JANUARY/FEBRUARY, 1980. ISSUE 2. $2.00

COMPUTE.

The Journal for Progressive Computing™

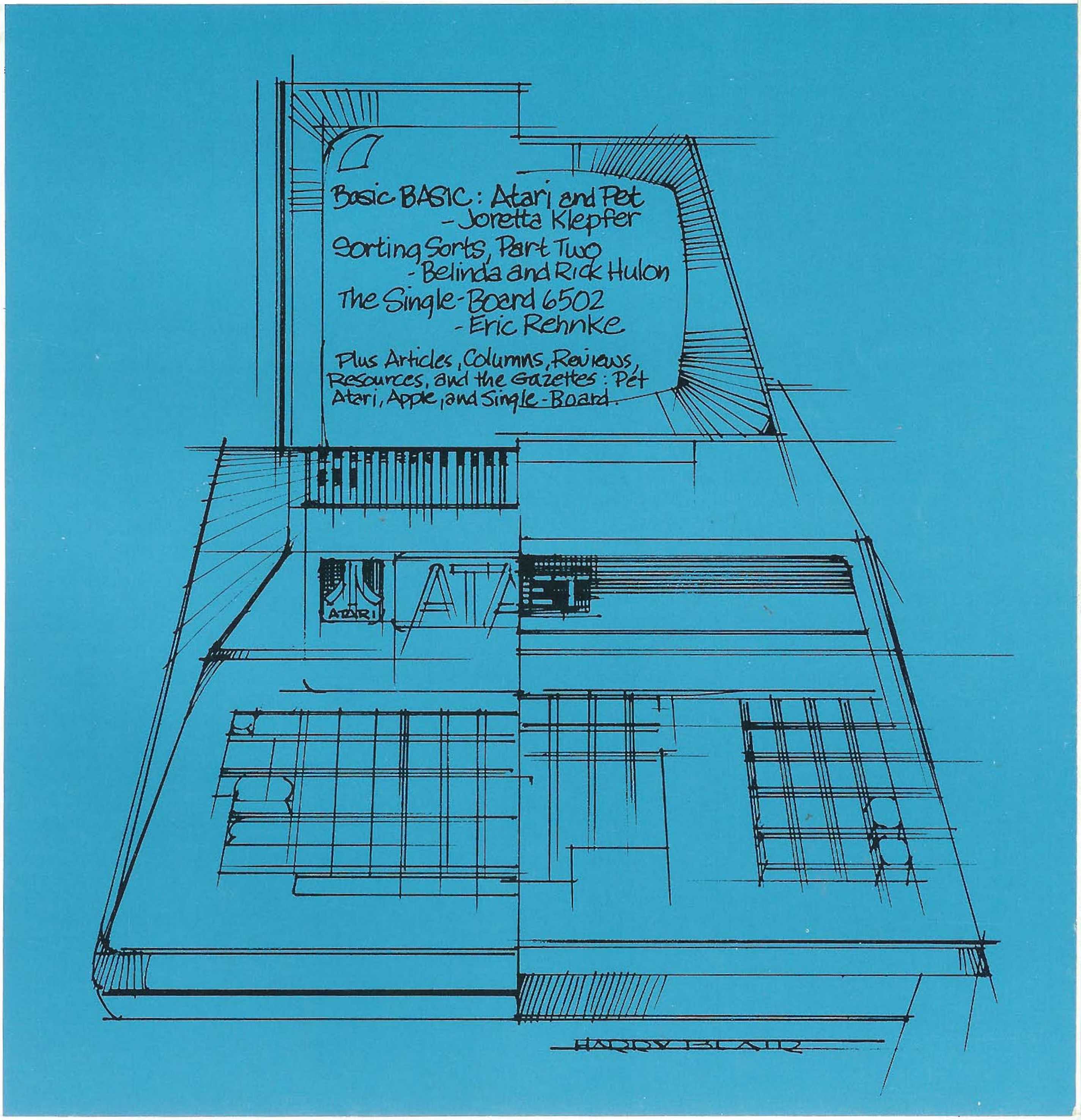
Basic BASIC: Atari and Pet
- Joretta Klepfer
Sorting Sorts, Part Two
- Belinda and Rick Hulon
The Single-Board 6502
- Eric Rehnke
Plus Articles, Columns, Reviews, Resources, and the Gazettes: Pet Atari, Apple, and Single-Board.
ATARI
ATA
HARRY BLAIR

Table of Contents

Staff of COMPUTE:
Robert C. Lock, Editor/Publisher
Carol Holmquist Lock, Circulation Manager.
Larry Isaacs, Software/Hardware Lab.
Joretta Klepfer, Editorial Assistant.
Marlene Pratto, Educational Assistant.

