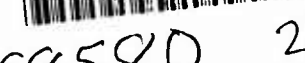
MTL MS 92-2

(2)

AN INTERACTIVE OFFICE ACCOUNTING PROGRAM WRITTEN IN UNIX AND C

KAREN M. KINSLEY
POLYMER RESEARCH BRANCH

July 1992



US ARMY
LABORATORY COMMAND
MATERIALS TECHNOLOGY LABORATORY

U.S. ARMY MATERIALS TECHNOLOGY LABORATORY
Watertown, Massachusetts 02172-0001

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

Mention of any trade names or manufacturers in this report shall not be construed as advertising nor as an official indorsement or approval of such products or companies by the United States Government.

DISPOSITION INSTRUCTIONS

Destroy this report when it is no longer needed.
Do not return it to the originator

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER MTL MS 92-2	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AN INTERACTIVE OFFICE ACCOUNTING PROGRAM WRITTEN IN UNIX AND C		5. TYPE OF REPORT & PERIOD COVERED Final Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Karen M. Kinsley		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS U.S. Army Materials Technology Laboratory Watertown, Massachusetts 02172-0001 SLCMT-EMP		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Laboratory Command 2800 Powder Mill Road Adelphi, Maryland 20783-1145		12. REPORT DATE July 1992 13. NUMBER OF PAGES 234
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer programs
Accounting
UNIX language
C language
Programming languages

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(SEE REVERSE SIDE)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block No. 20

ABSTRACT

This program is written in UNIX (Version 5.1, PO1) and C (pre-ANSI) and provides an office with the ability to create and maintain labor and non-labor accounts without requiring anyone in the office (except for the installer) to learn a computer language. Security for the accounts and account directories is automatically provided, and menus are used to create the accounts and to specify the charges to them. The program also includes creation of a summary file for all accounts, key word search functions for both the summary file and general accounts, and a printout program. A separate program, which allows an operator to view, print, and/or search the accounts for key words, but which will not allow the addition of charges to the accounts, is also provided.

CONTENTS

DISCLAIMER

THESE PROGRAMS ARE SUPPLIED AS-IS. NEITHER THE AUTHOR NOR THE UNITED STATES GOVERNMENT ASSUMES LIABILITY FOR THE USE OR MISUSE OF THE SOFTWARE CONTAINED IN THIS REPORT. THE USER ASSUMES FULL RESPONSIBLITY FOR ALL CONSEQUENCES OF THE USE OR MISUSE OF THE CONTENTS OF THIS REPORT.

LISTING OF TRADEMARKS

THE FOLLOWING CONSTITUTE THE KNOWN TRADEMARKS USED IN THIS REPORT. NO ENDORSEMENT OF THE COMPANIES AND PRODUCTS LISTED BELOW BY THE AUTHOR OF THIS REPORT OR BY THE UNITED STATES GOVERNMENT IS INTENDED OR IMPLIED.

MS-DOS is a registered trademark of Microsoft Corporation.

PC is a registered trademark of International Business Machines Corporation.

UNIX is a registered trademark of American Telephone and Telegraph Corporation (AT&T).

INTRODUCTION

This program enables those who have accesss to a computer with a UNIX operating system and C Language capability (such as are frequently used to generate electronic mail) to keep office accounts on that computer. It is designed for the office manager at a branch level, but may be suitable for other levels of users.

The software is user-friendly, and utilizes a question-and-answer format for determining which files to open and which actions to perform. More than one person may access the programs and files (for example, a manager, a timekeeper, and one or more secretaries), without the necessity of learning one or more computer languages. The only person who is assumed to have some knowlege of both UNIX and C is the installer, but the the operator in whose directories the actual data files reside (where different from the installer) should have some knowlege of UNIX and/or one of the editors generally available to e-mail users (as long as it is able to be used on non-mail text files). The other operators may use the system without having to learn a computer language.

General functions performed by this software include setting up file directories (with appropriate permissions for use), creating files, facilitating user input of accounting data, viewing and printing data files, generating an updated account status file, and searching the status file and account files for operator-selected key words or phrases.

There are two types of data files generated by the software: labor and non-labor. The types of charges that can be applied to accounts are dependent upon the nature of the account, and are as follows:

Labor	Non-Labor
1) Lump-sum addition or deletion of funds.	1) Lump-sum addition or deletion of funds.
2) Changes in expiration date.	2) Changes in expiration date.
3) General labor charges to multiple accounts.	3) General purchase costs.
	4) Travel charges.
4) Addition or deletion of labor charges to individual accounts.	5) Award costs.
5) Revise avg. cost/hour and recalc. balance.	6) Training costs.
6) Miscellaneous charges.	7) Miscellaneous charges.

Corrections to the labor and non-labor accounts can also be made, when necessary. In addition, the accounts are protected from overcharging by disallowing any transaction that will cause the balance to fall below $0.00 (although charges that will cause the balance to become exactly $0.00 are allowed after the user is warned that this will occur). No account with a balance of $0.00 can be used unless and until more money is added to the account.

One additional protection of data files included in the software is that

no charges are permitted to be added to a file if its expiration date has been exceeded, regardless of the account's current balance. Error messages are generated which inform the operator of the account's status, and the oportunity to change the expiration date is provided.

I would like to thank Dr. Wenzel E. Davidsohn for initiating this project and for his support during the development of this program, Maureen Woodward for her suggestions regarding the labor programs, and Jean McCarthy for beta testing this complete package. I would especially like to thank Amada Lugo, who described what functions she needed for the non-labor programs, and alpha tested the program with me. Her comments on the clarity of the prompts, the need for printing as well as simply scrolling the files on the screen, the necessity of searching all files for specific data, and on the presentation of material in the Program Users' Guide were invaluable to the success of the project.

It is my hope that those who use this program will find that it speeds the tracking of funds and makes their jobs a little easier. Please feel free to write to me with any comments or questions.

Karen M. Kinsley
US Army Materials Technology Laboratory
Mail Stop SLCMT-EMP
Arsenal Street
Watertown, MA 02170-0001

DSN: 955-5503
Comm: 617-923-5503
E-mail: kkinsley@watertown-emh1.army.mil

INSTALLATION GUIDE

GENERAL INFORMATION

The programs used in this software package were written in two languages, C (pre-ANSI, no function prototyping allowed), and UNIX, version 5.1 P01 (Bourne Shell only). This guide assumes that the installer (who is not necessarily one of the users--see the Program Users' Guide) is familiar enough with both languages to modify the programs as his or her office requires. One such modification could be to take advantage of windows, which were not available to me and were not therefore included in the programming.

The programs should be checked for compatibility with your system; I recommend using lint for the C programs and the shell command sh -vx for the UNIX programs. This shell command can also be used within the vi editor, at the colon (:) prompt, if it is preceded by an exclamation point (!).

GENERAL ORGANIZATION

The program used to initiate commands is startup.acc. It is assumed to reside in one person's top-level directory initially (not necessarily the installer's); this person shall be called OPDAT (for OPerator--DATa holder) from this point onward. Recommended permissions for this program are:-rwxr-----. Other operators will have to copy or link it to their top-level directories. (See the Program Users' Guide. If they link to the file, change the permission to -rwxr-x---.) Most other programs and data are to be stored in a subdirectory of OPDAT's, named account.d in this report. (See Note 1, below.) This directory should be granted permissions of drwxr-x--- for all times except at the beginning of the fiscal year, when a new data directory is to be added (see below). Group write permission should be added at this time, if OPDAT is not the person to create the data directory. For this reason, OPDAT (if not the installer) should have a working knowlege of basic UNIX commands.

Most other UNIX and C programs, and necessary subdirectories reside within account.d (see directory trees in Appendix A, and Note 1, below). Subdirectory temp.d is the location of most of the temporary files generated by the programs, and is both user and group writeable. The directories for the actual data files are also listed within account.d, in the form account.fyXX.d; the XX is substituted by the relevant fiscal year. The data directories and their permissions are created by the UNIX program setup.1 (see below), and contain several subdirectories.

Even though account.d itself is (assumed to be) protected from tampering by others, read, write, and execution privileges should be denied to others for all programs and directories (see below), and write privileges for programs should be denied to the group. Data directories also grant write privileges to the group and are created via setup.1 (below). (Note: Directory temp.d is created by the installer, not by one of the programs.)

The programs and directories within account.d are:

account.fyXX.d	chgnonlb.x	form.gen.c	keyword	setup.1
anscheck.c	date.get	gen.labor	labor.form	setup.2
anscheck.x	date_it.c	getinfo.c	masslabr.c	sum.search
cfunct.h	date_it.x	getinfo.x	masslabr.x	temp.d
checkdir	edit.data	indlabor.c	nonlabor.form	update
chgnonlb.c	findit	indlabor.x	printit	yes.or.no

The conventions used to designate files and directories are:

Suffix	File Type	Mode
.c	C Language Program.	640
.d	Directory.	770
.h	C Language Header File.	640
.x	Executable Form of C Program.	710

All other files contain UNIX Bourne Shell programs (mode 750), except for labor.form and nonlabor.form (mode 640), which contain headers for the labor and non-labor data files respectively.

DATA FILES--STRUCTURE AND ORGANIZATION

Data files are considered to be either labor or non-labor, as noted in the introduction. Therefore, within each directory of the form account.fyXX.d, the program setup.1 creates two subdirectories, labor.d and nonlabor.d. It also will automatically create several subdirectories to labor.d and nonlabor.d, which the user is then free to modify (see Program Users' Guide). As noted above, the data directory modes are set to 770.

Individual data files will then be placed in one of the subdirectories to labor.d or nonlabor.d by the program setup.2 and grant read and write permission to the user and the group, but deny any privilege to others (mode 660).

The subdirectories under labor.d are:

6.1.d	6.2.d	6.3.d	ILIR.d	customer.d	overhead.d

The subdirectories under nonlabor.d are:

6.1.d	ILIR.d	overhead.d
6.2.d	customer.d	training.d
6.3.d	misc.d	travel.d

Samples of labor and non-labor data files are included in the Program Users' Guide.

A brief description of each program is given below, as well as a listing of the programs that each calls and of those that call it:

Program	Description	Calls	Called By
anscheck.c anscheck.x	Verifies input answers to questions; called by UNIX programs. This is more general than yes.or.no (see below).	None.	edit.data gen.labor keyword, setup.1 setup.2 startup.acc sum.search
checkdir	Creates a named directory if it does not exist and adds permission protections.	None.	setup.1
cfunct.h	C Language Header File.	None.	chgnonlb.c (.x) date_it.c (.x) getinfo.c (.x) indlabor.c (.x) masslabr.c (.x)
chgnonlb.c chgnonlb.x	Adds data to the non-labor files.	cfunct.h	edit.data
date.get	Creates a file with today's date and day within the fiscal year.	date_it.x	edit.data gen.labor update
date_it.c date_it.x	Reads today's date, calculates the day in the fiscal year and adds the information to the date file.	cfunct.h	date.get
edit.data	Adds data to individual labor or non-labor files, and allows view and/or print of individual data files.	anscheck.x chgnonlb.x date.get indlabor.x yes.or.no	startup.acc
findit	Determines the path of an account file and the number of files with the same name but which occur in different directories.	None.	masslabr.c (.x)
form.gen.c (*) form.gen.x	Creates the files which contain header forms for the labor and non-labor data files.	None.	None.
gen.labor	Adds general labor charges to batches of 25 accounts at one time.	anscheck.x date.get masslabr.x yes.or.no	startup.acc

Program	Description	Calls	Called By
getinfo.c getinfo.x	Creates the information needed for input to the summary file.	cfunct.h	update
indlabor.c indlabor.x	Adds data to the individual labor account files.	cfunct.h	edit.data
keyword	Searches all labor and non-labor files for user-selected key words or phrases.	anscheck.x nonlabor.form yes.or.no	startup.acc
masslabr.c	Adds general labor charges to up	cfunct.h	gen.labor
printit	Prints all labor and/or non-labor accounts, and/or the summary file.	update yes.or.no	startup.acc
setup.1	Creates the data directory and the labor and non-labor subdirectories, and sets directory permissions.	anscheck.x checkdir yes.or.no	startup.acc
setup.2	Writes the initial information to each labor and non-labor account. Also sets the file permissions.	anscheck.x labor.form nonlabor.form yes.or.no	startup.acc
startup.acc	Overall control program for the accounting software package.	anscheck.x edit.data gen.labor keyword printit setup.1 setup.2 sum.search update yes.or.no	None.
sum.search	Searches the summary file for a key word or phrase.	anscheck.x yes.or.no	startup.acc
update	Creates the updated summary file.	date.get getinfo.x yes.or.no	startup.acc printit
yes.or.no	Returns a value of "Y" or "N" to a calling program, in response to a yes or no question.	None.	edit.data gen.labor keyword, printit setup.1, setup.2 startup.acc sum.search update

Note: *) Form.gen.x must be run before any of the account files can be set up. The labor and non-labor file forms must be available to their calling programs (see above).

1) If files are to be secured from tampering by the group (i.e, by denying write privileges to the group), but access to the data for report generation or other reasons by group members is necessary, an abbreviated version of startup.acc exists, and is called startup (see Appendix E).

   This program can be copied to (or linked with) the root directories of those who require access to that data. The UNIX programs called by startup reside in a directory of OPDAT's called /usr/OPDAT/small.acc.d (mode 750), and are listed in Appendix E. The permission mode of these programs is 750.

   These UNIX programs are used to conduct keyword searches of account files and the summary file, for printing the results of such searches (and/or storing the results in the user's root directory), and/or for viewing and/or printing the contents of one or all data files. Entering data to the account files is not permitted by startup. The use of these programs is detailed at the end of the Program Users' Guide.

   In addition to the programs listed in Appendix E, small.acc.d contains a subdirectory called temp.d, which is user and group writeable (mode 770). It also contains the executeable version of anscheck.c (anscheck.x, mode 710), and the two header files (labor.form and nonlabor.form, mode 640). The complete directory listing for small.acc.d is therefore:

anscheck.x	keyword.2	nonlabor.form	sum.search.2	yes.or.no.2
edit.data.2	labor.form	printit.2	temp.d	

2) If more than one operator is to input data, but only certain members of a group should see the account data, one suggestion is to consult with your system operator to redefine the group to include only those who should access the files.

   If this is not feasible, you may wish to consider using filter programs, which use the /usr/bin/id shell command to identify the user, and compare the user to a list of acceptable operators, or which establish a password for identification. Startup.acc could then be called from within the filter program after verification of identity. If an unauthorized user is detected, the program will halt execution and exit back to the user's directory, with an error message. Data file permissions should be set to -rw------ with executable files incorporating set user identification (SUID) permissions and commands. See reference 5, chapter 4, and your system operator for the proper method of using SUID commands within shell and C programs, and the dangers involved in granting that permission.

   In conclusion, for specific information on your system's capabilities and the possible levels of protection available, consult with your system operator. See reference 5 for general information regarding UNIX security programming and for programming examples.

3) Entrance to OPDAT's account.d directory, and exit back to the user's

directory, are accomplished within startup.acc. All other programs assume that the user is within account.d before that program is called.

This is not true of the programs called by the abbreviated program, startup (listed in Appendix E, and described in Note 1, above). These programs must be modified by listing the correct path to "/usr/OPDAT/account.d" at the start of each program.

4) There are a number of temporary files generated by the software package; most have the prefix "temp.". If the programs are to reside in an existing directory (i.e., if temp.d is not created), OPDAT should be certain that none of these files will overwrite current files. Temporary files generated by these programs will be deleted by the programs themselves.

5) All printouts are made via the command "wpr", which sends the output file to the local printer. The programs which call for printouts are edit.data, gen.labor, keyword, printit, and sum.search. (See also edit.data.2, keyword.2, printit.2, and sum.search.2, which are listed in Appendix E.)

6) All files are displayed on the screen via the "pg" command. The use of the "more" filter is also explained in the Program Users' Guide, if "pg" is not available on your system. The programs which use "pg" are edit.data, gen.labor, keyword, and sum.search. (See also edit.data.2, keyword.2, and sum.search.2, which are listed in Appendix E.)

7) The following programs assume a fiscal year of 1993 when the current fiscal year is requested: edit.data, gen.labor, keyword, printit, setup.1, setup.2, sum.search, and update. This value should be updated yearly. (See also edit.data.2, keyword.2, printit.2, and sum.seach.2, which are listed in Appendix E.)

8) The UNIX programs edit.data, gen.labor, setup.1, and setup.2 contain warnings not to use the <backspace> or <rubout> keys, because they will not be interpreted correctly. This may or may not apply to your system. (See also edit.data.2, listed in Appendix E.)

9) C Language programs open the account data files in "a+" mode, to prevent data from being overwritten. Corrections to individual accounts can be made by the user at the time the error is detected, via edit.data (see the Program Users' Guide, startup.acc menu selection C) or a UNIX editor. However, if a UNIX editor is used, the file format must be preserved: the C programs require formatted I/O to properly enter and retrieve data.

10) The account programs assume that all LABOR account (or XO) numbers are no more than five characters in length (non-labor files may be up to fourteen characters, which is the limit of my system). This is important only to the program masslabr.c, within the labhour structure variable, and is easily altered by changing the defined constant ACCLEN.

Masslabr.c also assumes that a file path which begins at the labor.d directory will define the location of that file within 24 characters (e.g. ./6.1.d/TEST1). If this is not a sufficient length, it may be

altered by changing the defined constant PATHLEN.

11) Indlabor.c and masslabr.c both use a defined constant MINWAGE, which sets the current minumum wage for the programs' use. At the time of this writing, this is $3.75/hour, but should be updated when new rates are determined.

12) It is advisable to check the chgnonlb.c and the indlabor.c program functions to be certain that the transactions described are relevant to your office (e.g., rewards may not be granted at your level) and that sufficient space is allotted to document numbers, to allow tracing of charges. The non-labor purchase function, for example, assumes that two document numbers are required for tracing: an office file number, and the actual purchase order number.

    For all charges, a transaction description string is 30 characters long, with the thirtieth character being the '\0' terminator character. (For reference, see the sample labor and non-labor account files appended to the Program Users' Guide.)

13) Conversion of the UNIX programs to MS-DOS batch files will depend upon the version of MS-DOS the installer is running on his or her PC. Some considerations include MS-DOS support of while loops and case blocks, and the formats of the if block and for loop.

    Some standard substitutions of MS-DOS commands for UNIX include: "del" for "rm", "rem" for "#", "cls" for "clear", "%1" for "$1", and "ren" for "mv". Please refer to your manual for other replacements.

    If required, a combination of password protection for startup.bat (startup.acc) and turning on the hidden and read-only attributes whenever the other batch, executable, and data files are not in use will provide some degree of protection against casual users of the system. The best guard against tampering of the data files would be to physically limit access to the PC, and to periodically back up all files to portable diskettes, which can then be placed in a secure location.

Starting the Program

This guide assumes that all programs reside initially within the directories of a single person, who shall be called OPDAT (for OPerator--DATa holder) from this time on. This guide also assumes that the user knows nothing of UNIX, but that OPDAT does have some knowledge of basic UNIX commands and/or of text file editors which are often available through electronic mail programs.

Before you can start using the accounting program package, you have to get the program called "startup.acc" from OPDAT, and make it available to you (i.e., "link" the program to your directory). In order to do this, you need the path name, or computer address, of where she or he has stored startup.acc; it should look something like this: "/usr/OPDAT/startup.acc". In order to use the program, enter the following at the computer prompt (here assumed to be "$>"):

```
$> ln /usr/OPDAT/startup.acc .    (Note:  One space is required between "ln" and
                                          "/", and between "startup.acc" and "."
                                          (the "." is a part of the command).)
```

Once you have linked the startup program to your account, you can access the accounting package at any time by simply entering:

```
$> startup.acc
```

Your screen will clear, and you will see the following question:

```
Do you want to use the accounting program?
Enter y or n (for yes or no):
```

You can answer this, and all following questions, in either upper or lower case letters, as you please. Just typing the <enter> or <return> key will cause the following message to appear:

```
Please enter one of the following:  y Y n N.
```

This kind of message will appear throughout the programs in the accounting package, whenever you don't choose any of the available options. It will keep showing up until you do enter the data the program requires, so don't worry too much about hitting the wrong key.

Entering "n" or "N" (the quotes are used here and throughout the guide to set off the letters---don't enter the quotes in the answer!) will exit you from the program; you should next see the "$>" prompt on your screen.

Entering "y" or "Y" will bring the following menu to the screen:

Here are the available programs:

A) Create the directories (file categories).
B) Set up the account files.
C) View, print, and/or add charges to files.
D) Enter general labor charges to up to 25 separate accounts.
E) Create the file containing the updated status summary of all accounts in the specified fiscal year.
F) Search the summary file for operator-specified keywords.
G) Search the account files for operator-specified keywords.
H) Print all labor and/or non-labor files, and/or the summary file.

Enter a letter (A-H) or Q to quit the accounting program:

--Choose a letter for the line containing the desired program, or "q" to leave the program and get the "$>" prompt.

Introduction to the Programs

Before going into a detailed explanation of how each program works, here is a short introduction to the first four programs listed above:

A) Create the directories (file categories).

Before any accounting can begin, the file categories must be set up. This program will create them, from a pre-set listing (such as 6.1.d, travel.d, etc., where the ".d" suffix indicates that what is being created is a directory, or accounting category, not an actual account file). The listing may contain different titles, depending on OPDAT's choice of directories. You can also call it at any time to create a new directory.

Note: This is the first program to be called if you have never used the accounting program before in the current fiscal year (it must be called at least once in a fiscal year). If you have used this program once in the fiscal year, and are satisfied with the file categories, you do not have to call it again.

B) Set up the account files.

This program lets you enter the basic information required to set up new files (both labor and non-labor) and depends on menu selection A to have already established the required directories (file categories). The best way to use this program is to set up several accounts in one sitting. You can not add any charges to existing accounts by using this program (for that you need to choose C or D from the menu), but you can create new accounts at any time.

C) View, print, and/or add charges to files.

This is a multi-purpose program, which allows you to access, add charges to, view, and/or print the individual labor and/or non-labor files. All charges to non-labor accounts are entered through this program, as are charges to individual labor accounts. This is the program which adds all the types of charges mentioned in the introduction, except for the batch processing of labor charges. Those are entered via menu selection D.

D) Enter general labor charges to up to 25 separate accounts.

The sole purpose of this program is to speed the entering of labor charges made to multiple accounts. If there are any errors encountered in processing (for example, file balance is $0.00), they will be noted in a separate error file, which you can later print, if you wish. This program cannot be used to add any other charges to a labor account, nor can it be used to add any charges to a non-labor account.

PROGRAMS---DETAILED INSTRUCTIONS

Note: To avoid confusion between what is being displayed on the screen, and my explanation of it, I will start my comments with a double dash (--) where necessary; my comments will also be indented.

A) CREATE THE DIRECTORIES (FILE CATEGORIES).

--Entering choice A from the startup.acc menu will bring the following message to the screen:

This is the initial program used to set up the accounting files.
It prepares the directories (file categories). Usually you need to call
this program only once in the fiscal year.

A help message: for these programs, DO NOT use the backspace
key to erase a character. The correct key is <ctrl><h>.

Do you want to set up the file categories for the account files?
Enter y or n (for yes or no):

--Answering "n" or "N" will return you to the startup.acc menu.

The help message may or may not apply to your system. On the UNIX system that I used, hitting the arrow keys or the backspace key either generated strange-looking characters on the screen, or entered erroneous commands.

If you do choose to continue, you will see:

Enter the last two digits of the fiscal year (i.e. 91 for 1991),
or else just hit the <enter> key if the fiscal year is 1993:

--If OPDAT has updated the program, hitting the enter key should enter the current fiscal year, if not 1993. Please enter two digits only; the program will not accept the entire year as an answer. The reason is that the program uses the fiscal year as part of the account directory title, and space for names is limited by the operating system.

The program will then repeat the year back to you, and ask if it's acceptable. If not, you can re-enter it.

Once the fiscal year has been established, the next message should look something like this:

Here are the directories (file categories) for the labor accounts:

6.1.d 6.2.d 6.3.d ILIR.d customer.d overhead.d

Are these acceptable?
Enter y or n:

--Entering "y" or "Y" means that the directories are fine, and that there is no need to rename, delete, or add a directory. (By the way, the ".d" suffix is used to indicate that these are directories, not actual account files.)

Entering "n" or "N" will clear the screen, re-list the directories, and bring the following menu to the screen:

```
Add a directory (a), change a directory name (c)
delete a directory (d), or go to the next part of the setup
program (g)?
```

******

--Entering "g" or "G" means that the directories as listed above the menu are now acceptable (before or after changing---you may decide that the initial listing is OK after all).

******

--Inputting "a" or "A" causes the screen to re-list the directories, and the following message to appear:

```
Enter the name of the directory to be added.  It can have no more
than 15 characters, and the following characters cannot be used:
single or double quotes / \ * ; - ? [ ] ( ) ~ ! $ { } > <
or spaces at the start of or within a directory name.
```

--Enter the name of the new directory. (For a beginner's reference to UNIX naming rules, and UNIX in general, see ref. 4.) UNIX does distinguish between upper and lower case letters for file and directory names, so be careful to enter the new name as you want it to appear. After the name is read, the screen will redisplay the directories, and more changes can be entered, as desired.

If you should input the name of an existing directory, the computer will give you an error message.

If the computer does not accept the name you have entered (for example, if you just hit <enter> without typing any letters), you will see the following message:

```
The directory could not be added as written.
Enter any key to continue:
```

--You can just hit <enter> or <return>, if you want, and the directories will be displayed.

******

--Entering "c" or "C" will cause the directories to be repeated to the screen, and the following lines will be written:

```
Enter the old name of the directory.  Please copy it exactly,
using lower or upper case letters as shown above:
```

--Input the name of the directory. If you just type <enter> or <return> without any other letters, you will get the following error message:

```
The directory could not be renamed.
Enter any key to continue:
```

--The directory listing will again appear on the screen, once the key is entered.

If you have misspelled the old name of the directory, the screen displays:

```
The directory you have chosen does not exist.
Enter any key to continue:
```

--The directory listing will be shown, once the key is entered.

If you have entered the old name of the directory correctly, the screen will next display:

```
Enter the new name of the directory.  It can have no more than
15 characters, and the following characters cannot be used:
single or double quotes  / \ * ; - ? [ ] ( ) ~ ! $ { } > <
or spaces at the start of or within a name.  It also can not
be the same name as another directory.
```

--Enter the new name. If you have hit the <enter> or <return> key only, you will see:

```
The directory could not be renamed.
Enter any key to continue:
```

--The directory listing will again appear on the screen, once the key is entered.

If another directory already has the new name, you will see:

```
The directory could not be renamed because another
directory already has that name.

Enter any key to continue:
```

--The directory listing will again be shown.

Note: If at any time you see an error message such as: "mv: permission denied", then OPDAT will have to rename the directory for you.

******

--Entering "d" or "D" will re-list the directories, and cause the following message to appear:

```
Enter the name of the directory to delete (upper and lower case
letters must exactly match).  If you don't want to delete a
directory, then just hit the <return> or <enter> key:
```

--If you should see a message like: "rmdir: permission denied", then OPDAT will have to delete the directory for you. If you see this message:

```
Your directory could not be deleted.
Please enter any key to continue:
```

--then you have probably misspelled (or used upper case instead of lower case letters or vice versa) the directory name. Entering a key (or just hitting <enter> or <return>) will redisplay the directories.

******

--You may see a message such as: "chmod: permission denied". If so, have OPDAT make sure that the files are secure from tampering by people outside the group, and that you are able to add files to the directory.

*****

--Once the labor account directories are corrected, entering "g" or "G" will display the non-labor directories. They are edited in exactly the same way as the labor directories above.

*****

--At the end of the program, the screen will clear, and you will see:

This is the end of the file category (directory) setup program. The next step is to create the actual accounts. You do this by choosing menu selection B on the startup.acc menu screen (which follows).

Enter any key to continue:

--Just press <enter> or <return> to go back to the startup.acc menu.

B) SET UP THE ACCOUNT FILES.

--Selecting choice B from the startup.acc menu will clear the screen and display the following message:

This is the program used to create the account files. It may be used at any time in the fiscal year.

Warning: You must have used menu selection A from the startup.acc menu to create the file directories (categories). If you have not yet done so, quit this program by entering N or n to answer the question below, then select choice A on the startup.acc menu screen.

A useful hint: DO NOT USE the <rubout> or <backspace> key to correct the screen. Use <crtl><h> instead.

Do you want to set up the account files?
Enter y or n (for yes or no):

--Entering "n" or "N" will halt this function and return you to the startup.acc menu.

As before, the warning about not using the <rubout> or <backspace> keys may or may not apply to your system.

If you decide to continue, the program will request the fiscal year for the files. There are two important reasons for this: the fiscal year is a part of the account directory name, and while you may create account files for years other than that of the current fiscal year, you cannot add charges to them.

Enter the last two digits of the fiscal year, or just hit <enter>
or <return> if the year is 93:

--If the account directory (which should have been created by using choice A from the startup.acc menu) could not be found, and if the year 91 were entered, you will see:

There are no files for fiscal year 91. Choose another year?
Enter y or n:

--If the answer is "n" or "N", the accounting program will stop this function and you will next see the startup.acc menu.

Once the year is successfully input, the program will ask for information regarding the account itself:

Enter the account (or XO) number:
(Note: If this is a labor account, please use 5 characters only. If this is a non-labor account, you can use up to 14 characters.)

--Enter the account number. The labor restriction applies because of the way batch labor data (startup.acc menu choice D) is entered. If this is a problem, see OPDAT: the batch program is easily adjusted to meet your

requirements.

The account number will be repeated to your screen. For example:

```
This is the input account (or XO) no.: TEST1.
Is this number OK?
Enter y or n:
```

--If the account number is not acceptable, you will be able to re-enter it.

If you have not entered an account number (i.e., you just hit the <enter> or <return> key), you will see the following error message:

```
This is the input account (or XO) no.:  None.

*********************************************************
*  WARNING:  Not entering an account number will cause  *
*            the program to terminate.                  *
*********************************************************

Is this number OK?
Enter y or n:
```

--If you enter "y" or "Y", the account set-up program will end, and you will be returned to the startup.acc menu.

If you input "n" or "N", you will be asked to re-enter the acount number.

Once the account number is satisfactory, the program will ask:

```
Is this a labor (L) or non-labor (N) account? (Enter L or N:)
```

--The only difference between setting up a labor account vs. a non-labor is that a labor account will ask you for an average labor rate per unit hour, while a non-labor account will not have that line. Therefore, while the following discussion will be for labor accounts, it will also apply to non-labor.

The screen will next show the account directories available. These were discussed in menu choice A, above. (The directories given below may not be the same as those you will see on your screen.)

```
Here are the available file categories (directories):

6.1.d       6.2.d       6.3.d       ILIR.d      customer.d  overhead.d

Choose a directory (file category).  Copy only enough letters
to fully identify the directory.  Use upper and lower case
letters as shown:
```

--If you do not choose an existing directory, the following will appear:

```
The directory you have entered does not exist.  Choose another directory?
(If the answer is "n" or "N", this part of the program will end.)
Enter y or n:
```

--If the answer is "y" or "Y", the directory list will then be re-displayed.

If the the directory already has an existing account with the name you have chosen, you will see an error message. For example:

Account No. TEST1 already exists in this directory.
In a moment, this program will ask you for a new account no.

--The program will then re-start at the "Enter the account (or XO) number" prompt.

If there are no problems, the computer will ask you for the account information; enter the information as requested. Once the data has been input, it will be repeated back to the screen for correction. Just hitting the <enter> or <return> key will cause an entry of "None." or "None Entered." to the file, unless otherwise noted.

The program will request:

Enter the purpose for the account (i.e. CW Research, Office Purchases, etc.) (50 characters or less):

Enter the name of the Principle Investigator (if any), or just hit the <enter> or <return> key (if none):

Enter the starting balance, in dollars. Do not include letters, commas, or the dollar sign (e.g. use 10000.00, not 10K or $ 10,000). (Simply hitting <enter> (without a number) will set the balance to $ 0.00.)

Enter the average cost per hour of the labor charges, in dollars. Do not use commas or the dollar sign. (Just hitting <enter> or <return> will automatically enter a value of $ 40.00:)

--It is important to enter the starting balance and the average cost per hour as requested because the language in which the programs to add labor and non-labor costs are written do not recognize numbers with commas in the middle. The dollar sign will be input automatically by this program to the file; including it here could cause the computer to interpret the number to be a file name rather than a simple number.

If there are no cents, however, the ".00" at the end of the balance can be omitted.

Note: The cost-per-hour prompt will not appear if the account is non-labor.

Enter the month, day, and year of the expiration date, as follows: mm/dd/yr, or hit <enter> or <return> if the date is 9/30/93:

--If OPDAT has updated the program, the proper expiration date should be displayed.

Once the data is entered, the screen will clear, and your input data will be repeated back to you. For example:

```
Here are the data you've entered:

  A) Account Number (or XO Number): TEST1
  B) Classification: Labor
  C) Purpose for Account: Test Setup Program
  D) Principle Investigator: Me
  E) Starting Balance: $ 2000
  F) Account Expiration Date: 9/30/93
  G) Rate/Hour: $ 40.00

Are these data OK?
Enter y or n:
```

--If these are all correct, the program will then create a file, and ask you if you want to add another account (see below). Otherwise, the program will request:

```
Enter the letter of the line to be changed
(A, B, C, D, E, F, G), or if no changes are desired,
just hit the <enter> or <return> key:
```

--Entering "a" or "A" will cause the program to assume that none of the information that has been entered so far is correct, and will set the program back to the beginning of the input section.

If you enter "b" or "B", the program will assume that only the account number is right, and will start asking for information from the "Is this a labor (L) or non-labor (N) account?" prompt.

Any other letter will ask you for the corrected data for that line, and will then repeat all the lines to the screen; the process can continue indefinately, until the data is all correct.

Note: Line G will not appear if the account is non-labor.

*****

--Once the data is correct, the computer will ask:

```
Add another account?
Enter y or n:
```

--Entering "y" or "Y" will cause the screen to display:

```
Enter the account (or XO) number:
(Note: If this is a labor account, please use 5 characters only. If
       this is a non-labor account, you can use up to 14 characters.)
```

--As before, you can enter an account number, or none. The same error message as before will result if no account is named.

If the account number is acceptable (and if you have chosen directory 6.1

(short for 6.1.d)), then you will see:

```
Is the account in the same file category (6.1)?
Enter y or n:
```

--If the answer is yes, the computer will check to see if another account with that name already exists; if so, then you will be prompted for a new account number, and all information.

If the answer is yes, and the account doesn't yet exist in that directory, then the program will ask for data from starting from the "Enter the purpose for the account..." prompt.

If the answer is no, the program will display:

```
Is this another labor account?
Enter y or n:
```

--If the answer is yes, then the labor directories will be displayed, and the data will be requested from that point, as before. If the answer is no, the program will return to the "Is this a labor (L) or non-labor (N) account?" prompt, and request all information from that point.

Note: For a non-labor account, the prompt will read:
"Is this another non-labor account?"

******

--When all of the new accounts have been added for this session, entering "n" or "N" at the "Add another account?" prompt will cause the following message to be displayed:

```
Choice C on the startup.acc menu (which follows) will activate the
program that adds charges to individual labor and non-labor accounts.
To add general labor charges to multiple accounts, select choice D from the
startup.acc menu screen.  You can run these programs at any time,
provided that the account file already exists.

Enter any key to continue:
```

--Entering any key, or just hitting the <return> or <enter> key, will return you to the startup.acc menu screen.

C) VIEW, PRINT, AND/OR ADD CHARGES TO FILES.

--Selecting option C from the startup.acc menu generates the following message:

This program finds files and enables the user to add to, and/or view, and/or print the charges in the individual account files.

Warning: You must already have created the files and directories (file categories) using Choices A and B from the startup.acc menu. If you haven't created the directories, or need to add one, enter n or N at the next question to exit this program, then choose A from the startup.acc menu.

If you need to create a file under an existing directory, exit this program as above and then choose B from the startup.acc menu.

A helpful hint: DO NOT use the <backspace> or <rubout> keys to correct input errors on the screen. Use <crtl><h> instead.

Do you want to use this program?
Enter y or n (for yes or no):

--If you choose to continue, you will be asked to enter the fiscal year. As noted for startup.acc menu choice B (above), the choice of fiscal year is important, because it forms part of the name of the file directory, and also because you can only view and/or print a file in a directory outside the current fiscal year (you cannot add charges to it).

Enter the last two digits of the fiscal year, or just hit <enter>
or <return> if the year is 93:

--If the program cannot find the directory containing the files for the specified year (for ex., 89), you will see:

There are no files for fiscal year 89. Choose another year?
Enter y or n:

--If the answer is "n" or "N", you will be returned to the startup.acc menu.

Once the year is correct, the program will ask you for the name of the account file. The process of entering the name of a labor or non-labor account, and choosing a directory (if more than one account exist with the same name, but reside in different directories---see below), are identical for both types of accounts. The real difference is in the kinds of information that can be added to each account, and that will be noted where appropriate.

For both labor and non-labor accounts, note that only one file may be worked on at a time. It is best, therefore, to gather all of the travel orders, purchase requests, etc., that apply to each single account, and to enter all of the changes to that account before going on to the next.

The program will request the account number:

```
Enter the account (or XO) number (same as the file number in the
computer), or enter q to quit:
```

--Entering "q" or "Q" will return you to the startup.acc menu.

If you enter a file number, the computer will repeat the number to the screen. For example:

```
This is the input account (or XO) number: TEST1.
Is this number OK?
Enter y or n:
```

--Entering "n" or "N" will generate a request to input the account number.

Not entering a number will cause the account number to be set as "None" and generate the following message:

```
This is the input account (or XO) number: None.

**********************************************************
*  WARNING:  Not entering an account number will cause  *
*            the program to terminate.                  *
**********************************************************

Is this number OK?
Enter y or n:
```

--Entering "y" or "Y" will return you to the startup.acc menu.

If the account number is correct (and not the same as "None"), the program will ask:

```
Is this a labor (L) or non-labor account (N)? (Enter L or N:)
```

--Enter the appropriate account type. The next question is:

```
Add charges to account (A), or simply look at (and/or print) the
account (L)?  (Enter A or L:)
```

--Enter "A" or "a" or "L" or "l" as appropriate. "L" or "l" will cause the program to skip over the charge input programs (see below).

The program next looks for the account file within the appropriate (labor or non-labor) directory. If it can't find the file (here, TEST1, a labor account), it will state:

```
Account #TEST1 either does not exist, or
does not exist as a labor account.
```

--Or, if TEST1 was supposed to be a non-labor account:

```
Account #TEST1 either does not exist, or
does not exist as a non-labor account.
```

--The computer will then go back and ask you for the account number, as above.

If the program finds a file TEST1 in two different file categories (for example, travel.d and purchases.d), it will ask you to pick the correct file:

```
Account #TEST1 was found under these directories:

travel.d
purchases.d

Please enter the correct directory name, copying only enough
characters from the list above (using upper and lower cases
as shown) to completely distinguish the directory from its neighbors:
```

--Enter the correct name (remember that UNIX does distinguish between lower and upper case letters, unlike most personal computer command languages). If it can't find the directory as you've entered it, you will see:

```
The directory you have entered does not exist.
Enter another account (a), try another directory (d),
or quit this program (q)?  (Enter a, d, or q:)
```

--Enter the appropriate response.

Once the account file has been located (for example, TEST1), and if you have decided to add data to the file, the program will ask:

```
Look at account TEST1 before making changes to it?
Enter y or n:
```

--If you enter "y" or "Y", the program will display the file. The display will occur in one of two ways: if you see the word "More" at the bottom of the screen, then hit the space bar to advance to the next screen, or to advance to adding charges to the file. If you see ":" or "(EOF):" at the lower left corner, hit <enter> or <return> to advance to the next screen or to go on to add the charges to the file, respectively.

Notes

1) When entering labor and non-labor charges in terms of dollars:

   If the charge you're entering doesn't have any cents, you don't have to put them in. The program will put in the ".00" for you, if it is not there. If there are cents, however, please be sure to enter them.

2) If you find that you have selected the wrong labor or non-labor function, and start to enter charges, entering total costs of $0.00 or total hours equal to zero will stop the program from asking you to repeat the entry (see below); the entry will also not be written to the file. The exceptions to this are changing the expiration

date, where no dollar or hour values are input, and changing the current wage rate/hour (labor accounts only), where a zero entry is an accepted value (but entering a rate/hour LESS than zero will allow you to escape without a file entry).

3) The non-labor program offers a correction function for entries that were made which were incorrect or were altered after entry. This is most useful for correcting miscellaneous items, but may be used for other (monetary) corrections. However, for corrections to travel, award, etc., entries, it is simpler to select the function and enter the negative amount that was previously charged, and the reason for the change in place of the name or other data requested. For ex., if a $500.00 trip were suddenly cancelled, you could enter:

   Travel Type = OCONUS,
   Travel Order Number = XX-123,
   Traveller = Cancel XX-120,
   Total Cost = $-500.00.

   (See below for the proper method of entry.)

   This is a better correction method for pre-set cost functions because a keyword search of the accounts (see below) will find all such corrections, and you will not have to remember the abbreviations (or worry about entering them incorrectly) used to represent the pre-set cost types.

4) Correcting erroneous labor charges to individual accounts may be accomplished through the add and delete labor charge functions, or via adding a negative value of the original cost if the miscellaneous charge function were used.

--When the computer opens the data file, it opens an additional file as well. If there are any errors in opening these files, the program will generate one of the following error messages, and stop the data input to the account files. If you see either of them, report them to OPDAT, and/or contact your UNIX system operator:

Can't open file containing today's date. Exiting program.

Can't open account file. Exiting program.

--If you have tried to open a file that is not in the current fiscal year, you will see the following message:

The current year does not match the fiscal year.
You may only view and/or print the file.

Enter any key to continue:

--The computer will then prompt you if you wish to view and/or print the file.

If the account is not new, the computer will then display the last transaction in the file. For example:

```
Initial Balance= $1800.00
Flag= --------
Date of Last Entry= 10/23/92
Prev. Transaction= Add Lab. ODN=106-92
Rate/Hour= $40.00
Prev. Hours= 10.00
Prev. Transaction Total= $400.00
Current Balance= $1400.00
Prev. Days Left= 39
Cum. Hours= 10.00
```

Enter any character to continue:

--The flags are as follows: B= Bankrupt, B*= Less than 20% remains of the original balance, TE*= Less than 30 days remain before the account expires, and TE= Time Expired.

Hitting the <return> or <enter> key will continue the program.

If the account were bankrupt, you would see:

This account is bankrupt (i.e. current balance = $0.00).
Enter A to add money to the account or Q to quit this part of
the program:

--Entering "q" or "Q" will bring you out of the "add labor charges" part of the program; you will then see a prompt to view and/or print the data in the file.