COMPUTE. receives continuing assistance from the following persons:
Harvey B. Herman, University of North Carolina at Greensboro, Dept. of Chemistry. Editorial Assistance.
David Gibbs, The Design Group, Greensboro, N.C., Art Direction/Design Consultation.
Jim Butterfield, Toronto. Editorial Assistance.

The following writers contribute on a regular basis as Associate Editors:
Gene Beals, 115 E. Stump Road, Montgomeryville, Pennsylvania 18936.
Len Lindsay, 1929 Northport Drive #6, Madison, Wisconsin 53704.
Roy O'Brien, Post Office Box 426, Beaumont, California 92223.
Eric Rehnke, # 61, 540 S. Ranchview Circle, Anaheim Hills, California 92807.
COMPUTE. is published by Small System Services, Inc., 900-902 Spring Garden Street, Greensboro, North Carolina 27403. Telephone: (919) 272-4867.
PET is a trademark of Commodore Business Machines, Inc.
Apple is a trademark of Apple Computer Company.
Atari is a trademark of Atari, Inc.

Authors of manuscripts warrant that all materials submitted to COMPUTE. are original materials with full ownership rights resident in said authors By submitting articles to COMPUTE. authors acknowledge that such materials, upon acceptance for publication, become the exclusive property of Small System Services, Inc. Unsolicited manuscripts not accepted for publication by COMPUTE. will be returned if author provides a self-addressed, stamped envelope. Program listings should be provided in printed form as well as machine readable form. Articles should be furnished as typed copy with double spacing. Each page of your article should bear the title of the article, date and name of the author.

Address all manuscripts and correspondence to COMPUTE. Post Office Box 5119, Greensboro, N.C. 27403. Materials (advertising art work, hardware, etc.) should be addressed to COMPUTE., 900 Spring Garden Street, Greensboro, N.C. 27403.

ISSN 0194-357X.

COMPUTE. assumes no liability for errors in articles or advertisements. Opinions expressed by authors are not necessarily those of COMPUTE.

ADVERTISERS INDEX

Aardvark	114
AB Computers	95
Abacus Software	38
Andromeda	69
Applesced	68
Automated Simulations	91
Harry Briley	90
CGRS	16
CMS Software Systems	28,53
Cognitive Products	38
Commodore Business Machines, Inc.	BC
Competitive Software	60
COMPUTE's Book Corner	97
Computer Associates	58
Computer Power for Business	31
Computer World	57,67
Computhink	13
Connecticut Microcomputing	20,27,84,85
Creative Computing Software	56
Cursor	79
Dr. Daley	17,64
D.T.I. Office Products	25
Eastern House Software	38,61,64,92,101
Electronic Specialists	114,88
Excert	115
H. Geller Computer Systems	22
Hayden	9
Hudson Digital Electronics	37,59,105
Innovision	4
Instant Software	50-51
Iridis	73
International Technical Systems	61,88
Kobetek Systems	101
Micro Ed	45,54
Microphys	46
Micro Computer Industries	25
Micro Technology Unlimited	10,63,IBC
NCE/Compumart	21
NEECO	14
Optimal Technology	108
Optimized Data Systems	61
Osborne & Associates	2
Peninsula Computer Project	48
Pet Shack	88
Programma	83
Recreational Computing	6
Eric C. Rehnke	113
RNB Enterprises	52,107
Sawyer Software	62
Seawell Marketing	109
Skyles Electric Works	I-1,19,55,98,99
Softside Software	IFC
Total Information Services	47
Xymec	35
Ziegler Electronic Products	86

Subscription Information:
U.S. $9.00/6 issue year
CANADA $12.00/6 issue year (U.S. funds)
Europe: Air Subscription $22.50 U.S./6 issue year
Other areas. Inquire for air rates.

In the United Kingdom and Europe, new subscribers and dealers should contact:

L.P. Enterprises
8-11 Cambridge House
Cambridge Road
Barking, Essex
England IG1 18NT
(United Kingdom and Northern Europe)

Ing. W. Hofacker GMBH
8 Munchen 75
Postfach 437
West Germany
(Germany, Switzerland, and Austria)

Editor's Note: We're happy to welcome Bob Baker to the pages of COMPUTE. Bob is the new author of Kilobaud Microcomputing's PET-Pourri Column. He replaced Len Lindsay who's quite busy these days with various writing tasks.

AN EASIER METHOD OF SAVING DATA PLUS HomeAccounting

Robert W Baker, 15 Windsor Drive, Atco, NJ 08004

Whenever a program must save specific data for the next time it is run, the data is normally savėd in a tape data file. This requires inserting a tape with the data file and reading the previous values every time the program is run. When the program is done, the tape must be rewound or changed and another data file written to save the new data. This procedure can waste a good deal of time, especially if only a small amount of data is needed and the program is normally run quite frequently. It can take as long as 10 to 15 seconds on the PET just to find the data file and read the header record before actually reading any data. In addition, you now have a tape for the program and another tape for the data file. If you only have several values to save, using data files is awkward, cumbersome, and not worth the trouble. Several applications that could be done very easily on a computer, in reality become useless when requiring data files.

Another possible method of saving data is to change the BASIC pointers and save the data along with the program on tape. The next time the program is loaded, the BASIC pointers are then reset and a GOTO xxxxx command is used to execute the program. If a RUN command is used instead of a GOTO, the data is lost and the program must be re-loaded. This method is too complicated and requires a number of functions the user must perform each time they save or load the program and data. It also runs the risk of permanently losing the data.

There is a way, however, that a program can save small amounts of data within the program itself using a very simple procedure. The basic theory is to include DATA statements in the program with initial data specified for the first time the program is run. The DATA statements and their associated data define space within the program for the data that is to be saved after each time the program is run. Before terminating, the program simply POKE's the new values to be saved back into the DATA statement(s) to replace the original data. The program itself is then saved after each execution and the latest data is automatically included without any special actions by the user. Whenever the program is loaded, the previous data is readily available using the standard READ command of BASIC. Saving data using this method is extremely simple, but it does require knowing the format of BASIC lines stored in memory. The necessary information on the PET has been described in various newsletters and several magazine articles, so it will not be repeated here. This article was written for the PET primarily, but the technique will work for other machines as well.

The listing for a Home Budget program that I've been using for several months on my 8K PET is included to help illustrate this simple data saving technique. Looking at the start of the program, lines 10 and 20 contain DATA statements to reserve space for 12 numeric values to be saved after each time the program is run. The DATA statements are located at the very beginning of the program, making it easier to know where to do the POKE's. Each value saved can be up to 6 digits in length, since this is the length of each field specified in the original DATA statements. Disregarding how the program actually works for now, the data from lines 10 and 20 would normally be read into elements of array "M" by lines 510 and 520. When the program is done, the value of a single element of M or M(x) is converted to a 6 character string with leading zeros blanked as spaces in line 1010. This insures that all 6 characters of each field in the data statements are changed every time the new data is saved in the program. The ASCII value of each character in the string representation of M(x) is then poked into a DATA statement by line 1030. This loop is repeated for all 12 values and a reminder is then printed so the user will not forget to save the program with the updated data. The program could have even printed the actual SAVE command for the user if desired. I intentionally left this out, however, in case the user decided the new data was not correct and wanted to re-run the program without saving the updated values.

If you should use this technique in your own program, don't forget it can be used to save strings or numbers. Be careful you don't destroy the DATA statement itself or the separating commas when poking characters into a DATA statement. Also, don't forget to step over the end-of-line flag, the 2-byte link, the 2-byte line number, and the 1-byte DATA statement "token" when more than one line is used to save data in the program. Each field definition should reserve enough space for the maximum length expected to be encountered by the program. Numeric values must be converted to strings before being saved. Quotes should be used at the beginning and end of each field when saving text strings. Don't forget to step past the quotes when poking the strings into the data statements. Strings should be changed to the length of the field being poked into by appending spaces as was done in the example with the numeric values after converting them to text strings. This will insure the entire field is updated each time the program is run and the correct data is always saved. The DATA statements must remain at a constant location in memory. Being at the beginning of the program avoids problems with changing locations caused by editing the program before the DATA statements. If the DATA statements are moved, the address used for the POKEs must be changed accordingly.