======

--If you do elect to add money to the account, you will see the following:

```
Add money to account TEST1?
Enter y or n (for yes or no):
```

--Entering "n" or "N" will exit you from the "add money" function.

If you decide to continue this function, it will ask you for documentation for adding the money:

```
Enter reason to add money to account (18 char. or less;
for ex.: ODN 47-92):
```

--If you don't have any documentation, just hit the <enter> or <return> key.

After entering the reason, you will be asked for the amount:

```
Enter amount of money to be added, in dollars.  DO NOT
use letters, the dollar sign ($), or commas (i.e. $1,000
or $10k are not acceptable):
```

--If you do enter the "$" or commas or "K", the computer either will not read the amount, or not read it correctly. If it can't read the amount to add, you will see:

```
Please re-enter amount of money to be added:
```

--If you try to enter a value less than zero, the screen will display:

```
Entering a value less than zero is not acceptable.
Please re-enter amount:
```

--Once the data is entered, it will be repeated back to you. For example:

```
The reason to add money is I want to, and the amount to be
added is $1400.00.  Are these values correct?
Enter y or n:
```

--If the entered values are not correct, the computer will ask you for them until they are correct.

If you do enter a value of zero for the money, there will be no entry made to the account, and the computer will exit the "add money" routine at this point. If the money entry were greater than $0.00, the computer will add the data to the account file, display the revised (current) balance, and will then display the following prompt:

```
Add more money to account TEST1? (Enter y or n:)
```

--Entering "y" or "Y" will re-start the process of adding money; "N" or "n" will exit you out of the "add money" mode.

Notes:
---

1) Since all charges to the account are entered as positive numbers, the addition of money (a credit to the account) is shown as a negative charge in the file.

2) If the account were initially bankrupt, and you did not add any money (i.e., you either entered "n" or "N" at the "Add money to account..." prompt, or entered a value of $0.00 when adding money), the program will exit from the "add labor charges" mode, and you will see a message to view and/or print the data file.

======

--If the account expiration date is today, you will see:

Today is the last day this account can be worked on.
Do you want to add more time? (Enter y or n):

--Entering "y" or "Y" will allow you to revise the expiration date; a value of "N" or "n" will simply continue the program.

If you have run out of time for this account, you will see the following:

The expiration date for the account balance has been exceeded.
Enter A to add time or Q to quit this part of the program:

--If you don't add time (change the expiration date), the "add labor charges" part of the program will end, and the screen will prompt you as to whether you want to view and/or print the data.

If you elect to change the expiration date, you will not be allowed to choose an expiration date earlier than your current date.

======

--If you choose to add time, you will then see:

Change the expiration date for account TEST1?
Enter y or n (for yes or no):

--Entering "n" or "N" will stop the "add expiration date" mode, without adding a change to the file.

A value of "y" or "Y" will cause the program to ask you for the date. If you don't enter the month, day, and year in terms of numbers, the computer will not be able to read the data and will ask you to repeat it.

Enter the number of the month (1-12) of the new date:

Enter the day of the new expiration date:

Now input the last two digits of the year:

--Enter each value as requested. Once the data has been entered, the program will repeat it back to you, to see if it's correct. For example:

You have entered the following date: 2/33/93.
Is this correct? (Enter y or n:)

--If the date is not correct, you will be asked to re-enter it. If it is correct, the computer will then check its validity. If the expiration date is outside that of the current fiscal year, or if you have entered an impossible date (i.e. 2/33/93), you will get the following error message:

The date you have entered is not acceptable.

--You will then be asked to re-enter the date.

If you have input a date that is earlier than the current date, but is still within the fiscal year, the program will state:

The date you have entered is earlier than today.
Please re-enter the date:

--Once the date is correct, the program will ask:

Please enter reason for changing the exp. date (18 char.
or less). (For ex.: Ref Memo 91-92):

--If you have no documentation, just hit the <enter> or <return> key. The computer will repeat the reason back to you, and ask if the reason is correct. If not, it will request the reason again.

When the date and reason for change are acceptable, the change will be written to the program file, and you will exit the "change expiration date" function.

======

--Note: If the expiration date for the file had been exceeded, and you had elected to change the date, but answered "n" or "N" at the "Change the expiration date..." question, the program will exit the "add labor charges" mode, and you will see a prompt to view and/or print the account file.

======

--Once the account balance and/or expiration dates have been changed (if the program had forced the revision(s)), you may also see the following warning message (if applicable):

The rate per unit hour of this account is currently set
to below the minimum wage of $3.75. This program will therefore ask for labor charges in terms of dollars, not hours.
To change this, see the change menu (next screen).

Enter any key to continue:

--Entering a key will clear the screen and the account number, current balance, and change menu will be shown. For example:

Account No. TEST1 has a balance of $1400.00 and expires in 38 days.

Here is the change menu:

A) Add money to account.
B) Delete money from account.
C) Change expiration date.
D) Change cost/hour.
E) Enter labor charges.
F) Delete labor charges.
G) Add miscellaneous charges.

Enter a letter, or Q to quit adding labor charges:

======

--Entering "q" or "Q" will stop all further addition of data to this account and will bring you to the view and/or print data prompt (see below).

======

--Entering "a" or "A" will allow you to add money to the account. This is the same function as discussed above, when adding money to the account to avoid bankruptcy.

======

--Choosing "b" or "B" will allow lump-sum deletions of money from the account file. You will see the following prompt:

Delete money from account TEST1?
Enter y or n (for yes or no):

--A value of "n" or "N" will exit you from the "delete money" function without making any changes to the file. A value of "y" or "Y" will cause the program to ask:

Enter reason to delete money (17 char or less;
for ex.: Ref. ODN 391-92):

--If you don't have any documentation, just hit <enter> or <return>. Once the reason is entered, you will be asked:

Enter amount of money to be deleted, in dollars. DO NOT
use letters, the dollar sign ($), or commas (i.e. $1,000
or $10k are not acceptable):

--As before, if you add commas or the dollar sign or the letter "K", the dollar value to be deleted will not be properly read by the program.

If the computer cannot read the amount to be deleted, you will see:

Please re-enter amount of money to be deleted:

--You may also see (if applicable):

Entering a value less than zero is not acceptable.
Please re-enter amount:

--When the amount is acceptable to the program, the program will repeat both the reason for the change and the amount to be deleted, and ask you if they are correct. Answering "n" or "N" will cause the program to once more ask you for both the reason and the amount.

Note: A (correct) value of $0.00 will not be entered into the account file; in fact, the program will leave the "delete money" mode, giving you a quick exit from this function.

If both the reason and the amount are correct, the computer will then attempt to deduct the amount from the current balance.

You may see one of the following messages, if applicable:

This transaction will cause the account balance to be
less than zero, and will not be allowed.

--Or,

Warning: This transaction will zero (bankrupt) the
account. No further deletions are allowed.

--If the account were not bankrupted, you will be given the current balance in the account, and you will then be asked:

Remove more money from account TEST1? (Enter y or n:):

--Entering "y" or "Y" will restart the "delete money" function; entering "N" or "n" will exit you from this function.

======

--Choice "C" or "c" changes the expiration date. This is the same function discussed above, where the account expiration date had to be changed in order to allow charges to be added to the file.

======

--Selecting choice "D" or "d" from the change menu give you the following prompt:

Change the avg. labor charge per unit hour for account TEST1?
Enter y or n (for yes or no):

--Entering "n" or "N" will exit you from this function. If you enter "Y" or "y", the computer will ask you:

Enter office no. of document authorizing change (9 char.
or less):

--If none, just hit the <enter> or <return> key. You will next see:

Enter the new average cost/hour, in dollars. DO NOT USE
commas, the dollar sign ($), or letters (i.e. don't enter
1,000 or $10K):

--If the data is not entered properly, the program will either not be able to read it (in which case the program will ask you to repeat the entry), or will not read the new value correctly.

The computer will then repeat the values to the screen and ask if they are correct. For example:

```
You have entered the following:

  Rate= $18.25 per hour,
  Office Document No.= 401-93.

Are these correct? (Enter y or n:)
```

--If the rate as entered is correct, one of the following messages may be displayed, if applicable:

```
You have entered a value less than zero, which is not
acceptable.  Enter another value (e) or quit this function (q)?
```

--Or,

```
You have entered a value less than the minimum wage of
$3.75 per hour.  All new labor charges will be allowed to
be entered in terms of dollars only, until the rate is reset.
Is this acceptable?  (Enter y or n:)
```

--If the rate entered is 1.0, then this message will not appear (since the hours or dollars to be input will be equivalent). If the rate is not acceptable, entering "n" or "N" will re-start this function.

When the values entered for the rate and the reason for change have been corrected, the computer will advise you:

```
Revising current balance to reflect new rate...
```

--There are three possible messages that the computer may issue if it has trouble writing the change to the file:

```
The old rate on line XX could not be read.
```

--If you see this message, do not add any more charges to this file. In order for it to occur, some data would have to be out of place in the file. You should print this file, and have OPDAT restore the data that was altered.

The other messages are:

```
This change in labor cost would reduce the account bal-
ance to less than zero, and is therefore not allowed.

This charge will zero (bankrupt) the account.  Is this
acceptable?  (Enter y or n:)
```

--Entering "n" or "N" will stop the program from writing the change to the account file.

If the second message were displayed, or if you answered "n" or "N" to the third, the program will ask:

```
Enter a new rate (e) or quit this function (q)?
```

--If no error messages were issued, the function will end automatically.

======

--In selecting "E" or "e" (add labor charges) and "F" or "f" (delete labor charges), you are choosing practically identical functions. In the discussion below, I will give the responses to the "add labor charges" function. Where the "delete labor charges" function is different, the responses will be given in square brackets ("[]").

At the start of the function, the program will ask:

```
Add labor charges to account TEST1?
[Delete labor charges from account TEST1?]
Enter y or n (for yes or no):
```

--Entering "n" or "N" will exit you from this segment of the program. If you have selected "Y" or "y", and if your average rate/hour is both less than the minimum wage (set to $3.75 as of this writing---it may have been modified since by OPDAT) and not equal to 1.0, you will see the following warning:

```
WARNING:  All labor charges will be assumed to be entered in
          terms of dollars, because the current wage rate
          is less than the minimum wage.  Do you want to go
          back to the change menu?  (Enter y or n:)
```

--If you do decide to go back to the change menu, you can adjust the average rate per unit hour (choice "D" on the menu).

If you answer "n" or "N", the function will continue.

--If the rate per hour is either 1.0 or greater than the minimum wage (here, the input rate is assumed to be $40.00/hour), the program will display:

```
The current rate/hour is $40.00.  This program will ask for
data in terms of hours.  Are these OK?  (Enter y or n:)
```

--Answering "N" or "n" will exit you from this function. If you answer "Y" or "y", the next prompt will be:

```
Please enter the Office Document No. (14 char. or less:)
```

--If there is no document number, just hit <enter> or <return>.

Should the time be accounted in terms of hours, the program will request:

```
Enter the number of hours to charge (no fewer than 0):
```

[Enter the number of hours to delete (no fewer than 0):]

--If you just hit <enter> or <return>, or enter a value less than zero, the last prompt will be repeated.

Should the time be accounted in terms of money, you will see:

Please enter the amount in dollars (no less than 0.0).
DO NOT USE commas, letters, or the the dollar sign
(i.e. don't enter $10K, or 1,000, or 10K, etc.):

--If the program can't read the input value, or if it is less than zero, the above message will be repeated.

When the data is successfully entered, it will be repeated to the screen, and you will be asked if it is acceptable. If not, the program will repeat the data requests, for both the documentation and the time. Should the input values be correct, the computer will attempt to add the changes to the file.

If you entered a total of zero hours, or $0.00, you will see:

No entry was made to this account since the total cost = $0.00.

--If you are adding labor charges which total more than $0.00 when calculated, and if the program encounters difficulty in adding the charge to the file, you may see:

This change will bankrupt the account. Do you want to
add it to the account? (Enter y or n:)

--If the answer is "Y" or "y", this warning will be given:

You must add money to this account before further
charges can be made.

--Another possible error message is:

This transaction will cause the balance to be less than
$0.00, and is not allowed.

--Whether or not the corrected data has been successfully added to the account file, the current balance will be displayed and you will be asked:

Add more labor charges to account TEST1?
[Delete more labor charges from account TEST1?]
Enter y or n:

--Answering "n" or "N" will exit this function.

======

--Selecting "G" or "g" will access the "add miscelleaneous charges" function. If necessary, you can correct a previous charge (for example, a purchase charged to the wrong account) by entering a negative value for the cost and the reason for the correction (see below).

```
Add miscellaneous charges to account TEST1?
Enter y or n (for yes or no):
```

--Entering "n" or "N" will exit you from this function. If you enter "y" or "Y", the program will ask:

```
Enter reason for charge (29 char. or less):
```

--Note: If you find yourself adding the several of the same type of charge, it is best to be consistent when describing the reason for that charge. That way, you can use the account file search program (choice G on the startup.acc menu) to obtain a listing of all entries for that particular charge, instead of going through the printouts of all files and making a list of such charges by hand.

```
Enter the total cost of the charge, in dollars.  DO NOT
use a comma, dollar sign ($), or letter (i.e. $10K or
1,000 are not acceptable):
```

--If the program cannot read the cost, it will ask you to re-enter the data.

Once the data has been read, the program will repeat the input values to the screen. For example:

```
Here are the values you've entered:

  Reason for Charge = Stamps for Air Mail,
  Total Cost = $35.98.

Are these OK?  (Enter y or n:)
```

--If you answer "n" or "N", the program will again ask you for the cost and the reason for the charge.

If you had entered a total value of $0.00, the screen will display:

```
No entry was made to the account since the total cost = $0.00.
```

--If the data is correct, and you have entered a non-zero cost, the program will attempt to add the cost to the account file. One of the following messages could be generated:

```
This charge can not be applied to this account because
the balance would then be less than zero.
```

--Or,

```
This charge will bankrupt the account.  Do you want
to add this cost to the account?  (Enter y or n:)
```

--Answering "n" or "N" will stop the program from entering the input.

If the account is not bankrupt, the program will display the current balance, and will then ask:

Add another misc. charge to account TEST1? (Enter y or n:)

--Replying "n" or "N" will cause the program to exit from this function.

======

--At the end of each labor function, you will see the following message:

Another charge (of any kind--misc., add money, etc.)
to account TEST1? (Enter y or n:)

--Entering "y" or "Y" will bring you back to the change menu (you can use "q" or "Q" to exit from the change menu); replying "n" or "N" will stop the changes to the account and will bring you to the view and/or print data prompt.

--When the account has been selected, the program will attempt to open the account file, and an additional file needed to operate this program. If either of the following error messages is displayed, you need to contact OPDAT and/or your UNIX system operator for help:

Can't open file containing today's date. Exiting program.

Can't open account file. Exiting program.

--If you see one of these messages, the program will exit the "add non-labor charges" segment, and you will see a prompt to view and/or print the data in the file (see below).

If you are trying to modify a file that is not in the current fiscal year, the program will generate the following:

The current year does not match the fiscal year.
You may only view and/or print data in this account.

Enter any key to continue:

--After inputting a key, the program will leave the "add non-labor charges" mode and you will see a prompt to view and/or print the data in the file.

When the file (for this example, called TEST2) has been opened, it may already have had an entry made to it. The program will read the last entry, if any, and print it on the screen. For example:

Initial Balance= $4500.00

Prev. Flags= -------
Date of Last Entry= 11/7/91

Prev. Transaction= EP: ODN=108-93 RN=XX3456-7890
Transact. Amount= $450.50

Previous Account Balance= $2549.50
Prev. Days Remaining= 58

Enter any key to continue:

--Just hit the <enter> or <return> key to continue.

The flags that may found on an account are: B= Bankrupt, B*= Less than 20% of original balance remains, TE= Time Expired, and TE*= Less than 30 days remain to the account.

If the account were bankrupt, you would be sent the following message:

This account is bankrupt (i.e. current balance = $0.00).
Enter A to add money to the account or Q to quit this part of
the program:

--Answering with "q" or "Q" will cause the "add non-labor charges" part of this program to end, and you would see a prompt to view and/or print data in the file.

======

--If you do elect to add money to the account, you will see the following:

```
Add money to account TEST2?
Enter y or n (for yes or no):
```

--Entering "n" or "N" will exit you from the "add money" function.

If you choose to continue this function, the program will ask you for documentation for adding the money:

```
Enter reason to add money to account (18 char. or less;
for ex.: ODN 47-92):
```

--If you don't have any documentation, just hit the <enter> or <return> key. (For information on the standard transaction descriptors used in this program, see the sample files appended to this guide, and the section on startup.acc menu selection G, below.)

After entering the reason, you will be asked for the amount:

```
Enter amount of money to be added, in dollars.  DO NOT
use letters, the dollar sign ($), or commas (i.e. $1,000
or $10k are not acceptable):
```

--If you do enter the "$", or commas, or "K", the computer either will not read the amount, or not read it correctly. If it can't read the entry, you will see:

```
Please re-enter amount of money to be added:
```

--If you try to enter a value less than zero, the screen will display:

```
Entering a value less than zero is not acceptable.
Please re-enter amount:
```

--Once the data is entered, it will be repeated back to you. For example:

```
The reason to add money is It needs money, and the amount to be
added is $2100.50.  Are these values correct?
Enter y or n:
```

--If the entered values were wrong, the computer will continue to ask for the data until they are correct.

Note: If you do input a (correct) value of zero for the money, there will be no entry made to the account, and the program will exit the "add money" routine at this point. In effect, this is a quick exit from the function.

If the money entry were greater than $0.00, the program will add the

data to the account file, list the current balance on the screen, and then give you the following prompt:

Add more money to account TEST2? (Enter y or n:)

--Entering "N" or "n" will exit you out of the "add money" mode.

Note: Since all charges to the account are entered as positive numbers, the addition of money (a credit to the account) is shown as a negative charge in the account file.

======

--If the account were bankrupt, and if you chose to activate the "add money" function but then did not make any changes (either by entering "n" or "N" at the "Add money to account..." prompt, or by inputting a value of $0.00, the program will exit from the "add non-labor charges" mode, and you will then receive a query to view and/or print the data in the file.

======

--If the current date is the same as the expiration date, you will see:

Today is the last day this account can be worked on. Do
you want to add more time? (Enter y or n (for yes or no):)

--Replying with "y" or "Y" will start the "change expiration date" function (see below); "N" or "n" will continue the program.

If the expiration date has already been exceeded, the computer will generate the following error message:

The expiration date for the account balance has been exceeded.
Enter A to add time or Q to quit this part of the program:

--Answering with "Q" or "q" will cause the program to exit from the "add non-labor charges" mode, and you will see a prompt to view and/or print the data in the file.

======

--If you do decide to add time by changing the expiration date, the computer will ask:

Change the expiration date for account TEST2? (Enter y or n:)

--Answering with "N" or "n" will exit you from this function. If you choose to add more time, the computer will prompt you for the new date. Please enter the date using NUMBERS for the month, day, and year. If the program cannot read the data as you've entered it, it will ask you to repeat the entry:

Enter the number of the month (1-12) of the new date:

Enter the day of the new expiration date:

Now input the last two digits of the year:

--The program will repeat the information back to you, and ask if it is acceptable. For example:

You have entered the following date: 3/22/93.
Is this correct? (Enter y or n (for yes or no):)

--If the reply is "n" or "N", the program will ask you to repeat the date.

If the date you have input is correct, the computer will check if the date is valid. If you have entered an impossible date (for example, 4/91/93), you will receive the following error message:

The date you have entered is not acceptable.

--You will then be asked to re-enter the date.

You may also see the following message, if applicable:

The date you have entered is earlier than today.
Please re-enter the date:

--When the date is acceptable, the computer will then ask:

Please enter reason for changing the exp. date (18 char.
or less). (For ex.: Ref Memo 91-92):

--After entering the reason, it will be repeated back to you, and you will be asked if it is correct. If so, the file will be updated, and you will be returned to the main program. If the reason is not correct, the program will ask you to input the reason again.

======

--If the file had originally exceeded its expiration date, and if you had initially decided to change the date, but then entered "n" or "N" at the "Change expiration date..." prompt, the program will then exit from the "add non-labor charges" mode, and you will see a message asking if you wish to view and/or print the data in the file.

======

--When you have finished adding the money and/or changing the expiration date (if necessary), the screen will clear and the current balance and the change menu will be displayed:

Account No. TEST2 has a balance of $2549.50 and expires in 45 days.

Here is the change menu:

A) Add money to account.
B) Delete money from account.
C) Change expiration date.
D) Enter purchase costs.
E) Enter travel charges.
F) Enter award costs.
G) Input training costs.
H) Add miscellaneous charges.
I) Correct a previous entry.

Enter a letter, or Q to quit adding non-labor charges:

======

--Choosing "Q" or "q" will exit you from the "add non-labor charges" mode of the accounting program. You will next see a prompt to view and/or print the data in the file.

======

--Selecting "A" or "a" will add a lump sum to the account balance. This is the same function as was used to add money to the account to avoid bankruptcy, as discussed above.

======

--Entering "B" or "b" allows you to delete lump sums from the account balance. The initial prompt is:

Delete money from account TEST2?
Enter y or n (for yes or no):

--Responding with "n" or "N" will exit you from this function. If your answer was "y" or "Y", you will then see:

Enter reason to delete money (17 char or less;
for ex.: Ref. ODN 391-93):

--If you have no documentation, hit <enter> or <return>.

The next message is:

Enter amount of money to be deleted, in dollars. DO NOT
use letters, the dollar sign ($), or commas (i.e. $1,000
or $10k are not acceptable):

--Please enter the money in the given manner. If you don't, the program either will be unable to read the value, or will not read it correctly. If it can't read the value, you will be asked to repeat the entry.

You may also see (if applicable):

Entering a value less than zero is not acceptable.
Please re-enter amount of money to be deleted:

--Once the values have been entered, the computer will repeat them back to you. For example:

```
The reason to delete money is Xmas Party Stuff, and the amount to be
deleted is $250.00.  Are these values correct?
Enter y or n:
```

--If they are not correct, the program will ask you to input the values once more. If they are, the program will attempt to subtract the entered sum from the current balance.

Note: A value of $0.00 will not be entered into the account file. In fact, the program will stop the "delete money" function if this value is correct. This is, in effect, a quick exit from this function.

If the program has any trouble deleting the money from the account, it will display one of these error messages:

```
This transaction will cause the account balance to be
less than zero, and will not be allowed.
```

--Or,

```
Warning: This transaction will zero (bankrupt) the
         account. No further deletions are allowed.
```

--If the account balance is still greater than zero, the new balance will be displayed. The program will next ask:

```
Remove more money from account TEST2? (Enter y or n:)
```

--Entering "n" or "N" will exit you from the "delete money" function.

======

--Selecting "C" or "c" activates the same function to change the expiration date as was used above, in the discussion regarding time expired accounts.

======

--Replying with "D" or "d" will generate the following prompt:

```
Add purchases to account TEST2?
Enter y or n (for yes or no):
```

--Entering "N" or "n" will stop the "add purchases" mode and return you to the main program. If you choose to continue, the program will request:

```
Enter Office Number for Purchase Requests (6 char. or less):

Enter 11 char. Requisition Number:
```

--The office number and the requisition number may contain letters, dashes, etc., if needed. If none, just hit <enter> or <return>. The next query is more structured:

Enter the total cost of the purchase, in dollars.
DO NOT use a comma, dollar sign ($), or letter (i.e. $10K
or 1,000 are not acceptable):

--Please enter the cost as requested. If you don't, the program either will not read the input, or will not read it correctly. If it can't read the input, you will be asked to re-enter the purchase price.

The program will then repeat the input information. For example:

Here are the values you've entered:

Office Number = 108-93,
Requisition Number = XX1234-5678,
Purchase Cost = $1200.48.

Are these OK? (Enter y or n:)

--If the values are incorrect, you will be asked to re-enter them. If they are good, then the program will attempt to write the change to the file.

If you had entered a total value of $0.00, the screen will display:

No entry was made to the account since the total cost = $0.00.

There are two possible messages that will be generated if the program has trouble entering the data and if you had entered a total cost other than $0.00:

This purchase can not be applied to this account because
the balance would then be less than zero.

--Or,

This purchase will bankrupt the account. Do you still
want to charge this purchase to this account? (Enter y or n:)

--Entering "N" or "n" will stop the purchase from being added to the file.

If the account is not bankrupt, the current balance (updated, if any changes were made) will be displayed, and the program will ask:

Charge another purchase to account TEST2? (Enter y or n:)

--Answering "N" or "n" will stop the "add purchase" function, and you will next see the change menu.

======

--If you have selected "E" or "e" from the change menu, you will see the following message:

Add travel charges to account TEST2?
Enter y or n (for yes or no):

--Entering "N" or "n" will stop the travel function. If you have entered "Y" or "y", you will be shown the travel menu:

Choose the type of travel:

A) CONUS
B) Invitational
C) Local
D) OCONUS
E) Other

Enter a, b, c, d, or e:

--You will then be asked to input the following information:

Enter the travel order number (up to 8 characters):

Enter the last name of the traveller (up to 14 char.):

--The answers to both of these questions may contain letters, dashes, etc. (If you don't have a travel order number, just hit <enter> or <return>.) The next request is not so forgiving:

Enter the total travel cost, in dollars. DO NOT use commas, the dollar sign ($), or a letter (i.e. $10,000 or 10K are not acceptable):

--If you don't enter the cost as requested, the program will either not read it, or won't read it correctly. If it can't read the cost, you will be asked to repeat it.

When all the data has been entered, the computer will repeat it back to you. For example:

You have entered the following:

Travel Type = Invitational,
Travel Order Number = 101-93,
Traveller = Me,
Total Cost = $450.00;

Are these OK? (Enter y or n:)

--If you answer "n" or "N", the program will ask you for the data, from the beginning. If the reply is "y" or "Y", the program will attempt to write the charge to the file.

If you had entered a total value of $0.00, the screen will display:

No entry was made to the account since the total cost = $0.00.

If you had input a non-zero cost, there are two possible error messages that can be generated if the program has trouble storing the data. If the account balance were $248.00:

This travel charge will force the account balance of
$248.00 to go below zero. This transaction is not allowed.

--Or,

This charge will bankrupt the account. Do you want to
add this charge? (Enter y or n:)

--Entering "n" or "N" will stop the computer from writing the charge to the account file.

If the account is not bankrupt, the program will give you the current balance (here, $248.00) and ask if you want to continue:

The current balance is $248.00. Do you want to charge
more travel to account TEST2? (Enter y or n:)

--Answering "n" or "N" will stop the program from adding more travel charges.

======

--Choosing "F" or "f" from the change menu will generate the following prompt:

Add award charges to account TEST2?
Enter y or n (for yes or no):

--Entering "n" or "N" will exit you from the "add awards" function. If you answer "y" or "Y", the program will ask you for infomation:

Enter the last name (up to 12 char.) of the person
receiving the award:

Enter the office document no. (up to 6 char.) of the award:

--The office document number can have dashes, letters, etc, as required. If no office document exits, just hit <enter> or <return>. The next request is:

Enter the amount of the award, in dollars. DO NOT use
commas, the dollar sign ($), or a letter (i.e. $10,000 or
10K are not acceptable):

--If you do not enter the cost as just a number, the program will either be unable to read the answer, or will not read it correctly. If the award function can't read the cost, you will be asked to re-enter it.

When all the information has been entered, the computer will repeat it back to you. For example:

You have entered the following:

Recipient = Firstaward,
Office Doc. Number = 302-93,
Award Cost = $700.00.

Are these OK? (Enter y or n:)

--Responding with "n" or "N" will re-start the data entry part of this function.

Once the input data is correct, the program will then try to write the data to the file. If the total award cost entered were $0.00, you will see the following message:

No entry was made to the account since the total cost = $0.00.

--If the award cost were non-zero, and if the program could not add the data immediately to the file, one of two error messages will be shown. If the account balance were $350.00, then:

This award charge will force the account balance of
$350.00 to go below zero. This transaction is not allowed.

--Or, if the account balance were $700.00:

This award will bankrupt the account. Do you want to
add this charge? (Enter y or n:)

--If the response is "n" or "N", the charge will not be added to the account file.

If the account is not bankrupt, the program will display the current balance and ask if you want to continue deducting award costs. For example:

The current balance is $350.00. Do you want to charge
another award to account TEST2? (Enter y or n:)

--Answering "n" or "N" will stop the "add awards" function.

======

--Entering "G" or "g" as a change menu selection will cause the program to ask:

Add training costs to account TEST2?
Enter y or n (for yes or no):

--If the reply is "n" or "N", the program will exit from the "add training costs" mode. If the answer is "y" or "Y", you will see:

Enter name of person requesting training and other necessary
data (e.g, course title), if any (19 char. or less):

--After inputting the name, you will be prompted for the cost:

Enter the total training cost, in dollars. DO NOT
use a comma, dollar sign ($), or letter (i.e. $10K or
1,000 are not acceptable):

--Please enter the cost as NUMBERS. If you don't, the program will either not read the cost properly, or it won't be able to read it at all. If it

can't read the input value, it will ask you to re-enter the cost.

Once the data has been input, the program will repeat it to the screen. For example:

```
Here are the values you've entered:

  Trainee Information = Me, CPR Course,
  Total Cost = $235.40.

Are these OK?  (Enter y or n:)
```

--If the data are not correct, the program will ask you to re-enter them. If the values are correct, the program will attempt to write the data to the account file.

If you had entered a total value of $0.00, the screen will display:

```
No entry was made to the account since the total cost = $0.00.
```

--Should the account balance be in danger of bankruptcy, you will recieve one of the following messages:

```
This charge can not be applied to this account because
the balance would then be less than zero.
```

--Or,

```
This charge will bankrupt the account.  Do you want
to add the charge to this account?  (Enter y or n:)
```

--If you enter "y" or "Y", the charge will be added. However, you will not be allowed to add any further costs to the account until you add more money (choice A on the change menu). A reply of "N" or "n" will stop the addition of this charge.

If the account is not bankrupt (regardless of whether or not the charge was successfully added to the file), the program will give you the current balance, and request:

```
Add another training charge to account TEST2?  (Enter y or n:)
```

--Answering entering "N" or "n" will exit the program from this function.

--Entering "H" or "h" from the change menu will give you the following message:

```
Add miscellaneous charges to account TEST2?
Enter y or n (for yes or no):
```

--Replying "N" or "n" will stop the program from entering the "add misc. charges" function. If you have answered "y" or "Y", you will be asked for information:

Enter reason for charge (29 char. or less):

Enter the total cost of the charge, in dollars. DO NOT use a comma, dollar sign ($), or letter (i.e. $10K or 1,000 are not acceptable):

--Please enter the data as requested. If the program can't read the cost as you've input it, the program will ask you to re-enter the cost.

Note: If you find yourself adding the several of the same type of charge (for example: vouchers, stamps, publishing costs, etc.), it is best to be consistent when describing the reason for that charge. That way, you can use the account file search program (choice G on the startup.acc menu) to get a listing of all entries for that particular charge, instead of going through the printouts of all files and making a list of such charges by hand.

When the information has been input, the program will repeat the data back to you. For example:

Here are the values you've entered:

Reason for Charge = Music for Xmas Party,
Total Cost = $350.80.

Are these OK? (Enter y or n:)

--If the values are not correct, the program will once more ask you for the data. When the information is correct, the program will try to add the charge to the account file.

If you had entered a total value of $0.00, the screen will display:

No entry was made to the account since the total cost = $0.00.

--If the account could become bankrupted due to this entry, one of two error messages will be displayed:

This charge can not be applied to this account because
the balance would then be less than zero.

--Or,

This charge will bankrupt the account. Do you want
to add the charge to this account? (Enter y or n:)

--Answering "n" or "N" will stop the cost from being added to the account file.

If the account were not bankrupted, the current balance will be displayed, and the program will ask:

Add another misc. charge to account TEST2? (Enter y or n:)

--Replying "n" or "N" will bring you out of the "add miscellaneous charges"

function.

======

--Requesting "I" or "i" from the "add non-labor charges" menu causes the following to be displayed on the screen:

```
Correct previous charges to account TEST2?
Enter y or n (for yes or no):
```

--Entering "n" or "N" will exit you from the "corrction" function. If you enter "Y" or "y", you will be asked to enter data:

```
Enter reason to correct old charge (29 char. or less):

Enter the total cost of the old charge, in dollars.  DO NOT
use a comma, dollar sign ($), or letter (i.e. $10K or
1,000 are not acceptable):
```

--Please input the data as requested.

Note: When you enter the reason to correct the charge, please use the same abbreviation (if any) as the program uses to write the type of charge being corrected, or if a misc. charge, then use the same description. For example, to cancel a travel order (type CONUS), use "TOC " (space is needed) in the message. This will let you use the keyword search program (choice G on the startup.acc menu) to search for both the initial charge and its correction, without your having to go through the printouts of all files and making a separate list by hand. (For a listing of abbreviations entered by the program, see the section on the account file search program or the sample output file (below).)

If you don't enter the cost as requested, the program will either not be able to read it, or will not read it correctly. If the program can't read the cost, it will ask:

```
Please re-enter cost:
```

--If you enter a negative number, you will see:

```
Entering a value less than zero is not allowed.
Please re-enter the old cost:
```

--When the values have been entered, they are printed again on the screen. For example:

```
Here are the values you've entered:

  Reason for Correction = Cancel Music for Xmas Party,
  Total Amount = $350.80.

Are these OK?  (Enter y or n:)
```

--If the input data is correct, the computer will add the old cost to the current balance, cancelling the effect of the orginal charge. Please note,

however, that when you look at the file, or print it, the correction will be entered as a negative number in the "Transaction Amount ($)" column, to show that it is a correction of an earlier charge, not a new charge (charges to the account are shown as positive numbers).

If the input is not correct, you will be asked to re-enter the data.

If you had entered a total value of $0.00, the screen will display:

No entry was made to the account since the total cost = $0.00.

Once the entry has been made to the file, you will be given the current account balance, and the program will request:

Enter another correction to account TEST2? (Enter y or n:)

--If the answer is "n" or "N", the program will exit the "correct previous entries" mode.

======

--Once you have finished with an "add non-labor charge" function, the program will ask you:

Another charge (of any kind---travel, award, etc.) to account TEST2?
Enter y or n:

--Entering "y" or "Y" will clear your screen and re-display the "add non-labor charges" menu.

Replying "n" or "N" will exit you from the "add non-labor charges" part of this program.

--Whether or not you have chosen to modify the data in an account, or simply to look at the data, at this point, the program will clear your screen, then ask you:

View data in file (v), print data (p), both view and print (b),
or skip view/print of account (s)?

******

--Entering "s" or "S" will bring you to the "Another account?" prompt without reviewing the data in the file.

******

--If you answered "v" or "V", the computer will display the file one screen at a time. There are two methods for this, depending on what your system allows. If you see a ":" at the lower left corner, hit <enter> or <return> to advance to the next screen. (The final screen will have "(EOF):" at the lower left corner; hit <enter> or <return> to finish the view.)

If you see the word "More" at the bottom of the screen, hit the space bar to advance to the next screen. Hitting <enter> or <return> may only advance the display by one line, depending on your system.

Note: The labor data files are more than 80 character spaces wide. The rows will overlap when displayed on a normal monitor.

******

--Answering "p" or "P" will print the account file on your local printer.

******

--A reply of "b" or "B" will allow you to first view the account file, then the program will print the file on your local printer.

******

--When you have finished viewing and/or printing the account (or if you have chosen to skip the review of the file), the program will ask:

Another account?
Enter y or n:

--Entering "N" or "n" will quit you from this function; you will next see the startup.acc menu. An answer of "Y" or "y" will generate this prompt:

Enter the account (or XO) number (same as the file
number in the computer), or enter q to quit:

--Entering "q" or "Q" will exit you from the program. You will next see the startup.acc menu.

Entering an account (or XO) number will have the program repeat the number. For example:

This is the input account number: NN123.
Is this number OK?

Enter y or n:

--If the number is not correct, you will be asked to re-enter it.

Note: If you just hit <enter> or <return> when asked for the account file number, the account number will be given as "None", and you will see the following error message:

This is the input account number: None.

```
*********************************************************
*  WARNING:  Entering no account number will cause the  *
*            program to terminate.                      *
*********************************************************
```

Is this number OK?
Enter y or n:

--Entering "y" or "Y" will bring you to the startup.acc menu. A reply of "n" or "N" will have the program ask you again for the account file number.

******

--After the new file name has been entered, the program will ask if the file is the same type (labor or non-labor) as before:

Is this another labor account?
Enter y or n:

--Or,

Is this another non-labor account?
Enter y or n:

******

--If the answer is "y" or "Y", the next question will be:

Add charges to account (A) or simply look at (and/or print)
the account (L)? (Enter A or L:)

--Please enter "a" or "A" or "l" or "L" as appropriate. The program will now proceed to search for duplicate file names, and will proceed from that point, as above.

******

--If the answer is "N" or "n", the program will then ask you:

Is this a labor (L) or non-labor (N) account? (Enter L or N:)

--The remainder of the program will proceed from this prompt, as above.

D) ENTER GENERAL LABOR CHARGES.

--Choosing Option "D" or "d" from the startup.acc menu brings the following message to the screen:

This program inputs general labor charges to accounts. It assumes that the directories (file categories) and accounts have been set up already, using Choices A and B on the startup.acc menu.

If this is not the case, answer N to the question below to exit this program, and choose from the startup.acc menu to input the required information.

A helpful hint: DO NOT USE the <rubout> or <backspace> keys to erase errors that you make when inputting data. Use <crtl><h> instead.

Do you want add general labor charges?
Enter y or n (for yes or no):

--If the answer is "n" or "N", the program will halt this function, and you will be returned to the startup.acc menu.

If you do continue, the program will ask you for the fiscal year. As noted for startup.acc menu selection B (above), the fiscal year forms a part of the file directory name. Also, if the fiscal year of the directory is not the same as that of the current fiscal year, you will not be able to enter the labor charges.

Enter the last two digits of the fiscal year, or just hit <enter>
or <return> if the year is 93:

--If the program can not find the file directory (for ex., for the year 93), you will get this error message:

There are no files for fiscal year 93. Select another year?
Enter y or n:

--If you answer "n" or "N", the program will stop this function and return you to the startup.acc menu.

If you have entered "Y" or "y" and you see the following error message, you need to talk to OPDAT and/or your UNIX system operator:

Can't open file containing today's date. Exiting program.

--If you did not enter the last two digits of the current fiscal year, as requested above, you will see the following:

The current year is not the same as the fiscal year. You
may only view and/or print the data in the files, using
choices C, E+F, or G from the startup.acc menu.

Enter any key to continue:

--Just hit the <return> or <enter> key. You will next see the startup.acc

menu.

If you did enter the correct fiscal year, you will next be asked for the name of the account, or its XO number, (must be the same as the file name). You can enter up to 25 separate account names; the names themselves can be entered in any order and may be used more than once in a single sitting. (The program will add any new charges to an already-named account, as they occur.)

If the year is correct, the next request will be:

Enter account no. (or XO no.) (up to 5 characters):

--Once entered, the program will repeat the name back to you. For example:

You have entered account no. TEST3. Is this OK? (Enter y or n:)

--If the answer is "n" or "N", the program will ask you for another account file name.

The program will then attempt to find the file. In order to do that, it needs to set up some files that allow the computer operating system to communicate with the program. If you see one of the two following error messages, you need to talk to OPDAT and/or your UNIX system operator.

Can't open file for storing account name. Exiting program.

--Or,

Can't open file containing account pathway. Exiting program.

--If the computer can't find your account, you will see:

This account either does not exist, or it does not exist
as a labor account.

--You will next be asked if you want to enter data for another account.

If the program finds the same account name under two different directories (file categories) (for example, test.d and overhead.d), the file name will be listed on the screen, preceded by the directory name. For example:

This account was found in multiple directories:

1) ./overhead.d/TEST3
2) ./test.d/TEST3

Enter the line no. of the correct directory listing:

--The "." is a part of the file path name. If you enter something other than the line number of the choices above, you will be asked to re-enter the line number.

Once you have established the correct account name, the program will ask you to input the labor charges. You can enter more than one charge to

that account on the same line:

The labor charges can be entered in terms of hours, or dollars (the program will automatically put in the correct value, based on the rate/hour in the account).

Enter the charges for TEST3. If entering more than 1 number per line, leave a space or comma between numbers. (Do not use commas in the middle of a number.) If entering charges in terms of dollars, you do not have to enter the dollar sign ($).

Use <crtl><h> to erase input errors:

--For example, you could enter:

8.5, 7, 6, 5 4

--When the line has been entered, the total will be repeated back to you:

The total entered for this line is 30.50.
Is this correct? (Enter y or n:)

--If the line total is acceptable, it is added to the cumulative charges which will be added to the account file (see below). Whether or not the input data is correct, you will be asked:

Another line of data for account TEST3? (Enter y or n:)

--If the answer is "n" or "N", and if the total number of different accounts is less than 25, the program will inquire:

Add charges to another account? (Enter y or n:)

--If the reply is "n" or "N", the program will process the data it has accumulated to that point.

Note: After entering charges to TEST3, you can select another account (for ex., TEST4), and enter costs to that file. When you're finished with TEST4, you can then add more costs to TEST3, within the same session. You can add charges to accounts in any order, to suit your timekeeping needs.

The total number of accounts which can be processed at one time is 25, but OPDAT can easily adjust that limit, as you require.

If the total number of different accounts is greater than 25, you will see:

The number of accounts to be processed exceeds the limit of 25 established by this program. Please stand by while accounts are processed....

--The batch processing of accounts will now begin. If any error conditions are encountered (such as trying to add charges to a bankrupt account), they will be written to an error file which you can access once the processing

is finished. The computer will list each file as it is accessed.

Note: If the total number of hours entered for any account is zero, or if the total dollar cost entered is $0.00, no entry will be made to that account file.

If you see the following error message, you need to talk to OPDAT and/or your UNIX system operator:

Can't open the error file. Exiting program.

--Sample error messages recorded in the error file are:

Can't open account file for account ./test.d/TEST3
with 30.50 additional hours or dollars charged.

Account No. ./test.d/TEST3 is bankrupt and
$30.50 could not be added.

Account No. ./test.d/TEST3 is bankrupt and
30.50 hours could not be added.

Account No. ./test.d/TEST3 has exceeded its expiration
date and $30.50 could not be added.

Account No. ./test.d/TEST3 has exceeded its expiration
date and 30.50 hours could not be added.

Account No. ./test.d/TEST3 is now bankrupt.

Account No. ./test.d/TEST3 could not be charged with 30.50 hours
because the charge would bankrupt the account.

Account No. ./test.d/TEST3 could not be charged with $30.50
because the charge would bankrupt the account.