Home Accounting

I don't claim to be an accounting expert but the Home Budget program works and serves a very useful purpose for me. It is based on an original budget system I devised that used an accounting book to record all income and expenditures. Various "accounts"within the budget help allocate what money from each paycheck is to be reserved for which bills in order to meet the projected expenses. Accounts for bills that are paid at least once a month are kept in the family checking account where they are readily available. Accounts for all other bills, paid at longer intervals, are normally kept in the savings account until needed. An account is established for each major expenditure, such as: insurances, home mortgage, utilities, telephone, auto loan, auto expenses, charge accounts, Christmas presents, vacation, etc. All smaller expenses are grouped into a miscellaneous account that is kept in the checking account. An additional account is reserved in the savings account for all "excess" funds, as the true "savings" total.

This simple BASIC program provides all the desired functions to keep an accurate home budget with a minimum of effort. It does not have any fancy features, instead it provides the necessary information in an easy to use format. It displays each account total along with the current checking and savings balances for fast and easy verification. Each transaction is entered by selecting the appropriate account number and the value to credit (+) or debit(-) the specified account. Positive values indicate deposits (credit) and negative values indicate expenses or bills paid (debit). The actual transactions are not recorded, only the running totals for each account are retained to keep the amount of saved data at a minimum. An additional feature of this program is the ability to set the amount to be credited to each account for a paycheck deposit. Thus, come payday, you simply enter the amounts deposited to the checking and savings account and the program does the rest. An account total can become negative if expenses exceed current funds allocated for that expense. This effectively indicates "borrowing" money from other accounts and should be corrected by transferring money from another account or changing the pay deposit value for the account. A negative checking or savings balance should be avoided as this indicates a very serious problem such as an overdrawn checking account. The first step in setting up the budget is to decide what accounts are needed and how many will be in checking or savings. In line 500 of the program, the variable "C" is defined as the number of budget accounts in checking (7), and "S" is defined as the number of accounts in savings (5). The variable "A" is computed as the total number of budget accounts (C + S = 12), and the money (M) and name (N$) arrays are dimensioned in the same line.

Since we are going to save the data within the program, we must define storage for the values in data statements. Line 10 contains the initial values for the checking accounts and line 20 is for the savings accounts. Separate DATA statements were used for checking and savings to allow easy addition or deletion of accounts as required. All values will be kept as whole numbers by multiplying each value by 100. This will help avoid decimal points and problems associated with fractions, besides making the data easier to save using pokes. With 6 digits per field, the limiting values for any account value are: -999.99 to +9999.99 since the minus sign takes up one digit space for negative numbers.

The actual account names are stored in lines 100-210. Lines 100 and 110 are for the checking accounts while lines 200 and 210 define the savings account names. Each name should be limited to 28 chracters for the program to function properly. In addition, the last checking account must be the MISC account and the last savings account must be the excess SAVINGS account. The amount to be deposited from a pay to each checking account is specified in line 300 with a zero value shown for the MISC account, the last value. This account automatically gets any remainder from the pay deposit after all the required checking account deposits are made. If the pay deposit is not large enough to meet the required checking budget total, the difference is subtracted from the MISC account. Line 400 contains the corresponding savings pay deposit values, with a zero value for the excess SAVINGS account, the last value. This account acts just as the MISC account does for the checking account. Any savings pay deposit excess/shortage is added/subtracted to this account.

To customize the program for your own use, simply set the correct values of C and S in line 500. Then add or delete the required DATA fields in lines 10 and 20, and the account names in lines 100-210. Change any account names as required but keep each to a maximum of 28 characters. Set the PAY deposit values for each account in lines 300 and 400 by taking into consideration the related expenses and frequency of payment. Remember to keep the MISC and SAVINGS accounts as the last accounts in the checking and savings, with zero pay deposit values for each. That should be all the changes required to convert the program for your own situation. Individual accounts can be added or deleted at any time by similar changes. Don't forget to set an account value to zero by transferring any money to other accounts before deleting the account. This will keep your checking and savings balances correct.