--When the batch processing is completed, and only if you exceeded the limit on the accounts to be processed, the program will ask:

Process another set of accounts? (Enter y or n:)

--If the answer is "y" or "Y", the program will ask for data for another set of accounts.

If you reply "N" or "n", the program will determine whether an error file were created. If not, the next display will be the startup.acc menu. If an error file were generated, you would be asked:

This program has generated an error file. Do you want to view it (v),
print it (p), both (b), or neither (n)?

--Printouts are sent to the local printer.

There are two usual methods of viewing the data in a file. If you see the word "More" at the bottom of your screen as the data is displayed,

hit the space bar to advance the screen. If you see a colon at the lower left corner of your screen, hit <enter> or <return> to see the next screen of data. If you see "(EOF):" at the lower left corner, hit <enter> or <return> to finish viewing the data.

When the view and/or print data modes have finished, or if the selection you entered were "n" or "N", the program will inquire:

```
Copy error file to a permanent file in your directory?
(If not, then the error file will be erased.)
Enter y or n:
```

--If the answer is "Y" or "y", the program will ask you for a new name for the file (see startup.acc menu choice A (above), for a short summary of UNIX file name conventions):

```
Enter file name for error file:
```

--If you do not enter a name, then you will see:

```
You must enter a file name.
```

--You will then be asked to input the file name.

You may also see (if applicable):

```
The file name you have selected already exists.
Choose another file name?
Enter y or n:
```

--If you enter "y" or "Y", you will then be asked to enter another file name.

If you select "n" or "N", the error file will not be copied to your directory and it will then be erased.

Once the file has been copied (if you've selected that option), the program will end, and you will see the startup.acc menu once again.

E) CREATE THE UPDATED SUMMARY FILE.

--Choosing "E" or "e" from the startup.acc menu begins the summary program. This program generates a file (sum.info) which summarizes the status of the accounts in file, within any fiscal year, current or otherwise. The first question is:

Do you want to create the updated summary file?
Enter y or n (for yes or no):

--If you reply "n" or "N", the program will return you to the startup.acc menu.

If you answer "y" or "Y", the program will next ask you for the fiscal year of the files to be updated:

Enter the last two digits of the fiscal year, or just hit <enter>
or <return> if the year is 93:

--If the program can not find the account directory for those files (for ex., fiscal year 91), you will see:

There are no files for fiscal year 91. Choose another year?
Enter y or n:

--If you answer "n" or "N", the program will exit the update function and return you to the startup.acc menu.

If you want to generate the file, the final message will be:

This program generally takes 1-4 min. to run, depending on the number of files being summarized, and the number of users on the system. There are no further questions for the operator to answer.

--Once the file is generated, the program will return you to the startup.acc menu.

If you see any of the following error messages, then the summary file could not be created. Notify OPDAT and/or your UNIX system operator if one appears:

Can't open account file XXXXX.

Can't open temporary output file.

Can't open datefile.

****

Note: A key is simply a word, abbreviation, or phrase that occurs only when certain kinds of data are input in a file; a computer can search a file for that key and print the line in which it appears.

The next two functions will search the updated summary file and all account files (respectively) for keys.

****

F) SEARCH THE SUMMARY FILE.

--Selecting "F" or "f" from the startup.acc menu activates the search program. You must already have used menu choice E to create the summary file.

The initial prompt is:

Do you want to search the summary file for a key word or phrase?
Enter y or n (for yes or no):

--If you answer "N" or "n", the function will stop, and you will next see the startup.acc menu.

If you elect to continue, you will be prompted for the fiscal year:

Enter the last two digits of the fiscal year, or just hit <enter>
or <return> if the year is 93:

--If the summary file for that year (for ex., 89) could not be found, then you will be given the following error message:

The summary file for fiscal year 89 does
not exist. Select another year?
Enter y or n:

--If you answer "n" or "N", the program will say:

To generate the updated summary file, please choose option E from
the startup.acc menu, which follows.

Enter any key to continue:

--Just hit <enter> or <return> to go back to the startup.acc menu.

******

--If the summary file exists, the screen will clear, and the following will be displayed:

This program searches the summary file for key words or phrases, which
you enter below. You can use this program to search for all information available on a specific account number, account balance, principle
investigator, etc.

--A sample output is:

Here are the first 11 lines of the summary file:

```
Account Directory:  6.1.d
Account No. 122HK:  Time of Update:  Fri Nov  1 15:33:03 EST 1991
122HK : Classification:  Labor
122HK : Purpose for Account:  Test for Accounting Program
122HK : Principle Investigator: Me
122HK : Account Flags: -------
Flags:  B*= <20% Orig. Balance, B= Bankrupt
        TE*= <30 Days Remaining, TE= Time Expired
122HK : Current Balance: $123456.78
122HK : Days Remaining to Account: 334
```

Please enter the word or phrase to search for (i.e. the actual name of the principle investigator, the account no., etc. The case of the letters will be ignored.):

--The word or phrase can be in upper or lower case letters, as you prefer. The program will display any line that contains the string to be matched.

If you choose to use a phrase, please note that the spacing between words is important, and it is better to be as specific as possible. For example, the string "account flags" will return the flags on all accounts, while "122HK : account flags" will display only the flags for account 122HK (the letters "HK" will be used in the search, but the program will return all flags for any upper or lower case letter combination of "hk"). Simply entering "122HK" will print on the screen all lines displaying "122hk", or "122HK", or "122Hk", or "122hK".

If you just hit <enter> or <return> without inputting a string, you will get the following error message:

WARNING: Entering nothing will cause the program to terminate.

You have entered: nothing. Is this OK?
Enter y or n:

--If you enter "y" or "Y", the search function will halt, and you will next see the startup.acc menu. If you enter "N" or "n", you will be asked to re-enter the key word or phrase.

If you have entered a search string (e.g."account balance"), the program will repeat it to the screen:

You have entered: account balance. Is this OK?
Enter y or n:

--If you enter "n" or "N", you will be able to re-enter the search string.

The search function will now check each line in the summary file to see if any part of the line matches the search string. If so, the entire line will be stored in a temporary file.

If there is no match to your word or phrase, then the program advises:

```
There were no lines to match:  account balance.
Another search?
Enter y or n:
```

--Entering "n" or "N" will return you to the startup.acc menu.

If there are lines to match your input string, the screen will clear and the lines will be displayed. For example, if you had entered "current balance", you would see the current balances of all accounts (first labor, then non-labor):

```
Here are the lines that match your search input (current balance):

122HJ : Current Balance: $123456.78
122HK : Current Balance: $3456.78
122HL : Current Balance: $456.78
122HM : Current Balance: $56.78
122HN : Current Balance: $123.45
122HO : Current Balance: $1234.56

Print these results (p), save data to a permanent file (s),
do both (b), or none of the above (n)?  (Enter p, s, b, or n:)
```

--Note: There are two methods usually used to display data one screen at a time. If you see the word "More" highlighted at the bottom of the screen, hit the space bar to advance the screen. If you see ":" or "(EOF):" at the lower left corner of the screen, hit <enter> or <return>.

Printing is done via the local printer.

******

--If you want to save (or both print and save) the file, you will be prompted:

```
Enter file name for saving this data (it will be saved to
your own directory):
```

--For UNIX naming conventions, see startup.acc menu choice A (above). The reason that it says that the file will be saved to your directory is that you are currently in OPDAT's directory (remember that all files are located there). If you want to access this data later, you should have the file in your own directory.

If you did not enter a file name (i.e., you just pressed the <return> or <enter> key), you will see the following error message:

```
You must enter a file name.
```

--You will then be asked to re-enter the file name.

You may also receive the following message, if applicable:

The file you have selected already exists. Choose another file name?
Enter y or n:

--If you answer "n" or "N", the program will not store the results to a file.

If you reply "Y" or "y", you will be asked to re-enter the file name.

******

--When printing and/or saving the file is completed (or if "n" or "N" were chosen), the next prompt is:

Another search?
Enter y or n:

--If the reply is "y" or "Y", the search will start from the beginning. If the answer is "N" or "n", the program will exit from search mode, and you will again see the startup.acc menu.

G) SEARCH THE ACCOUNT FILES.

--This program will search both the labor and non-labor files for the entered key word or phrase.

When you choose this option, therefore, you will see:

This is the program which will search through all files to match a keyword (e.g. training, travel, purchases, etc.), and will collect the data, to be printed and/or saved to a file, as you wish.

Do you want to use this program?
Enter y or n (for yes or no):

--Entering "n" or "N" will return you to the startup.acc menu.

If you do choose to continue, the program will display:

Enter the last two digits of the fiscal year to be searched,
or just press <enter> or <return> if the fiscal year is 93:

--This program can search any existing data directory, and as noted above, the fiscal year is a part of the directory name. If you enter more than two digits, you will be asked to re-enter the year.

If you have entered a year for which there are no files (e.g. "89") then you will see the following error message:

There are no files for fiscal year 89. Choose another year?
Enter y or n:

--Replying with "n" or "N" will return you to the startup.acc menu.

When the year is correct, the program will display the keyword change menu:

Here are the available pre-set keywords:

A) Award,
B) Purchases,
C) Training,
D) Travel.

Enter a letter (A-D) to select a pre-set keyword, or
enter E to select your own keyword, or Q to quit this program:

******
--Selecting "q" or "Q" will return you to the startup.acc menu.

******
--Entering "D" or "d" lets you choose the type of travel:

Here are the types of travel available:

A) CONUS,
B) Invitational,
C) Local,
D) OCONUS,
E) Other,
F) All of the above.

Enter the letter for the correct line:

******
--Answering with "e" or "E" allows you to enter your own keyword(s):

Enter the word or phrase to search for; the case (upper or lower) of the letters will be ignored:

--If you just hit <enter> or <return> you will recieve these messages:

WARNING: Entering nothing will terminate the program.

You have entered: nothing. Is this OK?
Enter y or n:

--If you answer "y" or "Y", the program will return you to the startup.acc menu.

Notes:
---

1) Any other reply than <enter> or <return> becomes your keyword. It should have enough characters to uniquely identify it, and should only appear on a file's "Transaction Description" column. For example, a person's name could appear on the "Principle Investigator" line, but would not show up at all for an equipment purchase. This function will print the line with the principle investigator's name, but it since it is set up to print lines according to a certain format, that line will look out of place. (See the sample output, below.) You may have to save the output file to your own directory and edit it later, to make the results acceptable.

2) A complete listing of abbreviations used in the "Transaction Description" column of the account files is appended to this section.

3) If you need a printout of all that was charged in a given month, use a search string which consists of a vertical line, followed by the number of the month (use two digits), followed by a slash. For example, if you need to know the account entries for all of January, your key would be: "|01/" (again, the quotes are used only to set off the string; don't use them as part of the actual search string). This is the way that the month appears in the "Date Of Entry" column in the data files (see the sample files appended to this guide).

After you have selected your keyword (for example, "voucher"), it will be repeated to the screen:

You have entered: voucher. Is this OK?
Enter y or n:

--Replying "n" or "N" allows you to re-enter your own keyword.

******

--Once the search string (e.g., "voucher") has been selcted, the screen will clear, and the program will display:

This program will search the files for: voucher.

The process may take between 1-4 min., depending on the number of files to be searched and on how many users are on the system.

******

--If the program can not find any lines in any file to match your keyword (for example, "voucher"), it will send the following prompt:

The search program could not find any match to: voucher.

Another search?
Enter y or n:

--Answering "n" or "N" will return you to the startup.acc menu. If you want to do another search, the program assumes that you want to search through the files for the same fiscal year as entered at the start of the program.

******

--If there is a match, the program will create a temporary file to hold all lines that match the keyword, and will display that file at the end of the search.

There are two ways (depending on your UNIX system) that the data files may be displayed. If you see a ":" or "(EOF):" at the lower left-hand corner of the screen, hit <enter> or <return> to get to the next screen. If you see the word "More" highlighted at the bottom of the screen, hit the space bar to advance the screen.

This is a sample output from a search for "All Travel" (choice F from the travel menu, above):

Here is the data that matched your search:

B= Bankrupt, B*= <20% of Starting Balance Remains, EP= Equip. or Other Purchase,
ODN= Office Document Number, ON= Order Number, RN= Requisition Number,
TE= Time Expired, TE*= <30 days remain, TOC= Travel Orders (CONUS),
TOI= Travel Orders (Invitational), TOL= Travel Orders (Local),
TOO= Travel Orders (OCONUS), TOX= Travel Orders (Other)

Account Directory: 6.2.d
Account Number: 122XX
Classification: Non-Labor
Purpose for Account: Test of Accounting Program
Current Balance = $ 12345.67

Date Of Entry	Transaction Description	Transact. Amount ($)
10/28/91	TOC ON=10-023 Smith3	1234.00
10/13/91	TOI ON=10-021 Jones2	567.89
10/28/91	TOL ON=10-120 Jones2	87.00
11/04/91	TOL ON=10-157 Jones1	456.00
12/11/91	TOO ON=12-034 Jones1	2345.67
10/28/91	TOX ON=10-022 Smith2	789.10
11/04/91	TOX ON=10-156 Smith1	543.21
11/06/91	TOX ON=11-021 Cancels 10-156	-543.21
11/13/91	TOX 11-044 CANCELLED 10-022	-789.10

--Notes:

1) The first 5 lines are always at the start of any "found data" file.

2) The last two entries are corrected entries. The next-to-last line was entered by choosing a negative travel amount and the inputting the cancelled order number in place of the traveller's name. The final line was made via the "Correct Previous Entry" function. Since the string "TOX " (the space after "X" is important) was used for this correction, this search program was able to present a complete record of the transactions.

3) All lines of data matching the search are grouped according to their originating file and directory.

4) As noted at the start of this section, both the labor and non-labor files are searched for the keyword.

5) For the "All Travel" search only, all entries are grouped by each type of travel, then by chronological order within each travel type.

6) The "Date Of Entry" is the date that the file entry was made. It is automatically entered by the program at the time of data input, and is common to all transactions (labor and non-labor). It therefore has nothing to do with the actual travel date (or that of any other type of charge).

******

--After the file is displayed on the screen, the program asks:

Do you want to print the results?
Enter y or n:

--Answering "y" or "Y" will print the results to your local printer. When the file is finished printing, or if no results were printed, the program requests:

The file containing the results is a temporary file.
Do you want a copy stored on your directory?
Enter y or n:

--The reason the program is asking this question is that you are currently in OPDAT's directory (where the files are located---see the introduction at the start of the program guide). To edit the file, or even to reprint it, you should have a copy in your own directory. The temporary copy of the search results is deleted at the end of this program.

If the answer is "Y" or "y", the program asks:

```
Enter file name:
```

--For UNIX file naming rules, see startup.acc menu choice A, above.

If you just hit <enter> or <return> without entering a file name, the program gives you the following error message:

```
You must enter a file name.
```

--You will then be asked to re-enter the file name.

If you enter the name of an existing file, you will notified of that fact, and asked if you want to enter a new file name. (If not, then the temporary file will simply be erased.)

When the file name has been successfully entered (or if no storage were requested), the program asks:

```
Another search?
Enter y or n:
```

--Answering "n" or "N" will return you to the startup.acc menu. If you want to do another search, the program assumes that you want to search through the files for the same fiscal year as entered at the start of the program.

APPENDIX TO ACCOUNT FILE KEYWORD SEARCH PROGRAM

The standard descriptors used in this program to indicate a particular type of transaction are:

A) Labor:

Transaction	Descriptor
1) Add money to account.	"Add Money:"
2) Delete money from account.	"Del. Money:"
3) Change expiration date.	"Exp. Date:"
4) Change avg. cost/hour.	"Change Rate"
5) Enter labor charges (individual accounts only).	"Add Lab."
6) Delete labor charges (individual accounts only).	"Del. Lab."
7) Add general labor charges.	"Add general labor charges."

B) Non-Labor

Transaction	Descriptor
1) Add money to account.	"Add Money:"
2) Delete money from account.	"Del. Money:"
3) Change expiration date.	"Exp. Date:"
4) Enter purchase costs.	"EP:"
5) Enter travel order charges:	
a) CONUS,	"TOC "
b) Invitational,	"TOI "
c) Local,	"TOL "
d) OCONUS,	"TOO "
e) Other.	"TOX "
6) Enter Award costs.	"Award"
7) Input training costs.	"Training:"

Notes:
---

1) There is no standard abbreviation for labor and non-labor miscellaneous charges. There is also no standard descriptor for the non-labor correction function (but entries may be known by the negative charges, as well as the transaction description itself).

2) As noted above, the case of the letters does not matter when the charge types are input by hand.

3) The space at the end of the travel order abbreviations is necessary. If you try to hand input them, however, the space may or may not be picked up, depending on your system. It is therefore best to use the pre-set menu for the travel charges.

H) PRINT THE LABOR, NON-LABOR, AND/OR UPDATED SUMMARY FILES.

--This function prints the above files on the local printer.

Selecting this function clears the screen, and writes:

```
Do you want to print files?
Enter y or n (for yes or no):
```

--Answering "n" or "N" will return you to the startup.acc menu.

If you elect to print the files, you will be asked:

```
Enter the last two digits of the fiscal year, or just hit <enter>
or <return> if the year is 93:
```

--As with the search functions described above, this function will print the files for any fiscal year with records.

If there are no files for the year entered (for example, "99"), you will be told:

```
There are no files for fiscal year 99.  Select another year?
Enter y or n:
```

--Answering "n" or "N" will stop this function and display the startup.acc menu.

If the fiscal year selected does have files, the program will ask you if you want to print them:

```
Print all labor accounts?
Enter y or n:

Print all non-labor accounts?
Enter y or n:

Print the file containing the summary of all accounts?
Enter y or n:
```

--If you answered "y" or "Y" to one or more of the above, the function will look to see if there are any files to print; if the labor or non-labor files don't exist, you will see an error message to that effect. If the summary file was requested, but didn't exist, the screen will display:

```
The summary file does not exist.  This program will create it.
This process could take 1-4 min., depending on the number of
files and the number of users on the system.
```

--When the requested files are ready to be printed, you will see the following messages (as applicable):

```
Creating the labor printout file...

Creating the non-labor printout file...
```

Creating the summary printout file...

--The final message is:

This program has ended. Enter any key to return to the main menu:

--Just press <enter> or <return> to go back to the startup.acc menu.

SAMPLE ACCOUNT FILES

Account Number: TEST1
Classification: Labor
Purpose for Account: Test Labor Programs
Principle Investigator: None.
Starting Balance: $ 0
Labor Charge (Rate per Hour): $ 40
Date Created: 12/19/91
Initial Expiration Date: 4/33/93

B=Bankrupt, B*= <20% of Starting Balance Remains, ODN= Office Document Number,
TE= Time Expired, TE*= <30 Days Remain

Acct. Flags	Date Of Entry	Transaction Description	Rate ($)/ Hour	Hours Entered	Transact. Amount ($)	Current Balance ($)	Days Remaining	Cumulative Hours
TE	12/19/91	Add Money: Funding Approved.	40.00	0.00	-45000.00	45000.00	-80	0.00
	12/19/91	Exp. Date: Input Correction	40.00	0.00	0.00	45000.00	285	0.00
	12/19/91	Add general labor charges.	40.00	42.50	1700.00	43300.00	285	42.50
	12/19/91	Del. Money: Funding Cutback.	40.00	0.00	2000.00	41300.00	285	42.50
	12/19/91	Change Rate per ODN=1234-5678	20.00	0.00	-850.00	42150.00	285	42.50
	12/19/91	Add Lab. ODN= Maint. Memo	20.00	20.00	400.00	41750.00	285	62.50
	12/19/91	Del. Lab. ODN= Memo Rescinded	20.00	-20.00	-400.00	42150.00	285	42.50
	12/19/91	Equipment Repair (Misc. Chg.)	20.00	0.00	45.50	42104.50	285	42.50
	12/20/91	Exp. Date: None.	20.00	0.00	0.00	42104.50	155	42.50
	12/20/91	Change Rate per ODN=None	3.00	0.00	0.00	42104.50	155	42.50
	12/20/91	Misc. Computer Repair	3.00	0.00	200.00	41904.50	155	42.50
	12/20/91	Change Rate per ODN=None.	40.00	0.00	1700.00	41054.50	155	42.50
	12/20/91	Add general labor charges.	40.00	70.00	2800.00	38254.50	155	112.50

Note: The Date of Entry is the date on which the transaction was added to the data file, and is entered automatically. It has nothing to do with the actual date on which the charge was incurred.

Account Number: TEST2
Classification: Non-Labor
Purpose for Account: Test the Non-Labor Program
Principle Investigator: None.
Starting Balance: $ -50000ZZ

Date Created: 12/19/91
Initial Expiration Date: 45/21/92

B= Bankrupt, B*= <20% of Starting Balance Remains, EP= Equip. or Other Purchase, ODN= Office Document Number, ON= Order Number, RN= Requisition Number, TE= Time Expired, TE*= <30 days remain, TOC= Travel Orders (CONUS), TOI= Travel Orders (Invitational), TOL= Travel Orders (Local), TOO= Travel Orders (OCONUS), TOX= Travel Orders (Other)

Acct. Flags	Date Of Entry	Transaction Description	Transact. Amount ($)	Current Balance ($)	Days Remaining
-----TE	12/19/91	Add Money: It needs money.	-4500.45	4500.45	-80
-------	12/19/91	Exp. Date: Time Correction.	0.00	4500.45	117
-------	12/19/91	Del. Money: I need money.	100.00	4400.45	117
-------	12/19/91	EP: ODN=123-45 RN=XX9876-5432	500.00	3900.45	117
-------	12/19/91	TOX ON=Aspen.Go Me	1567.89	2332.56	117
-------	12/19/91	Award-Me. ODN=None.	1.00	2331.56	117
-------	12/19/91	Training: Me, Basket Weaving	123.45	2208.11	117
B*-----	12/19/91	Retirement Party	2002.02	206.09	117
B*-----	12/19/91	Training: Basket Weave Cancel	-123.45	329.54	117

Notes:

1) The Date of Entry is the date the charge is added to the file, and is entered automatically. It has nothing to do with the actual date on which any purchases were made, travel taken, etc.

2) The operations performed on this sample account were (in order):

   A) Add money to the account.

   B) Change the expiration date.

   C) Delete money from the account.

   D) Equipment (or other) purchase made.

   E) Travel Order (Other) entered.

   F) Award charged to account.

   G) Training cost input.

   H) Miscellaneous charge added.

   I) Correction to Training charge.

Account Number: TEST3
Classification: Labor
Purpose for Account: Test the Labor Programs
Principle Investigator: None.
Starting Balance: $ 12345.67
Labor Charge (Rate per Hour): $ 0
Date Created: 12/19/91
Initial Expiration Date: 9/30/92

B=Bankrupt, B*= <20% of Starting Balance Remains, ODN= Office Document Number,
TE= Time Expired, TE*= <30 Days Remain

Acct. Flags	Date Of Entry	Transaction Description	Rate ($)/ Hour	Hours Entered	Transact. Amount ($)	Current Balance ($)	Days Remaining	Cumulative Hours
	12/19/91	Add general labor charges.	0.00	0.00	450.00	11895.67	286	0.00
	12/20/91	Change Rate per ODN=None.	30.00	0.00	0.00	11895.67	285	0.00
	12/20/91	Add general labor charges.	30.00	35.00	1050.00	10845.67	285	35.00

USERS' GUIDE TO THE ABBREVIATED SEARCH, VIEW, AND PRINT PROGRAMS

STARTING THE PROGRAM

The abbreviated program is called startup (as opposed to startup.acc). As noted in the Installation Guide, this was developed for those who need to see or print what is in the account files, but who have no need to enter data to them.

This program is initially stored in a directory of OPDAT's. In order to use the command "startup" without having to type in OPDAT's directory name each time you need to access the program, you can link the program to your current directory.

In order to do this, you do need to have OPDAT provide you with the full path name of the program. It should have the form: /usr/OPDAT/startup and you can then link it to your current directory by entering the following command at the computer's command prompt (here assumed to be "$>"):

$> ln /usr/OPDAT/startup . <enter> (Note: There is one space between "ln" and "/usr/OPDAT/startup" and another between the program name and ".". (The period is a part of the command.))

You can then access the program at any time after that, by simply entering "startup" at the command prompt:

$> startup

There is one potential problem: if you already have a command or program called startup on your directory, this procedure will simply erase the old program, unless the file permissions on the old program do not permit erasure (i.e., you see an error message such as "ln: permission denied"). If this is the case, replace the period at the end of the link command with the name of a new file that is not in use. Substitute the new name for "startup" in the rest of this guide.

==========

Note: In order to separate my comments from the on-screen prompts, I will precede my comments with "--" as necessary. The comments will also be indented.

==========

--When you enter "startup", your screen will clear, and you will be asked the following:

Do you want to look at and/or print account files?
Enter y or n (for yes or no):

--Entering "n" or "N" will stop the program and return you to the command prompt ("$>"). (Note: For now, and for the rest of this guide, the letters used as choices to be input to commands will be set off in the comments by enclosing them in quotes. When inputting the letters to the program, do not

enclose them in quotes.)

If you input "y" or "Y", you will next see the startup menu.

Note: If you enter something other than "y" or "Y" or "n" or "N", you will receive the following message:

Please enter one of the following: y Y n N.

--This type of message will occur whenever the program requests a decision from you. Thus, you don't have to worry too much if you hit the wrong key.

*****

--If you decide to continue with the program, the screen will clear, and the startup menu will be displayed:

Here are the available programs:

A) View or print individual labor and non-labor accounts.
B) Search the summary file for operator-specified keywords.
C) Search the account files for operator-specified keywords.
D) Print all labor and/or non-labor files, and/or the summary file.

Enter a letter (A-D) or Q to quit the accounting program:

*****

--If you select "A" (or "a"), the program will clear your screen, and you will be prompted:

This program finds files and enables the user to view and/or print the charges in the individual account files.

A helpful hint: DO NOT use the <backspace> or <rubout> keys to correct input errors on the screen. Use <crtl><h> instead.

Do you want to use this program?
Enter y or n (for yes or no):

--Entering "n" or "N" will return you to the startup menu.

The remainder of this program is the same as the "View, Print, and/or Add Charges to Files" option in the startup.acc menu (Choice C), except that after entering the account number, the program will skip the "add charges to files" option and will procede directly to the "View data in file..." prompt.

At the end of the program, you will be returned to the startup menu.

*****

--If you selected "b" or "B", you will see the following question:

Do you want to search the summary file for a key word or phrase?
Enter y or n (for yes or no):

--Entering "n" or "N" will return you to the startup menu.

This program is the same as the keyword search for summary files as detailed for startup.acc menu choice F in the Program Users' Guide. The only significant difference is that if the summary file does not exist (for example, for the year 1992) you will see the following error message:

There is no summary file for fiscal year 92. Please have
OPDAT run the summary creation program.

--At the conclusion of the program, you will be returned to the startup menu.

*****

--If you enter "c" or "C", the following will be displayed:

This is the program which will search through all files to match a
keyword (e.g. training, travel, purchases, etc.), and will collect the
data, to be printed and/or saved to a file, as you wish.

Do you want to use this program?
Enter y or n (for yes or no):

--Entering "n" or "N" will return you to the startup menu.

The remainder of this program is the the same as that of the keyword search of all files discussed for startup.acc menu choice G. At the end of the program, you will again see the startup menu.

*****

--Selecting menu option "D" (or "d") will clear the screen, and display the following question:

Do you want to print files?
Enter y or n (for yes or no):

--Inputting "n" or "N" will return you to the startup menu.

The printing program is essentially the same at that discribed for startup.acc menu choice H in the Program Users' Guide. The only difference is that if you ask to print the summary file, and it does not exist, you will shown the following error message:

The summary file does not exist. Please ask OPDAT to create it.

--The other files, if they exist and printouts are requested, will be printed. Upon completion of the program, you will return to the startup menu.

*****

--When you are finished with the startup program, enter "q" or "Q" to exit. you will then see the computer's command prompt (here assumed to be "$>").

REFERENCES

1) Arthur, Lowell Jay; UNIX Shell Programming,Second Edition; John Wiley and Sons, Inc.; New York, NY; 1990.

2) Kernighan, Brian W. and Rithchie, Dennis M.; The C Programming Language, Second Edition; Prentice Hall; Englewood Cliffs, NJ; 1988.

3) Waite, Mitchell and Prata, Stephen; The Waite Group's New C Primer Plus; Howard W. Sams and Company; Carmel, IN; 1990.

4) Waite, Michell, and Martin, Donald, and Prata, Stephen; The Waite Group's UNIX Primer Plus; Howard W. Sams and Company; Carmel, IN; 1989.

5) Wood, Patrick H. and Kochan, Stephen G.; UNIX System Security; Hayden Books; Macmillan Publishing Co.; Carmel, IN; 1985.

Appendix A: Directory Trees for the Accounting Programs

/usr/OPDAT

startup.acc

startup

/usr/OPDAT/account.d

anscheck.c	indlabor.c
anscheck.x	indlabor.x
cfunct.h	keyword
checkdir	labor.form
chgnonlb.c	masslabr.c
chgnonlb.x	masslabr.x
date.get	nonlabor.form
date_it.c	printit
date_it.x	setup.1
edit.data	setup.2
findit	sum.search
form.gen.c	update
gen.labor	yes.or.no
getinfo.c	temp.d
getinfo.x	account.fy??.d

/usr/OPDAT/small.acc.d

anscheck.x
edit.data.2
keyword.2
labor.form
nonlabor.form
printit.2
sum.search.2
yes.or.no.2
temp.d

/usr/OPDAT/small.acc.d/temp.d

temp.ans

/usr/OPDAT/account.d/temp.d

foundfile	temp.matchfile
matchfile.temp	temp.out
temp.1	temp.prnt.l
temp.ans	temp.prnt.n
temp.date	temp.prnt.s
temp.info	

/usr/OPDAT/account.d/account.fy??.d

sum.info labor.d nonlabor.d

/usr/OPDAT/account.d/account.fy??.d/labor.d

temp.err temp.find1 temp.find2

(Note: These files are temporary and are automatically deleted when no longer required by gen.labor and masslabr.x.)

6.1.d 6.2.d 6.3.d ILIR.d customer.d overhead.d

(Note: These are the sub-directories in which the labor account files are stored.)

/usr/OPDAT/account.d/account.fy??.d/nonlabor.d

6.1.d	6.3.d	customer.d	overhead.d	travel.d
6.2.d	ILIR.d	misc.d	training.d	

(Note: These are the sub-directories in which the non-labor account files are stored.)

Appendix B: C-Language Programs

```
/*anscheck.c is used in conjunction with almost all of the UNIX programs to
  verify that input answers to questions are from a list of allowed variables.
  If not, the program repeats the list to the screen until a value has bee
  selected.

  Argv[1] is the user-input variable, argv[2] is the name of the file contain-
  ing the final answer to the question, and argv[3+] is the list of allowed
  answers to the question asked by the shell.                              */

#include <stdio.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
#define MAX 81

int main(argc,argv)
  int argc;
  char **argv;
  {
    int i, nogood;
    char outval[MAX], *invalue, query;
    FILE *fout;

    invalue= argv[1];
    query= *invalue;
    nogood= TRUE;
    while (nogood == TRUE)
     {
       for(i=3; i<argc; i++)
        {
          if(query == *argv[i])
            break;                  /*Compare the input value to those allowed.*/
        }                           /*In this case, for all shell prog., one- */
                                    /*letter answers are used. */
                                    /*If the function goes through the */
                                    /*entire argument list, then the answer */
        if(i < argc)                /*that the operator input matches none  */
          nogood = FALSE;           /*of the allowed values.  */
        else {
          printf("\nPlease enter one of the following: ");
          for (i=3; i< argc; i++)
            printf("%s ", argv[i]);
          printf(".\n");            /*Reset the value of the pointer to the */
          invalue = gets(outval);   /*new input array, and repeat the loop  */
          query= *invalue;
        }
     }

    if ((fout = fopen(argv[2],"w+")) == NULL)
      printf("The temporary file could not be opened for writing.\n");
    else{
      fprintf(fout,"%c\n", query);
      rewind(fout);
      fclose(fout);
    }
```

```
    return 0;
}
```

```
/* cfunct.h is the header file which contains common routines for the various
   accounting programs.  */

#include <stdio.h>
#include <string.h>

#define TRUE 1
#define FALSE 0

int checkans();
int dayfiltr();
int dayinfy();
void flagcalc();
void initstr();
void labform();
void strfiltr();

/* Checkans compares an input character value against the value in a response
   string.  It returns the matching character to the main program.  This
   function is type int in order to ensure compatibility with various com-
   pilers.       */

int checkans(anstring)
  char *anstring;

  {
    char *ptchr, linestr[81], query;
    int len, nogood, i;

    len= strlen(anstring);
    nogood= TRUE;

    gets(linestr);      /* Read in answer line from terminal. */
    query= *linestr;    /* Actual value of answer is the first character */
                        /* of the answer string.  */
    while(nogood == TRUE)
     { ptchr= anstring;
       for(i=0; i < len; i++, ptchr++)
         { if(query == *ptchr)
             break; }

       if(i < len)      /* If a match has been found, then i will have a */
        nogood= FALSE;  /* value less than that of the answer string (-1 */
       else {           /* for the '\0' character). */
        ptchr= anstring;
        printf("\nPlease enter one of the following: ");
        for(i=0; i< len; i++, ptchr++)
          printf("%c ", *ptchr);
        printf(".\n");
        gets(linestr);
        query= *linestr; }
     }
    return *ptchr;
  }
```

```
/* Dayfiltr checks the days, months, and years input from the main program,
   and returns a value of TRUE if the date is possible, or FALSE if it is not.
   This is essentially a guard against someone playing games with the records.*/

int dayfiltr(month, day, year, fiscyear)
  int month, day, year, fiscyear;  /* fiscyear= directory fiscal year */
   {
     int gooday;

     gooday= TRUE;

     if(year > fiscyear || year < (fiscyear-1))  /* This is not a valid */
      gooday= FALSE;                              /* year.              */

     if(gooday == TRUE){          /* Check if the input month is valid: */
       if(month < 1 || month >12)
        gooday= FALSE;
       else if(month > 9  &&  year==fiscyear)   /* October--December are part */
        gooday= FALSE;                          /* of fiscal year numbered for*/
     }                                          /* the next year.             */

     if(gooday == TRUE){           /* Check if input days are valid: */
       if(day < 1  ||  day > 31)
         gooday= FALSE;
       else if((month==4 || month==6 || month==9 || month==1) && day > 30)
         gooday= FALSE;
       else if(month==2) {
         if((fiscyear % 4) == 0){
           if(day > 29) gooday= FALSE;}
         else{
           if(day > 28) gooday= FALSE;}
       }
     }
    return gooday;
  }

/* Dayinfy is designed to read the current month, day, and year from a
   program and return the number of the day in the current fiscal year
   (which starts on 1 October). */

int dayinfy(month, day, year,fiscyear)
  int month, day, year,fiscyear;
  {
    int fy_day, leap_yr;  /* fy_day= #days into fiscal year*/
    {

/* Check if this current year is a leap year: */

     if ((year % 4) == 0)
      leap_yr= TRUE;
     else
      leap_yr= FALSE;

/* Calculate the day into the fiscal year and return that value. */
```

```
/* Check if the current day is a part of the current fiscal year (fiscyear): */

      if (year > fiscyear){        /* The given day is not a part of the */
       fy_day= 0;}                 /* fiscal year. */

      else if ((month < 10) && (year < fiscyear)) {
       fy_day= 0;}                /* If these months were in the fiscal year,*/
                                  /* they would have the same calendar year  */
                                  /*  as the fiscal year.                    */

      else if ((month >= 10) && (year < (fiscyear-1) || year==fiscyear)) {
       fy_day= 0;}                /* For a current fiscal year, these months */
                                  /* are always 1 calendar year behind the   */
                                  /* current fiscal year.                    */
      else if (month == 10) {
       fy_day= day;}

      else if(month == 11){
       fy_day= day + 31;}

      else if(month == 12){
       fy_day= day + 61;}

      else if(month == 1){
       fy_day= day + 92;}

      else if(month == 2){
       fy_day= day + 123;}

      else if(month == 3){
       { if(leap_yr == TRUE)
          fy_day= day + 152;
         else
          fy_day= day + 151; }
      }
      else if(month == 4) {
       { if(leap_yr == TRUE)
          fy_day= day + 183;
         else
          fy_day= day + 182; }
      }
      else if(month == 5) {
       { if(leap_yr == TRUE)
          fy_day= day + 213;
         else
          fy_day= day + 212; }
      }
      else if(month == 6) {
       { if(leap_yr == TRUE)
          fy_day= day + 244;
         else
          fy_day= day + 243; }
      }
      else if(month == 7) {
       { if(leap_yr == TRUE)
```

```
            fy_day= day + 274;
          else
            fy_day= day + 273; }
        }
        else if(month == 8) {
         { if(leap_yr == TRUE)
            fy_day= day + 305;
          else
            fy_day= day + 304; }
        }
        else {
          if(leap_yr == TRUE)
            fy_day= day + 336;
          else
            fy_day= day + 335;
        }

      }
      return fy_day;
    }

/* Flagcalc calculates the value of the flags on an account: */

void flagcalc(startbal, currbal, timeleft, flagptr)
  float startbal, currbal;
  int timeleft;
  char *flagptr;
  {
    char flagit[8];
    static char t1[5]= "--TE";
    static char t2[5]= "-TE*";
    static char space[5]= "----";

    if(currbal < 0.01)
     sprintf(flagit, "B--");
    else if(currbal < (0.2*startbal))
     sprintf(flagit, "B*-");
    else
     sprintf(flagit, "---");

    if(timeleft <= 0)
      strcat(flagit, t1);
    else if(timeleft < 30)
      strcat(flagit, t2);
    else
      strcat(flagit, space);
    strcpy(flagptr, flagit);

    return;
  }

/* Function initstr reinitializes the value of an input string. */

void initstr(string1)
  char *string1;
```

```
  {
    int len, i;
    char ch= '\0';

    len= strlen(string1);

    for(i=0; i< (len+1); i++)
      *(string1 + i) = ch;

    return;
  }

/* Function labform adds the labor header as required to the account file: */

void labform(accfile)
  FILE *accfile;
  {
    static char *form[6]= {
"B=Bankrupt, B*= <20% of Starting Balance Remains, ODN=Office Document Number,",
"TE= Time Expired, TE*= <30 Days Remain",
"-------|---------|----------------------------|---------|---------|----------|
-----------|---------|----------|",
" Acct. | Date Of |         Transaction        |Rate ($)/|  Hours  | Transact.|
  Current  |  Days   |Cumulative|",
" Flags |  Entry  |         Description        |  Hour   | Entered |Amount ($)|
Balance ($)|Remaining|  Hours   |",
"-------|---------|----------------------------|---------|---------|----------|
-----------|---------|----------|"};

/* Note: The last 4 pointers are each 1 line long, but are printed as two lines
         due to the format of this report page.  For a true representation of
         the form of this header, see the sample account files at the end of the
         Program Users' Guide.  */

   static char ch='\n';
   int i;

   for(i=0; i<6; i++)
     fprintf(accfile, "%s%c", form[i], ch);

   return;
}

/* String Filter provides a means of filtering a response string of up to 81
   characters (string2) into a string of proper length for the main program
   (string1, len1)  */

void strfiltr(string1, len1)
  char *string1;
  int len1;
   {
     char string2[81];
     int i, len2;

     gets(string2);
```

```
    len2= strlen(string2);

    for(i=0; i<(len1 -1) && i<len2; i++)
      *(string1 + i) = *(string2 + i);

    *(string1 + i) = '\0';

    return;
}
```

```
/* chgnonlb.c is the program to change individual non-labor accounts.  It is
   called from a UNIX program named edit.data.  This program takes 5 command
   line arguments: argv[1] = account file name with path, argv[2] = account no.,
   argv[3]= the number of lines in the file as it is opened, argv[4] = name of
   file containing today's date and day in the fiscal year, and argv[5] = the
   fiscal year, itself.
*/
#include <stdio.h>
#include <string.h>
#include "cfunct.h"

#define TRUE 1
#define FALSE 0
#define NLBFMT "%-7s|%-9s|%-29s|%10.2f|%11.2f|%6d\n"

void exit();

int checkans(); /* These functions are listed in header file cfunct.h. */
int dayinfy();
int dayfiltr();
void flagcalc();
void strfiltr();

float addmoney();  /* These functions are listed below. */
void delmoney();
int changedt();
void purch();
void travel();
void award();
void train();
void misc();
void correct();
void writenl();
void formprnt();

struct nonlabor {
  char flag[8];
  char date[9];
  char transact[30];
  float transtot;
  float currbal;
  int daysleft; };

int main(argc,argv)
  int argc;
  char **argv;
  {
    struct nonlabor newchrg;
    int todaynum, fiscyear, numline, newline, month, day, year,
        oldday, oldexday, remday, expmonth, expday, expyear, expdayyr,
        badtime, badmoney, goodate, changeflg, ch;
    float initbal;
    FILE *datefile, *accfile;
    char todate[9], readstr1[32], resp, mm[4], dd[4], yy[3];
    static char yestring[5]= "yYnN";
```

```
    static char menu[21]= "aAbBcCdDeEfFgGhHiIqQ";
    static char quitstr[5]= "aAqQ";
    long jump= (long) sizeof(char);

/* Open datefile, read today's date and day in year, close file: */

   if((datefile= fopen(argv[4], "r")) == NULL) {
     printf("\n Can't open file containing today's date.  Exiting program.\n");
     exit(1); }

   fgets(todate, 9, datefile);
   fscanf(datefile, "%d", &todaynum);
   rewind(datefile);
   fclose(datefile);
   if(todaynum==0){
     printf("\nThe current year does not match the fiscal year.");
     printf("\nYou may only view and/or print data in this account.");
     printf("\n\nEnter any key to continue: ");
     ch= getchar();
     exit(0); }

/* Open account file for reading and appending data: */

   if ((accfile= fopen(argv[1], "a+")) == NULL) {
     printf("\nCan't open account file.  Exiting program.\n");
     exit(1); }

/* Get the following from command line data (converting from string to int): */

   fiscyear= atoi(argv[5]); /* Directory fiscal year (not necessarily current)*/
   numline= atoi(argv[3]); /* Number of lines in the file */

   newline=1;

   if(numline < 20) {  /* If this is true, the account file is new. */
     while (newline < 5)                    /* Go to beginning of 5th line. */
      if((ch= getc(accfile))  == '\n') newline++;

     fgets(readstr1,20,accfile);            /* Skip over the header label. */
     newchrg.currbal= 0.0;
     while((ch= getc(accfile)) != '\n')
     { if((ch > '/' && ch < ':') || ch=='.' || ch=='-') {
        ungetc(ch, accfile);
        if((fscanf(accfile, "%f", &newchrg.currbal) == 1) &&
          (newchrg.currbal < 0.0)) newchrg.currbal= 0.0;
       }
     }
     newline++;

     while (newline < 8)
      {if((ch=getc(accfile)) == '\n')       /* Go to beginning of 8th line. */
         newline++;
      }

/* Read in account expiration date: */
```

```
    fgets(readstr1,25,accfile);
    if(fscanf(accfile,"%d %c %d %c %d", &expmonth, &ch, &expday, &ch,
        &expyear) != 5) {
      expmonth= 0;
      expday= 0;
      expyear= 0; }
    expdayyr= dayinfy(expmonth, expday, expyear, fiscyear);
    goodate= dayfiltr(expmonth, expday, expyear, fiscyear);

    remday= expdayyr - todaynum;
    newchrg.daysleft= remday;
    initbal= newchrg.currbal;
    newchrg.transtot= 0.0;

    if(initbal > 0.0  &&  remday >= 0  &&  goodate==TRUE){
      numline++;
      badmoney= FALSE;
      badtime= FALSE;
    }
    else {
      if(initbal <= 0.0)
        badmoney= TRUE;
      else
        badmoney= FALSE;

      if(remday < 0  || goodate==FALSE)
        badtime= TRUE;
      else
        badtime= FALSE;
    }

  }
  else{   /* The file is not new. */

/* The file should be set to the start of line 20.  The initial balance */
/* needs to be read from the first line of data: */

    while(newline < 20)
      if((ch= getc(accfile)) == '\n') newline++;

    fseek(accfile, 60L*jump, 1);
    fscanf(accfile, "%f", &initbal);

    /* Go to the last line of the file and read in the last transaction: */

    fseek(accfile, -78L*jump, 2);
    fgets(newchrg.flag, 8, accfile);
    fscanf(accfile,"%c %d %c %d %c %d %c ",&ch, &month, &ch, &day, &ch,
          &year, &ch );
    fgets(newchrg.transact, 30, accfile);
    fscanf(accfile, "%c %f %c %f %c %d", &ch, &newchrg.transtot, &ch,
          &newchrg.currbal, &ch, &newchrg.daysleft);

    sprintf(mm, "%d%c", month, '/');
```

```
      sprintf(dd, "%d%c", day, '/');
      sprintf(yy, "%d", year);
      strcpy(newchrg.date, mm);
      strcat(newchrg.date, dd);
      strcat(newchrg.date, yy);

      printf("\n Initial Balance= $%.2f\n", initbal);
      printf("\n Prev. Flags= %s\n Date of Last Entry= %s\n", newchrg.flag,
             newchrg.date);
      printf("\n Prev. Transaction= %s\n Transact. Amount= $%.2f\n",
             newchrg.transact, newchrg.transtot);
      printf("\n Previous Account Balance= $%.2f", newchrg.currbal);
      printf("\n Prev. Days Remaining= %d\n", newchrg.daysleft);
      printf("\nEnter any key to continue: ");
      ch= getchar();

   /* In order to calculate the days remaining until the end of the account,
      use the date of last entry, and its days remaining, and refigure
      today's time remaining.  (This is required in case the expiration date has
      already been altered from the original.)                                  */

      oldday= dayinfy(month, day, year, fiscyear);
      oldexday= oldday + newchrg.daysleft;
      remday= oldexday - todaynum;

      if(remday < 0)
        badtime= TRUE;
      else
        badtime= FALSE;

      if(newchrg.currbal <= 0.0)
        badmoney= TRUE;
      else
        badmoney= FALSE;
    }

   /* Position computer file pointer to EOF: */

      fseek(accfile, 0L, 2);

   /* Check whether time or money need to be changed:    */

      if(badmoney == TRUE){
        printf("\n\nThis account is bankrupt (i.e. current balance = $0.00).\n");
        printf("Enter A to add money to the account or Q to quit this part of\n");
        printf("the program: ");
        resp= checkans(quitstr);
        if(resp == 'q' || resp == 'Q'){
          rewind(accfile);
          fclose(accfile);
          exit(0); }
        else {
          initbal= addmoney(&newchrg, accfile, todate, remday, initbal, &numline,
                            argv[2]);
          if(newchrg.currbal <= 0.0){
```

```
      rewind(accfile);
      fclose(accfile);
      exit(0); }
  }
}
if(remday==0){
  printf("\nToday is the last day this account can be worked on.  Do\n");
  printf("you want to add more time?  (Enter y or n (for yes or no):) ");
  resp=checkans(yestring);
  if(resp=='Y' || resp=='y') badtime= TRUE; }
else if(remday < 0){
  printf("\n\nThe expiration date for the account balance has been exceeded.");
  printf("\nEnter A to add time or Q to quit this part of the program: ");
  resp= checkans(quitstr);
  if(resp=='q' || resp=='Q'){
    rewind(accfile);
    fclose(accfile);
    exit(0); }
}
if(badtime == TRUE){
  remday= changedt(&newchrg, accfile, todate, remday, initbal, fiscyear,
                   todaynum, &numline, argv[2]);
  if(remday < 0){
    rewind(accfile);
    fclose(accfile);
    exit(0); }
}

changeflg= TRUE;
while (changeflg == TRUE)
  { system("clear");    /* Clear screen */
    printf("\nAccount No. %s has a balance of $%.2f and expires",
             argv[2], newchrg.currbal);
    if(remday > 1)
      printf(" in %d days.", remday);
    else if(remday == 1)
      printf(" in 1 day.");
    else
      printf(" today.");

    printf("\n\nHere is the change menu:\n\n    A) Add money to account.\n");
    printf("    B) Delete money from account.\n    C) Change expiration");
    printf(" date.\n    D) Enter purchase costs.\n    E) Enter travel ");
    printf("charges.\n    F) Enter award costs.\n    G) Input training ");
    printf("costs.\n    H) Add miscellaneous charges.\n    I) Correct a ");
    printf("previous entry.");
    printf("\n\nEnter a letter, or Q to quit adding non-labor charges: ");
    resp= checkans(menu);
    if(resp == 'A' || resp=='a')
      initbal= addmoney(&newchrg, accfile, todate, remday, initbal, &numline,
                argv[2]);
    else if(resp=='B' || resp=='b')
      delmoney(&newchrg, accfile, todate, remday, initbal, &numline,
               argv[2]);
    else if(resp=='C' || resp=='c')
```

```
        remday= changedt(&newchrg, accfile, todate, remday, initbal, fiscyear,
                todaynum, &numline, argv[2]);
      else if(resp=='D' || resp=='d')
        purch(&newchrg, accfile, todate, remday, initbal, &numline, argv[2]);
      else if(resp=='E' || resp=='e')
        travel(&newchrg, accfile, todate, remday, initbal, &numline, argv[2]);
      else if(resp=='F' || resp=='f')
        award(&newchrg, accfile, todate, remday, initbal, &numline, argv[2]);
      else if(resp=='G' || resp=='g')
        train(&newchrg, accfile, todate, remday, initbal, &numline, argv[2]);
      else if(resp=='H' || resp=='h')
        misc(&newchrg, accfile, todate, remday, initbal, &numline, argv[2]);
      else if(resp=='I' || resp=='i')
        correct(&newchrg, accfile, todate, remday, initbal, &numline, argv[2]);

      if(resp != 'q'  &&  resp != 'Q') {
        printf("\n\nAnother charge (of any kind---travel, award, etc.)");
        printf(" to account %s? \nEnter y or n: ", argv[2]);
        resp=checkans(yestring);
        if(resp=='n' || resp=='N')
          changeflg= FALSE;
      }
      else
        changeflg= FALSE;

    } /* end while changeflg */

   rewind(accfile);
   fclose(accfile);
   return 0;
 }

/* Addmoney adds money to an account, and writes the value into the file */

float addmoney(strucptr, fileptr, indate, remday, startbal, lines, accnum)
  struct nonlabor *strucptr;
  FILE *fileptr;
  char *indate, *accnum;
  int remday, *lines;
  float startbal;
  {
    int getgoing, getinfo;
    float newmoney, newbal;
    char resp, string1[30], string2[19], linestr[81], flag[8];
    static char yestring[5]= "yYnN";

    printf("\nAdd money to account %s?  (Enter y or n (for yes or no):) ",
            accnum);
    resp= checkans(yestring);
    if(resp=='n' || resp=='N')
      getgoing= FALSE;
    else
      getgoing= TRUE;

    while(getgoing == TRUE)
```

```
{ getinfo = TRUE;
  while(getinfo==TRUE)
   { printf("\nEnter reason to add money to account (18 char or less;");
     printf("\nfor ex.: ODN 47-92): ");
     strfiltr(string2, 19);
     printf("\nEnter amount of money to be added, in dollars.  DO NOT\n");
     printf("use letters, the dollar sign ($), or commas (i.e. $1,000\n");
     printf("or $10k are not acceptable): ");

     newmoney= -1.0;
     while(newmoney < 0.0)
     { while((scanf("%f", &newmoney)) == 0)
       { gets(linestr);
         printf("\nPlease re-enter amount of money to be added: ");
       }
       if(newmoney < 0.0){
         printf("\nEntering a value less than zero is not acceptable.\n");
         printf("Please re-enter amount: "); }
     }
     resp= getchar();
     printf("\nThe reason to add money is %s, and the amount to be",
              string2);
     printf("\nadded is $%.2f.  Are these values correct?", newmoney);
     printf("\nEnter y or n: ");
     resp= checkans(yestring);
     if(resp=='y' || resp=='Y') {
       getinfo= FALSE;
       if(newmoney < 0.01)
         getgoing= FALSE;    /* No change to acct. with a zero entry. */
     }                       /* This is also a quick exit.            */
  }  /* end while getinfo */

  if(getgoing == TRUE) {
    if(startbal < 0.01){
      startbal= newmoney;
      newbal= newmoney;
    }
    else
      newbal= strucptr->currbal + newmoney;

    newmoney *= -1.0; /* Charges are written as positive #'s.  Adding */
                      /* money is therefore a negative file entry.    */

    sprintf(string1, "Add Money: ");
    strcat(string1, string2);
    flagcalc(startbal, newbal, remday, flag);
    writenl(strucptr,fileptr,flag,indate,string1,newmoney,newbal,remday);
    (*lines)++;
    if((*lines % 56) == 0){       /* Write a form feed to the acct. */
      putc('\f', fileptr);        /* file, then have system append  */
      formprnt(fileptr);          /* header lines to acct. file.    */

      writenl(strucptr,fileptr,flag,indate,string1,newmoney,newbal,
              remday);
      *lines += 10;
```

```
        }
        printf("\n\nThe current balance is $%.2f.", strucptr->currbal);
        printf("\nAdd more money to account %s? (Enter y or n:) ", accnum);
        resp= checkans(yestring);
        if(resp=='n' || resp=='N')
          getgoing= FALSE;
      }
  } /* end while getgoing */

  return startbal;
}


/* Delmoney deletes money from an account, and writes the new balance into the
   account records.    */

void delmoney(strucptr, fileptr, indate, remday, startbal, lines, accnum)
  struct nonlabor *strucptr;
  FILE *fileptr;
  char *indate, *accnum;
  int remday, *lines;
  float startbal;
  {
    int getgoing, getinfo;
    float newmoney, newbal;
    char resp, string1[30], string2[18], linestr[81], flag[8];
    static char yestring[5]= "yYnN";

    printf("\nDelete money from account %s? \nEnter y or n ", accnum);
    printf("(for yes or no): ");
    resp= checkans(yestring);
    if(resp=='n' || resp=='N')
      getgoing= FALSE;
    else
      getgoing= TRUE;

    while(getgoing == TRUE)
      { getinfo = TRUE;
        while(getinfo==TRUE)
         { printf("\nEnter reason to delete money (17 char. or less;");
           printf("\nfor ex.: Ref. ODN 391-92): ");
           strfiltr(string2, 18);
         printf("\nEnter amount of money to be deleted, in dollars.  DO NOT\n");
           printf("use letters, the dollar sign ($), or commas (i.e. $1,000\n");
           printf("or $10k are not acceptable): ");
           newmoney= -1.0;
           while(newmoney < 0.0)
           { while((scanf("%f", &newmoney)) == 0)
             { gets(linestr);
               printf("\nPlease re-enter amount of money to be deleted: ");
             }
             if(newmoney < 0.0) {
               printf("\nEntering a value less than zero is not acceptable.\n");
               printf("Please re-enter amount of money to be deleted: "); }
           }
```

```
            resp= getchar();
            printf("\nThe reason to delete money is %s, and the amount to be",
                    string2);
            printf("\ndeleted is $%.2f.  Are these values correct?", newmoney);
            printf("\nEnter y or n: ");
            resp= checkans(yestring);
            if(resp=='y' || resp=='Y') {
              getinfo= FALSE;
              if(newmoney < 0.01)   /* No change to acct. file is made with */
                getgoing= FALSE;    /* a zero entry. This is also a quick exit*/
            }
          } /* end while getinfo */

          if(getgoing == TRUE) {
            newbal= strucptr->currbal - newmoney;
            if(newbal< 0.0){
              printf("\nThis transaction will cause the account balance to be\n");
              printf("less than zero, and will not be allowed.\n");
            }
            else {
              if(newbal<0.01){
                printf("\nWarning: This transaction will zero (bankrupt) the");
                printf("\n         account. No further deletions are allowed.\n");
                getgoing= FALSE;
              }
              sprintf(string1, "Del. Money: ");
              strcat(string1, string2);
              flagcalc(startbal, newbal, remday, flag);
              writenl(strucptr,fileptr,flag,indate,string1,newmoney,newbal,
                      remday);
              (*lines)++;
              if((*lines % 56) == 0){
                putc('\f', fileptr);
                formprnt(fileptr);
                writenl(strucptr,fileptr,flag,indate,string1,newmoney,newbal,
                        remday);
                *lines += 10;
              }
            }
            if(getgoing== TRUE){
              printf("\n\nThe current balance is $%.2f.\n", strucptr->currbal);
              printf("\nRemove more money from account %s? (Enter y or n:) ",
                     accnum);
              resp= checkans(yestring);
              if(resp=='n' || resp=='N')
                getgoing= FALSE;
            }
          }
        } /* end while getgoing */

        return;
      }

/* Function changedate changes the expiration date, and determines the effect
   on the account.  It returns the new number of days remaining to the main
```

```
   program.   */

int changedt(strucptr, fileptr, todate, remday, initbal, fiscyr, thisday,
             lines,  accnum)
  struct nonlabor *strucptr;
  FILE *fileptr;
  char *todate, *accnum;
  int remday, fiscyr, thisday, *lines;
  float initbal;
  {
    int month, day, year, yr_day, getgoing, getinfo, goodate, moreinfo;
    char resp, string1[30], string2[19], linestr[81], flag[8];
    static char yestring[5]= "yYnN";

    printf("\nChange the expiration date for account %s? (Enter y or n:) ",
              accnum);
    resp= checkans(yestring);
    if(resp=='n' || resp=='N')
      getgoing= FALSE;
    else
      getgoing= TRUE;

    if(getgoing == TRUE) {
      getinfo= TRUE;
      while(getinfo==TRUE)
      { printf("\nEnter the number of the month (1-12) of the new date: ");
        while((scanf("%d", &month)) == 0)
        { gets(linestr);
          printf("\nPlease re-enter the month: ");
        }
        printf("\nEnter the day of the new expiration date: ");
        while((scanf("%d", &day)) == 0)
        { gets(linestr);
          printf("\nPlease re-enter the day: ");
        }
        year= -1;
        while(year< 0  || year >99)
        { printf("\nNow input the last two digits of the year: ");
          while((scanf("%d", &year))== 0)
          { gets(linestr);
            printf("\nPlease re-enter the year: ");
          }
        }
        resp= getchar();
        printf("\nYou have entered the following date: %d/%d/%d.", month,
              day, year);
        printf("\nIs this correct? (Enter y or n (for yes or no):) ");
        resp= checkans(yestring);
        if(resp=='y' || resp == 'Y') {
          goodate= dayfiltr(month, day, year,fiscyr);
          if(goodate==FALSE)
            printf("\nThe date you have entered is not acceptable.\n");
          else {
            yr_day= dayinfy(month, day, year, fiscyr);
            if((yr_day - thisday) >= 0){
```

```
            getinfo= FALSE;
            remday = yr_day - thisday;
          }
          else {
            printf("\nThe date you have entered is earlier than today.\n");
            printf("Please re-enter the date: ");
          }
        }
      }
    } /* end while getinfo */

    sprintf(string1, "Exp. Date: ");
    moreinfo= TRUE;
    while(moreinfo == TRUE)
    { printf("\nPlease enter reason for changing the exp. date (18 char.\n");
      printf("or less).  (For ex.:  Ref Memo 91-92): ");
      strfiltr(string2, 19);
      printf("\nYou have entered: %s.  Is this OK? (Enter y or n:) ",string2);
      resp= checkans(yestring);
      if(resp == 'y' || resp=='Y')
        moreinfo= FALSE;
    }
    strcat(string1,string2);
    flagcalc(initbal, strucptr->currbal, remday, flag);
    writenl(strucptr, fileptr, flag, todate, string1, 0.0, strucptr->currbal,
            remday);
    (*lines)++;
    if((*lines % 56) == 0){
      putc('\f', fileptr);
      formprnt(fileptr);
      writenl(strucptr, fileptr, flag, todate, string1, 0.0,
              strucptr->currbal, remday);
      *lines += 10;
    }

  } /* end if getgoing */

  return remday;
}

/* Function purch adds general purchase and equipment charges to an account: */

void purch(strucptr, fileptr, todate, remday, initbal, lines, accnum)
  struct nonlabor *strucptr;
  FILE *fileptr;
  char *todate, *accnum;
  int remday, *lines;
  float initbal;
  {
   int getgoing, getinfo, goodbuy;
   float tempamt, newtot;
   char resp, str1[30], str2[7], str3[5], str4[12], linestr[81];
   char flag[8];
   static char yestring[5]= "yYnN";
```

```
printf("\nAdd purchases to account %s? \nEnter y or n", accnum);
printf(" (for yes or no): ");
resp= checkans(yestring);
if(resp=='y' || resp=='Y')
  getgoing= TRUE;
else
  getgoing= FALSE;

while(getgoing == TRUE)
{ getinfo= TRUE;
  while(getinfo == TRUE)
  {  printf("\nEnter Office Document Number for Purchase Requests (6 char.");
     printf(" or less): ");
     strfiltr(str2, 7);
     printf("\nEnter 11 char. Requisition Number: ");
     strfiltr(str4, 12);

     printf("\nEnter the total cost of the purchase, in dollars.\n");
     printf("\nDO NOT use a comma, dollar sign ($), or letter (i.e. $10K\n");
     printf("or 1,000 are not acceptable): ");
     while((scanf("%f", &tempamt)) == 0)
     { gets(linestr);
       printf("\nPlease re-enter total purchase price: ");
     }
     resp= getchar();
     printf("\nHere are the values you've entered:\n\n  ");
     printf("Office Number = %s,\n  Requisition Number = %s,\n", str2, str4);
     printf("  Purchase Cost = $%.2f.", tempamt);
     printf("\n\nAre these OK?  (Enter y or n:) ");
     resp= checkans(yestring);
     if(resp=='y' || resp=='Y')
       getinfo= FALSE;
  }
  if(tempamt >= 0.01 || tempamt <= -0.01){
                        /* Add changes to account only if this is true. */
    newtot= strucptr->currbal - tempamt;

    if(newtot < 0.0) {
      printf("\nThis purchase can not be applied to this account because\n");
      printf("the balance would then be less than zero.\n");
      goodbuy= FALSE;
    }
    else if(newtot < 0.01){
      printf("\nThis purchase will bankrupt the account.  Do you still\n");
      printf("want to charge this purchase to this account?  (Enter y or n:) ");
      resp= checkans(yestring);
      if(resp=='Y' || resp=='y') {
        goodbuy= TRUE;
        getgoing= FALSE;
      }
      else
        goodbuy= FALSE;
    }
    else
      goodbuy= TRUE;
```

```
      if(goodbuy == TRUE) {
        sprintf(str1, "EP: ODN=");
        sprintf(str3, " RN=");
        strcat(str1,str2);
        strcat(str1,str3);
        strcat(str1,str4);
        flagcalc(initbal, newtot, remday, flag);
        writenl(strucptr, fileptr,flag, todate, str1, tempamt, newtot, remday);
        (*lines)++;
        if((*lines % 56) == 0){
          putc('\f', fileptr);
          formprnt(fileptr);
          writenl(strucptr, fileptr,flag, todate, str1, tempamt, newtot,
                  remday);
          *lines += 10;
        }
      }
    }
    else {
      printf("\nNo entry was made to the account since the total ");
      printf("cost = $0.00.\n");
    }

    if(getgoing == TRUE) {
      printf("\n\nThe current account balance is: $%.2f.\n",
                strucptr->currbal);
      printf("Charge another purchase to account %s? (Enter y or n:) ",
              accnum);
      resp= checkans(yestring);
      if(resp=='n' || resp=='N')
        getgoing= FALSE;
    }
  } /* end while getgoing */

   return;
}

/* Function travel adds travel charges to the account.  */

void travel(strucptr, fileptr, thisdate, timeleft, origbal, lines, accnum)
  struct nonlabor *strucptr;
  FILE *fileptr;
  char *thisdate, *accnum;
  int timeleft, *lines;
  float origbal;
  {
   int getgoing, getinfo, goodtrav;
   float cost, newbal;
   char flag[8];
   char linestr[81], trav1[30], resp, name[15], travnum[9], travtype[13];
   static char yestring[5]= "yYnN";
   static char choozit[11]= "aAbBcCdDeE";
   static char space[2]= " ";
```

```
printf("\nAdd travel charges to account %s? \nEnter y or n ", accnum);
printf("(for yes or no): ");
resp= checkans(yestring);
if(resp=='y' || resp=='Y')
  getgoing= TRUE;
else
  getgoing= FALSE;

while(getgoing == TRUE)
{ getinfo= TRUE;
  while(getinfo == TRUE)
  {  printf("\nChoose the type of travel:\n\n    A) CONUS\n");
     printf("    B) Invitational\n    C) Local\n    D) OCONUS\n");
     printf("    E) Other\n\nEnter a, b, c, d, or e: ");
     resp= checkans(choozit);
     if(resp=='a' || resp=='A') {
       sprintf(trav1, "TOC ON=");
       sprintf(travtype, "CONUS"); }
     else if(resp=='b' || resp=='B'){
       sprintf(trav1, "TOI ON=");
       sprintf(travtype, "Invitational"); }
     else if(resp=='c' || resp=='C') {
       sprintf(trav1, "TOL ON=");
       sprintf(travtype, "Local"); }
     else if(resp=='d' || resp=='D') {
       sprintf(trav1, "TOO ON=");
       sprintf(travtype, "OCONUS"); }
     else {
       sprintf(trav1, "TOX ON=");
       sprintf(travtype, "Other");
     }
     printf("\nEnter the travel order number (up to 8 characters): ");
     strfiltr(travnum, 9);
     printf("\nEnter the last name of the traveller (up to 14 char.): ");
     strfiltr(name, 15);
     printf("\nEnter the total travel cost, in dollars.  DO NOT use\n");
     printf("commas, the dollar sign ($), or a letter (i.e. $10,000 or\n");
     printf("10K are not acceptable): ");
     while((scanf("%f", &cost)) == 0)
     { gets(linestr);
       printf("\nPlease re-enter travel cost: ");
     }
     resp= getchar();
     printf("\n\nYou have entered the following:\n\n  Travel Type = %s,\n",
            travtype);
     printf("  Travel Order Number = %s,\n  Traveller = %s,\n", travnum,
            name);
     printf("  Total Cost = $%.2f.\n\nAre these OK?  (Enter y or n:) ",
            cost);
     resp= checkans(yestring);
     if(resp=='y' || resp=='Y')
       getinfo= FALSE;

  } /* end while getinfo */
```

```
  if(cost >= 0.01 || cost <= -0.01) {
                /* Add charges to account only if this is true. */
    newbal= strucptr->currbal - cost;

    if(newbal < 0.0) {
      printf("\n\nThis travel charge will force the account balance of\n");
      printf("$%.2f to go below zero.  This transaction is not allowed.\n",
             strucptr->currbal);
      goodtrav= FALSE;
    }
    else if(newbal < 0.01) {
      printf("\n\nThis charge will bankrupt the account. Do you want to");
      printf("\nadd this charge? (Enter y or n:) ");
      resp= checkans(yestring);
      if(resp=='n' || resp=='N')
        goodtrav= FALSE;
      else {
        goodtrav= TRUE;
        getgoing= FALSE;
      }
    }
    else
      goodtrav= TRUE;

    if(goodtrav == TRUE){
      strcat(trav1, travnum);
      strcat(trav1, space);
      strcat(trav1, name);
      flagcalc(origbal, newbal, timeleft, flag);
      writenl(strucptr, fileptr, flag,thisdate,trav1,cost,newbal,timeleft);
      (*lines)++;
      if((*lines % 56) == 0){
        putc('\f', fileptr);
        formprnt(fileptr);
        writenl(strucptr, fileptr, flag,thisdate,trav1,cost,newbal,timeleft);
        *lines += 10;
      }
    }
  }
  else {
    printf("\nNo entry was made to the account since the total ");
    printf("cost = $0.00.\n");
  }

  if(getgoing == TRUE) {
     printf("\n\nThe current balance is $%.2f.  Do you want to charge\n",
              strucptr->currbal);
     printf("more travel to account %s?  (Enter y or n:) ", accnum);
     resp= checkans(yestring);
     if(resp=='n' || resp=='N')
       getgoing= FALSE;
  }
} /* end while getgoing */

return;
```

```
}

/* Function award allows the operator to deduct award charges */

void award(strucptr, fileptr, thisdate, timeleft, origbal, lines, accnum)
  struct nonlabor *strucptr;
  FILE *fileptr;
  char *thisdate, *accnum;
  int timeleft, *lines;
  float origbal;
  {
   int getgoing, getinfo, goodawrd;
   float cost, newbal;
   char flag[8];
   char linestr[81], reason[30], resp, name[13], ref[6], docnum[7];
   static char yestring[5]= "yYnN";

   printf("\nAdd award charges to account %s? \nEnter y or n ", accnum);
   printf("(for yes or no): ");
   resp= checkans(yestring);
   if(resp=='y' || resp=='Y')
     getgoing= TRUE;
   else
     getgoing= FALSE;

   while(getgoing == TRUE)
   { getinfo= TRUE;
     while(getinfo == TRUE)
     {  printf("\nEnter the last name (up to 12 char.) of the person");
        printf("\nreceiving the award: ");
        strfiltr(name, 13);
        printf("\nEnter the office document no. (up to 6 char.) of the award: ");
        strfiltr(docnum, 7);
        printf("\nEnter the amount of the award, in dollars.  DO NOT use\n");
        printf("commas, the dollar sign ($), or a letter (i.e. $10,000 or\n");
        printf("10K are not acceptable): ");
        while((scanf("%f", &cost)) == 0)
        { gets(linestr);
          printf("\nPlease re-enter award cost: ");
        }
        resp= getchar();
        printf("\n\nYou have entered the following:\n\n  Recipient = %s,\n",
               name);
        printf("  Office Doc. Number = %s,\n  Award Cost = $%.2f.", docnum,
               cost);
        printf("\n\nAre these OK?  (Enter y or n:) ");