The program listing contains a number of REM lines to help document the program. If you should decide to use the example program, please don't bother entering these lines. They'll only make the program loading and saving much longer, since the program will be about 3 times larger than needed. This is exactly what was tried to avoid by saving the data within the program to minimize tape useage. Once typed in, a few minutes experimenting with the program should clearly indicate how it works. Enter a few transactions, then type D and list lines 10 and 20 to see what was saved within the program. If you have any problems, check for extra spaces in lines 10 and 20 or check the POKE address in line 1010.

```
10 DATAØØØØØØ,ØØØØØØ,ØØØØØØ,ØØØØØØ,ØØØØØØ,
      ØØØØØØ,ØØØØØØ
20 DATAØØØØØØ,ØØØØØØ,ØØØØØØ,ØØØØØØ,ØØØØØØ
3Ø :
31 REM *******************************
32 REM *    HOME BUDGET PROGRAM      *
33 REM *-----------------------------*
34 REM *    BY: ROBERT W. BAKER      *
35 REM *        15 WINDSOR DRIVE     *
36 REM *        ATCO, NJ Ø8ØØ4       *
39 REM *******************************
5Ø :
51 REM ===============================
52 REM DATA STATEMENTS TO SAVE VALUES
53 REM MUST BE THE FIRST STATEMENTS
54 REM IN THE PROGRAM TO MAKE THEM
55 REM EASY TO FIND.
56 REM ACCOUNT DESCRIPTIONS FOLLOW -
57 REM ===============================
58 :
1ØØ DATA "CHARGES"
1Ø2 DATA "GAS & AUTO EXPENSES"
1Ø5 DATA "MORTGAGE"
11Ø DATA "TELEPHONE", "UTILITIES"
115 DATA "AUTO LOAN","MISC"
2ØØ DATA "AUTO INSURANCE"
2Ø5 DATA "HOMEOWNERS INSURANCE"
21Ø DATA "LIFE INSURANCE", "CHRISTMAS"
215 DATA "SAVINGS"
25Ø :
251 REM ==============================
252 REM FOLLOWING DATA STATEMENT
253 REM CONTAINS STANDARD DEPOSIT
254 REM VALUES FOR PAY DEPOSIT.
255 REM ==============================
256 :
3ØØ DATA 25,4Ø,15Ø,1Ø,5Ø,45,Ø
4ØØ DATA 3Ø,7,25,15,Ø
45Ø :
451 REM ==============================
452 REM MAJOR VARIABLE DEFINITIONS:
453 REM  C  = # OF ITEMS IN CHECKING
454 REM  S  = # OF ITEMS IN SAVINGS
455 REM  A  = TOTAL NUMBER OF 'ACCTS'
456 REM  CB = CHECKING BALANCE
457 REM  SD = TOTAL SAVINGS DEPOSIT
458 REM  M(.)  = CURRENT ACCT VALUES
459 REM  N$(.) = ACCT NAMES FROM DATA
47Ø REM ==============================
471 REM READ VALUES FROM DATA
472 REM STATEMENTS TO INITIALIZE.
479 REM ==============================
48Ø :
5ØØ C=7:S=5:A=C+S:DIM M(A),N$(A)
5Ø5 CB=Ø:SD=Ø
51Ø FOR X=1 TO C:READ M(X):CB=CB+M(X)
515 NEXT
52Ø FOR X=C+1 TO A:READ M(X)
525 SD=SD+M(X):NEXT
54Ø FOR X=1 TO A:READ N$(X):NEXT
55Ø L$="........................... $"
59Ø :
591 REM =============================
592 REM DISPLAY ACCT #, NAME, & VALUE
593 REM ALONG WITH CHECKING/SAVINGS
594 REM TOTALS, THEN PROMPT FOR INPUT
595 REM =============================
596 :
6ØØ PRINT"[CLR]";:C2=Ø:S2=Ø
6Ø5 FOR X=1 TO C:V=M(X):GOSUB 9ØØ:NEXT
6Ø6 PRINT TAB(3Ø)"["DDDDDDDD"]"
61Ø PRINT"   [RV]TOTAL CHECKING ";
611 PRINT"BALANCE[RVOFF]..... $";
615 V=CB:GOSUB 91Ø:PRINT
62Ø FOR X=C+1 TO A:V=M(X):GOSUB 9ØØ
625 NEXT: PRINT TAB(3Ø)"["DDDDDDDD"]"