        resp= checkans(yestring);
        if(resp=='y' || resp=='Y')
          getinfo= FALSE;

     } /* end while getinfo */

     if(cost >= 0.01 || cost <= -0.01) {
       newbal= strucptr->currbal - cost;
```

```
      if(newbal < 0.0) {
        printf("\n\nThis award charge will force the account balance of\n");
        printf("$%.2f to go below zero.  This transaction is not allowed.\n",
               strucptr->currbal);
        goodawrd= FALSE; }
      else if(newbal < 0.01) {
        printf("\n\nThis award will bankrupt the account. Do you want to");
        printf("\nadd this charge? (Enter y or n:) ");
        resp= checkans(yestring);
        if(resp=='n' || resp=='N')
          goodawrd= FALSE;
        else {
          goodawrd= TRUE;
          getgoing= FALSE;
        }
      }
      else
        goodawrd= TRUE;

      if(goodawrd == TRUE){
        sprintf(reason, "Award-");
        sprintf(ref, " ODN=");
        strcat(reason, name);
        strcat(reason, ref);
        strcat(reason, docnum);
        flagcalc(origbal, newbal, timeleft, flag);
        writenl(strucptr, fileptr, flag,thisdate,reason,cost,newbal,timeleft);
        (*lines)++;
        if((*lines % 56) == 0){
          putc('\f', fileptr);
          formprnt(fileptr);
          writenl(strucptr,fileptr,flag,thisdate,reason,cost,newbal,timeleft);
          *lines += 10;
        }
      }
    }
    else {
      printf("\nNo entry was made to the account since the total ");
      printf("cost = $0.00.\n");
    }

    if(getgoing == TRUE) {
       printf("\n\nThe current balance is $%.2f.  Do you want to charge\n",
              strucptr->currbal);
       printf("another award to account %s?  (Enter y or n:) ", accnum);
       resp= checkans(yestring);
       if(resp=='n' || resp=='N')
         getgoing= FALSE;
    }
  } /* end while getgoing */

  return;
}

/* Function train adds training charges to the accounts: */
```

```
void train(strucptr, fileptr, todate, remday, initbal, lines, accnum)
  struct nonlabor *strucptr;
  FILE *fileptr;
  char *todate, *accnum;
  int remday, *lines;
  float initbal;
  {
   int getgoing, getinfo, trainOK;
   float tempamt, newtot;
   char resp, name[20], str1[30], linestr[81];
   char flag[8];
   static char yestring[5]= "yYnN";
   static char reason[11]= "Training: ";

   printf("\nAdd training costs to account %s? \nEnter y or n", accnum);
   printf(" (for yes or no): ");
   resp= checkans(yestring);
   if(resp=='y' || resp=='Y')
     getgoing= TRUE;
   else
     getgoing= FALSE;

   while(getgoing == TRUE)
   { getinfo= TRUE;
     while(getinfo == TRUE)
     { printf("\nEnter name of person requesting training and other necessary");
       printf("\ndata (e.g., course title), if any (19 char. or less):\n");
       strfiltr(name, 20);

        printf("\nEnter the total training cost, in dollars.  DO NOT\n");
        printf("use a comma, dollar sign ($), or letter (i.e. $10K or\n");
        printf("1,000 are not acceptable): ");
        while((scanf("%f", &tempamt)) == 0)
        { gets(linestr);
          printf("\nPlease re-enter cost: ");
        }
        resp= getchar();
        printf("\n\nHere are the values you've entered:\n\n");
        printf("  Trainee Information = %s,\n", name);
        printf("  Total Cost = $%.2f.", tempamt);
        printf("\n\nAre these OK?  (Enter y or n:) ");
        resp= checkans(yestring);
        if(resp=='y' || resp=='Y')
          getinfo= FALSE;
     }
     if(tempamt >= 0.01 || tempamt <= -0.01) {
       newtot= strucptr->currbal - tempamt;

       if(newtot < 0.0) {
         printf("\nThis charge can not be applied to this account because\n");
         printf("the balance would then be less than zero.\n");
         trainOK= FALSE;
       }
       else if(newtot < 0.01){
```

```
        printf("\nThis charge will bankrupt the account.  Do you want\n");
        printf("to add the charge to this account?  (Enter y or n:) ");
        resp= checkans(yestring);
        if(resp=='Y' || resp=='y') {
          trainOK= TRUE;
          getgoing= FALSE;
        }
        else
          trainOK= FALSE;
      }
      else
        trainOK= TRUE;

      if(trainOK == TRUE) {
        strcpy(str1, reason);
        strcat(str1, name);
        flagcalc(initbal, newtot, remday, flag);
        writenl(strucptr, fileptr,flag, todate, str1, tempamt, newtot, remday);
        (*lines)++;
        if((*lines % 56) == 0){
          putc('\f', fileptr);
          formprnt(fileptr);
          writenl(strucptr, fileptr,flag, todate, str1, tempamt, newtot,
                  remday);
          *lines += 10;
        }
      }
    }
    else {
      printf("\nNo entry was made to the account since the total ");
      printf("cost = $0.00.\n");
    }

    if(getgoing == TRUE) {
      printf("\n\nThe current account balance is: $%.2f.\n",
              strucptr->currbal);
      printf("Add another training charge to account %s? (Enter y or n:) ",
              accnum);
      resp= checkans(yestring);
      if(resp=='n' || resp=='N')
        getgoing= FALSE;
    }
  } /* end while getgoing */

  return;
}

/* Function misc adds charges from miscellaneous sources: */

void misc(strucptr, fileptr, todate, remday, initbal, lines, accnum)
  struct nonlabor *strucptr;
  FILE *fileptr;
  char *todate, *accnum;
  int remday, *lines;
  float initbal;
```

```
{
 int getgoing, getinfo, goodbuy;
 float tempamt, newtot;
 char resp, str1[30], linestr[81];
 char flag[8];
 static char yestring[5]= "yYnN";

 printf("\nAdd miscellaneous charges to account %s? \nEnter y or n", accnum);
 printf(" (for yes or no): ");
 resp= checkans(yestring);
 if(resp=='y' || resp=='Y')
   getgoing= TRUE;
 else
   getgoing= FALSE;

 while(getgoing == TRUE)
 { getinfo= TRUE;
   while(getinfo == TRUE)
   { printf("\nEnter reason for charge (29 char. or less):\n");
     strfiltr(str1, 30);

     printf("\nEnter the total cost of the charge, in dollars.  DO NOT\n");
     printf("use a comma, dollar sign ($), or letter (i.e. $10K or\n");
     printf("1,000 are not acceptable): ");
     while((scanf("%f", &tempamt)) == 0)
     { gets(linestr);
       printf("\nPlease re-enter cost: ");
     }
     resp= getchar();
     printf("\n\nHere are the values you've entered:\n\n");
     printf("  Reason for Charge = %s,\n", str1);
     printf("  Total Cost = $%.2f.", tempamt);
     printf("\n\nAre these OK?  (Enter y or n:) ");
     resp= checkans(yestring);
     if(resp=='y' || resp=='Y')
       getinfo= FALSE;
   }
   if(tempamt >= 0.01 || tempamt <= -0.01) {
                                        /* Add costs only if this is true. */
     newtot= strucptr->currbal - tempamt;

     if(newtot < 0.0) {
       printf("\nThis charge can not be applied to this account because\n");
       printf("the balance would then be less than zero.\n");
       goodbuy= FALSE;
     }
     else if(newtot < 0.01){
       printf("\nThis charge will bankrupt the account.  Do you want\n");
       printf("to add the charge to this account?  (Enter y or n:) ");
       resp= checkans(yestring);
       if(resp=='Y' || resp=='y') {
         goodbuy= TRUE;
         getgoing= FALSE;
       }
       else
```

```
            goodbuy= FALSE;
        }
        else
          goodbuy= TRUE;

        if(goodbuy == TRUE) {
          flagcalc(initbal, newtot, remday, flag);
          writenl(strucptr, fileptr,flag, todate, str1, tempamt, newtot, remday);
          (*lines)++;
          if((*lines % 56) == 0){
            putc('\f', fileptr);
            formprnt(fileptr);
            writenl(strucptr, fileptr,flag, todate, str1, tempamt, newtot,
                    remday);
            *lines += 10;
          }
        }
      }
      else {
        printf("\nNo entry was made to the account since the total ");
        printf("cost = $0.00.\n");
      }

      if(getgoing == TRUE) {
        printf("\n\nThe current account balance is: $%.2f.\n",
                strucptr->currbal);
        printf("Add another misc. charge to account %s? (Enter y or n:) ",
                accnum);
        resp= checkans(yestring);
        if(resp=='n' || resp=='N')
          getgoing= FALSE;
      }
    } /* end while getgoing */

    return;
  }

/* Function correct is used to correct previous charges (i.e. add the money back
   to the account) */

void correct(strucptr, fileptr, todate, remday, initbal, lines, accnum)
  struct nonlabor *strucptr;
  FILE *fileptr;
  char *todate, *accnum;
  int remday, *lines;
  float initbal;
  {
   int getgoing, getinfo;
   float tempamt, newtot;
   char resp, str1[30], linestr[81];
   char flag[8];
   static char yestring[5]= "yYnN";

   printf("\nCorrect previous charges to account %s? \nEnter y or n", accnum);
   printf(" (for yes or no): ");
```

```
resp= checkans(yestring);
if(resp=='y' || resp=='Y')
  getgoing= TRUE;
else
  getgoing= FALSE;

while(getgoing == TRUE)
{ getinfo= TRUE;
  while(getinfo == TRUE)
  { printf("\nEnter reason to correct old charge (29 char. or less):\n");
     strfiltr(str1, 30);

     printf("\nEnter the total cost of the old charge, in dollars.  DO ");
     printf("NOT\nuse a comma, dollar sign ($), or letter (i.e. $10K or\n");
     printf("1,000 are not acceptable): ");
     tempamt= -1.0;
     while(tempamt < 0.0)
     { while((scanf("%f", &tempamt)) == 0)
       { gets(linestr);
         printf("\nPlease re-enter cost: ");
       }
       if(tempamt < 0.0){
         printf("\nEntering a value less than zero is not allowed.");
         printf("\nPlease re-enter the old cost: "); }
     }
     resp= getchar();
     printf("\n\nHere are the values you've entered:\n\n");
     printf("  Reason for Correction = %s,\n", str1);
     printf("  Total Amount = $%.2f.", tempamt);
     printf("\n\nAre these OK?  (Enter y or n:) ");
     resp= checkans(yestring);
     if(resp=='y' || resp=='Y')
       getinfo= FALSE;
  }
  if(tempamt >= 0.01) {  /* Add charges to account only if this is true. */
    newtot= strucptr->currbal + tempamt;
    tempamt *= -1.0;  /* This causes a negative number to be input to the
                         account file to show a corrected value, because
                         costs are entered as positive numbers in the file.*/

    flagcalc(initbal, newtot, remday, flag);
    writenl(strucptr, fileptr,flag, todate, str1, tempamt, newtot, remday);
    (*lines)++;
    if((*lines % 56) == 0){
      putc('\f', fileptr);
      formprnt(fileptr);
      writenl(strucptr, fileptr,flag, todate, str1, tempamt, newtot, remday);
      *lines += 10;
    }
  }
  else {
    printf("\nNo entry was made to the account since the total ");
    printf("cost = $0.00.\n");
  }
```

```
      printf("\n\nThe current account balance is:  $%.2f.\n",
              strucptr->currbal);
      printf("\nEnter another correction to account %s? (Enter y or n:) ",
                accnum);
      resp= checkans(yestring);
      if(resp=='n' || resp=='N')
        getgoing= FALSE;

    } /* end while getgoing */

    return;
  }

/* This function revises the current value of the structural variable new charge
   and writes the formatted values of the structural variable to the file. */

void writenl(nlptr, fileptr, str1, str2, str3, cost, newbal, timeleft)
  struct nonlabor *nlptr;
  FILE *fileptr;
  char *str1, *str2, *str3;
  float cost, newbal;
  int timeleft;
  {
    static char space1[8]= "       ";
    static char space2[9]= "        ";
    static char space3[30]= "                              ";

    strcpy(nlptr->flag, space1);  /* Re-initialize the strings. */
    strcpy(nlptr->date, space2);
    strcpy(nlptr->transact, space3);

    strcpy(nlptr->flag, str1);
    strcpy(nlptr->date, str2);
    strcpy(nlptr->transact, str3);
    nlptr->transtot= cost;
    nlptr->currbal= newbal;
    nlptr->daysleft= timeleft;

    fseek(fileptr, 0L, 2);  /* Go to EOF */
    fprintf(fileptr, NLBFMT, nlptr->flag, nlptr->date, nlptr->transact,
           cost, newbal, timeleft);

    return;
  }

/*Function formprnt copies the following forms into a file: */

void formprnt(accfile)
 FILE *accfile;
 {
   static char *formnl[9]= {
"B=Bankrupt, B*= <20% of Starting Balance Remains, EP= Equip. or Other Purchase,",
"ODN= Office Document Number, ON= Order Number, RN= Requisition Number",
"TE= Time Expired, TE*= <30 days remain, TOC= Travel Orders (CONUS)",
"TOI= Travel Orders (Invitational), TOL= Travel Orders (Local)",
```

```
"TOO= Travel Orders (OCONUS), TOX= Travel Orders (Other)",
"-------|---------|-----------------------------|----------|-----------|---------",
" Acct. | Date Of |         Transaction         | Transact.|  Current  |  Days",
" Flags |  Entry  |         Description         |Amount ($)|Balance ($)|Remaining",
"-------|---------|-----------------------------|----------|-----------|---------"};

   int i;
   char ch= '\n';

   for(i=0; i<8; i++)
    fprintf(accfile, "%s%c", formnl[i], ch);

   return;
}
```

```
/* date_it.c is designed to read the current month, day, and year from
   a file and return the number of the day in the current fiscal year */

#include <stdio.h>
#include <string.h>
#include "cfunct.h"
#define TRUE 1
#define FALSE 0

int dayinfy();
void exit();
int main (argc, argv)
  int argc;
  char **argv;
  {
    char ch;
    int day, month, year, fy_day, fiscyear;  /* fy_day= #days into fiscal year*/
    FILE *fp;                                /* fiscyear= directory fisc. year*/
    {
      if ((fp = fopen(argv[1], "a+")) == NULL)
       {
         printf("Can't open %s! \n", argv[1]);
         exit(1);
       }
/*
/* Retrieve the data regarding today's date from the the date file, and get the
   fiscal year of the directory in question.  If the account is prepared ahead
   of the directory fiscal year, or is being searched after that year has
   passed, then the current day should not affect the time-to-expire calcula-
   tions required elsewhere.                                                  */
      fscanf(fp,"%d %c %d %c %d ", &month, &ch, &day, &ch, &year);
      fiscyear= atoi(argv[2]);
      fy_day = dayinfy(month, day, year, fiscyear);

      fseek(fp, 0L, 2);             /* Set file pointer to end-of-file. */
      fprintf(fp, "%d\n", fy_day); /* File opened in append mode.  This there-
                                      fore adds fy_day to the end of the file.*/
      fclose(fp);
   }
   return 0;
  }
```

```
/* form.gen.c generates the UNIX files labor.form and nonlabor.form, which are
   used by UNIX program setup.2 when creating the initial data files. */

#include <stdio.h>
#include <string.h>

void exit();

int main()
{
  int i, j;
  FILE *labfile, *nlfile;
  char ch= '\n';

  static char *labform[6] = {
"B=Bankrupt, B*= <20% of Starting Balance Remains, ODN= Office Document Number,",
"TE= Time Expired, TE*= <30 Days Remain",
"-------|--------|---------------------------|--------|--------|---------|
----------|--------|---------|",
" Acct. | Date Of |        Transaction        |Rate ($)/|  Hours  | Transact.|
  Current  | Days   |Cumulative|",
" Flags |  Entry  |        Description        |  Hour   | Entered |Amount ($)|
Balance ($)|Remaining|  Hours   |",
"-------|--------|---------------------------|--------|--------|---------|
----------|--------|---------|" };

/* Note:  The last four pointers are each one line in length.  They are printed
          as two lines due to the format of this report page.  For a true rep-
          resentation of the header file appearance, see Appendix D of this
          report.  */

  static char *nlform[9]= {
"B=Bankrupt, B*= <20% of Starting Balance Remains, EP= Equip. or Other Purchase,",
"ODN= Office Document Number, ON= Order Number, RN= Requisition Number,",
"TE= Time Expired, TE*= <30 days remain, TOC= Travel Orders (CONUS),",
"TOI= Travel Orders (Invitational), TOL= Travel Orders (Local),",
"TOO= Travel Orders (OCONUS), TOX= Travel Orders (Other)",
"-------|--------|---------------------------|---------|----------|--------",
" Acct. | Date Of |        Transaction        | Transact.|  Current  |  Days",
" Flags |  Entry  |        Description        |Amount ($)|Balance ($)|Remaining",
"-------|--------|---------------------------|---------|----------|--------"};

  if((labfile= fopen("labor.form", "w+")) == NULL)
  { printf("\nCannot open file to store form for header of labor accounts.\n");
    printf("Exiting program.\n");
    exit(1);
  }
  for(i=0; i < 6; i++)
    fprintf(labfile, "%s%c", labform[i], ch);

  rewind(labfile);
  fclose(labfile);

  if((nlfile= fopen("nonlabor.form", "w+")) == NULL)
  { printf("\nCannot open file to store form for header of non-labor");
```

```
    printf(" accounts.\nExiting program.\n");
    exit(1);
  }
  for(j=0; j < 9; j++)
    fprintf(nlfile, "%s%c", nlform[j], ch);

  rewind(nlfile);
  fclose(nlfile);


  return 0;
}
```

```
/* getinfo.c creates the sum.info data of current account information.  Input
   from the UNIX program (called update) is as follows:  argv[1]= account file,
   argv[2]= labor or nonlabor indicator, argv[3]= output file name, argv[4]
   is the number of lines in the file (needed to determine if file is new or
   not, argv[5] is the value of the file containing today's date and day in
   year, argv[6] is the value of the fiscal year, and argv[7] is a string con-
   taining the file name. */

#include <stdio.h>
#include <string.h>
#include "cfunct.h"

void exit();

int dayinfy();   /* These functions are defined in the header file cfunct.h. */
void flagcalc();
void initstr();

int main (argc, argv)
  int argc;
  char **argv;
  {
   static char daystr[30]= ": Days Remaining to Account: ";
   static char balstr[21]= ": Current Balance: $";
   static char flag1[18]= ": Account Flags: ";
   static char flag2[45]= "Flags:  B*= <20% Orig. Balance, B= Bankrupt\n";
   static char flag3[51]= "        TE*= <30 Days Remaining, TE= Time Expired\n";
   static char rateln[34]= ": Labor Charge (Rate per Hour): $";
   char flags[8], readstr[81], todate[9], account[15], printstr[110];
   int numline, remday, today, fiscyear, oldday, oldexday, oldrem,
       month, day, year, newline, ch;
   float startbal, bal, rate, transamt, hours;
   FILE *accfile, *outfile, *datefile;
   long jump= (long) sizeof(char);

   if((accfile= fopen(argv[1], "r"))== NULL){
     printf("\nCan't open account file %s.\n", argv[1]);
     exit(1); }

   if((outfile= fopen(argv[3], "w+")) == NULL) {
     printf("\nCan't open temporary output file.\n");
     exit(1); }

   if((datefile= fopen(argv[5], "r")) == NULL) {
     printf("\nCan't open datefile.\n");
     exit(1); }
   fgets(todate, 9, datefile);
   fscanf(datefile, "%d", &today);
   rewind(datefile);
   fclose(datefile);

   numline= atoi(argv[4]); /* Get number of lines in the file. */
   fiscyear= atoi(argv[6]); /* Get directory fiscal year. */
   strcpy(account, argv[7]); /* Get the account number. */
```

```
while((ch=getc(accfile)) != '\n');  /* Read until the first newline char. */

fgets(readstr, 81, accfile); /* Get account classif. and write to outfile. */
sprintf(printstr, "%s : %s", account, readstr);
fputs(printstr, outfile);
initstr(printstr);
initstr(readstr);   /* Re-initialize strings. */

fgets(readstr, 81, accfile); /* Get purpose for account; write to outfile. */
sprintf(printstr, "%s : %s", account, readstr);
fputs(printstr, outfile);
initstr(printstr);
initstr(readstr);

fgets(readstr, 81, accfile); /* Get name of princ. investigator. */
sprintf(printstr, "%s : %s", account, readstr);
fputs(printstr, outfile);
initstr(printstr);
initstr(readstr);

newline=5;

if(numline < 20) {  /* File is a new file */
  bal= 0.0;
  while((ch= getc(accfile)) != '\n')  /* Read up to the new balance: */
  { if((ch > '/' && ch < ':') || ch=='.' || ch=='-') {
      ungetc(ch, accfile);
      if((fscanf(accfile, "%f", &bal) == 1) && (bal < 0.0)) bal= 0.0; }
  }
  newline++;
  if(*(argv[2]) == 'l' || *(argv[2]) == 'L') {
    rate= 0.0;
    while((ch= getc(accfile)) != '\n')  /* Read up to the rate/hour */
    { if((ch > '/' && ch < ':') || ch=='.' || ch=='-') {
       ungetc(ch, accfile);
       if((fscanf(accfile, "%f", &rate) == 1) && (rate < 0.0)) rate= 0.0; }
    }
    newline++;
  }
  while(newline < 8)                    /* Read past the date created to */
  {if((ch= getc(accfile)) == '\n')      /* the expiration date. */
     newline++; }
  fgets(readstr,25, accfile);
  if(fscanf(accfile, "%d %c %d %c %d", &month, &ch, &day, &ch, &year) != 5)
  { month= 0;
    day= 0;
    year= 0; }
  oldexday= dayinfy(month, day, year, fiscyear);
  if(oldexday == 0)
    remday= 365- today;
  else
    remday= oldexday- today;

  flagcalc(bal, bal, remday, flags);
}
```

```
else { /* File is not new: */
  while(newline < 20)
    if((ch= getc(accfile)) == '\n') newline++;

  if(*(argv[2]) == 'l' || *(argv[2]) == 'L') {
      /* The account is a labor account. */

    /* Obtain the starting balance from line 20 of file: */

    fseek(accfile, 80L*jump, 1);
    fscanf(accfile, "%f", &startbal);

    /* Read and print data from last line of account file: */

    fseek(accfile, 0L, 2);          /* Go to EOF. */
    fseek(accfile, -105L*jump, 2); /* Go 105 spaces back from EOF. */
    fscanf(accfile, "%d %c %d %c %d %c %c", &month, &ch, &day, &ch,
         &year, &ch, &ch);
    fgets(readstr, 30, accfile);
    fscanf(accfile, "%c %f %c %f %c %f %c %f %c %d", &ch, &rate, &ch, &hours,
           &ch, &transamt, &ch, &bal, &ch, &oldrem);
    if(today == 0)
      remday= oldrem;
    else {
      oldday= dayinfy(month, day, year, fiscyear);
      remday= oldday + oldrem - today; }
    flagcalc(startbal,bal,remday,flags);
    sprintf(printstr, "%s %s %.2f\n", account, rateln, rate);
    fputs(printstr, outfile);
    initstr(printstr);
    initstr(readstr);
  }
  else {   /* This is a nonlabor account. */

    /* Obtain the starting balance from line 20 of file: */

    fseek(accfile, 60L*jump, 1);
    fscanf(accfile, "%f", &startbal);
    fseek(accfile, 0L, 2);           /* Go to EOF. */
    fseek(accfile, -70L*jump, 2);    /* Go 70 spaces from EOF. */
    fscanf(accfile, "%d %c %d %c %d %c %c", &month, &ch, &day, &ch,
         &year, &ch, &ch);
    fgets(readstr, 30, accfile);
    fscanf(accfile, "%c %f %c %f %c %d", &ch, &transamt, &ch, &bal, &ch,
           &oldrem);
    if(today == 0)
      remday= oldrem;
    else {
      oldday= dayinfy(month, day, year, fiscyear);
      remday= oldday + oldrem - today; }
    flagcalc(startbal, bal, remday, flags);
  }
}
sprintf(printstr, "%s %s%s\n", account, flag1, flags);
fputs(printstr, outfile);
```

```
  fputs(flag2, outfile);     /* Print flags to outfile. */
  fputs(flag3, outfile);
  initstr(printstr);

  sprintf(printstr, "%s %s%.2f\n", account, balstr, bal);
  fputs(printstr, outfile);  /* Print current balance. */
  initstr(printstr);

  sprintf(printstr, "%s %s%d\n\n\n", account, daystr, remday);
  fputs(printstr, outfile);  /* Print days left in account. */

  rewind(accfile);
  fclose(accfile);
  rewind(outfile);
  fclose(outfile);

  return 0;
}
```

```
/* indlabor.c is the program to change individual labor accounts.  It is
   called from a UNIX program named edit.data.  This program takes 5 command
   line arguments: argv[1] = account file name with path, argv[2] = account no.,
   argv[3]= the number of lines in the file as it is opened, argv[4] = name of
   file containing today's date and day in the fiscal year, and argv[5] = the
   fiscal year, itself.
*/
#include <stdio.h>
#include <string.h>
#include "cfunct.h"

#define TRUE 1
#define FALSE 0
#define LABFMT "%-7s|%-9s|%-29s|%9.2f|%9.2f|%10.2f|%11.2f|   %3d   |%10.2f|\n"
#define MINWAGE 3.75

void exit();

int checkans();   /* These functions are listed in header file cfunct.h. */
int dayfiltr();
int dayinfy();
void flagcalc();
void labform();
void strfiltr();

float addmoney();  /* These functions are listed within this program. */
void delmoney();
int changedt();
void costhour();
void laborchg();
void misc();
void writelab();

struct labor {
  char flag[8];
  char date[9];
  char transact[30];
  float hourate;
  float hours;
  float transtot;
  float currbal;
  int daysleft;
  float tothour; };

int main(argc,argv)
  int argc;
  char **argv;
  {
    struct labor newlabor;

    int todaynum, fiscyear, numline, newline, month, day, year,
        oldday, oldexday, remday, expmonth, expday, expyear, expdayyr,
        badtime, badmoney, goodate, changeflg, badrate, ch;

    float initbal;
```

```
    FILE *datefile, *accfile;
    char todate[9], readstr1[32], resp, linestr[81], mm[4], dd[4], yy[3];

    static char yestring[5]= "yYnN";
    static char menu[17]= "aAbBcCdDeEfFgGqQ";
    static char quitstr[5]= "aAqQ";

    long jump= (long) sizeof(char);


/* Open datefile, read today's date and day in year, close file: */

   if((datefile= fopen(argv[4], "r")) == NULL) {
     printf("\n Can't open file containing today's date.  Exiting program.\n");
     exit(1); }
   fgets(todate, 9, datefile);
   fscanf(datefile,"%d", &todaynum);
   rewind(datefile);
   fclose(datefile);
   if(todaynum==0){
     printf("\nThe current year does not match the fiscal year.");
     printf("\nYou may only view and/or print the file.");
     printf("\n\nEnter any key to continue: ");
     ch= getchar();
     exit(0); }

/* Open account file for reading and appending data: */

   if ((accfile= fopen(argv[1], "a+")) == NULL) {
     printf("\nCan't open account file.  Exiting program.\n");
     exit(1); }

/* Get the following from command line data (converting from string to int): */

   fiscyear= atoi(argv[5]); /* Directory fiscal year (not necessarily current)*/
   numline= atoi(argv[3]); /* Number of lines in the file */

   newline=1;

   if(numline < 20) {       /* True if this is a new file. */
     while (newline < 5)    /* Go to beginning of 5th line. */
       if((ch= getc(accfile)) == '\n') newline++;

     newlabor.currbal= 0.0;
     while((ch= getc(accfile)) != '\n')        /* Skip over the header label. */
     { if((ch > '/' && ch < ':') || ch=='.' || ch=='-'){
         ungetc(ch, accfile);
         if((fscanf(accfile, "%f", &newlabor.currbal) == 1) &&
           (newlabor.currbal < 0.0)) newlabor.currbal= 0.0;
       }
     }
     newline++;
     newlabor.hourate= 0.0;
     while((ch= getc(accfile)) != '\n')
```

```
    { if((ch > '/' && ch < ':') || ch=='.' || ch=='-'){
        ungetc(ch, accfile);
        if((fscanf(accfile, "%f", &newlabor.hourate) == 1) &&
            (newlabor.hourate < 0.0)) newlabor.hourate= 0.0;
      }
    }
    newline++;

    while(newline < 8)
      if((ch= getc(accfile)) == '\n') newline++;

/* Read in account expiration date: */

    fgets(readstr1,25,accfile);
    if((fscanf(accfile, "%d %c %d %c %d", &expmonth, &ch, &expday, &ch,
        &expyear) != 5)){
      expmonth= 0;
      expday = 0;
      expyear = 0; }
    expdayyr= dayinfy(expmonth, expday, expyear, fiscyear);
    goodate= dayfiltr(expmonth, expday, expyear, fiscyear);

    remday= expdayyr - todaynum;
    newlabor.daysleft= remday;
    initbal= newlabor.currbal;
    newlabor.hours= 0.0;
    newlabor.transtot= 0.0;
    newlabor.tothour= 0.0;
    if(initbal>0.0 && remday>=0 && (newlabor.hourate>=MINWAGE ||
      (newlabor.hourate>0.99 && newlabor.hourate<1.01)) && goodate==TRUE)
    { badrate= FALSE;
      badmoney= FALSE;
      badtime= FALSE;
    }
    else {
      if(initbal <= 0.0)
        badmoney= TRUE;
      else
        badmoney= FALSE;

      if(remday < 0  || goodate==FALSE)
        badtime= TRUE;
      else
        badtime= FALSE;

      if(newlabor.hourate < MINWAGE && (newlabor.hourate <= 0.99 ||
         newlabor.hourate >= 1.01))
        badrate= TRUE;
      else
        badrate= FALSE;
    }

  }
 else{  /* The file is not new. */
```

```
  /* The file should be set to the start of line 20.  The initial balance*/
  /* needs to be read from the first line of data: */

    while(newline < 20)
      if((ch= getc(accfile)) == '\n') newline++;

    fseek(accfile, 80L*jump, 1);
    fscanf(accfile, "%f", &initbal);

     /* Go to the last line of the file and read in the last transaction: */

    fseek(accfile, -113L*jump, 2);
    fgets(newlabor.flag, 8, accfile);
    fscanf(accfile,"%c%d%c%d%c%d%c%c",&ch, &month, &ch, &day, &ch, &year, &ch,
           &ch);
    fgets(newlabor.transact,30,accfile);
    fscanf(accfile, "%c %f %c %f %c %f %c %f %c %d %c %f",
           &ch, &newlabor.hourate, &ch, &newlabor.hours, &ch,
           &newlabor.transtot, &ch, &newlabor.currbal,
           &ch, &newlabor.daysleft, &ch, &newlabor.tothour);

    /* Re-create the last date entered: */
    sprintf(mm, "%d%c", month, '/');
    sprintf(dd, "%d%c", day, '/');
    sprintf(yy, "%d", year);
    strcpy(newlabor.date, mm);
    strcat(newlabor.date, dd);
    strcat(newlabor.date, yy);

    printf("\n Initial Balance= $%.2f\n Flag= %s\n Date of Last Entry= %s\n",
             initbal, newlabor.flag, newlabor.date);
    printf(" Prev. Transaction= %s\n Rate/Hour= $%.2f\n Prev. Hours= %.2f\n",
             newlabor.transact, newlabor.hourate, newlabor.hours);
    printf(" Prev. Transaction Total= $%.2f\n Current Balance= $%.2f\n",
             newlabor.transtot, newlabor.currbal);
    printf(" Prev. Days Left= %d\n Cum. Hours= %.2f\n", newlabor.daysleft,
             newlabor.tothour);
   printf("\nEnter any character to continue:");
   ch= getchar();

/* In order to calculate the days remaining until the end of the account,
   calculate the day in fiscal year of the last entry, and refigure today's
   time remaining.  (This is required in case the expiration date has already
   been altered from the original.)                                    */

     oldday= dayinfy(month, day, year, fiscyear);
     oldexday= oldday + newlabor.daysleft;
     remday= oldexday - todaynum;

     if(remday < 0)
       badtime= TRUE;
     else
       badtime= FALSE;

     if(newlabor.currbal <= 0.0)
```

```
      badmoney= TRUE;
    else
      badmoney= FALSE;

    if(newlabor.hourate < MINWAGE && (newlabor.hourate <= 0.99 ||
       newlabor.hourate >= 1.01))
      badrate=TRUE;
    else
      badrate= FALSE;
  }

/* Position computer file pointer to EOF: *\

  fseek(accfile, 0L, 2);

/* Check whether time, money, or labor rate need to be changed:    */

  if(badmoney == TRUE){
    printf("\n\nThis account is bankrupt (i.e. current balance = $0.00).\n");
    printf("Enter A to add money to the account or Q to quit this part of\n");
    printf("the program: ");
    resp= checkans(quitstr);
    if(resp == 'q' || resp == 'Q'){
      rewind(accfile);
      fclose(accfile);
      exit(0); }
    else {
      initbal= addmoney(&newlabor, accfile, todate, remday, initbal, &numline,
                        argv[2]);
      if(newlabor.currbal <= 0.0){
        rewind(accfile);
        fclose(accfile);
        exit(0); }
    }
  }
  if(remday==0){
    printf("\nToday is the last day this account can be worked on.");
    printf("\nDo you want to add more time?  (Enter y or n): ");
    resp=checkans(yestring);
    if(resp=='Y' || resp=='y') badtime= TRUE; }
  else if(remday < 0){
    printf("\n\nThe expiration date for the account balance has been ");
    printf("exceeded.");
    printf("\nEnter A to add time or Q to quit this part of the program: ");
    resp= checkans(quitstr);
    if(resp=='q' || resp=='Q'){
      rewind(accfile);
      fclose(accfile);
      exit(0); }
  }
  if(badtime == TRUE){
     remday= changedt(&newlabor, accfile, todate, remday, initbal, fiscyear,
                      todaynum, &numline, argv[2]);
     if(remday < 0){
       rewind(accfile);
```

```
    fclose(accfile);
    exit(0); }
}
if(badrate==TRUE){
  printf("\n\nThe rate per unit hour of this account is currently set");
  printf("\nto below the minimum wage of $%.2f.  This program will there-",
        MINWAGE);
  printf("\nfore ask for labor charges in terms of dollars, not hours.\n");
  printf("To change this, see the change menu (next screen).");
  printf("\n\nEnter any key to continue: ");
  gets(linestr);
}
changeflg= TRUE;
while (changeflg == TRUE)
{ system("clear");    /* Clear screen */
  printf("\nAccount No. %s has a balance of $%.2f and expires",
          argv[2], newlabor.currbal);
  if(remday > 1)
    printf(" in %d days.", remday);
  else if(remday == 1)
    printf(" in 1 day.");
  else
    printf(" today.");

  printf("\n\nHere is the change menu:\n\n    A) Add money to account.\n");
  printf("    B) Delete money from account.\n    C) Change expiration ");
  printf("date.\n    D) Change cost/hour.\n    E) Enter labor charges.\n");
  printf("    F) Delete labor charges.\n    G) Add miscellaneous charges.");
  printf("\n\nEnter a letter, or Q to quit adding labor charges: ");
  resp= checkans(menu);
  if(resp == 'A' || resp=='a')
    initbal= addmoney(&newlabor, accfile, todate, remday, initbal, &numline,
                      argv[2]);
  else if(resp=='B' || resp=='b')
     delmoney(&newlabor, accfile, todate, remday, initbal, &numline,
                argv[2]);
  else if(resp=='C' || resp=='c')
      remday= changedt(&newlabor, accfile, todate, remday, initbal, fiscyear,
                todaynum, &numline, argv[2]);
  else if(resp=='D' || resp=='d')
      costhour(&newlabor, accfile, todate, remday, initbal, &numline,
                 argv[2]);
  else if(resp=='E' || resp=='e')
      laborchg(&newlabor, accfile, todate, remday, initbal, TRUE, &numline,
                 argv[2]);
  else if(resp=='F' || resp=='f')
      laborchg(&newlabor, accfile, todate, remday, initbal, FALSE, &numline,
                argv[2]);
  else if(resp=='G' || resp=='g')
      misc(&newlabor, accfile, todate, remday, initbal, &numline, argv[2]);

  if(resp != 'q' && resp != 'Q') {
     printf("\n\nAnother charge (of any kind--misc., add money, etc.)");
     printf("\nto account %s? (Enter y or n:) ", argv[2]);
     resp=checkans(yestring);
```

```
        if(resp=='n' || resp=='N')
          changeflg= FALSE;
      }
      else
        changeflg= FALSE;

    } /* end while changeflg */

    rewind(accfile);
    fclose(accfile);
    return 0;
  }

/* Addmoney adds lump sums to an account, and writes the value into the file. */

float addmoney(strucptr, fileptr, indate, remday, startbal, lines, accnum)
  struct labor *strucptr;
  FILE *fileptr;
  char *indate, *accnum;
  int remday, *lines;
  float startbal;
  {
    int getgoing, getinfo;
    float newmoney, newbal;
    char resp, string1[30], string2[19], linestr[81], flag[8];
    static char yestring[5]= "yYnN";

    printf("\nAdd money to account %s? \nEnter y or n (for yes or no): ",
          accnum);
    resp= checkans(yestring);
    if(resp=='n' || resp=='N')
      getgoing= FALSE;
    else
      getgoing= TRUE;

    while(getgoing == TRUE)
      { getinfo = TRUE;
        while(getinfo==TRUE)
         { sprintf(string1, "Add Money: ");
           printf("\nEnter reason to add money to account (18 char or less;");
           printf("\nfor ex.: ODN 47-92): ");
           strfiltr(string2, 19);
           strcat(string1, string2);

           printf("\nEnter amount of money to be added, in dollars.  DO NOT\n");
           printf("use letters, the dollar sign ($), or commas (i.e. $1,000\n");
           printf("or $10k are not acceptable): ");
           newmoney= -1.0;
           while(newmoney < 0.0)
           { while((scanf("%f", &newmoney)) == 0)
             { gets(linestr);
               printf("\nPlease re-enter amount of money to be added:");
             }
             if(newmoney < 0.0){
               printf("\nEntering a value less than zero is not acceptable.");
```

```
            printf("\nPlease re-enter amount: "); }
          }
          resp=getchar();
          printf("\nThe reason to add money is %s, and the amount to be",
                  string2);
          printf("\nadded is $%.2f.  Are these values correct?", newmoney);
          printf("\nEnter y or n: ");
          resp= checkans(yestring);
          if(resp=='y' || resp=='Y') {
            getinfo= FALSE;
            if(newmoney < 0.01)   /* No change for zero entry.  Quick exit */
              getgoing= FALSE;    /* from subroutine.  */
          }
        } /* end while getinfo */

        if(getgoing == TRUE) {
          if(startbal < 0.01){
            startbal= newmoney;
            newbal= newmoney;
          }
          else
            newbal= strucptr->currbal + newmoney;

          newmoney *= -1.0;  /* Charges to an account are entered as positive */
                             /* numbers.  Adding money to an account is there-*/
                             /* fore shown on the account file as a negative  */
                             /* transaction. */

          flagcalc(startbal, newbal, remday, flag);
          writelab(strucptr, fileptr, flag, indate, string1, strucptr->hourate,
                   0.0, newmoney, newbal, remday, strucptr->tothour);
          (*lines)++;                    /* Write form feed char. and append */
          if((*lines % 56) == 0) {       /* the headers to the account file. */
            putc('\f', fileptr);
            labform(fileptr);
            writelab(strucptr, fileptr, flag, indate, string1,
                     strucptr->hourate, 0.0, newmoney, newbal, remday,
                     strucptr->tothour);
            *lines += 7;
          }
          printf("\n\nThe current balance is $%.2f.", strucptr->currbal);
          printf("\nAdd more money to account %s? (Enter y or n:) ", accnum);
          resp= checkans(yestring);
          if(resp=='n' || resp=='N')
            getgoing= FALSE;
        }
      } /* end while getgoing */

      return startbal;
    }


/* Delmoney deletes lump sums from an account, and writes the new balance into
   the account records.    */
```

```
void delmoney(strucptr, fileptr, indate, remday, startbal, lines, accnum)
  struct labor *strucptr;
  FILE *fileptr;
  char *indate, *accnum;
  int remday, *lines;
  float startbal;
  {
    int getgoing, getinfo;
    float newmoney, newbal;
    char resp, string1[30], string2[18], linestr[81], flag[8];
    static char yestring[5]= "yYnN";

    printf("\nDelete money from account %s? \nEnter y or n ", accnum);
    printf("(for yes or no): ");
    resp= checkans(yestring);
    if(resp=='n' || resp=='N')
      getgoing= FALSE;
    else
      getgoing= TRUE;

    while(getgoing == TRUE)
      { getinfo = TRUE;
        while(getinfo==TRUE)
         { sprintf(string1, "Del. Money: ");
           printf("\nEnter reason to delete money (17 char or less;");
           printf("\nfor ex.: Ref. ODN 391-92): ");
           strfiltr(string2, 18);
           strcat(string1, string2);

           printf("\nEnter amount of money to be deleted, in dollars.  DO NOT\n");
           printf("use letters, the dollar sign ($), or commas (i.e. $1,000\n");
           printf("or $10k are not acceptable): ");
           newmoney= -1.0;
           while(newmoney < 0.0)
           { while((scanf("%f", &newmoney)) == 0)
             { gets(linestr);
               printf("\nPlease re-enter amount of money to be deleted:");
             }
             if(newmoney < 0.0){
               printf("\nEntering a value less than zero is not acceptable.");
               printf("\nPlease re-enter amount: "); }
           }
           resp=getchar();
           printf("\nThe reason to delete money is %s, and the amount to be",
                   string2);
           printf("\ndeleted is $%.2f.  Are these values correct?", newmoney);
           printf("\nEnter y or n: ");
           resp= checkans(yestring);
           if(resp=='y' || resp=='Y') {
             getinfo= FALSE;
             if(newmoney < 0.01)
               getgoing= FALSE;
           }
        } /* end while getinfo */
```

```
        if(getgoing == TRUE) {
          newbal= strucptr->currbal - newmoney;
          if(newbal< 0.0){
            printf("\nThis transaction will cause the account balance to be\n");
            printf("less than zero, and will not be allowed.\n");
          }
          else {
            if(newbal<0.01){
              printf("\nWarning: This transaction will zero (bankrupt) the");
              printf("\n         account. No further deletions are allowed.\n");
              getgoing= FALSE;
            }
           flagcalc(startbal, newbal, remday, flag);
           writelab(strucptr, fileptr, flag, indate, string1, strucptr->hourate,
                    0.0, newmoney, newbal, remday, strucptr->tothour);
           (*lines)++;
           if((*lines % 56) == 0) {
             putc('\f', fileptr);
             labform(fileptr);
             writelab(strucptr, fileptr, flag, indate, string1,
                      strucptr->hourate, 0.0, newmoney, newbal, remday,
                      strucptr->tothour);
             *lines += 7;
           }
          }
          if(getgoing == TRUE){
            printf("\nThe current balance is $%.2f.\n", strucptr->currbal);
            printf("\nRemove more money from account %s? (Enter y or n:) ",
                   accnum);
            resp= checkans(yestring);
            if(resp=='n' || resp=='N')
              getgoing= FALSE;
          }
        }
    } /* end while getgoing */

    return;
  }

/* Function changedate changes the expiration date, and determines the effect
   on the account.  It returns the new number of days remaining to the main
   program.   */

int changedt(strucptr, fileptr, todate, remday, initbal, fiscyr, thisday,
             lines, accnum)
  struct labor *strucptr;
  FILE *fileptr;
  char *todate, *accnum;
  int remday, fiscyr, thisday, *lines;
  float initbal;
  {
    int month, day, year, yr_day, getgoing, getinfo, goodate, moreinfo;
    char resp, string1[30], string2[19], linestr[81], flag[8];
    static char yestring[5]= "yYnN";
```

```
printf("\nChange the expiration date for account %s?", accnum);
printf("\nEnter y or n (for yes or no): ");
resp= checkans(yestring);
if(resp=='n' || resp=='N')
  getgoing= FALSE;
else
  getgoing= TRUE;

if(getgoing == TRUE) {
  getinfo= TRUE;
  while(getinfo==TRUE)
  { printf("\nEnter the number of the month (1-12) of the new date: ");
    while((scanf("%d", &month)) == 0)
    { gets(linestr);
      printf("\nPlease re-enter the month: ");
    }
    printf("\nEnter the day of the new expiration date: ");
    while((scanf("%d", &day)) == 0)
    { gets(linestr);
      printf("\nPlease re-enter the day: ");
    }
    year= -1;
    while(year< 0  || year >99)
    { printf("\nNow input the last two digits of the year: ");
      while((scanf("%d", &year))== 0)
      { gets(linestr);
        printf("\nPlease re-enter the year: ");
      }
    }
    resp=getchar();
    printf("\nYou have entered the following date: %d/%d/%d.", month,
          day, year);
    printf("\nIs this correct? (Enter y or n:) ");
    resp= checkans(yestring);
    if(resp=='y' || resp == 'Y') {
      goodate= dayfiltr(month, day, year,fiscyr);
      if(goodate==FALSE)
        printf("\nThe date you have entered is not acceptable.\n");
      else {
        yr_day= dayinfy(month, day, year, fiscyr);
        if((yr_day - thisday) >= 0){
          getinfo= FALSE;
          remday = yr_day - thisday;
        }
        else {
          printf("\nThe date you have entered is earlier than today.\n");
          printf("Please re-enter the date:\n");
        }
      }
    }
  } /* end while getinfo */

  sprintf(string1, "Exp. Date: ");
  moreinfo= TRUE;
  while(moreinfo == TRUE)
```

```
      { printf("\nPlease enter reason for changing the exp. date (18 char.\n");
        printf("or less).  (For ex.:  Ref Memo 91-92): ");
        strfiltr(string2, 19);
        printf("\nYou have entered: %s.  Is this OK? (Enter y or n:) ",string2);
        resp= checkans(yestring);
        if(resp == 'y' || resp=='Y')
          moreinfo= FALSE;
      }
      strcat(string1,string2);
      flagcalc(initbal, strucptr->currbal, remday, flag);
      writelab(strucptr, fileptr, flag, todate, string1, strucptr->hourate,
            0.0, 0.0, strucptr->currbal, remday, strucptr->tothour);
      (*lines)++;
      if((*lines % 56) == 0) {
        putc('\f', fileptr);
        labform(fileptr);
        writelab(strucptr, fileptr, flag, todate, string1, strucptr->hourate,
            0.0, 0.0, strucptr->currbal, remday, strucptr->tothour);
        *lines += 7;
      }

    } /* end if getgoing */

    return remday;
  }

/* Function costhour changes the cost per unit hour of the average labor
   charge.  */

void costhour(labptr, fileptr, thisdate, timeleft, origbal, lines,
              accnum)
  struct labor *labptr;
  FILE *fileptr;
  char *thisdate, *accnum;
  int timeleft, *lines;
  float origbal;
  {
    int getgoing, getinfo, hourOK, addit, count;
    float newrate, oldrate, oldcost, newcost, newbal;
    char flag[8], resp, linestr[81], reason[30], docnum[10], ans, quitans;
    static char yestring[5]= "yYnN";
    static char quitstr[5]= "eEqQ";
    long jump= (long) sizeof(char);

    printf("\n\nChange the avg. labor charge per unit hour for account %s?",
           accnum);
    printf("\nEnter y or n (for yes or no): ");
    resp= checkans(yestring);
    if(resp=='y' || resp=='Y')
     getgoing= TRUE;
    else
     getgoing= FALSE;

    while(getgoing == TRUE)
    { getinfo= TRUE;
```

```
while(getinfo == TRUE)
{ printf("\nEnter office no. of document authorizing change (9 char.\n");
  printf("or less): ");
  strfiltr(docnum, 10);
  printf("\nEnter the new average cost/hour, in dollars.  DO NOT USE\n");
  printf("commas, the dollar sign ($), or letters (i.e. don't enter\n");
  printf("1,000 or $10K): ");
  while((scanf("%f", &newrate)) == 0)
  { gets(linestr);
    printf("\nPlease re-enter cost per hour: ");
  }
  resp= getchar();
  printf("\n\nYou have entered the following:\n\n  Rate= $%.2f per hour,",
         newrate);
  printf("\n  Office Document No.= %s.", docnum);
  printf("\n\nAre these correct? (Enter y or n:) ");
  resp= checkans(yestring);
  if(resp=='y' || resp=='Y'){
    if(newrate < 0.0) {
      printf("\nYou have entered a value less than zero, which is not\n");
      printf("acceptable.  Enter another value (e) or quit this");
      printf(" function (q)? ");
      quitans= checkans(quitstr);
      if(quitans=='q' || quitans=='Q'){
        getinfo= FALSE;
        getgoing= FALSE; }
    }
    else if(newrate < MINWAGE && (newrate >= 1.01 || newrate <= 0.99)) {
     printf("\nYou have entered a value less than the minimum wage of\n");
     printf("$%.2f per hour.  All new labor charges will be allowed to\n",
            MINWAGE);
     printf("be entered in terms of dollars only, until the rate is ");
     printf("reset.\nIs this acceptable?  (Enter y or n:) ");
     ans= checkans(yestring);
     if(ans=='y' || ans=='Y'){
       addit= TRUE;
       getinfo= FALSE;
       hourOK= FALSE; }
     else {
       addit= FALSE;
       hourOK= FALSE;
       getinfo= FALSE; }
    }
    else {
      addit= TRUE;
      getinfo= FALSE;
      hourOK= TRUE;
    }
  } /*end if resp */
} /* end while getinfo */

if(getgoing == TRUE) {
  if(hourOK == TRUE)  {
    printf("\n\nRevising current balance to reflect new rate...\n\n");
```

```
  if(labptr->hourate >= MINWAGE  || (labptr->hourate < 1.01 &&
      labptr->hourate > 0.99))
    oldcost= labptr->tothour * labptr->hourate;
  else if(labptr->tothour < 0.01)
    oldcost=0.0;
  else {
        /* The last rate to be input caused the previous entries to be
           entered in terms of dollars, rather than numbers.  Since
           there were hours input at some place in the record (the
           total number of hours entered previously > 0), the program
           will search for the last suitable value of the rate.*/

    oldrate= 0.0;
    oldcost= 0.0;
    fseek(fileptr, 0L, 2);  /* Be sure you're at EOF. */
    fseek(fileptr, -178L*jump, 2);
    for(count= (*lines -1); count >= 20 && oldrate < 0.01; count--)
    { if((fscanf(fileptr, "%f", &oldrate))== 0){
        printf("\nThe old rate on line %d couldn't be read.\n", count);
        fseek(fileptr, -122L*jump, 1); }
      else{
        if(oldrate >= MINWAGE  || (oldrate < 1.01 && oldrate > 0.99))
          oldcost= labptr->tothour * oldrate; /*find old hour costs */
        else {
          oldrate= 0.0;
          fseek(fileptr, -122L*jump, 1); }
      }
    }
  }
  newcost= labptr->tothour * newrate;
  newbal= labptr->currbal + oldcost - newcost;

  if(newbal < 0.0) {
    addit= FALSE;
    printf("\nThis change in labor cost would reduce the account bal-");
    printf("\nance to less than zero, and is therefore not allowed.\n");
  }
  else if(newbal < 0.01) {
    printf("\nThis charge will zero (bankrupt) the account.  Is this");
    printf("\nacceptable?  (Enter y or n:) ");
    ans= checkans(yestring);
    if(ans=='y' || ans=='Y'){
      addit= TRUE;
      getgoing= FALSE; }
    else
      addit= FALSE;
  }
  else {
    addit= TRUE;
    getgoing= FALSE;
  }
}
else if(addit == TRUE) {
  newcost= 0.0;
  newbal= labptr->currbal;
```

```
        }                              /* end if hourOK */

        if(addit == TRUE) {
          if(newbal > labptr->currbal) newcost *= -1.0; /* Money added to */
          flagcalc(origbal, newbal, timeleft, flag);    /* acct. is negative */
          sprintf(reason, "Change Rate per ODN=");      /* by convention. */
          strcat(reason, docnum);
          writelab(labptr, fileptr, flag, thisdate, reason, newrate, 0.0,
                   newcost, newbal, timeleft, labptr->tothour);
          (*lines)++;
          if((*lines % 56) == 0) {
            putc('\f', fileptr);
            labform(fileptr);
            writelab(labptr, fileptr, flag, thisdate, reason, newrate, 0.0,
                     newcost, newbal, timeleft, labptr->tothour);
            *lines += 7;
          }
        }
        if(getgoing == TRUE){
          printf("\n\nEnter a new rate (e) or quit this function (q)? ");
          ans= checkans(quitstr);
          if(ans=='q' || ans=='Q')
          getgoing= FALSE;
        }
      } /* end if getgoing */
    } /* end while getgoing */

    return;
  }

/* Function laborchg adds or deletes labor charges from a file, depending on
   the value of addit (TRUE= add charge, FALSE= delete charge (add money to
   account)).   */

void laborchg(labptr, fileptr, thisdate, timeleft, origbal, addit,
              lines, accnum)
  struct labor *labptr;
  FILE *fileptr;
  char *thisdate, *accnum;
  int timeleft, addit, *lines;
  float origbal;
  {
    int getgoing, getinfo, hourok, balok;
    float newbal, newtot, newhour, newcost;
    char resp, linestr[81], reason[30], docnum[15], flag[8];
    static char yestring[5]= "yYnN";
    static char ref[7]= " ODN= ";
    static char addlab[10]= "Add Lab. ";
    static char dellab[10]= "Del. Lab.";

    if(addit == TRUE)
      printf("\nAdd labor charges to account %s?", accnum);
    else
      printf("\nDelete labor charges from account %s?", accnum);
```

```
printf("\nEnter y or n (for yes or no): ");
resp= checkans(yestring);
if(resp=='y' || resp=='Y')
  getgoing= TRUE;
else
  getgoing= FALSE;

while(getgoing == TRUE)
{ if(labptr->hourate < MINWAGE && (labptr->hourate <= 0.99 ||
       labptr->hourate >= 1.01)) {
    hourok= FALSE;
    printf("WARNING:  All labor charges will be assumed to be entered in");
    printf("\n          terms of dollars, because the current wage rate");
    printf("\n          is less than the minimum wage.  Do you want to go");
    printf("\n          back to the change menu?  (Enter y or n:) ");
    resp= checkans(yestring);
    if(resp=='y' || resp=='Y') {
      getgoing= FALSE;
      getinfo= FALSE; }
    else
      getinfo= TRUE;
  }
  else {
    hourok= TRUE;
    printf("\nThe current rate/hour is $%.2f.  This program will ask for",
            labptr->hourate);
    printf("\ndata in terms of hours.  Are these OK?  (Enter y or n:) ");
    resp= checkans(yestring);
    if(resp=='y' || resp=='Y')
      getinfo= TRUE;
    else {
      getinfo = FALSE;
      getgoing= FALSE; }
  }
  while(getinfo == TRUE)
  { printf("\nPlease enter the Office Document No. (14 char. or less:)\n");
    strfiltr(docnum, 15);
    if(hourok == TRUE) {
     newhour= -1.0;
     while(newhour < 0.0)
     {if(addit == TRUE)
       printf("\nEnter the number of hours to charge (no fewer than 0): ");
      else
       printf("\nEnter the number of hours to delete (no fewer than 0): ");
      if((scanf("%f", &newhour)) == 0)
       gets(linestr);
     }
    }
    else {
      newcost= -1.0;
      while(newcost < 0.0)
      { printf("\nPlease enter the amount in dollars (no less than 0.0).");
        printf("\nDO NOT USE commas, letters, or the the dollar sign\n");
        printf("(i.e. don't enter $10K, or 1,000, or 10K, etc.): ");
        if((scanf("%f", &newcost)) == 0)
```

```
        gets(linestr);
      }
    }
    resp= getchar();
    if(hourok == TRUE)
      printf("\nYou have entered:\n  Hours = %.2f,\n  Ref. Doc. = %s.",
              newhour, docnum);
    else
      printf("\nYou have entered:\n  Cost = $%.2f,\n  Ref. Doc. = %s.",
              newcost, docnum);
    printf("\n\nIs this OK?  (Enter y or n:) ");
    resp= checkans(yestring);
    if(resp=='y' || resp=='Y')
      getinfo= FALSE;

  } /* end while getinfo */

  if(getgoing == TRUE) {
    if(hourok == TRUE){
      if(addit == FALSE) newhour *= -1.0;  /* Deleted charges are shown */
      newcost= labptr->hourate * newhour;  /* as negative in account file,*/
      newtot= labptr->tothour += newhour;  /* since the addition of money*/
    }                                      /* is negative by the conven- */
                                           /* tion used in this program. */
    else {
      newhour= 0.0;
      newtot= labptr->tothour;
      if(addit == FALSE) newcost *= -1.0;
    }
    if(newcost <= -0.01 || newcost >= 0.01) { /* Add charges to account */
      newbal= labptr->currbal - newcost;      /* only if true.          */
      if(newbal > 0.01)
        balok= TRUE;
      else if(newbal > -0.001){
        printf("\nThis change will bankrupt the account.  Do you want to");
        printf("\nadd it to the account?  (Enter y or n:) ");
        resp= checkans(yestring);
        if(resp=='y' || resp=='Y'){
          balok = TRUE;
          printf("\nYou must add money to this account before further\n");
          printf("charges can be made.\n\n"); }
        else
          balok= FALSE;
        }
      else {
        printf("\nThis transaction will cause the balance to be less than\n");
        printf("$0.00, and is not allowed.\n\n");
        balok= FALSE;
      }
      if(balok == TRUE){
        if(addit == TRUE)
          strcpy(reason, addlab);
        else
          strcpy(reason, dellab);
```

```
            strcat(reason, ref);
            strcat(reason, docnum);
            flagcalc(origbal, newbal, timeleft, flag);
            writelab(labptr, fileptr, flag, thisdate, reason, labptr->hourate,
                    newhour, newcost, newbal, timeleft, newtot);
            (*lines)++;
            if((*lines % 56) == 0) {
              putc('\f', fileptr);
              labform(fileptr);
              writelab(labptr, fileptr, flag, thisdate, reason, labptr->hourate,
                      newhour, newcost, newbal, timeleft, newtot);
              *lines += 7;
            }
          }
        }
        else {
          printf("\nNo entry was made to the account since the total ");
          printf("cost = $0.00.\n");
        }

        printf("\n\nThe current balance is $%.2f.\n", labptr->currbal);

        if(addit == TRUE)
          printf("\nAdd more labor charges to account %s?", accnum);
        else
          printf("\nDelete more labor charges from account %s?", accnum);

        printf("\nEnter y or n: ");
        resp= checkans(yestring);
        if(resp=='n' || resp=='N')
          getgoing= FALSE;

      } /* end if getgoing */

    } /* end while getgoing */

    return;
  }

/* Function misc adds charges from miscellaneous sources: */

void misc(strucptr, fileptr, todate, remday, initbal, lines, accnum)
  struct labor *strucptr;
  FILE *fileptr;
  char *todate, *accnum;
  int remday, *lines;
  float initbal;
  {
   int getgoing, getinfo, goodbuy;
   float tempamt, newtot;
   char resp, str1[30], linestr[81], flag[8];
   static char yestring[5]= "yYnN";

   printf("\nAdd miscellaneous charges to account %s? \nEnter y or n ", accnum);
   printf("(for yes or no): ");
```

```
resp= checkans(yestring);
if(resp=='y' || resp=='Y')
  getgoing= TRUE;
else
  getgoing= FALSE;

while(getgoing == TRUE)
{ getinfo= TRUE;
  while(getinfo == TRUE)
  {  printf("\nEnter reason for charge (29 char. or less):\n");
     strfiltr(str1, 30);

     printf("\nEnter the total cost of the charge, in dollars.  DO NOT");
     printf("\nuse a comma, dollar sign ($), or letter (i.e. $10K or\n");
     printf("1,000 are not acceptable): ");
     while((scanf("%f", &tempamt)) == 0)
     { gets(linestr);
       printf("\nPlease re-enter cost: ");
     }
     resp= getchar();
     printf("\nHere are the values you've entered:\n\n");
     printf("  Reason for Charge = %s,\n", str1);
     printf("  Total Cost = $%.2f.", tempamt);
     printf("\n\nAre these OK?  (Enter y or n:) ");
     resp= checkans(yestring);
     if(resp=='y' || resp=='Y')
       getinfo= FALSE;
  }
  if(tempamt >= 0.01 || tempamt <= -0.01) {
                            /* Add charges to account if this is true. */

    newtot= strucptr->currbal - tempamt;

    if(newtot < 0.0) {
      printf("\nThis charge can not be applied to this account because\n");
      printf("the balance would then be less than zero.\n");
      goodbuy= FALSE;
    }
    else if(newtot < 0.01){
      printf("\nThis charge will bankrupt the account.  Do you want\n");
      printf("to add this cost to the account?  (Enter y or n:) ");
      resp= checkans(yestring);
      if(resp=='Y' || resp=='y') {
        goodbuy= TRUE;
        getgoing= FALSE;
      }
      else
        goodbuy= FALSE;
    }
    else
      goodbuy= TRUE;

    if(goodbuy == TRUE) {
      flagcalc(initbal, newtot, remday, flag);
      writelab(strucptr, fileptr, flag, todate, str1, strucptr->hourate, 0.0,
```

```
                tempamt, newtot, remday, strucptr->tothour);
        (*lines)++;
        if((*lines % 56) == 0) {
          putc('\f', fileptr);
          labform(fileptr);
          writelab(strucptr, fileptr, flag, todate, str1, strucptr->hourate,
                   0.0, tempamt, newtot, remday, strucptr->tothour);
          *lines += 7;
        }
      }
    }
    else {
      printf("\nNo entry was made to the account since the total ");
      printf("cost = $0.00.\n");
    }

    if(getgoing == TRUE) {
      printf("\n\nThe current account balance is $%.2f.\n",strucptr->currbal);
      printf("Add another misc. charge to account %s? (Enter y or n:) ",
              accnum);
      resp= checkans(yestring);
      if(resp=='n' || resp=='N')
        getgoing= FALSE;
    }
  } /* end while getgoing */

  return;
}

/* Function writelab takes arguments from either the main program or calling
   functions, revises the current structure (labor) variable, and writes the
   new data to file.            */

void writelab(labptr, fileptr, str1, str2, str3, rate, numhour, cost, newbal,
              timeleft, sumhour)
  struct labor *labptr;
  FILE *fileptr;
  char *str1, *str2, *str3;
  float rate, numhour, cost, newbal, sumhour;
  int timeleft;
  {
    static char space1[8]= "       ";
    static char space2[9]= "        ";
    static char space3[30]= "                              ";

    strcpy(labptr->flag, space1);   /* The space string is used to clear out */
    strcpy(labptr->flag, str1);     /* any letters that remain left over from */
    strcpy(labptr->date, space2);   /* previous data.  */
    strcpy(labptr->date, str2);
    strcpy(labptr->transact, space3);
    strcpy(labptr->transact, str3);
    labptr->hourate= rate;
    labptr->hours= numhour;
    labptr->transtot= cost;
    labptr->currbal= newbal;
```

```
    labptr->daysleft= timeleft;
    labptr->tothour= sumhour;

    fseek(fileptr, 0L, 2);  /* Go to EOF */
    fprintf(fileptr, LABFMT, labptr->flag, labptr->date, labptr->transact,
            rate, numhour, cost, newbal, timeleft, sumhour);

    return;
}
```

```
/* masslabr.c is the program to add general labor charges to accounts, in
   groups of up to 25 accounts at one time.  The charges will be entered in
   terms of hours or of dollars, depending upon the current average labor rate
   per unit hour.  This program is called from a UNIX program called gen.labor
   and is not a part of edit.data, as there is only limited interactive control
   over input data.  Gen.labor is a part of the startup. acc menu (Choice D).

   This program takes the following command line arguements: argv[1] = name of
   file containing today's date, argv[2] = directory fiscal year, argv[3]=
   the pathname of the starting directory.  */

#include <stdio.h>
#include <string.h>
#include "cfunct.h"

#define TRUE 1
#define FALSE 0
#define LABFMT "%-7s|%-9s|%-29s|%9.2f|%9.2f|%10.2f|%11.2f|   %3d   |%10.2f|\n"
#define MINWAGE 3.75
#define LIMIT 25    /* The # of accounts that this prog. handles in 1 pass. */
#define DIRLIMIT 8  /* The current # of labor directories +2, just in case. */
#define PATHLEN 25  /* The length of the account file path descriptor. */
#define ACCLEN 6    /* The length of the string holding the account number. */

void exit();

int checkans(); /* These functions are defined in header file cfunct.h. */
int dayinfy();
int dayfiltr();
void flagcalc();
void initstr();
void labform();
void strfiltr();

void laborchg();  /* These functions are defined below. */
void writelab();

struct labor {
  char flag[8];
  char date[9];
  char transact[30];
  float hourate;
  float hours;
  float transtot;
  float currbal;
  int daysleft;
  float tothour; };

struct labtime{
   char name[ACCLEN];
   float charges;
   char path[PATHLEN];};

int main(argc,argv)
  int argc;
```

```
char **argv;
{
  struct labor newlabor;

  struct labtime getacct[LIMIT];

  int todaynum, fiscyear, numline, newline, month, day, year,
      oldday, oldexday, remday, expmonth, expday, expyear, expdayyr,
      badrate,badtime, badmoney, goodate, i, badacct, startup, numpath,
      linenum, numacct, addname, addacct, badnum, accopen, addhours, ch;

  float initbal, time, tot;

  long jump= (long) sizeof(char);

  FILE *datefile, *accfile, *errfile, *findfile;

  char todate[9], readstr1[32], resp, linestr[81],
       account[ACCLEN], pathway[DIRLIMIT][PATHLEN],
       tempath[PATHLEN], command[100];

  char dash= '-';
  static char yestring[5]= "yYnN";
  static char find1[11]= "temp.find1";
  static char find2[11]= "temp.find2";

  /* Open datefile, read today's date and day in year, close file: */

  if((datefile= fopen(argv[1], "r")) == NULL) {
    printf("\n Can't open file containing today's date.  Exiting program.\n");
    exit(1); }
  fgets(todate, 9, datefile);
  fscanf(datefile,"%d", &todaynum);
  rewind(datefile);
  fclose(datefile);
  if(todaynum == 0) {
     printf("\nThe current year is not the same as the fiscal year.  You ");
     printf("\nmay only view and/or print the data in the files, using");
     printf("\nchoices C, or E+F, or G from the startup.acc menu.");
     printf("\n\nEnter any key to continue: ");
     ch= getchar();
     exit(0);
   }
  fiscyear= atoi(argv[2]);

  startup= TRUE;
  while(startup == TRUE)
  { /* Initialize structural variables: */
    for(i=0; i < LIMIT; i++){
      getacct[i].name[0]= dash;
      getacct[i].charges= 0.0; }

    /* Enter account name & test if it exists in 1 or more directories: */
    numacct= 0;       /* Number of accounts added to list. */
    addacct= TRUE;
```

```
while(addacct == TRUE && numacct < LIMIT)
{ badacct= TRUE;
  while(badacct == TRUE)
  { printf("\nEnter account no. (or XO no.) (up to %d characters): ",
             ACCLEN-1);
    strfiltr(account, ACCLEN);
    printf("\nYou have entered account no. %s.  Is this OK?  (Enter y ",
             account);
    printf("or n:) ");
    resp= checkans(yestring);
    if(resp=='y' || resp=='Y') badacct=FALSE;
  }
  if((findfile= fopen(find1, "w+")) == NULL) {
   printf("\nCan't open file for storing account name. Exiting program.");
   exit(1); }

  fprintf(findfile, "%s\n", account);
  rewind(findfile);
  fclose(findfile);

  sprintf(command, "%s/findit", argv[3]);
  system(command);
  initstr(command);
  if((findfile= fopen(find2, "r")) == NULL) {
    printf("\nCan't open file containing account pathway.");
    printf("  Exiting program.");
    exit(1); }

  fscanf(findfile,"%d%c", &numpath,&ch);  /* # of dir. file exists in. */
  if(numpath == 0) {
    printf("\nThis account either does not exist, or it does not exist");
    printf("\nas a labor account.\n");
    addname= FALSE; }
  else if(numpath == 1) {
    addname= TRUE;
    fscanf(findfile, "%s", tempath);}
  else {
    system("clear");
    printf("This account was found in multiple directories:\n");
    for(i=0; i< numpath && i< DIRLIMIT; i++)
    { fscanf(findfile, "%s%c", pathway[i], &ch);
      printf("\n  %d)  %s", i+1, pathway[i]);
    }
    printf("\n\nEnter the line no. of the correct directory listing: ");
    badnum= TRUE;
    while(badnum == TRUE)
    { if((scanf("%d", &linenum)) == 0) {
        gets(linestr);
        printf("\nPlease re-enter line number: "); }
      else if(linenum < 1 || linenum > i)
        printf("\nPlease re-enter line number: ");
      else
        badnum= FALSE;
    }
    ch=getchar();
```

```
      strcpy(tempath, pathway[linenum-1]);
      addname= TRUE;
    }
    rewind(findfile);
    fclose(findfile);

    if(addname == TRUE) {
/* Determine if account has been used before, if adding the name would
   cause the limit on accounts to be exceeded, or add it to the list: */

      for(i=0; i< LIMIT; i++)
      { if(getacct[i].name[0] == dash) { /*The element has not been used.*/
          strcpy(getacct[i].name, account);
          strcpy(getacct[i].path, tempath);
          numacct++;
          break; }
        else if((strcmp(getacct[i].path, tempath)) == 0)
          break;
      }
      initstr(tempath);
      if(i < LIMIT) {
       addhours= TRUE;
       printf("\n\nThe labor charges can be entered in terms of hours, or");
       printf("\ndollars (the program will automatically put in the ");
       printf("correct\nvalue, based on the rate/hour in the account).\n\n");
        while(addhours == TRUE)
        { tot= 0.0;
          printf("\nEnter the charges for %s.  If entering more than 1\n",
                  getacct[i].name);
          printf("number per line, leave a space or comma between numbers.");
          printf("\n(Do not use commas in the middle of a number.)  If\n");
          printf("entering charges in terms of dollars, you do not have\n");
          printf("to enter the dollar sign ($).\n");
          printf("\nUse <crtl><h> to erase input errors:\n");
          while ((ch=getchar()) != '\n')
          { if((ch > '/' && ch < ':') || ch=='.' || ch=='-'){
              ungetc(ch, stdin);
              if((scanf("%f", &time)) == 1)
                tot += time;
              else if((ch=getchar()) == '\n') break;}
          }
          printf("\nThe total entered for this line is %.2f.", tot);
          printf("\nIs this correct?  (Enter y or n:) ");
          resp= checkans(yestring);
          if(resp=='y' || resp=='Y') getacct[i].charges += tot;
          printf("\nAnother line of data for account %s?  (Enter y or n:) ",
                  getacct[i].name);
          resp= checkans(yestring);
          if(resp=='n' || resp=='N') addhours= FALSE;
        }
      }
      else {
        printf("\nThe number of accounts to be processed exceeds the limit");
        printf("\nof %d established by this program. Please stand by while",
                LIMIT);
```

```
        printf("\naccounts are processed....\n");
      }
    } /* end if addname==TRUE */

    if(numacct < LIMIT) {
      printf("\nAdd charges to another account?  (Enter y or n:) ");
      resp= checkans(yestring);
      if(resp=='n' || resp=='N') addacct= FALSE; }

  } /* End while(addacct & numacct) */

    /* Open individual accounts, determine if the account is new, and add
       hours or dollars, as is required per account (see below).  Send any
       error messages to the error file errfile.  */

if(numacct > 0) {
  if((errfile= fopen("temp.err", "a+")) == NULL) {
    printf("\nCan't open the error file.  Exiting program.");
    exit(1); }

  printf("\nProcessing accounts...\n");

  for(i=0; i < numacct; i++)        /* Open account if entry is non-zero. */
  { if(getacct[i].charges <= -0.01 || getacct[i].charges >= 0.01){
      accopen= TRUE;
      if((accfile= fopen(getacct[i].path, "a+")) == NULL) {
       fprintf(errfile,"Can't open account file for account %s\n",
                getacct[i].name);
       fprintf(errfile,"with %.2f additional hours or dollars charged.\n\n",
                getacct[i].charges);
       accopen= FALSE; }
    }
    else
      accopen= FALSE;

    if(accopen == TRUE) {
      printf("\nAccount %s .\n", getacct[i].path);

    /* Determine the number of lines in the current account file: */

      numline=0;
      while((ch=getc(accfile)) != EOF)
        if(ch == '\n') numline++;

      fseek(accfile, 0L, 0);
      newline=1;

      if(numline < 20) {              /* True if this is a new file. */
        while (newline < 5)           /* Go to beginning of 5th line. */
        { if((ch= getc(accfile)) == '\n')
          newline++;
        }
        fgets(readstr1,20,accfile);    /* Skip over the header label. */
        newlabor.currbal= 0.0;
        while((ch= getc(accfile)) != '\n')
```

```
{ if((ch > '/' && ch < ':') || ch == '.' || ch== '-') {
    ungetc(ch, accfile);
    if((fscanf(accfile, "%f", &newlabor.currbal) == 1) &&
      (newlabor.currbal < 0.0 )) newlabor.currbal= 0.0;
  }
}
newline++;
newlabor.hourate= 0.0;
while((ch= getc(accfile)) != '\n')  /* Skip to the rate per hour. */
{ if((ch > '/' && ch < ':') || ch=='.' || ch=='-') {
    ungetc(ch, accfile);
    if((fscanf(accfile, "%f", &newlabor.hourate) == 1) &&
      (newlabor.hourate < 0.0)) newlabor.hourate= 0.0;
  }
}
newline++;
while(newline < 8 )
{ if((ch= getc(accfile)) == '\n')
  newline++;
}

/* Read in account expiration date: */

fgets(readstr1,25,accfile);
if(fscanf(accfile, "%d %c %d %c %d", &expmonth, &ch, &expday,
     &ch, &expyear) != 5){
    expmonth= 0;
    expday= 0;
    expyear= 0; }
expdayyr= dayinfy(expmonth, expday, expyear, fiscyear);
goodate= dayfiltr(expmonth, expday, expyear, fiscyear);

fseek(accfile, 0L, 2);   /* Go to EOF. */

remday= expdayyr - todaynum;
newlabor.daysleft= remday;
initbal= newlabor.currbal;
newlabor.hours= 0.0;
newlabor.tothour= 0.0;
newlabor.transtot= 0.0;

if(newlabor.hourate < MINWAGE && (newlabor.hourate <= 0.99 ||
    newlabor.hourate >= 1.01))
 badrate= TRUE;
else
 badrate= FALSE;

if(initbal>0.0 && remday>=0 && goodate==TRUE)
{ badmoney= FALSE;
  badtime= FALSE;
}
else {
  if(initbal <= 0.0)
    badmoney= TRUE;
  else
```

```
            badmoney= FALSE;

          if(remday < 0  || goodate==FALSE)
            badtime= TRUE;
          else
            badtime= FALSE;
        }
      }
      else {  /* The file is not new. */

  /* The file should be set to the start of line 20.  The initial balance*/
  /* needs to be read from the first line of data: */

          while(newline < 20)
            if((ch= getc(accfile)) == '\n') newline++;

          fseek(accfile, 80L*jump, 1);
          fscanf(accfile, "%f", &initbal);

    /* Go to the last line of the file and read in the last transaction: */

          fseek(accfile, -113L*jump, 2);
          fgets(newlabor.flag, 8, accfile);
          fscanf(accfile,"%c%d%c%d%c%d%c%c",&ch, &month, &ch, &day, &ch,
               &year, &ch, &ch);
          fgets(newlabor.transact,30,accfile);
          fscanf(accfile, "%c %f %c %f %c %f %c %f %c %d %c %f",
               &ch, &newlabor.hourate, &ch, &newlabor.hours, &ch,
               &newlabor.transtot, &ch, &newlabor.currbal,
               &ch, &newlabor.daysleft, &ch, &newlabor.tothour);

/* In order to calculate the days remaining until the end of the account,
   calculate the day in fiscal year of the last entry, and refigure today's
   time remaining.  (This is required in case the expiration date has already
   been altered from the original.)                                    */

          oldday= dayinfy(month, day, year, fiscyear);
          oldexday= oldday + newlabor.daysleft;
          remday= oldexday - todaynum;

          if(newlabor.hourate < MINWAGE && (newlabor.hourate <= 0.99 ||
              newlabor.hourate >= 1.01)) /* Floating pt. numbers can't be */
           badrate= TRUE;                /* used for equality comparisons. */
          else
           badrate= FALSE;

          if(remday < 0)
            badtime= TRUE;
          else
            badtime= FALSE;

          if(newlabor.currbal <= 0.0)
            badmoney= TRUE;
          else
            badmoney= FALSE;
```

```
          }

/* Position computer file pointer to EOF: *\

          fseek(accfile, 0L, 2);

/* Check whether time, money, or rate prevent addition of hours to acct.:*/

          if(badmoney == TRUE){
            fprintf(errfile,"Account No. %s is bankrupt and", getacct[i].path);
            if(badrate == TRUE)
              fprintf(errfile,"\n$%.2f could not be added.\n\n",
                      getacct[i].charges);
            else
              fprintf(errfile,"\n%.2f hours could not be added.\n\n",
                      getacct[i].charges);
            rewind(accfile);
            fclose(accfile);
          }
          else if(badtime == TRUE){
            fprintf(errfile,"Account No. %s has exceeded its expiration",
                getacct[i].path);
            if(badrate == TRUE)
              fprintf(errfile,"\ndate and $%.2f could not be added.\n\n",
                getacct[i].charges);
            else
              fprintf(errfile,"\ndate and %.2f hours could not be added.\n\n",
                getacct[i].charges);
            rewind(accfile);
            fclose(accfile);
          }
          else {
            laborchg(&newlabor, accfile, todate, remday, initbal, numline,
                     &getacct[i], errfile);
            rewind(accfile);
            fclose(accfile);
          }
        }
      }
      rewind(errfile);
      fclose(errfile);
      if(addacct == TRUE) {
        printf("\nProcess another set of accounts? (Enter y or n:) ");
        resp= checkans(yestring);
        if(resp=='n' || resp=='N') startup= FALSE; }
      else
        startup= FALSE;
    }
    else
      startup= FALSE;  /* End if(numacct > 0) */

  } /* End while startup */

  return 0;
}
```

```
/* Function laborchg adds labor charges to a file. */

void laborchg(labptr, fileptr, thisdate, timeleft, origbal,
              lines, newchrg, errfile)
  struct labor *labptr;
  struct labtime *newchrg;
  FILE *fileptr, *errfile;
  char *thisdate;
  int timeleft, lines;
  float origbal;
  {
    int hourOK, balOK;
    float newbal, newtot, newhour, newcost;
    char flag[8];
    static char reason[30]= "Add general labor charges.   ";

    if(labptr->hourate < MINWAGE && (labptr->hourate <= 0.99 ||
          labptr->hourate >= 1.01)) {
      hourOK= FALSE;
      newhour= 0.0;
      newtot= labptr->tothour;
      newcost= newchrg->charges; }
    else {
      hourOK= TRUE;
      newhour= newchrg->charges;
      newcost= labptr->hourate * newhour;
      newtot= labptr->tothour += newhour;
    }
    newbal= labptr->currbal - newcost;
    if(newbal > 0.00)
      balOK= TRUE;
    else if(newbal > -0.01){
      balOK= TRUE;
      fprintf(errfile, "Account No. %s is now bankrupt.\n\n", newchrg->path);
    }
    else {
      if(hourOK == TRUE)
        fprintf(errfile,"Account No. %s could not be charged with %.2f hours\n",
                newchrg->path, newchrg->charges);
      else
        fprintf(errfile,"Account No. %s could not be charged with $%.2f\n",
                newchrg->path, newchrg->charges);

      fprintf(errfile,"because the charge would bankrupt the account.\n\n");
      balOK= FALSE;
    }
    if(balOK == TRUE){
      flagcalc(origbal, newbal, timeleft, flag);
      writelab(labptr, fileptr, flag, thisdate, reason, labptr->hourate,
               newhour, newcost, newbal, timeleft, newtot);
      lines++;
      if((lines % 56) == 0) {
        putc('\f', fileptr);
        labform(fileptr);
```

```
        writelab(labptr, fileptr, flag, thisdate, reason, labptr->hourate,
                 newhour, newcost, newbal, timeleft, newtot);
      }
    }
    return;
  }

/* Function writelab takes arguments from either the main program or calling
   functions, revises the current structure (labor) variable, and writes the
   new data to file.          */

void writelab(labptr, fileptr, str1, str2, str3, rate, numhour, cost, newbal,
              timeleft, sumhour)
  struct labor *labptr;
  FILE *fileptr;
  char *str1, *str2, *str3;
  float rate, numhour, cost, newbal, sumhour;
  int timeleft;
  {
    static char space1[8]= "       ";
    static char space2[9]= "        ";
    static char space3[30]= "                              ";

    strcpy(labptr->flag, space1);   /* The space string is used to clear out */
    strcpy(labptr->flag, str1);     /* any letters that remain left over from */
    strcpy(labptr->date, space2);   /* previous data.  */
    strcpy(labptr->date, str2);
    strcpy(labptr->transact, space3);
    strcpy(labptr->transact, str3);
    labptr->hourate= rate;
    labptr->hours= numhour;
    labptr->transtot= cost;
    labptr->currbal= newbal;
    labptr->daysleft= timeleft;
    labptr->tothour= sumhour;

    fseek(fileptr, 0L, 2);  /* Go to EOF */
    fprintf(fileptr, LABFMT, labptr->flag, labptr->date, labptr->transact,
            rate, numhour, cost, newbal, timeleft, sumhour);

    return;
  }
```

Appendix C: UNIX Language Programs Related to Program "startup.acc"

```
#checkdir checks the named directory and creates it if it does not exist:
#
while [ "$1" ]
do
  if [ ! -d $1 ]
  then
    mkdir $1
    chmod 770 $1
  fi
  shift
done
```

```
#date.get obtains today's date (month,day, year only) and stores the value in
#a file ($1) and sends it, along with the desired fiscal year ($2) to a
#C-language program (date_it), which calculates the day in the requested fiscal
#year and writes that value to $1.
#
if [ -f $1 ]
then
  rm $1
fi
date '+%D' > $1
date_it.x $1 $2
chmod 660 $1
```

```
#Program edit.data
#
true=0
false=1
#
path1='pwd'
pathtemp="${path1}/temp.d"
#
clear
echo "This program finds files and enables the user to add to, and/or view,"
echo "and/or print the charges in the individual account files.\n"
echo "Warning:  You must already have created the files and directories (file"
echo "          categories) using Choices A and B from the startup.acc menu."
echo "          If you haven't created the directories, or need to add one,"
echo "          enter n or N at the next question to exit this program, then"
echo "          choose A from the startup.acc menu."
echo "\n          If you need to create a file under an existing directory,"
echo "          exit this program as above and then choose B from the"
echo "          startup.acc menu.\n"
echo "A helpful hint:  DO NOT use the <backspace> or <rubout> keys to correct"
echo "input errors on the screen.  Use <crtl><h> instead. \n\n"
#
echo "Do you want to use this program?"
yes.or.no 1
resp='cat ${pathtemp}/temp.ans'
rm ${pathtemp}/temp.ans
case ${resp} in
  y|Y)
      startup=${true}
      ;;
  n|N)
      startup=${false}
esac
#
if [ ${startup} -eq ${true} ]
then
  baddir=${true}
  while [ ${baddir} -eq ${true} ]
  do
    fy=100
    while [ ${fy} -lt 0 -o ${fy} -gt 99 ]
    do
      echo "\nEnter the last two digits of the fiscal year, or just hit <enter>"
      echo "or <return> if the year is 93: \c"
      read fy
      case ${fy} in
        "")
            fy=93
            ;;
      esac
    done
    if [ -d account.fy${fy}.d ]
    then
      baddir=${false}
    else
```

```
      echo "\nThere are no files for fiscal year ${fy}.  Choose another year?"
      yes.or.no 2
      resp='cat ${pathtemp}/temp.ans'
      rm ${pathtemp}/temp.ans
      case ${resp} in
        n|N)
            baddir=${false}
            startup=${false}
            ;;
      esac
    fi
  done
#
fi
#
if [ ${startup} -eq ${true} ]
then
#
#Get the current date, and the day in the fiscal year; store the data in
#the file ${pathtemp)/temp.date:
#
 path2="${path1}/account.fy${fy}.d"
 pathlab="${path2}/labor.d"
 pathnl="${path2}/nonlabor.d"
 date.get ${pathtemp}/temp.date ${fy}
#
 cd account.fy${fy}.d
 getgoing=${true}
 while [ ${getgoing} -eq ${true} ]
 do
  badans=${true}
  while [ ${badans} -eq ${true} ]
  do
    echo "\nEnter the account (or XO) number (same as the file number in the"
    echo "computer), or enter q to quit: \c"
    read account
    case ${account} in
      "")
         account="None"
         accgood=${false}
         getgoing=${false}
         quit=${false}
         ;;
      Q|q)
         quit=${true}
         accgood=${false}
         getgoing=${false}
         badans=${false}
         ;;
      *)
         quit=${false}
         accgood=${true}
         getgoing=${true}
         ;;
    esac
```

```
#
    if [ ${quit} -eq ${false} ]
    then
      echo "\nThis is the input account (or XO) number: ${account}."
      if [ ${getgoing} -eq ${false} ]
      then
        echo "\n*************************************************************"
        echo "*  WARNING:  Not entering an account number will cause  *"
        echo "*            the program to terminate.                  *"
        echo "*************************************************************"
      fi
      echo "\nIs this number OK?"
      ${path1}/yes.or.no 3
      resp=`cat ${pathtemp}/temp.ans`
      rm ${pathtemp}/temp.ans
      case ${resp} in
        y|Y)
            badans=${false}
            ;;
      esac
    fi
  done
#
  while [ ${accgood} -eq ${true} ]
  do
    echo "\nIs this a labor (L) or non-labor account (N)? (Enter L or N:) \c"
    read acctype
    case ${acctype} in
      "")
         acctype="?"
         ;;
       *)
         acctype=`echo "${acctype}" | cut -f1 -d" "`
        ;;
    esac
    ${path1}/anscheck.x ${acctype} ${pathtemp}/temp.ans "l" "L" "n" "N"
    acctype=`cat ${pathtemp}/temp.ans`
    rm ${pathtemp}/temp.ans
    echo "\nAdd charges to account (A), or simply look at (and/or print) the"
    echo "account (L)?  (Enter A or L :) \c"
    read edtype
    case ${edtype} in
      "")
         edtype="?"
         ;;
       *)
         edtype=`echo "${edtype}" | cut -f1 -d" "`
         ;;
    esac
    ${path1}/anscheck.x ${edtype} ${pathtemp}/temp.ans "a" "A" "l" "L"
    edtype=`cat ${pathtemp}/temp.ans`
    rm ${pathtemp}/temp.ans
    case ${acctype} in
      l|L)
         cd ${pathlab}
```

```
        ;;
      n|N)
         cd ${pathnl}
         ;;
    esac
#
    sametype=${true}
    while [ ${sametype} -eq ${true} ]
    do
      clear
      filepath=`find . -name ${account} -print`
      case ${filepath} in
        "")
           echo "\nAccount #${account} either does not exist, or"
           case ${acctype} in
             l|L)
                 echo "does not exist as a labor account."
                 ;;
             n|N)
                 echo "does not exist as a non-labor account."
                 ;;
           esac
           accgood=${false}
           sametype=${false}
           ;;
#
       *)
#Check to see if more than one file exists with the input name given (if the
#variable filepath has more than one line):
#
        numline=`echo "${filepath}" | wc -l | cut -c7`
        if [ ${numline} -eq 1 ]
        then
          infname=${filepath}
        else
          clear
          keepon=${true}
          while [ ${keepon} -eq ${true} ]
          do
           echo "\n\nAccount #${account} was found under these directories:\n"
           echo "${filepath}" | cut -f2 -d"/"
           echo "\nPlease enter the correct directory name, copying only enough"
           echo "characters from the list above (using upper and lower cases as"
           echo "shown) to completely distinguish the directory from its \
neighbors:"
            read dirname
            dirname="${dirname}*"
            if [ -d ${dirname} ]
            then
              infname="${dirname}/${account}"
              keepon=${false}
            else
              echo "\nThe directory you have entered does not exist."
              echo "Enter another account (a), try another directory (d),"
              echo "or quit this program (q)?  (Enter a, d, or q:) \c"
```

```
                read resp
                case ${resp} in
                  "")
                     resp="?"
                     ;;
                  *)
                     resp=`echo "${resp}" | cut -f1 -d" "`
                     ;;
                esac
                ${path1}/anscheck.x ${resp} ${pathtemp}/temp.ans "a" "A" "d" "D" \
"q" "Q"
                resp=`cat ${pathtemp}/temp.ans`
                rm ${pathtemp}/temp.ans
                case ${resp} in
                  a|A)
                      accgood=${false}
                      sametype=${false}
                      keepon=${false}
                      ;;
                  q|Q)
                      getgoing=${false}
                      accgood=${false}
                      sametype=${false}
                      keepon=${false}
                      ;;
                esac
#
             fi   #End if dirname.
#
           done   #End while keepon.
#
         fi  #End if numline.
#
         ;;
       esac  #End case filepath.
#
       if [ ${sametype} -eq ${true} ]
       then
#
#Modify and/or look at the account files:
#
         case ${edtype} in
          a|A)
           echo "\nLook at account ${account} before making changes to it?"
           ${path1}/yes.or.no 4
           lookat=`cat ${pathtemp}/temp.ans`
           rm ${pathtemp}/temp.ans
           case ${lookat} in
             Y)
               pg ${infname}
               ;;
           esac
           clear
#
#Call the C programs that modify individual accounts:
#
```

```
        linenum='wc -l ${infname} | cut -c1-7'
        case ${acctype} in
          l|L)
             ${path1}/indlabor.x ${infname} ${account} ${linenum} \
${pathtemp}/temp.date ${fy}
              ;;
          n|N)
             ${path1}/chgnonlb.x ${infname} ${account} ${linenum} \
${pathtemp}/temp.date ${fy}
              ;;
        esac
#
        ;;
       esac
       clear
       echo "View data in file (v), print data (p), both view and print (b),"
       echo "or skip view/print of account (s)?  (Enter v, p, b, or s:) \c"
       read vuit
       case ${vuit} in
        "")
           vuit="?"
           ;;
         *)
           vuit='echo "${vuit}" | cut -f1 -d" "'
           ;;
       esac
       ${path1}/anscheck.x ${vuit} ${pathtemp}/temp.ans "v" "V" "p" "P" "b" \
"B" "s" "S"
       vuit='cat ${pathtemp}/temp.ans'
       rm ${pathtemp}/temp.ans
       case ${vuit} in
        v|V)
            pg ${infname}
            ;;
        p|P)
            wpr ${infname}
            ;;
        b|B)
            pg ${infname}
            wpr ${infname}
            ;;
       esac
        echo "\n\nAnother account?"
        ${path1}/yes.or.no 5
        resp='cat ${pathtemp}/temp.ans'
        rm ${pathtemp}/temp.ans
        case ${resp} in
          Y)
            badans=${true}
            while [ ${badans} -eq ${true} ]
            do
             echo "\nEnter the account (or XO) number (same as the file"
             echo "number in the computer), or enter q to quit: \c"
             read account
```

```
      case ${account} in
        "")
            account="None"
            accgood=${false}
            getgoing=${false}
            sametype=${false}
            quit=${false}
            ;;
        Q|q)
            quit=${true}
            sametype=${false}
            accgood=${false}
            getgoing=${false}
            badans=${false}
            ;;
         *)
            quit=${false}
            accgood=${true}
            getgoing=${true}
            ;;
      esac
#
      if [ ${quit} -eq ${false} ]
      then
        echo "\nThis is the input account number: ${account}."
        if [ ${getgoing} -eq ${false} ]
        then
          echo "\n************************************************************"
          echo "*  WARNING:  Entering no account number will cause the  *"
          echo "*            program to terminate.                      *"
          echo "************************************************************"
        fi
        echo "\nIs this number OK?"
        ${path1}/yes.or.no 6
        ans='cat ${pathtemp}/temp.ans'
        rm ${pathtemp}/temp.ans
        case ${ans} in
          Y)
            badans=${false}
            ;;
        esac
    fi
  done
#
  if [ ${getgoing} -eq ${true} ]
  then
    case ${acctype} in
      l|L)
          echo "\nIs this another labor account?"
          ;;
      n|N)
          echo "\nIs this another non-labor account?"
          ;;
    esac
```

```
              ${path1}/yes.or.no 7
              ans='cat ${pathtemp}/temp.ans'
              rm ${pathtemp}/temp.ans
              case ${ans} in
                Y)
                   sametype=${true}
                   echo "\nAdd charges to account (A), or simply look at (and/or \
print)"
                   echo "the account (L)?  (Enter A or L:) \c"
                   read edtype
                   case ${edtype} in
                     "")
                        edtype="?"
                        ;;
                      *)
                        edtype='echo "${edtype}" | cut -f1 -d" "'
                        ;;
                    esac
                    ${path1}/anscheck.x ${edtype} ${pathtemp}/temp.ans "a" "A" \
"l" "L"
                    edtype='cat ${pathtemp}/temp.ans'
                    rm ${pathtemp}/temp.ans
                    ;;
                N)
                   sametype=${false}
                   ;;
              esac
#
            fi
           ;;
#
         N)
           getgoing=${false}
           accgood=${false}
           sametype=${false}
           ;;
#
       esac  #End case $resp in...
#
     fi  #End if sametype.
#
   done  #End while sametype.
#
   cd ..  #Go to path1/account.fyxx.d.
#
  done  #End while accgood.
#
 done #End while getgoing.
 cd ${path1}
#
fi  #End if startup= true.
#
if [ -f ${pathtemp}/temp.date ]
then
  rm ${pathtemp}/temp.date
fi
```

```
#Program findit gets the listing of an account file path, and determines the
#number of possible listings of that account.
#
#For use with the gen.labor program and is called by masslabr.x (executable
#form of the C Language program masslabr.c).
#
account=`cat temp.find1`
filepath=`find . -name ${account} -print`
case ${filepath} in
  "")
     echo "0" > temp.find2
     ;;
   *)
     numline=`echo "${filepath}" | wc -l | cut -c1-7`
     echo "${numline}" > temp.find2
     echo "${filepath}" >> temp.find2
     ;;
esac
```

```
#Program gen.labor
#
path1='pwd'
pathtemp="${path1}/temp.d"
true=0
false=1
#
clear
echo "This program inputs general labor charges to accounts.  It assumes that"
echo "the directories (file categories) and accounts have been set up already,"
echo "using Choices A and B on the startup.acc menu.\n"
echo "If this is not the case, answer N to the question below to exit this pro-"
echo "gram, and choose from the startup.acc menu to input the required infor-"
echo "mation.\n"
echo "A helpful hint:  DO NOT USE the <rubout> or <backspace> keys to erase"
echo "errors that you make when inputting data.  Use <crtl><h> instead.\n\n"
#
echo "Do you want add general labor charges?"
yes.or.no 1
resp='cat ${pathtemp}/temp.ans'
rm ${pathtemp}/temp.ans
case ${resp} in
  Y)
    startup=${true}
    ;;
  N)
    startup=${false}
esac
if [ ${startup} -eq ${true} ]
then
  baddir=${true}
  while [ ${baddir} -eq ${true} ]
  do
    fy=100
    while [ ${fy} -lt 0 -o ${fy} -gt 99 ]
    do
      echo "\nEnter the last two digits of the fiscal year, or just hit <enter>"
      echo "or <return> if the year is 93: \c"
      read fy
      case ${fy} in
        "")
            fy=93
            ;;
      esac
    done
    if [ -d account.fy${fy}.d ]
    then
      baddir=${false}
    else
      echo "\nThere are no files for fiscal year ${fy}.  Select another year?"
      yes.or.no 2
      resp='cat ${pathtemp}/temp.ans'
      rm ${pathtemp}/temp.ans
      case ${resp} in
        N)
```

```
          baddir=${false}
          startup=${false}
          ;;
      esac
    fi
  done
#
 if [ ${startup} -eq ${true} ]
 then
#
  date.get ${pathtemp}/temp.date ${fy} #Get today's date and day in fiscal year.
  path2="account.fy${fy}.d/labor.d"
  cd ${path2}
  clear
  ${path1}/masslabr.x ${pathtemp}/temp.date ${fy} ${path1}
  if [ -s temp.err ]
  then
  echo "This program has generated an error file.  Do you want to view it (v),"
    echo "print it (p), both (b), or neither (n)?  (Enter v, p, b, or n:) \c"
    read resp
    case ${resp} in
      "")
         resp="?"
         ;;
       *)
         resp=`echo "${resp}" | cut -f1 -d" "`
         ;;
    esac
    ${path1}/anscheck.x ${resp} ${pathtemp}/temp.ans "v" "V" "p" "P" "b" "B" \
"n" "N"
    resp=`cat ${pathtemp}/temp.ans`
    rm ${pathtemp}/temp.ans
    case ${resp} in
      v|V)
          pg temp.err
          ;;
      p|P)
          wpr temp.err
          ;;
      b|B)
          pg temp.err
          wpr temp.err
          ;;
    esac
    echo "Copy the error file to a permanent file in your directory?"
    echo "(If not, then the error file will be erased.)\n"
    ${path1}/yes.or.no 3
    resp=`cat ${pathtemp}/temp.ans`
    rm ${pathtemp}/temp.ans
    case ${resp} in
      Y)
        goodfile=${false}
        while [ ${goodfile} -eq ${false} ]
        do
```

```
          echo "\nEnter file name for error file:"
          read fname
          case ${fname} in
            "")
               echo "\nYou must enter a file name."
               ;;
             *)
               if [ -f $1/${fname} ]
               then
                 echo "\nThe file name you have selected already exists."
                 echo "Choose another file name?"
                 yes.or.no 4
                 ans=`cat ${pathtemp}/temp.ans`
                 rm ${pathtemp}/temp.ans
                 case ${ans} in
                   N)
                      goodfile=${true}
                      ;;
                 esac
               else
                 cat temp.err >> $1/${fname}
                 goodfile=${true}
               fi
               ;;
          esac
        done
        ;;
    esac
    rm temp.err
#
  else
    if [ -f temp.err ]
    then
      rm temp.err
    fi
  fi
#
  if [ -f temp.find? ]
  then
    rm temp.find?
  fi
  cd ${path1}
#
  if [ -f ${pathtemp}/temp.date ]
  then
    rm ${pathtemp}/temp.date
  fi
#
 fi #End if startup= true.
#
fi  #End if startup= true.
```

```
#Program keyword
#
clear
path1='pwd'
pathtemp="${path1}/temp.d"
true=1
false=0
#
echo "This is the program which will search through all files to match a"
echo "keyword (e.g. training, travel, purchases, etc.), and will collect the"
echo "data, to be printed and/or saved to a file, as you wish."
echo "\nDo you want to use this program?"
yes.or.no 1
resp='cat ${pathtemp}/temp.ans'
rm ${pathtemp}/temp.ans
case ${resp} in
  Y)
     getgoing=${true}
     ;;
  N)
     getgoing=${false}
     ;;
esac
#
if [ ${getgoing} -eq ${true} ]
then
  baddir=${true}
  while [ ${baddir} -eq ${true} ]
  do
    fy=100
    while [ ${fy} -lt 0 -o ${fy} -gt 99 ]
    do
      echo "\nEnter the last two digits of the fiscal year to be searched,"
      echo "or just press <enter> or <return> if the fiscal year is 93: \c"
      read fy
      case ${fy} in
       "")
          fy=93
          ;;
      esac
    done
    if [ -d account.fy${fy}.d ]
    then
      baddir=${false}
    else
      echo "\nThere are no files for fiscal year ${fy}.  Choose another year?"
      yes.or.no 2
      resp='cat ${pathtemp}/temp.ans'
      rm ${pathtemp}/temp.ans
      case ${resp} in
        N)
          baddir=${false}
          getgoing=${false}
          ;;
      esac
```

```
    fi
  done
#
  while [ ${getgoing} -eq ${true} ]
  do
    clear
    echo "Here are the available pre-set keywords:\n"
    echo "  A) Award,\n  B) Purchases,\n  C) Training,\n  D) Travel."
    echo "\nEnter a letter (A-D) to select a pre-set keyword, or"
    echo "enter E to select your own keyword, or Q to quit this program: \c"
    read resp
    case ${resp} in
      "")
          resp="?"
          ;;
       *)
          resp=`echo "${resp}" | cut -f1 -d" "`
          ;;
    esac
    anscheck.x ${resp} ${pathtemp}/temp.ans "a" "A" "b" "B" "c" "C" "d" "D" \
"e" "E" "q" "Q"
    resp=`cat ${pathtemp}/temp.ans`
    rm ${pathtemp}/temp.ans
    case ${resp} in
      q|Q)
          getgoing=${false}
          ;;
      a|A)
          searchit="Award"
          ;;
      b|B)
          searchit="EP: "
          ;;
      c|C)
          searchit="Training: "
          ;;
      d|D)
          clear
          echo "Here are the types of travel available:\n"
          echo "  A) CONUS,\n  B) Invitational,\n  C) Local,\n  D) OCONUS,"
          echo "  E) Other,\n  F) All of the above.\n"
          echo "Enter the letter for the correct line: \c"
          read travtype
          case ${travtype} in
            "")
                travtype="?"
                ;;
             *)
                travtype=`echo "${travtype}" | cut -f1 -d" "`
                ;;        #This insures a one-word answer to input to anscheck.x.
          esac
          anscheck.x ${travtype} ${pathtemp}/temp.ans "a" "A" "b" "B" "c" "C" \
"d" "D" "e" "E" "f" "F"
          travtype=`cat ${pathtemp}/temp.ans`
          rm ${pathtemp}/temp.ans
```

```
        case ${travtype} in
          a|A)
             searchit="TOC "
             ;;
          b|B)
             searchit="TOI "
             ;;
          c|C)
             searchit="TOL "
             ;;
          d|D)
             searchit="TOO "
             ;;
          e|E)
             searchit="TOX "
             ;;
          f|F)
             searchit="All Travel"
             ;;
        esac
        ;;
    e|E)
        badstring=${true}
        while [ ${badstring} -eq ${true} ]
        do
          echo "\nEnter the word or phrase to search for; the case of the"
          echo "letters (upper or lower) will be ignored:"
          read searchit
          case ${searchit} in
            "")
               searchit="nothing"
               echo "\nWARNING: Entering nothing will terminate the program."
               ;;
          esac
          echo "\nYou have entered: ${searchit}.  Is this OK?"
          yes.or.no 3
          resp=`cat ${pathtemp}/temp.ans`
          rm ${pathtemp}/temp.ans
          case ${resp} in
            Y)
               badstring=${false}
               ;;
          esac
          case ${searchit} in
            nothing)
                     getgoing=${false}
                      ;;
          esac
        done
        ;;
    esac    #End of determination of searchit string.
#
    if [ ${getgoing} -eq ${true} ]
    then
      clear
```

```
      echo "This program will search the files for:  ${searchit}.\n"
      echo "The process may take between 1-4 min., depending on the number of"
      echo "files to be searched and on how many users are on the system."
#
      cd account.fy${fy}.d
      for dir in *        #$dir= labor.d or nonlabor.d.
      do
        if [ -d ${dir} ]
        then
          cd ${dir}
#
          for subdir in *    #$subdir= 6.1.d, travel.d, etc.
          do
            if [ -d ${subdir} ]
            then
              cd ${subdir}
#
              for file in *
              do
                if [ -f ${file} ]
                then
                  linenum=`wc -l ${file} | cut -c1-7`
                  if [ ${linenum} -gt 19 ]   #Data has been input to the file.
                  then
                    case ${searchit} in
                      "All Travel")
#
                        for trav in "TOC " "TOI " "TOL " "TOO " "TOX "
                        do
                          case ${dir} in
                            labor.d)
                              grep "${trav}" ${file} | cut -f2,3,4,5,6 -d\| >>\
${pathtemp}/temp.matchfile
                              ;;
                            nonlabor.d)
                              grep "${trav}" ${file} | cut -f2,3,4 -d\| >>\
${pathtemp}/temp.matchfile
                              ;;
                          esac
                        done
                        ;;
#
                      *)
                        case ${dir} in
                          labor.d)
                           grep -i "${searchit}" ${file} |cut -f2,3,4,5,6 -d\| \
>> ${pathtemp}/temp.matchfile
                            ;;
                          nonlabor.d)
                            grep -i "${searchit}" ${file} | cut -f2,3,4 -d\| >>\
${pathtemp}/temp.matchfile
                            ;;
                        esac
                        ;;
                    esac   #End case searchit.
```

```
#
                if [ -s ${pathtemp}/temp.matchfile ]
                then                    #If the file is generated & has data in
                                        #it, then add data in temp file to file
                                        #holding all matched data:
#
                  if [ ! -f ${pathtemp}/foundfile ]
                  then                              #<-Create file if not exist.
                   sed -e '6,$d' ${path1}/nonlabor.form > \
${pathtemp}/foundfile
                  fi
                  echo "\nAccount Directory:  ${subdir}" >> \
${pathtemp}/foundfile
              #Copy account information to foundfile:
#
                  sed -e '4,$d' ${file} >> ${pathtemp}/foundfile
                  case ${dir} in
                    labor.d)
                      currbal='tail -1 ${file} | cut -f7 -d\|'
                      ;;
                    nonlabor.d)
                      currbal='tail -1 ${file} | cut -f5 -d\|'
                      ;;
                  esac
                  echo "Current Balance = $ ${currbal}\n" >> \
${pathtemp}/foundfile
                  case ${dir} in    #Copy col. headings to foundfile.
                    labor.d)
                      tail -4 ${path1}/labor.form |cut -f2,3,4,5,6 -d\| >>\
${pathtemp}/foundfile
                      ;;
                    nonlabor.d)
                      tail -4 ${path1}/nonlabor.form | cut -f2,3,4 -d\| >>\
${pathtemp}/foundfile
                      ;;
                  esac
                  cat ${pathtemp}/temp.matchfile >> ${pathtemp}/foundfile
                  echo "\f\n" >> ${pathtemp}/foundfile
                  rm ${pathtemp}/temp.matchfile
                else
                  if [ -f ${pathtemp}/temp.matchfile ]
                  then
                    rm ${pathtemp}/temp.matchfile
                  fi
                fi
                #
              fi  #End if linenum > 19.
              #
            fi  #End if file exists.
            #
          done  #End for file in subdir.
          #
          cd ..  #Go back to dir (labor.d, nonlabor.d).
          #
        fi  #End if subdir is a directory.
```

```
        #
      done  #End for subdir in dir.
      #
      cd .. #Go back to account.fyxx.d.
      #
    fi  #End if dir exists as a directory.
    #
  done #End for dir in account.fyxx.d.
#
  cd ${pathtemp}
  clear
  if [ -f foundfile ]    #If a data file were generated:
  then
    echo "Here is the data that matched your search:\n"
    pg foundfile
    echo "\nDo you want to print the results?"
    ${path1}/yes.or.no 4
    resp='cat temp.ans'
    rm temp.ans
    case ${resp} in
      Y)
        wpr foundfile
        ;;
    esac
    echo "\nThe file containing the results is a temporary file."
    echo "Do you want a copy stored on your directory?"
    ${path1}/yes.or.no 5
    resp='cat temp.ans'
    rm temp.ans
    case ${resp} in
      Y)
        badfile=${true}
        while [ ${badfile} -eq ${true} ]
        do
          echo "\nEnter file name: \c"
          read fname
          case ${fname} in
            "")
              echo "\nYou must enter a file name."
              ;;
             *)
              if [ -f $1/${fname} ]
              then
                echo "\nThe file you have entered already exists."
                echo "Select another file name?"
                yes.or.no 6
                ans='cat ${pathtemp}/temp.ans'
                rm ${pathtemp}/temp.ans
                case ${ans} in
                  N)
                    badfile=${false}
                    ;;
                esac
              else
                badfile=${false}
```

```
                    cat foundfile >> $1/${fname}
                  fi
                  ;;
              esac
            done
            ;;
        esac
        rm foundfile
#
      else
        echo "The search program could not find any match to:  ${searchit}."
      fi
#
      cd ${path1}
      echo "\nAnother search?"
      yes.or.no 7
      resp='cat ${pathtemp}/temp.ans'
      rm ${pathtemp}/temp.ans
      case ${resp} in
        N)
          getgoing=${false}
          ;;
      esac
#
    fi  #End if getgoing = true.
#
  done  #End while getgoing = true.
#
fi #End if getgoing = true.
```

```
#Program printit prints the data found in the labor and/or nonlabor accounts,
#and/or the update file sum.info.
#
true=0
false=1
path1='pwd'
pathtemp="${path1}/temp.d"
#
clear
echo "Do you want to print files?"
yes.or.no 1
resp='cat ${pathtemp}/temp.ans'
rm ${pathtemp}/temp.ans
case ${resp} in
  Y)
    getgoing=${true}
    ;;
  N)
    getgoing=${false}
    ;;
esac
if [ ${getgoing} -eq ${true} ]
then
  baddir=${true}
  while [ ${baddir} -eq ${true} ]
  do
    fy=100
    while [ ${fy} -lt 0 -o ${fy} -gt 99 ]
    do
      echo "\nEnter the last two digits of the fiscal year, or simply"
      echo "press <enter> or <return> if the fiscal year is 93: \c"
      read fy
      case ${fy} in
        "")
            fy=93
            ;;
      esac
    done
    if [ -d account.fy${fy}.d ]
    then
      baddir=${false}
    else
      echo "\nThere are no files for fiscal year ${fy}.  Select another year?"
      yes.or.no 2
      resp='cat ${pathtemp}/temp.ans'
      rm ${pathtemp}/temp.ans
      case ${resp} in
        N)
          baddir=${false}
          getgoing=${false}
          ;;
      esac
    fi
  done
#
```

```
  if [ ${getgoing} -eq ${true} ]
  then
    clear
    echo "\nPrint all labor accounts?"
    yes.or.no 3
    resplab='cat ${pathtemp}/temp.ans'
    rm ${pathtemp}/temp.ans
#
    echo "\nPrint all non-labor accounts?"
    yes.or.no 4
    respnl='cat ${pathtemp}/temp.ans'
    rm ${pathtemp}/temp.ans
#
    echo "\nPrint the file containing the summary of all accounts?"
    yes.or.no 5
    respsum='cat ${pathtemp}/temp.ans'
    rm ${pathtemp}/temp.ans

    case ${resplab} in
      Y)
          echo "\nCreating the labor printout file..."
          cd account.fy${fy}.d/labor.d
          for dir in *
          do
            if [ -d ${dir} ]
            then
              cd ${dir}
              for file in *
              do
                if [ -f ${file} ]
                then
                  echo "Account Directory:  ${dir}" >> ${pathtemp}/temp.prnt.l
                  cat ${file} >> ${pathtemp}/temp.prnt.l
                  echo "\f" >> ${pathtemp}/temp.prnt.l
                fi
              done
              cd ..
            fi
          done
          cd ${path1}
          if [ ! -f ${pathtemp}/temp.prnt.l ]
          then
            echo "\nThis program could not find any labor accounts to print."
          fi
      ;;
    esac
#
    case ${respnl} in
      Y)
          echo "\nCreating the non-labor printout file..."
          cd account.fy${fy}.d/nonlabor.d
          for dir in *
          do
            if [ -d ${dir} ]
            then
```

```
            cd ${dir}
            for file in *
            do
              if [ -f ${file} ]
              then
                echo "Account Directory:  ${dir}" >> ${pathtemp}/temp.prnt.n
                cat ${file} >> ${pathtemp}/temp.prnt.n
                echo "\f" >> ${pathtemp}/temp.prnt.n
              fi
            done
            cd ..
          fi
        done
        cd ${path1}
        if [ ! -f ${pathtemp}/temp.prnt.n ]
        then
         echo "\nThis program could not find any non-labor accounts to print."
        fi
      ;;
    esac
#
    case ${respsum} in
      Y)
       path2="${path1}/account.fy${fy}.d"
#
       if [ ! -f ${path2}/sum.info ]    #If the file doesn't exist:
       then
        echo "\nThe summary file does not exist.  This program will create it."
        echo "This process could take 1-4 min., depending on the number of"
        echo "files and the number of users on the system."
        update 1 ${fy}
       fi
       echo "\nCreating the summary printout file..."
       cat ${path2}/sum.info > ${pathtemp}/temp.prnt.s
      ;;
    esac
#
# Print the requested files, if they exist:
#
    if [ -f ${pathtemp}/temp.prnt.? ]
    then
      wpr ${pathtemp}/temp.prnt.?
      rm ${pathtemp}/temp.prnt.?
    fi
#
  echo "\nThis program has ended.  Enter any key to return to the main menu: \c"
  read resp
#
  fi  #End if getgoing.
#
fi  #End if getgoing.
```

```
#Program setup.1
#
true=0
false=1
path1='pwd'  #path1= /Home dir/.../setup dir.
pathtemp="${path1}/temp.d"
#
echo "\n\nThis is the initial program used to set up the accounting files."
echo "It prepares the directories (file categories).  Usually, you need to"
echo "call this program only once in the fiscal year."
echo "\nA help message:  for these programs, DO NOT use the backspace"
echo "key to erase a character.  The correct key is <ctrl><h>.\n\n"
#
echo "Do you want to set up the file categories (directories)"
echo "for the account files?"
yes.or.no 1
resp='cat ${pathtemp}/temp.ans'
rm ${pathtemp}/temp.ans
case ${resp} in
  Y)
    startup=${true}
    ;;
  N)
    startup=${false}
    ;;
esac
if [ ${startup} -eq ${true} ]
then
#
#Start processing the directories by inputting the default directories:
#
#
  fy=100
  while [ ${fy} -lt 0  -o  ${fy} -gt 99 ]
  do
    echo "\nEnter the last two digits of the fiscal year (i.e. 91 for 1991),"
    echo "or else just hit the <enter> key if the fiscal year is 1993: \c"
    read fy
    case ${fy} in           #This construction supplies an answer in case the
      "")                   #enter key is hit with no value entered on the line.
        fy=93
        ;;
    esac
    echo "\nYou have entered ${fy}.  Is this OK?"
    yes.or.no 2
    resp='cat ${pathtemp}/temp.ans'
    rm ${pathtemp}/temp.ans
    case ${resp} in
      N)
        fy=100
        ;;
    esac
  done
#
  checkdir account.fy${fy}.d
```

```
#
  path2="${path1}/account.fy${fy}.d"
  cd account.fy${fy}.d
#
#Create the sub-directories if they do not already exist:
#
  ${path1}/checkdir labor.d nonlabor.d
#
  labcheck='ls labor.d'          #Since this program could be called more than
  nonlabcheck='ls nonlabor.d'    #once in the FY, all default directories will
                                 #be added only at the first use of this program
                                 #in case they were subsequently deleted or
                                 #otherwise altered.
  case ${labcheck} in
    "")
      ${path1}/checkdir labor.d/6.1.d labor.d/6.2.d labor.d/6.3.d \
labor.d/customer.d labor.d/overhead.d labor.d/ILIR.d
      ;;
  esac
#
  case ${nonlabcheck} in
    "")
      ${path1}/checkdir nonlabor.d/6.1.d nonlabor.d/6.2.d \
nonlabor.d/6.3.d nonlabor.d/ILIR.d nonlabor.d/customer.d \
nonlabor.d/misc.d nonlabor.d/overhead.d nonlabor.d/training.d \
nonlabor.d/travel.d
      ;;
  esac
#
#List the directories to the screen, ask if they are OK.  If they are not,
#ask which directories to add, delete, or rename.
#
  clear      #Clear the screen.
  count=0
  while [ ${count} -lt 2 ]
  do
    if [ ${count} -eq 0 ]
    then
      dirname="labor.d"
      echo "\nHere are the directories (file categories) for the labor \
accounts:\n"
    else
      dirname="nonlabor.d"
      echo "\nHere are the directories (file categories) for the non-labor \
accounts:\n"
    fi
#
    cd ${dirname}   #$path= path2/${dirname}.
    badans=${true}
    while [ ${badans} -eq ${true} ]
    do
     echo "\n"
     ls -C
     echo "\nAre these acceptable?"
     ${path1}/yes.or.no 3
     resp='cat ${pathtemp}/temp.ans'
```

```
    rm ${pathtemp}/temp.ans
    case ${resp} in
      Y)
        badans=${false}
        ;;
    esac
    if [ ${badans} -eq ${true} ]
    then
      clear
      echo "\n\n"
      ls -C
      echo "\nAdd a directory (a), change a directory name (c),"
      echo "delete a directory (d), or go to the next part of the setup"
      echo "program (g)?\n"
      echo "Enter a, c, d, or g: \c"
      read action
      case ${action} in
        "")
          action="?"
          ;;
        *)
          action=`echo "${action}" | cut -f1 -d" "`
          ;;
      esac
      ${path1}/anscheck.x ${action} ${pathtemp}/temp.1 "a" "A" "c" "C" "d" \
"D" "g" "G"
      action=`cat ${pathtemp}/temp.1`
      rm ${pathtemp}/temp.1
#
      case ${action} in
        a|A)
          clear
          echo "\n\n"
          ls -C
          echo "\nEnter the name of the directory to be added.  It can have no more"
          echo "than 15 characters, and the following characters cannot be used:"
          echo "\nsingle or double quotes / \ * ; - ? [ ] ( ) ~ ! $ { } > < "
          echo "or spaces at the start of or within a directory name.\n"
          read dname
          case ${dname} in
            "")
              dmake=${false}
              ;;
            *)
              dmake=${true}
              ;;
          esac
          if [ ${dmake} -eq ${false} ]
          then
            echo "\nThe directory could not be added as written."
            echo "Enter any key to continue: \c"
            read nextkey
          else
            ${path1}/checkdir ${dname}
          fi
```

```
        ;;
#
      c|C)
        clear
        echo "\n\n"
        ls -C
        echo "\nEnter the old name of the directory.  Please copy it"
        echo "exactly, using lower or upper case letters as shown above:"
        read dname1
        case ${dname1} in
          "")
            getdir=${false}
            ;;
          *)
            getdir=${true}
            ;;
        esac
        if [ getdir -eq ${true} ]
        then
         if [ -d ${dname1} ]
         then
#
          echo "\nEnter the new name of the directory.  It can have no more than"
          echo " 15 characters, and the following characters cannot be used:"
          echo "\nsingle or double quotes  / \ * ; - ? [ ] ( ) ~ ! $ { } > <"
          echo "or spaces at the start of or within a name.  It also can not"
          echo "be the same name as another directory.\n"
          read dname2
          case ${dname2} in
            "")
               getnew=${false}
                ;;
             *)
               getnew=${true}
               ;;
          esac
          if [ getnew -eq ${true} ]
          then
            if [ ! -d ${dname2} ]
            then
              mv ${dname1} ${dname2}
            else
              echo "\nThe directory could not be renamed because another"
              echo "directory already has that name.\n"
              echo "Enter any key to continue: \c"
              read nextkey
            fi
          else
            echo "\nThe directory cannot be renamed."
            echo "Enter any key to continue: \c"
            read nextkey
          fi
         else
          echo "\nThe directory you have chosen does not exist."
```

```
            echo "Enter any key to continue: \c"
            read nextkey
          fi
#
        else
          echo "\nThe directory could not be renamed."
          echo "Enter any key to continue: \c"
          read nextkey
        fi
      ;;
#
    d|D)
        clear
        echo "\n\n"
        ls -C
        echo "\nEnter the name of the directory to delete (upper and lower case"
        echo "letters must exactly match).  If you don't want to delete a"
        echo "directory, then just hit the <return> or <enter> key:"
        read dname
        case ${dname} in
          "")
            deldir=${false}
            ;;
          *)
            if [ -d ${dname} ]
            then
              deldir=${true}
            else
              deldir=${false}
            fi
            ;;
        esac
        if [ ${deldir} -eq ${true} ]
        then
          rmdir ${dname}
        else
          echo "\nYour directory could not be deleted."
          echo "Please enter any key to continue: \c"
          read nextkey
        fi
        ;;
#
    g|G)
        badans=${false}
        ;;
#
  esac
fi
#
done
#
clear
count=`expr ${count} + 1`
cd ${path2}    #Go back one directory, i.e. to account.fy??.d.
```

```
  done
#
  cd ..    #Go back one directory, i.e. to the setup directory (path1).
  clear
  echo "\nThis is the end of the file category (directory) setup program.  The"
  echo "next step is to create the actual accounts.  You do this by choosing"
  echo "menu selection B on the startup.acc menu screen (which follows)."
  echo "\nEnter any key to continue: \c"
  read resp
#
fi  #End if startup=true.
```

```
#Program setup.2
#
true=0
false=1
#
path1='pwd'
pathtemp="${path1}/temp.d"
#
clear
echo "This is the program used to create the account files.  It may be"
echo "used at any time in the fiscal year.\n"
echo "Warning: You must have used menu selection A from the startup.acc"
echo "         menu to create the file directories (categories).  If you"
echo "         have not yet done so, quit this program by entering N or n"
echo "         to answer the question below, then select choice A on the"
echo "         startup.acc menu screen.\n"
echo "A useful hint: DO NOT USE the <rubout> or <backspace> key to correct the"
echo "screen.  Use <crtl><h> instead.\n\n"
#
echo "Do you want to set up the account files?"
yes.or.no 1
resp='cat ${pathtemp}/temp.ans'
rm ${pathtemp}/temp.ans
case ${resp} in
  Y)
    startup=${true}
    ;;
  N)
    startup=${false}
    ;;
esac
if [ ${startup} -eq ${true} ]
then
  baddir=${true}
  while [ ${baddir} -eq ${true} ]
  do
    fy=100
    while [ ${fy} -lt 0 -o ${fy} -gt 99 ]
    do
    echo "\nEnter the last two digits of the fiscal year, or just hit <enter>"
    echo "or <return> if the year is 93: \c"
    read fy
    case ${fy} in
      "")
         fy=93
         ;;
    esac
    done
    if [ -d account.fy${fy}.d ]
    then
      baddir=${false}
    else
      echo "\nThere are no files for fiscal year ${fy}.  Choose another year?"
      yes.or.no 2
      resp='cat ${pathtemp}/temp.ans'
```

```
      rm ${pathtemp}/temp.ans
      case ${resp} in
        N)
          baddir=${false}
          startup=${false}
          ;;
      esac
    fi
  done
#
fi
if [ ${startup} -eq ${true} ]
then
#
#Get the current date:
#
 todate='date '+%D''
#
#Go to the current master account directory, then set up the directory paths:
#
 cd account.fy${fy}.d
 path2="${path1}/account.fy${fy}.d"
 pathlab="${path2}/labor.d"
 pathnl="${path2}/nonlabor.d"
#
 getgoing=${true}
#
 while [ ${getgoing} -eq ${true} ]
 do
  badans=${true}
  while [ ${badans} -eq ${true} ]
  do
    echo "\nEnter the account (or XO) number:"
    echo "(Note: If this is a labor account, please use 5 characters only. If"
    echo "       this is a non-labor account, you can use up to 14 characters.)"
    read account
    case ${account} in
      "")
        account="None"
        accgood=${false}
        getgoing=${false}
        ;;
      *)
        accgood=${true}
        getgoing=${true}
        ;;
    esac
    echo "\nThis is the input account (or XO) no.: ${account}."
    if [ ${getgoing} -eq ${false} ]
    then
      echo "\n*************************************************************"
      echo "*  WARNING:  Not entering an account number will cause  *"
      echo "*            the program to terminate.                  *"
      echo "*************************************************************\n"
    fi
```

```
    echo "Is this number OK?"
    ${path1}/yes.or.no 3
    resp=`cat ${pathtemp}/temp.ans`
    case ${resp} in
      Y)
        badans=${false}
        ;;
    esac
  done
#
  rm ${pathtemp}/temp.ans
  while [ ${accgood} -eq ${true} ]
  do
    echo "\nIs this a labor (L) or non-labor (N) account? (Enter L or N:) \c"
    read acctype
    case ${acctype} in
      "")
         acctype="?"
         ;;
       *)
         acctype=`echo "${acctype}" | cut -f1 -d" "`
         ;;
    esac
    ${path1}/anscheck.x ${acctype} ${pathtemp}/temp.ans "1" "L" "n" "N"
    acctype=`cat ${pathtemp}/temp.ans`
    rm ${pathtemp}/temp.ans
    case ${acctype} in
      1|L)
          cd ${pathlab}
          ;;
      n|N)
          cd ${pathnl}
          ;;
    esac
#
    sametype=${true}
    while [ ${sametype} -eq ${true} ]
    do
      clear
      badresp=${true}
      while [ ${badresp} -eq ${true} ]
      do
        echo "\n\nHere are the available file categories (directories):\n"
        ls -C
        echo "\nChoose a directory (file category).  Copy only enough letters"
        echo "to fully identify the directory.  Use upper and lower case"
        echo "letters as shown:\n"
        read dirname0
        case ${dirname0} in
          "")
             dirname0="_"
             ;;
        esac
        dirname="${dirname0}*"
        if [ -d ${dirname} ]       #make sure the directory exists
```

```
        then
          badresp=${false}
        else
          echo "\nThe directory you have entered does not exist.  Choose \
another directory?"
        echo "(If the answer is "n" or "N", this part of the program will end.)"
          ${path1}/yes.or.no 4
          resp='cat ${pathtemp}/temp.ans'
          rm ${pathtemp}/temp.ans
          case ${resp} in
            N)
              getgoing=${false}
              accgood=${false}
              sametype=${false}
              badresp=${false}
              samedir=${false}
              ;;
          esac
        fi
      done
#
      if [ -d ${dirname} ]
      then
        cd ${dirname}
        if [ -f ${account} ]   #Check whether or not this account
        then                   #already exists.
          samedir=${false}
          sametype=${false}
          accgood=${false}
          echo "\nAccount No. ${account} already exists in this directory."
        echo "In a moment, this program will ask you for a new account no.\n"
        else
          samedir=${true}
        fi
      fi
#
#Set up the input data for the account file:
#
      while [ ${samedir} -eq ${true} ]
      do
        case ${acctype} in
          l|L)
              classif="Labor"
              fileform="${path1}/labor.form"  #File containing heading lines.
              ;;
          n|N)
              classif="Non-Labor"
              fileform="${path1}/nonlabor.form"
              ;;
        esac
        echo "\nEnter the purpose for the account (i.e. CW Research, Office"
        echo "Purchases, etc.) (50 characters or less):"
        read purpose
        case ${purpose} in
          "")
```

```
            purpose="None Entered."
            ;;
        esac
        echo "\nEnter the name of the Principle Investigator (if any), or just"
        echo "hit the <enter> or <return> key (if none):"
        read prin
        case ${prin} in
          "")
            prin="None."
            ;;
        esac
        echo "\nEnter the starting balance, in dollars.  Do not include \
letters,"
        echo "commas, or the dollar sign (e.g. use 10000.00, not 10K or $ \
10,000)."
        echo "(Simply hitting <enter> (without a number) will set the balance \
to $ 0.00.)"
        read bal
        case ${bal} in
          "")
            bal=0.00
            ;;
        esac
        case ${acctype} in
          l|L)
            echo "\nEnter the average cost per hour of the labor charges, in"
            echo "dollars.  Do not use commas or the dollar sign. (Just"
            echo "hitting <enter> or <return> will automatically enter a"
            echo "value of $ 40.)"
            read hourate
            case ${hourate} in
              "")
                hourate=40.00
                ;;
            esac
            ;;
        esac
        echo "\nEnter the month, day, and year of the expiration date, as "
        echo "follows: mm/dd/yr, or hit <enter> or <return> if the date is"
        echo "9/30/93: \c"
        read expdate
        case ${expdate} in
          "")
            expdate="9/30/93"
            ;;
        esac
#
        badanswer=${true}
        while [ ${badanswer} -eq ${true} ]
        do
          clear
          echo "\n\nHere are the data you've entered:\n"
          echo "  A) Account Number (or XO Number): ${account}\n"
          echo "  B) Classification: ${classif}\n"
          echo "  C) Purpose for Account: ${purpose}\n"
```

```
            echo "  D) Principle Investigator: ${prin}\n"
            echo "  E) Starting Balance: $ ${bal}\n"
            echo "  F) Account Expiration Date: ${expdate}\n"
            case ${acctype} in
              l|L)
                 echo "  G) Rate/Hour: $ ${hourate}\n"
                 ;;
            esac
            echo "\nAre these data OK?"
            ${path1}/yes.or.no 5
            resp=`cat ${pathtemp}/temp.ans`
            rm ${pathtemp}/temp.ans
            case ${resp} in
                Y)
                  addfile=${true}
                  badanswer=${false}
                  ;;
                N)
                  addfile=${false}
                  echo "\nEnter the letter of the line to be changed"
                  case ${acctype} in
                    n|N)
                        echo "(A, B, C, D, E, F), or if no changes are desired,"
                        ;;
                    l|L)
                       echo "(A, B, C, D, E, F, G), or if no changes are desired,"
                        ;;
                  esac
                  echo "just hit the <enter> or <return> key: \c"
                  read linelet
                  case ${linelet} in
                    "")
                       badanswer=${false}
                       addfile=${true}
                       ;;
                     *)
                       linelet=`echo "${linelet}" | cut -f1 -d" "`
                       ;;       #This provides a one-word answer to anscheck.x.
                  esac
#
                  if [ ${badanswer} -eq ${true} ]
                  then
                    case ${acctype} in
                      n|N)
                    ${path1}/anscheck.x ${linelet} ${pathtemp}/temp.ans "a" "A" \
    "b" "B" "c" "C" "d" "D" "e" "E" "f" "F"
                         ;;
                      l|L)
                    ${path1}/anscheck.x ${linelet} ${pathtemp}/temp.ans "a" "A" \
    "b" "B" "c" "C" "d" "D" "e" "E" "f" "F" "g" "G"
                         ;;
                    esac
                    linelet=`cat ${pathtemp}/temp.ans`
                    rm ${pathtemp}/temp.ans
                    case ${linelet} in
```

```
a|A)
    badanswer=${false}  #Break out of all but the initial
    addfile=${false}    #loop (getgoing) to the beginning
    samedir=${false}    #of data entry for the accounts.
    sametype=${false}
    accgood=${false}
    ;;
b|B)
    badanswer=${false}  #Break back to the loop selecting
    addfile=${false}    #labor or non-labor accounts
    samedir=${false}    #(accgood).
    sametype=${false}
    ;;
c|C)
    echo "\nEnter the purpose for the account (office pur-"
    echo "chases, etc.) (50 Characters or less):"
    read purpose
    case ${purpose} in
      "")
         purpose="None Entered"
         ;;
    esac
    ;;
d|D)
    echo "\nEnter the name of the Principle Investigator,"
    echo "if any.  If none, just hit the <return> key:"
    read prin
    case ${prin} in
      "")
         prin="None"
         ;;
    esac
    ;;
e|E)
    echo "\nEnter the starting balance in dollars or hours,"
    echo "as desired.  Do not include commas or the dollar"
    echo "sign if entering a dollar balance (i.e. use num-"
    echo "bers only).  (Hitting <return> or <enter> will"
    echo "set the balance to zero.)"
    read bal
    case ${bal} in
     "")
        bal=0.0
        ;;
    esac
    ;;
f|F)
    echo "\nEnter the month, day, and year of the account"
    echo "expiration date, as follows: mm/dd/yy, or hit"
    echo "<enter> or <return> if the date is 9/30/93:"
    read expdate
    case ${expdate} in
      "")
         expdate="9/30/93"
         ;;
```

```
                    esac
                    ;;
                g|G)
                    echo "\nEnter the average rate per hour of the labor"
                    echo "charged to this account, in dollars.  Do not"
                    echo "include commas or the dollar sign: (Just hitting"
                    echo "<enter> or <return> will input a value of $ 40.00"
                    read ${hourate}
                   .case ${hourate} in
                      "")
                        hourate=40.00
                        ;;
                    esac
                ;;
              esac
#
            fi
            ;;
      esac
#
    done   #End of badanswer loop.
#
    if [ ${addfile} -eq ${true} ]
    then
      newfile="${account}"
      echo "Account Number:  ${account}" >> ${newfile}
      echo "Classification:  ${classif}" >> ${newfile}
      echo "Purpose for Account:  ${purpose}" >> ${newfile}
      echo "Principle Investigator: ${prin}" >> ${newfile}
      case ${acctype} in
        l|L)
            echo "Starting Balance: $ ${bal}" >> ${newfile}
            echo "Labor Charge (Rate per Hour): $ ${hourate}" >> ${newfile}
            ;;
        n|N)
            echo "Starting Balance: $ ${bal}\n" >> ${newfile}
            ;;
      esac
      echo "Date Created:  ${todate}" >> ${newfile}
      case ${acctype} in
        l|L)
            echo "Initial Expiration Date:  ${expdate}\n\n\n\n\n" >> \
${newfile}
            ;;
        n|N)
            echo "Initial Expiration Date:  ${expdate}\n\n" >> ${newfile}
            ;;
      esac
      cat ${fileform} >> ${newfile}
      chmod 660 ${newfile}
      echo "\n\nAdd another account?"
      ${path1}/yes.or.no 6
      moreacc='cat ${pathtemp}/temp.ans'
      rm ${pathtemp}/temp.ans
      case ${moreacc} in
```

```
            Y)
              badans=${true}
              while [ ${badans} -eq ${true} ]
              do
                echo "\nEnter the account (or XO) number:"
                echo "(Note: If this is a labor account, please use 5 characters \
only. If"
                echo "       this is a non-labor account, you can use up to 14 \
characters.)"
                read account
                case ${account} in
                  "")
                      account="None"
                      accgood=${false}
                      samedir=${false}
                      sametype=${false}
                      getgoing=${false}
                      ;;
                   *)
                      accgood=${true}
                      getgoing=${true}
                      ;;
                esac
                echo "\nYou have entered the following account: ${account}"
                if [ ${getgoing} -eq ${false} ]
                then
                echo "\n*************************************************************"
                 echo "*  WARNING:  Entering no account number will cause the  *"
                 echo "*            program to terminate.                      *"
                echo "*************************************************************\n"
                fi
                echo "Is this number OK?"
                ${path1}/yes.or.no 7
                resp=`cat ${pathtemp}/temp.ans`
                rm ${pathtemp}/temp.ans
                case ${resp} in
                    Y)
                      badans=${false}
                      ;;
                esac
#
               done    #End of badans loop.
#
               if [ ${accgood} -eq ${true} ]
               then
                 echo "\nIs the account in the same file category (${dirname0})?"
                 ${path1}/yes.or.no 8
                 resp=`cat ${pathtemp}/temp.ans`
                 rm ${pathtemp}/temp.ans
                 case ${resp} in
                     Y)
                       if [ -f ${account} ]
                       then
                          accgood=${false}
```

```
                        sametype=${false}
                        samedir=${false}
                        gooddir=${false}
                        echo "\nThe account #${account} already exists."
                        echo "In a moment, the program will ask you for"
                        echo "another account number./n/n"
                    else
                        gooddir=${true}
                        samedir=${true}
                        sametype=${true}
                    fi
                    ;;
                N)
                    gooddir=${true}
                    samedir=${false}
                    ;;
            esac
            if [ ${gooddir} -eq ${true} ]
            then
              if [ ${samedir} -eq ${false} ]
              then
                case ${acctype} in
                  l|L)
              echo "\nIs this another labor account?"
                    ;;
                  n|N)
              echo "\nIs this another non-labor account?"
                    ;;
                esac
                ${path1}/yes.or.no 9
                resp='cat ${pathtemp}/temp.ans'
                rm ${pathtemp}/temp.ans
                case ${resp} in
                  N)
                    sametype=${false}
                    ;;
                  Y)
                    sametype=${true}
                    ;;
                esac
              fi
            fi
#
        fi
        ;;
    n|N)
        samedir=${false}
        sametype=${false}
        accgood=${false}
        getgoing=${false}
        ;;
  esac
#
fi
#
```

```
        done    #Dir= pathlab/dirname or pathnl/dirname.
                #End of samedir loop.
#
        cd ..   #Dir= pathlab or pathnl (=path2/labor.d or path2/nonlabor.d).
     done      #End of sametype loop.
#
    cd ..     #Reset dir to path2 (=path1/account.fyxx.d).
  done    #End of accgood loop.
#
done   #End of getgoing loop.
#
cd ${path1}  #Return to the main (or starting) directory.
clear
echo "Choice C on the startup.acc menu (which follows) will activate the"
echo "program that adds charges to individual labor and non-labor accounts."
echo "To add general labor charges to multiple accounts, select choice D from"
echo "the startup.acc menu screen.  You can run these programs at any time,"
echo "provided that the account file already exists."
echo "\nEnter any key to continue:"
read resp
#
fi #End if startup= true.
```

```
#startup.acc is the program which controls the main functions of the accounting
#program.  This program is used by everyone with full access to the accounts,
#and is the only program that is distributed to all who may both write to and
#read the account data files.
#
usrpath='pwd'
cd /usr/OPDAT/account.d   #Installer to amend this command with proper name.
true=0
false=1
clear
echo "Do you want to use the accounting program?"
yes.or.no 1   #This is a shell program to return a yes-or-no answer.
ans='cat temp.d/temp.ans'
rm temp.d/temp.ans
case ${ans} in
  Y)
    getgoing=${true}
    ;;
  N)
    getgoing=${false}
    ;;
esac
while [ ${getgoing} -eq ${true} ]
do
   clear
   echo "Here are the available programs:\n"
   echo "  A)  Create the directories (file categories)."
   echo "  B)  Set up the account files."
   echo "  C)  View, print, and/or add charges to files."
   echo "  D)  Enter general labor charges to up to 25 separate accounts."
   echo "  E)  Create the file containing the updated status summary of "
   echo "        all accounts in the specified fiscal year."
   echo "  F)  Search the summary file for operator-specified keywords."
   echo "  G)  Search the account files for operator-specified keywords."
   echo "  H)  Print all labor and/or non-labor files, and/or the summary file."
   echo "\nEnter a letter (A-H) or Q to quit the accounting program: \c"
   read ans
   case ${ans} in
     "")
        ans="?"
        ;;
      *)
        ans='echo "${ans}" | cut -f1 -d" "'  #This insures a 1-word response.
        ;;
   esac
   anscheck.x ${ans} temp.d/temp.ans "a" "A" "b" "B" "c" "C" "d" "D" "e" "E" \
"f" "F" "g" "G" "h" "H" "q" "Q"
   ans='cat temp.d/temp.ans'
   rm temp.d/temp.ans
   case ${ans} in
     a|A)
          setup.1
          ;;
     b|B)
          setup.2
          ;;
```

```
      c|C)
           edit.data
           ;;
      d|D) gen.labor ${usrpath}
           ;;
      e|E)
           update 0
           ;;
      f|F)
           sum.search ${usrpath}
           ;;
      g|G)
           keyword ${usrpath}
           ;;
      h|H)
           printit
           ;;
      q|Q)
           getgoing=${false}
           ;;
    esac
done
cd ${usrpath}
```

```
#Program sum.search
#
#Note: This program is to be used with the program startup.acc.  The input
#argument ($1) is used to supply the user's own directory for file copying.
#
true=0
false=1
path1=`pwd`
pathtemp="${path1}/temp.d"
#
clear
echo "Do you want to searc! the summary file for a key word or phrase?"
yes.or.no 1
resp=`cat ${pathtemp}/temp.ans`
rm ${pathtemp}/temp.ans
case ${resp} in
  Y)
    startup=${true}
    ;;
  N)
    startup=${false}
    ;;
esac
if [ ${startup} -eq ${true} ]
then
  badfile=${true}
  while [ ${badfile} -eq ${true} ]
  do
    fy=100
    while [ ${fy} -lt 0 -o ${fy} -gt 99 ]
    do
      echo "\nEnter the last two digits of the fiscal year, or just hit <enter>"
      echo "or <return> if the year is 93: \c"
      read fy
      case ${fy} in
        "")
          fy=93
          ;;
      esac
    done
    if [ -f account.fy${fy}.d/sum.info ]
    then
      badfile=${false}
    else
      echo "\nThe summary file for fiscal year ${fy} does"
      echo "not exist.  Select another year?"
      yes.or.no 2
      resp=`cat ${pathtemp}/temp.ans`
      rm ${pathtemp}/temp.ans
      case ${resp} in
        N)
          badfile=${false}
          startup=${false}
          echo "\nTo generate the summary file, please choose option E from"
          echo "the startup.acc menu, which follows."
```

```
          echo "\nEnter any key to continue:"
          read resp
          ;;
      esac
    fi
  done
#
 if [ ${startup} -eq ${true} ]
 then
#
  path2=${path1}/account.fy${fy}.d
#
  clear
  echo "This program searches the summary file for key words or phrases, which"
  echo "you enter below.  You can use this program to search for all informa-"
  echo "tion available on a specific account number, account balance, principle"
  echo "investigator, etc."
#
  getgoing=${true}
  while [ ${getgoing} -eq ${true} ]
  do
    echo "\nHere are the first 11 lines of the summary file:\n"
    sed -e '12,$d' ${path2}/sum.info
#
    badinfo=${true}
    while [ ${badinfo} -eq ${true} ]
    do
      echo "\nPlease enter the word or phrase to search for (i.e. the actual name"
      echo "of the principle investigator, the account no., etc.  The case of the"
      echo "letters will be ignored.):"
      read searchit
      case ${searchit} in
        "")
          searchit="nothing"
          echo "WARNING: Entering nothing will cause the program to terminate."
          ;;
      esac
      echo "\nYou have entered:  ${searchit}.  Is this OK?"
      yes.or.no 3
      resp=`cat ${pathtemp}/temp.ans`
      rm ${pathtemp}/temp.ans
      case ${resp} in
        Y)
          badinfo=${false}
          case ${searchit} in
            nothing)
                    getgoing=${false}
                    ;;
          esac
          ;;
      esac
#
    done
#
```

```
    if [ ${getgoing} -eq ${true} ]
    then
      grep -i "${searchit}" ${path2}/sum.info > ${pathtemp}/temp.out
      if [ -s ${pathtemp}/temp.out ]
      then
       clear
       echo "Here are the lines that match your search input (${searchit}):\n"
       pg ${pathtemp}/temp.out
       echo "\nPrint these results (p), save data to a permanent file (s),"
       echo "do both (b), or none of the above (n)?  (Enter p, s, b, or n:) \c"
       read resp
       case ${resp} in
         "")
            resp="?"
            ;;
          *)
            resp=`echo "${resp}" | cut -f1 -d" "`
            ;;
       esac
       anscheck.x ${resp} ${pathtemp}/temp.ans "p" "P" "s" "S" "b" "B" "n" "N"
       resp=`cat ${pathtemp}/temp.ans`
       rm ${pathtemp}/temp.ans
       case ${resp} in
         p|P|b|B)
            wpr ${pathtemp}/temp.out
            ;;
       esac
       case ${resp} in
         s|S|b|B)
           badname=${true}
           while [ ${badname} -eq ${true} ]
           do
             echo "\nEnter file name for saving this data (it will be saved to"
             echo "your own directory):"
             read fname
             case ${fname} in
               "")
                 echo "\nYou must enter a file name."
                 ;;
               *)
                 if [ -s $1/${fname} ]
                 then
                   echo "\nThe file you have selected already exists.  Choose \
another file name?"
                   yes.or.no 4
                   ans=`cat ${pathtemp}/temp.ans`
                   rm ${pathtemp}/temp.ans
                   case ${ans} in
                     N)
                       badname=${false}
                       ;;
                   esac
                 else
                   badname=${false}
                   cat ${pathtemp}/temp.out >> $1/${fname}
```

```
                fi
                ;;
            esac
          done
          ;;
      esac
    else
      echo "\nThere were no lines to match:  ${searchit}."
    fi
    if [ -f ${pathtemp}/temp.out ]
    then
      rm ${pathtemp}/temp.out
    fi
    echo "\nAnother search?"
    yes.or.no 5
    resp=`cat ${pathtemp}/temp.ans`
    rm ${pathtemp}/temp.ans
    case ${resp} in
      N)
        getgoing=${false}
        ;;
      Y)
        clear
        ;;
    esac
#
  fi  #End if getgoing= true.
#
 done  #End while getgoing= true.
#
fi  #End if startup= true.
#
fi  #End if startup= true.
```

```
#Program update summarizes the data found in the labor and nonlabor accounts.
#It takes 2 input arguement: $1= 0 if the program is not called from printit,
#and $1= 1 if printit calls the function.  (Printit has already established that
#files exist for a given year (see below).)  $2 is used only with printit, and
#establishes the fiscal year for which an update file is requested.
#
true=0
false=1
path1='pwd'
pathtemp="${path1}/temp.d"
#
if [ $1 -eq 0 ]
then
  clear
  echo "Do you want to create the updated summary file?"
  yes.or.no 1
  resp='cat ${pathtemp}/temp.ans'
  rm ${pathtemp}/temp.ans
  case ${resp} in
    Y)
      startup=${true}
      ;;
    N)
      startup=${false}
      ;;
  esac
  if [ ${startup} -eq ${true} ]
  then
    baddir=${true}
    while [ ${baddir} -eq ${true} ]
    do
      fy=100
      while [ ${fy} -lt 0 -o ${fy} -gt 99 ]
      do
      echo "\nEnter the last two digits of the fiscal year, or just hit <enter>"
        echo "or <return> if the year is 93: \c"
        read fy
        case ${fy} in
          "")
             fy=93
             ;;
        esac
      done
      if [ -d account.fy${fy}.d ]
      then
        baddir=${false}
      else
        echo "\nThere are no files for fiscal year ${fy}.  Choose another year?"
        yes.or.no 2
        resp='cat ${pathtemp}/temp.ans'
        rm ${pathtemp}/temp.ans
        case ${resp} in
          N)
            baddir=${false}
            startup=${false}
```

```
            ;;
        esac
      fi
    done
#
  fi  #End if startup = true.
#
else
  startup=${true}
  fy=$2
fi                #End if [ $1= 0 ].
#
if [ ${startup} -eq ${true} ]
then
#
 if [ $1 -eq 0 ]
 then
   clear
 echo "This program generally takes 1-4 min. to run, depending on the number of"
 echo "files being summarized, and the number of users on the system. There are"
   echo "no further questions for the operator to answer."
 fi
#
 path2=${path1}/account.fy${fy}.d
#
 if [ -f ${path2}/sum.info ]
 then
   rm ${path2}/sum.info
 fi
 date.get ${pathtemp}/temp.date ${fy}
 cd account.fy${fy}.d
 numfile=0
 for dir in *
 do
   if [ -d ${dir} ]
   then
     case ${dir} in
       labor.d)
               labind="1"
                ;;
      *)
                labind="n"
                ;;
     esac
     cd ${dir}
     for subdir in *
     do
       if [ -d ${subdir} ]
       then
         cd ${subdir}
         for file in *
         do
           if [ -f ${file} ]
           then
             linenum=`wc -l ${file} | cut -c1-7`
```

```
            newdate='date'
            echo "Account Directory: ${subdir}" >> ${path2}/sum.info
            echo "Account No. ${file}:  Time of Update:  ${newdate}">> \
${path2}/sum.info
            ${path1}/getinfo.x ${file} ${labind} temp.info ${linenum} \
${pathtemp}/temp.date ${fy} ${file}
            cat temp.info >> ${path2}/sum.info   #sum.info is the summary file
            rm temp.info                         #name.
#
            numfile='expr ${numfile} + 1'    #After every 5 files, send a
            count='expr ${numfile} % 5'      #form-feed char. to file for later
            if [ ${count} -eq 0 ]            #printing.
            then
              echo "\f" >> ${path2}/sum.info
            fi
#
          fi
        done
        cd ..  #Go back to $dir.
      fi
    done
    cd ..  #Go back to account.fyxx.d.
  fi
done
cd ${path1}
chmod 660 ${path2}/sum.info
if [ -f ${pathtemp}/temp.date ]
then
  rm ${pathtemp}/temp.date
fi
#
fi #End if startup= true.
```

```
#yes.or.no returns a yes-or-no answer to the calling program.  It takes one
#argument:  $1= the number of times the function has been called within a given
#program.
#
true=0
false=1
pathstart='pwd'
cd /usr/OPDAT/account.d/temp.d   #Note: Installer must substitute for OPDAT's
                                 #path name.
if [ $1 -eq 1 ]
then
  echo "Enter y or n (for yes or no): \c"
else
  echo "Enter y or n: \c"
fi
read resp
checkit=${true}
while [ ${checkit} -eq ${true} ]
do
  case ${resp} in
    Y*|y*)
          echo "Y" > temp.ans
          checkit=${false}
          ;;
    N*|n*)
          echo "N" > temp.ans
          checkit=${false}
          ;;
        *)
          echo "\nPlease enter one of the following: y Y n N."
          read resp
          ;;
  esac
done
cd ${pathstart}
```

Appendix D: Header Files for Labor and Non-Labor Account Files

This is the listing of file "labor.form":

B=Bankrupt, B*= <20% of Starting Balance Remains, ODN= Office Document Number,
TE= Time Expired, TE*= <30 Days Remain

Acct. Flags	Date Of Entry	Transaction Description	Rate ($)/ Hour	Hours Entered	Transact. Amount ($)	Current Balance ($)	Days Remaining	Cumulative Hours

This is the listing of file "nonlabor.form":

B=Bankrupt, B*= <20% of Starting Balance Remains, EP= Equip. or Other Purchase,
ODN= Office Document Number, ON= Order Number, RN= Requisition Number,
TE= Time Expired, TE*= <30 days remain, TOC= Travel Orders (CONUS),
TOI= Travel Orders (Invitational), TOL= Travel Orders (Local),
TOO= Travel Orders (OCONUS), TOX= Travel Orders (Other)

Acct. Flags	Date Of Entry	Transaction Description	Transact. Amount ($)	Current Balance ($)	Days Remaining

Appendix E: UNIX Programs Related to the Use of Program "startup"

```
#Program edit.data.2
#
#This program is a abbreviated version of edit.data and is used in conjunction
#with the controller program startup (not startup.acc).
#
true=0
false=1
#
pathsmall='pwd'
smalltmp="${pathsmall}/temp.d"
path1="/usr/OPDAT/account.d"  #Replace with the correct path.
pathtemp="${path1}/temp.d"
#
clear
echo "This program finds files and enables the user to view and/or print the"
echo "charges in the individual account files.\n"
echo "A helpful hint:  DO NOT use the <backspace> or <rubout> keys to correct"
echo "input errors on the screen.  Use <crtl><h> instead. \n\n"
#
echo "Do you want to use this program?"
yes.or.no.2 1
resp='cat ${smalltmp}/temp.ans'
rm ${smalltmp}/temp.ans
case ${resp} in
  y|Y)
      startup=${true}
      ;;
  n|N)
      startup=${false}
esac
#
if [ ${startup} -eq ${true} ]
then
  cd ${path1}
  baddir=${true}
  while [ ${baddir} -eq ${true} ]
  do
    fy=100
    while [ ${fy} -lt 0 -o ${fy} -gt 99 ]
    do
      echo "\nEnter the last two digits of the fiscal year, or just hit <enter>"
      echo "or <return> if the year is 93: \c"
      read fy
      case ${fy} in
        "")
            fy=93
            ;;
      esac
    done
    if [ -d account.fy${fy}.d ]
    then
      baddir=${false}
    else
      echo "\nThere are no files for fiscal year ${fy}.  Choose another year?"
      yes.or.no 2
```

```
      resp='cat ${pathtemp}/temp.ans'
      rm ${pathtemp}/temp.ans
      case ${resp} in
        n|N)
            baddir=${false}
            startup=${false}
            ;;
      esac
    fi
  done
#
fi
#
if [ ${startup} -eq ${true} ]
then
#
 path2="${path1}/account.fy${fy}.d"
 pathlab="${path2}/labor.d"
 pathnl="${path2}/nonlabor.d"
#
 cd ${path2}
 getgoing=${true}
 while [ ${getgoing} -eq ${true} ]
 do
  badans=${true}
  while [ ${badans} -eq ${true} ]
  do
    echo "\nEnter the account (or XO) number (same as the file number in the"
    echo "computer), or enter q to quit: \c"
    read account
    case ${account} in
      "")
         account="None"
         accgood=${false}
         getgoing=${false}
         quit=${false}
         ;;
      Q|q)
         quit=${true}
         accgood=${false}
         getgoing=${false}
         badans=${false}
         ;;
      *)
         quit=${false}
         accgood=${true}
         getgoing=${true}
         ;;
    esac
#
    if [ ${quit} -eq ${false} ]
    then
      echo "\nThis is the input account (or XO) number: ${account}."
      if [ ${getgoing} -eq ${false} ]
      then
```

```
        echo "\n*************************************************************"
        echo "*  WARNING:  Not entering an account number will cause  *"
        echo "*            the program to terminate.                  *"
        echo "*************************************************************"
      fi
      echo "\nIs this number OK?"
      ${path1}/yes.or.no 3
      resp=`cat ${pathtemp}/temp.ans`
      rm ${pathtemp}/temp.ans
      case ${resp} in
        y|Y)
            badans=${false}
            ;;
      esac
    fi
  done
#
  while [ ${accgood} -eq ${true} ]
  do
    echo "\nIs this a labor (L) or non-labor account (N)? (Enter L or N:) \c"
    read acctype
    case ${acctype} in
      "")
          acctype="?"
          ;;
       *)
          acctype=`echo "${acctype}" | cut -f1 -d" "`
          ;;
    esac
    ${path1}/anscheck.x ${acctype} ${pathtemp}/temp.ans "l" "L" "n" "N"
    acctype=`cat ${pathtemp}/temp.ans`
    rm ${pathtemp}/temp.ans
    case ${acctype} in
      l|L)
          cd ${pathlab}
          ;;
      n|N)
          cd ${pathnl}
          ;;
    esac
#
    sametype=${true}
    while [ ${sametype} -eq ${true} ]
    do
      clear
      filepath=`find . -name ${account} -print`
      case ${filepath} in
        "")
          echo "\nAccount #${account} either does not exist, or "
          case ${acctype} in
            l|L)
                echo "does not exist as a labor account."
                ;;
            n|N)
                echo "does not exist as a non-labor account."
```

```
                ;;
            esac
            accgood=${false}
            sametype=${false}
            ;;
#
        *)
#Check to see if more than one file exists with the input name given (if the
#variable filepath has more than one line):
#
          numline=`echo "${filepath}" | wc -l | cut -c7`
          if [ ${numline} -eq 1 ]
          then
            infname=${filepath}
          else
            clear
            keepon=${true}
            while [ ${keepon} -eq ${true} ]
            do
             echo "\n\nAccount #${account} was found under these directories:\n"
             echo "${filepath}" | cut -f2 -d"/"
             echo "\nPlease enter the correct directory name, copying only enough"
             echo "characters from the list above (using upper and lower cases as"
             echo "shown) to completely distinguish the directory from its \
neighbors:"
             read dirname
             dirname="${dirname}*"
             if [ -d ${dirname} ]
             then
               infname="${dirname}/${account}"
               keepon=${false}
             else
               echo "\nThe directory you have entered does not exist."
               echo "Enter another account (a), try another directory (d),"
               echo "or quit this program (q)?  (Enter a, d, or q:) \c"
               read resp
               case ${resp} in
                 "")
                    resp="?"
                    ;;
                 *)
                    resp=`echo "${resp}" | cut -f1 -d" "`
                    ;;
               esac
               ${path1}/anscheck.x ${resp} ${pathtemp}/temp.ans "a" "A" "d" "D" \
"q" "Q"
               resp=`cat ${pathtemp}/temp.ans`
               rm ${pathtemp}/temp.ans
               case ${resp} in
                 a|A)
                     accgood=${false}
                     sametype=${false}
                     keepon=${false}
                     ;;
                 q|Q)
```

```
                    getgoing=${false}
                    accgood=${false}
                    sametype=${false}
                    keepon=${false}
                    ;;
               esac
#
          fi   #End if dirname.
#
        done   #End while keepon.
#
      fi  #End if numline.
#
      ;;
    esac  #End case filepath.
#
    if [ ${sametype} -eq ${true} ]
    then
#
#Look at the account files:
#
      echo "View data in file (v), print data (p), both view and print (b),"
      echo "or skip view/print of account (s)?  (Enter v, p, b, or s:) \c"
      read vuit
      case ${vuit} in
       "")
          vuit="?"
          ;;
        *)
          vuit=`echo "${vuit}" | cut -f1 -d" "`
    ;;
      esac
      ${path1}/anscheck.x ${vuit} ${pathtemp}/temp.ans "v" "V" "p" "P" "b" \
"B" "s" "S"
      vuit=`cat ${pathtemp}/temp.ans`
      rm ${pathtemp}/temp.ans
      case ${vuit} in
       v|V)
           pg ${infname}
           ;;
       p|P)
           wpr ${infname}
           ;;
       b|B)
           pg ${infname}
           wpr ${infname}
           ;;
      esac
      echo "\n\nAnother account?"
      ${path1}/yes.or.no 4
      resp=`cat ${pathtemp}/temp.ans`
      rm ${pathtemp}/temp.ans
      case ${resp} in
         y|Y)
            badans=${true}
```

```
    while [ ${badans} -eq ${true} ]
    do
     echo "\nEnter the account (or XO) number (same as the file"
     echo "number in the computer), or enter q to quit: \c"
     read account
     case ${account} in
       "")
          account="None"
          accgood=${false}
          getgoing=${false}
          sametype=${false}
          quit=${false}
          ;;
       Q|q)
          quit=${true}
          sametype=${false}
          accgood=${false}
          getgoing=${false}
          badans=${false}
          ;;
        *)
          quit=${false}
          accgood=${true}
          getgoing=${true}
          ;;
     esac
#
     if [ ${quit} -eq ${false} ]
     then
       echo "\nThis is the input account number: ${account}."
       if [ ${getgoing} -eq ${false} ]
       then
        echo "\n*************************************************************"
        echo "*  WARNING:  Entering no account number will cause the  *"
        echo "*            program to terminate.                      *"
        echo "*************************************************************"
       fi
       echo "\nIs this number OK?"
       ${path1}/yes.or.no 5
       ans=`cat ${pathtemp}/temp.ans`
       rm ${pathtemp}/temp.ans
       case ${ans} in
         y|Y)
             badans=${false}
             ;;
       esac
   fi
done
#
if [ ${getgoing} -eq ${true} ]
then
  case ${acctype} in
    l|L)
        echo "\nIs this another labor account?"
        ;;
```

```
                n|N)
                    echo "\nIs this another non-labor account?"
                    ;;
                esac
                ${path1}/yes.or.no 6
                ans='cat ${pathtemp}/temp.ans'
                rm ${pathtemp}/temp.ans
                case ${ans} in
                  y|Y)
                    sametype=${true}
                    ;;
                  n|N)
                    sametype=${false}
                    ;;
                esac
#             .
            fi
           ;;
#
          n|N)
              getgoing=${false}
              accgood=${false}
              sametype=${false}
              ;;
#
        esac  #End case resp.
#
      fi  #End if sametype.
#
    done  #End while sametype.
#
    cd ..  #Go to path1/account.fyxx.d.
#
  done   #End while accgood.
#
 done #End while getgoing.
 cd ${path1}
#
fi   #End if startup= true.
cd ${pathsmall}
```

```
#Program keyword.2
#
#This program is used by startup (not startup.acc) to conduct a keyword search
#of all files.
#
clear
pathsmall=`pwd`
smalltmp="${pathsmall}/temp.d"
path1="/usr/OPDAT/account.d"  #Replace with the correct path name.
pathtemp="${path1}/temp.d"
#
true=1
false=0
#
echo "This is the program which will search through all files to match a"
echo "keyword (e.g. training, travel, purchases, etc.); and will collect the"
echo "data, to be printed and/or saved to a file, as you wish."
echo "\nDo you want to use this program?"
yes.or.no.2 1
resp=`cat ${smalltmp}/temp.ans`
rm ${smalltmp}/temp.ans
case ${resp} in
  Y)
     getgoing=${true}
     ;;
  N)
     getgoing=${false}
     ;;
esac
#
if [ ${getgoing} -eq ${true} ]
then
  cd ${path1}
  baddir=${true}
  while [ ${baddir} -eq ${true} ]
  do
    fy=100
    while [ ${fy} -lt 0 -o ${fy} -gt 99 ]
    do
      echo "\nEnter the last two digits of the fiscal year to be searched,"
      echo "or press <enter> or <return> if the fiscal year is 93: \c"
      read fy
      case ${fy} in
       "")
          fy=93
          ;;
      esac
    done
    if [ -d account.fy${fy}.d ]
    then
      baddir=${false}
    else
      echo "\nThere are no files for fiscal year ${fy}.  Choose another year?"
      yes.or.no 2
      resp=`cat ${pathtemp}/temp.ans`
```

```
      rm ${pathtemp}/temp.ans
      case ${resp} in
        N)
          baddir=${false}
          getgoing=${false}
          ;;
      esac
    fi
  done
#
  while [ ${getgoing} -eq ${true} ]
  do
    clear
    echo "Here are the available pre-set keywords:\n"
    echo "  A) Award,\n  B) Purchases,\n  C) Training,\n  D) Travel."
    echo "\nEnter a letter (A-D) to select a pre-set keyword, or"
    echo "enter E to select your own keyword, or Q to quit this program: \c"
    read resp
    case ${resp} in
      "")
          resp="?"
          ;;
       *)
          resp=`echo "${resp}" | cut -f1 -d" "`
          ;;
    esac
    anscheck.x ${resp} ${pathtemp}/temp.ans "a" "A" "b" "B" "c" "C" "d" "D" \
"e" "E" "q" "Q"
    resp=`cat ${pathtemp}/temp.ans`
    rm ${pathtemp}/temp.ans
    case ${resp} in
      q|Q)
          getgoing=${false}
          ;;
      a|A)
          searchit="Award"
          ;;
      b|B)
          searchit="EP: "
          ;;
      c|C)
          searchit="Training: "
          ;;
      d|D)
          clear
          echo "Here are the types of travel available:\n"
          echo "  A) CONUS,\n  B) Invitational,\n  C) Local,\n  D) OCONUS,"
          echo "  E) Other,\n  F) All of the above.\n"
          echo "Enter the letter for the correct line: \c"
          read travtype
          case ${travtype} in
            "")
              travtype="?"
              ;;
           *)
```

```
                travtype='echo "${travtype}" | cut -f1 -d" "'
                ;;
            esac
            anscheck.x ${travtype} ${pathtemp}/temp.ans "a" "A" "b" "B" "c" "C" \
"d" "D" "e" "E" "f" "F"
            travtype='cat ${pathtemp}/temp.ans'
            rm ${pathtemp}/temp.ans
            case ${travtype} in
              a|A)
                 searchit="TOC "
                 ;;
              b|B)
                 searchit="TOI "
                 ;;
              c|C)
                 searchit="TOL "
                 ;;
              d|D)
                 searchit="TOO "
                 ;;
              e|E)
                 searchit="TOX "
                 ;;
              f|F)
                 searchit="All Travel"
                 ;;
            esac
            ;;
      e|E)
          badstring=${true}
          while [ ${badstring} -eq ${true} ]
          do
            echo "\nEnter the word or phrase to search for; the case of the"
            echo "letters (upper or lower) will be ignored:"
            read searchit
            case ${searchit} in
              "")
                 searchit="nothing"
                 echo "\nWARNING: Entering nothing will terminate the program."
                 ;;
            esac
            echo "\nYou have entered: ${searchit}.  Is this OK?"
            yes.or.no 3
            resp='cat ${pathtemp}/temp.ans'
            rm ${pathtemp}/temp.ans
            case ${resp} in
              Y)
                 badstring=${false}
                 ;;
            esac
            case ${searchit} in
              nothing)
                        getgoing=${false}
                         ;;
            esac
```

```
          done
          ;;
    esac    #End of determination of searchit string.
#
    if [ ${getgoing} -eq ${true} ]
    then
      clear
      echo "This program will search the files for:  ${searchit}.\n"
      echo "The process may take between 1-4 min., depending on the number of"
      echo "files to be searched and on how many users are on the system."
#
      cd account.fy${fy}.d
      for dir in *        #Dir= labor.d or nonlabor.d.
      do
        if [ -d ${dir} ]
        then
          cd ${dir}
#
          for subdir in *    #Subdir= 6.1.d, travel.d, etc.
          do
            if [ -d ${subdir} ]
            then
              cd ${subdir}
#
              for file in *
              do
                if [ -f ${file} ]
                then
                  linenum=`wc -l ${file} | cut -c1-7`
                  if [ ${linenum} -gt 19 ]   #Data has been input to the file.
                  then
                    case ${searchit} in
                      "All Travel")
#
                        for trav in "TOC " "TOI " "TOL " "TOO " "TOX "
                        do
                          case ${dir} in
                            labor.d)
                              grep "${trav}" ${file} | cut -f2,3,4,5,6 -d\| >>\
${pathtemp}/temp.matchfile
                              ;;
                            nonlabor.d)
                              grep "${trav}" ${file} | cut -f2,3,4 -d\| >>\
${pathtemp}/temp.matchfile
                              ;;
                          esac
                        done
                        ;;
#
                      *)
                        case ${dir} in
                          labor.d)
                           grep -i "${searchit}" ${file} |cut -f2,3,4,5,6 -d\| \
>> ${pathtemp}/temp.matchfile
                           ;;
```

```
                        nonlabor.d)
                          grep -i "${searchit}" ${file} | cut -f2,3,4 -d\| >>\
${pathtemp}/temp.matchfile
                          ;;
                      esac
                      ;;
                  esac   #End case searchit.
#
                  if [ -s ${pathtemp}/temp.matchfile ]
                  then                #If the file is generated & has data in
                                      #it, then add data in temp file to file
                                      #holding all matched data:
#
                    if [ ! -f ${pathtemp}/foundfile ]
                    then                              #<-Create file if not exist
                     sed -e '6,$d' ${path1}/nonlabor.form > \
${pathtemp}/foundfile
                    fi
                    echo "\nAccount Directory:  ${subdir}" >> \
${pathtemp}/foundfile
#
#Copy account information to foundfile:
#
                    sed -e '4,$d' ${file} >> ${pathtemp}/foundfile
                    case ${dir} in
                      labor.d)
                        currbal=`tail -1 ${file} | cut -f7 -d\|`
                        ;;
                      nonlabor.d)
                        currbal=`tail -1 ${file} | cut -f5 -d\|`
                        ;;
                    esac
                    echo "Current Balance = $ ${currbal}\n" >> \
${pathtemp}/foundfile
                    case ${dir} in    #Copy col. headings to foundfile.
                      labor.d)
                        tail -4 ${path1}/labor.form |cut -f2,3,4,5,6 -d\| >>\
${pathtemp}/foundfile
                        ;;
                      nonlabor.d)
                        tail -4 ${path1}/nonlabor.form | cut -f2,3,4 -d\| >>\
${pathtemp}/foundfile
                        ;;
                    esac
                    cat ${pathtemp}/temp.matchfile >> ${pathtemp}/foundfile
                    echo "\f\n" >> ${pathtemp}/foundfile
                    rm ${pathtemp}/temp.matchfile
                  else
                    if [ -f ${pathtemp}/temp.matchfile ]
                    then
                      rm ${pathtemp}/temp.matchfile
                    fi
                  fi
                  #
                fi  #End if linenum > 19.
```

```
            #
          fi  #End if file exists.
          #
        done  #End for file in subdir.
        #
        cd ..  #Go back to dir (labor.d, nonlabor.d).
        #
      fi  #End if subdir is a directory.
      #
    done  #End for subdir in dir.
    #
    cd .. #Go back to account.fyxx.d.
    #
  fi  #End if dir exists as a directory.
  #
done #End for dir in account.fyxx.d.
#
cd ${pathtemp}
clear
if [ -f foundfile ]    #If a data file were generated:
then
  echo "Here is the data that matched your search:\n"
  pg foundfile
  echo "\nDo you want to print the results?"
  ${path1}/yes.or.no 4
  resp='cat temp.ans'
  rm temp.ans
  case ${resp} in
    Y)
      wpr foundfile
      ;;
  esac
  echo "\nThe file containing the results is a temporary file."
  echo "Do you want a copy stored on your directory?"
  ${path1}/yes.or.no 5
  resp='cat temp.ans'
  rm temp.ans
  case ${resp} in
    Y)
      badfile=${true}
      while [ ${badfile} -eq ${true} ]
      do
        echo "\nEnter file name: \c"
        read fname
        case ${fname} in
          "")
            echo "\nYou must enter a file name."
            ;;
          *)
            if [ -f $1/${fname} ]
            then
              echo "\nThe file you have entered already exists."
              echo "Select another file name?"
              ${path1}/yes.or.no 6
              ans='cat temp.ans'
              rm temp.ans
```

```
                        case ${ans} in
                          N)
                            badfile=${false}
                            ;;
                        esac
                      else
                        badfile=${false}
                        cat foundfile >> $1/${fname}
                      fi
                      ;;
                  esac
              done
              ;;
        esac
        rm foundfile
#
      else
        echo "The search program could not find any match to:  ${searchit}."
      fi
#
      cd ${path1}
      echo "\nAnother search?"
      yes.or.no 7
      resp=`cat ${pathtemp}/temp.ans`
      rm ${pathtemp}/temp.ans
      case ${resp} in
        N)
          getgoing=${false}
          ;;
      esac
#
    fi  #End if getgoing = true.
#
  done  #End while getgoing = true.
#
fi #End if getgoing = true.
cd ${pathsmall}
```

```
#printit.2 prints the data found in the labor and/or nonlabor accounts,
#and/or the update file sum.info.  It is called by startup, not startup.acc.
#
true=0
false=1
pathsmall='pwd'
smalltemp="${pathsmall}/temp.d"
path1="/usr/OPDAT/account.d"  #Replace with the correct path name.
pathtemp="${path1}/temp.d"
#
clear
echo "Do you want to print files?"
yes.or.no.2 1
resp='cat ${smalltemp}/temp.ans'
rm ${smalltemp}/temp.ans
case ${resp} in
  Y)
    getgoing=${true}
    ;;
  N)
    getgoing=${false}
    ;;
esac
if [ ${getgoing} -eq ${true} ]
then
  cd ${path1}
  baddir=${true}
  while [ ${baddir} -eq ${true} ]
  do
    fy=100
    while [ ${fy} -lt 0 -o ${fy} -gt 99 ]
    do
      echo "\nEnter the last two digits of the fiscal year, or simply"
      echo "press <enter> or <return> if the fiscal year is 93: \c"
      read fy
      case ${fy} in
        "")
           fy=93
           ;;
      esac
    done
    if [ -d account.fy${fy}.d ]
    then
      baddir=${false}
    else
      echo "\nThere are no files for fiscal year ${fy}.  Select another year?"
      yes.or.no 2
      resp='cat ${pathtemp}/temp.ans'
      rm ${pathtemp}/temp.ans
      case ${resp} in
        N)
          baddir=${false}
          getgoing=${false}
          ;;
      esac
```

```
    fi
  done
#
  if [ ${getgoing} -eq ${true} ]
  then
    clear
    echo "\nPrint all labor accounts?"
    yes.or.no 3
    resplab='cat ${pathtemp}/temp.ans'
    rm ${pathtemp}/temp.ans
#
    echo "\nPrint all non-labor accounts?"
    yes.or.no 4
    respnl='cat ${pathtemp}/temp.ans'
    rm ${pathtemp}/temp.ans
#
    echo "\nPrint the file containing the summary of all accounts?"
    yes.or.no 5
    respsum='cat ${pathtemp}/temp.ans'
    rm ${pathtemp}/temp.ans

    case ${resplab} in
      Y)
          echo "\nCreating the labor printout file..."
          cd account.fy${fy}.d/labor.d
          for dir in *
          do
            if [ -d ${dir} ]
            then
              cd ${dir}
              for file in *
              do
                if [ -f ${file} ]
                then
                  echo "Account Directory:  ${dir}" >> ${pathtemp}/temp.prnt.l
                  cat ${file} >> ${pathtemp}/temp.prnt.l
                  echo "\f" >> ${pathtemp}/temp.prnt.l
                fi
              done
              cd ..
            fi
          done
          cd ${path1}
          if [ ! -f ${pathtemp}/temp.prnt.l ]
          then
            echo "\nThis program could not find any labor accounts to print."
          fi
      ;;
    esac
#
    case ${respnl} in
      Y)
          echo "\nCreating the non-labor printout file..."
          cd account.fy${fy}.d/nonlabor.d
          for dir in *
```

```
          do
            if [ -d ${dir} ]
            then
              cd ${dir}
              for file in *
              do
                if [ -f ${file} ]
                then
                  echo "Account Directory:  ${dir}" >> ${pathtemp}/temp.prnt.n
                  cat ${file} >> ${pathtemp}/temp.prnt.n
                  echo "\f" >> ${pathtemp}/temp.prnt.n
                fi
              done
              cd ..
            fi
          done
          cd ${path1}
          if [ ! -f ${pathtemp}/temp.prnt.n ]
          then
           echo "\nThis program could not find any non-labor accounts to print."
          fi
      ;;
    esac
#
    case ${respsum} in
     Y)
      path2=${path1}/account.fy${fy}.d
#
      if [ ! -f ${path2}/sum.info ]    #If the file doesn't exist:
      then
       echo "\nThe summary file does not exist.  Please ask OPDAT to create it."
      else
       echo "\nCreating the summary printout file..."
       cat ${path2}/sum.info > ${pathtemp}/temp.prnt.s
      fi
      ;;
    esac
#
# Print the requested files, if they exist:
#
    if [ -f ${pathtemp}/temp.prnt.? ]
    then
      wpr ${pathtemp}/temp.prnt.?
      rm ${pathtemp}/temp.prnt.?
    fi
#
  echo "\nThis program has ended.  Enter any key to return to the main menu: \c"
  read resp
#
  fi  #end if getgoing
#
  cd ${pathsmall}
#
fi  #end if getgoing
```

```
#startup is an abbreviated program to allow access to users to read the data
#in the account files, and create a keyword search, but will not include any
#ability to write to the data files.
#
usrpath='pwd'
pathsmall="/usr/OPDAT/small.acc.d"  #Replace this with the correct name.
smalltemp="${pathsmall}/temp.d"
#
true=0
false=1
#
cd ${pathsmall}
clear
echo "Do you want to look at and/or print account files?"
yes.or.no.2 1   #This is a shell program to return a yes-or-no answer.
ans='cat ${smalltemp}/temp.ans'
rm ${smalltemp}/temp.ans
case ${ans} in
  Y)
    getgoing=${true}
    ;;
  N)
    getgoing=${false}
    ;;
esac
while [ ${getgoing} -eq ${true} ]
do
   clear
   echo "Here are the available programs:\n"
   echo "  A)  View or print individual labor and non-labor accounts."
   echo "  B)  Search the summary file for operator-specified keywords."
   echo "  C)  Search the account files for operator-specified keywords."
   echo "  D)  Print all labor and/or non-labor files, and/or the summary file."
   echo "\nEnter a letter (A-D) or Q to quit the accounting program: \c"
   read ans
   case ${ans} in
     "")
        ans="?"
        ;;
      *)
        ans='echo "${ans}" | cut -f1 -d" "'  #This insures a 1-word response.
        ;;
   esac
   anscheck.x ${ans} ${smalltemp}/temp.ans "a" "A" "b" "B" "c" "C" "d" "D" \
"q" "Q"
   ans='cat ${smalltemp}/temp.ans'
   rm ${smalltemp}/temp.ans
   case ${ans} in
     a|A)
          edit.data.2
          ;;
     b|B)
          sum.search.2 ${usrpath}
          ;;
     c|C)
```

```
            keyword.2 ${usrpath}
            ;;
        d|D)
            printit.2
            ;;
        q|Q)
            getgoing=${false}
            ;;
    esac
done
cd ${usrpath}
```

```
#Program sum.search.2
#
#Note: This program is to be used with the program startup.  The input
#argument ($1) is used to supply the user's own directory for file copying.
#
true=0
false=1
#
pathsmall='pwd'
smalltmp="${pathsmall}/temp.d"
path1="/usr/OPDAT/account.d"  #Replace with the correct name.
pathtemp="${path1}/temp.d"
#
clear
echo "Do you want to search the summary file for a key word or phrase?"
yes.or.no.2 1
resp='cat ${smalltmp}/temp.ans'
rm ${smalltmp}/temp.ans
case ${resp} in
  Y)
    startup=${true}
    ;;
  N)
    startup=${false}
    ;;
esac
#
if [ ${startup} -eq ${true} ]
then
  cd ${path1}
  badfile=${true}
  while [ ${badfile} -eq ${true} ]
  do
    fy=100
    while [ ${fy} -lt 0 -o ${fy} -gt 99 ]
    do
      echo "\nEnter the last two digits of the fiscal year, or just hit <enter>"
      echo "or <return> if the year is 93: \c"
      read fy
      case ${fy} in
        "")
           fy=93
           ;;
      esac
    done
    if [ -d account.fy${fy}.d ]
    then
      if [ -f account.fy${fy}.d/sum.info ]
      then
        badfile=${false}
      else
        echo "\nThere is no summary file for fiscal year ${fy}.  Please have"
        echo "OPDAT run the summary creation program."
        echo "\nEnter any key to continue: \c"
        read resp
```

```
        badfile=${false}
        startup=${false}
      fi
#
    else
      echo "\nThere are no files for fiscal year ${fy}.  Select another year?"
      yes.or.no 2
      resp=`cat ${pathtemp}/temp.ans`
      rm ${pathtemp}/temp.ans
      case ${resp} in
        N)
           badfile=${false}
           startup=${false}
           echo "\nEnter any key to return to the main menu: \c"
           read resp
           ;;
      esac
    fi
  done
#
 if [ ${startup} -eq ${true} ]
 then
#
  path2=${path1}/account.fy${fy}.d
#
  clear
  echo "This program searches the summary file for key words or phrases, which"
  echo "you enter below.  You can use this program to search for all informa-"
  echo "tion available on a specific account number, account balance, principle"
  echo "investigator, etc."
#
  getgoing=${true}
  while [ ${getgoing} -eq ${true} ]
  do
    echo "\nHere are the first 11 lines of the summary file:\n"
    sed -e '12,$d' ${path2}/sum.info
#
    badinfo=${true}
    while [ ${badinfo} -eq ${true} ]
    do
      echo "\nPlease enter the word or phrase to search for.  The case of the"
      echo "letters will be ignored.:"
      read searchit
      case ${searchit} in
        "")
           searchit="nothing"
           echo "WARNING: Entering nothing will cause the program to terminate."
           ;;
      esac
      echo "\nYou have entered:  ${searchit}.  Is this OK?"
      yes.or.no 3
      resp=`cat ${pathtemp}/temp.ans`
      rm ${pathtemp}/temp.ans
      case ${resp} in
        Y)
```

```
            badinfo=${false}
            case ${searchit} in
              nothing)
                      getgoing=${false}
                      ;;
            esac
            ;;
      esac
    done
#
    if [ ${getgoing} -eq ${true} ]
    then
      grep -i "${searchit}" ${path2}/sum.info > ${pathtemp}/temp.out
      if [ -s ${pathtemp}/temp.out ]
      then
       clear
       echo "Here are the lines that match your search input (${searchit}):\n"
       pg ${pathtemp}/temp.out
       echo "\nPrint these results (p), save data to a permanent file (s),"
       echo "do both (b), or none of the above (n)?  (Enter p, s, b, or n:) \c"
       read resp
       case ${resp} in
         "")
            resp="?"
            ;;
          *)
            resp=`echo "${resp}" | cut -f1 -d" "`
            ;;
       esac
       anscheck.x ${resp} ${pathtemp}/temp.ans "p" "P" "s" "S" "b" "B" "n" "N"
       resp=`cat ${pathtemp}/temp.ans`
       rm ${pathtemp}/temp.ans
       case ${resp} in
         p|P|b|B)
             wpr ${pathtemp}/temp.out
             ;;
       esac
       case ${resp} in
         s|S|b|B)
           badname=${true}
           while [ ${badname} -eq ${true} ]
           do
             echo "\nEnter file name for saving this data (it will be saved to"
             echo "your own directory):"
             read fname
             case ${fname} in
               "")
                  echo "\nYou must enter a file name."
                  ;;
                *)
                  if [ -f $1/${fname} ]
                  then
                    echo "\nThe file you have selected already exists.  Choose \
another file name?"
                    yes.or.no 4
```

```
                    ans='cat ${pathtemp}/temp.ans'
                    rm ${pathtemp}/temp.ans
                    case ${ans} in
                      N)
                        badname=${false}
                        ;;
                    esac
                  else
                    badname=${false}
                    cat ${pathtemp}/temp.out >> $1/${fname}
                  fi
                  ;;
              esac
            done
            ;;
      esac
    else
      echo "\nThere were no lines to match:  ${searchit}."
    fi
    if [ -f ${pathtemp}/temp.out ]
    then
      rm ${pathtemp}/temp.out
    fi
    echo "\nAnother search?"
    yes.or.no 5
    resp='cat ${pathtemp}/temp.ans'
    rm ${pathtemp}/temp.ans
    case ${resp} in
      N)
        getgoing=${false}
        ;;
      Y)
        clear
        ;;
    esac
#
  fi  #End if getgoing= true.
#
 done  #End while getgoing= true.
#
fi  #End if startup= true.
#
fi  #End if startup= true.
cd ${pathsmall}
```

```
#yes.or.no.2 returns a yes-or-no answer to the calling program.  It takes one
#argument:  $1= the number of times the function has been called within a given
#program.  This program is used by those in the small.acc.d directory and by
#startup (not startup.acc).
#
true=0
false=1
#
pathstart='pwd'
cd temp.d
#
if [ $1 -eq 1 ]
then
  echo "Enter y or n (for yes or no): \c"
else
  echo "Enter y or n: \c"
fi
read resp
checkit=${true}
while [ ${checkit} -eq ${true} ]
do
  case ${resp} in
    Y*|y*)
          echo "Y" > temp.ans
          checkit=${false}
          ;;
    N*|n*)
          echo "N" > temp.ans
          checkit=${false}
          ;;
        *)
          echo "\nPlease enter one of the following: y Y n N."
          read resp
          ;;
  esac
done
cd ${pathstart}
```

U.S. Army Materials Technology Laboratory
Watertown, Massachusetts 02172-0001
AN INTERACTIVE OFFICE ACCOUNTING PROGRAM
WRITTEN IN UNIX AND C -
Karen M. Kinsley

AD ____________

UNCLASSIFIED
UNLIMITED DISTRIBUTION

Key Words

Monograph Series MTL MS 92-2, July 1992, 234 pp -

Computer programs
Accounting
UNIX language

This program is written in UNIX (Version 5.1, P01) and C (pre-ANSI) and provides an office with the ability to create and maintain labor and non-labor accounts without requiring anyone in the office (except for the installer) to learn a computer language. Security for the accounts and account directories is automatically provided, and menus are used to create the accounts and to specify the charges to them. The program also includes creation of a summary file for all accounts, key word search functions for both the summary file and general accounts, and a printout program. A separate program, which allows an operator to view, print, and/or search the accounts for key words, but which will not allow the addition of charges to the accounts, is also provided.

U.S. Army Materials Technology Laboratory
Watertown, Massachusetts 02172-0001
AN INTERACTIVE OFFICE ACCOUNTING PROGRAM
WRITTEN IN UNIX AND C -
Karen M. Kinsley

AD ____________

UNCLASSIFIED
UNLIMITED DISTRIBUTION

Key Words

Monograph Series MTL MS 92-2, July 1992, 234 pp -

Computer programs
Accounting
UNIX language

This program is written in UNIX (Version 5.1, P01) and C (pre-ANSI) and provides an office with the ability to create and maintain labor and non-labor accounts without requiring anyone in the office (except for the installer) to learn a computer language. Security for the accounts and account directories is automatically provided, and menus are used to create the accounts and to specify the charges to them. The program also includes creation of a summary file for all accounts, key word search functions for both the summary file and general accounts, and a printout program. A separate program, which allows an operator to view, print, and/or search the accounts for key words, but which will not allow the addition of charges to the accounts, is also provided.

U.S. Army Materials Technology Laboratory
Watertown, Massachusetts 02172-0001
AN INTERACTIVE OFFICE ACCOUNTING PROGRAM
WRITTEN IN UNIX AND C -
Karen M. Kinsley

AD ____________

UNCLASSIFIED
UNLIMITED DISTRIBUTION

Key Words

Monograph Series MTL MS 92-2, July 1992, 234 pp -

Computer programs
Accounting
UNIX language

This program is written in UNIX (Version 5.1, P01) and C (pre-ANSI) and provides an office with the ability to create and maintain labor and non-labor accounts without requiring anyone in the office (except for the installer) to learn a computer language. Security for the accounts and account directories is automatically provided, and menus are used to create the accounts and to specify the charges to them. The program also includes creation of a summary file for all accounts, key word search functions for both the summary file and general accounts, and a printout program. A separate program, which allows an operator to view, print, and/or search the accounts for key words, but which will not allow the addition of charges to the accounts, is also provided.

U.S. Army Materials Technology Laboratory
Watertown, Massachusetts 02172-0001
AN INTERACTIVE OFFICE ACCOUNTING PROGRAM
WRITTEN IN UNIX AND C -
Karen M. Kinsley

AD ____________

UNCLASSIFIED
UNLIMITED DISTRIBUTION

Key Words

Monograph Series MTL MS 92-2, July 1992, 234 pp -

Computer programs
Accounting
UNIX language

This program is written in UNIX (Version 5.1, P01) and C (pre-ANSI) and provides an office with the ability to create and maintain labor and non-labor accounts without requiring anyone in the office (except for the installer) to learn a computer language. Security for the accounts and account directories is automatically provided, and menus are used to create the accounts and to specify the charges to them. The program also includes creation of a summary file for all accounts, key word search functions for both the summary file and general accounts, and a printout program. A separate program, which allows an operator to view, print, and/or search the accounts for key words, but which will not allow the addition of charges to the accounts, is also provided.