63Ø PRINT"   [RV]TOTAL SAVINGS ";
631 PRINT"ON DEPOSIT[RVOFF]... $";
635 V=SD: GOSUB 91Ø
636 FOR X=1 TO 39
637 PRINT"["$",LC,DN,"E",UP]";:NEXT
64Ø PRINT:PRINT"[DN]^"
641 :
642 REM ==============================
643 REM GET USER INPUT & CHECK FOR
644 REM VALID INPUT -
645 REM ==============================
646 :
65Ø PRINT"["-@@"] ACCT#, [RV]P[RVOFF]";
651 INPUT"AY, OR, [RV]D[RVOFF]ONE";A$
66Ø X=VAL(A$):IF X>Ø AND X<=A THEN 8ØØ
```

```
664 :
665 REM ==============================
666 REM ******   CREDIT PAY   ********
667 REM ==============================
668 :
67Ø IFLEFT$(A$,1)<>"P"OR(C1<>Ø)THEN1ØØØ
671 :
672 REM =============================
673 REM GET CHECKING/SAVINGS DEPOSITS
674 REM TO CREDIT STANDARD PAY.
675 REM CAN ONLY USE ONCE PER RUN!
676 REM =============================
677 :
68Ø INPUT"    CHECKING DEPOSIT";C1
681 INPUT"    SAVINGS  DEPOSIT";S1
685 C1=INT(1ØØ*(C1+.ØØ1))
686 S1=INT(1ØØ*(S1+.ØØ1))
69Ø FOR X=1 TO A:READ V:V=V*1ØØ
695 IF X=C THEN V=C1-C2
7ØØ IF X=A THEN V=S1-S2
71Ø GOSUB 95Ø:NEXT:GOTO 6ØØ
75Ø :
751 REM ==============================
752 REM *****  CREDIT/DEBIT ACCT *****
753 REM ==============================
754 :
8ØØ PRINT"[DN]AMT TO CREDIT(+)";
8Ø5 INPUT" / DEBIT(-)";V
85Ø V=INT(1ØØ*(V+.ØØ1)):GOSUB 95Ø
855 GOTO6ØØ
89Ø :
891 REM =============================
892 REM SUBROUTINE TO GENERATE SINGLE
893 REM LINE OF DISPLAY WITH -
894 REM ACCT #, NAME, AND VALUE IN
895 REM STANDARD FORMAT.
896 REM =============================
897 :
9ØØ PRINT RIGHT$(STR$(X+1ØØ),2);" ";
9Ø1 PRINTN$(X);RIGHT$(L$,29-LEN(N$(X)));
9Ø5 T$=STR$(INT(ABS(V)/1ØØ)*SGN(V))
91Ø PRINT RIGHT$("    "+T$,4);
915 IFV<ØANDV>-1ØØTHENPRINT"[2 LC]-Ø";
92Ø PRINT".";RIGHT$(STR$(ABS(V)+1ØØ),2)
925 RETURN
94Ø :
941 REM ==============================
942 REM SUBROUTINE TO CREDIT/DEBIT
943 REM ACCT & UPDATE SAVINGS/CHECKING
944 REM TOTALS.
945 REM ==============================
946 :
95Ø M(X)=M(X)+V
955 IF X>C THEN SD=SD+V:S2=S2+V:RETURN
96Ø CB=CB+V:C2=C2+V:RETURN
99Ø :
991 REM ==============================
992 REM ******   CHECK IF DONE   *****
993 REM ==============================
994 :
1ØØØ IF LEFT$(A$,1) <> "D" THEN 6ØØ
1ØØ1 :
1ØØ2 REM ============================
1ØØ3 REM AT END OF PROGRAM SAVE DATA
1ØØ4 REM BACK INTO THE PROGRAM, THEN
1ØØ5 REM PRINT REMINDER TO SAVE PGM
1ØØ6 REM ============================
1ØØ7 :
1Ø1Ø N=1Ø3Ø:FOR X=1 TO A
1Ø15 L$=RIGHT$("      "+STR$(M(X)),6)
1Ø3Ø FOR Y=1 TO 6
1Ø35 POKE N,ASC(MID$(L$,Y,1)):N=N+1
1Ø4Ø NEXT:N=N+1:IF X=C THEN N=N+5
2ØØØ NEXT
2ØØ5 PRINT"[CLR]REWIND TAPE AND SAVE "
2ØØ6 PRINT"THE PROGRAM"
2Ø1Ø PRINT"[DN]TO RETAIN THE NEW DATA!"
2Ø11 PRINT"[2 DN]"
2Ø15 END
2Ø2Ø REM
2Ø3Ø REM CURSOR POSITIONING AND GRAPHICS
2Ø4Ø REM CHARACTERS ARE ENCLOSED WITHIN
2Ø5Ø REM BRACKETS. THE GRAPHIC CHARS.
2Ø6Ø REM ARE SHOWN AS UNSHIFTED CHARS.
2Ø7Ø REM BETWEEN QUOTES. THE CURSOR
2Ø8Ø REM CONTROL CHARACTERS ARE
2Ø9Ø REM INDICATED AS FOLLOWS:
21ØØ REM SP=SPACE; LC=LEFT CURSOR
211Ø REM UP=UP CURSOR; DN=DOWN CURSOR
212Ø REM CLR=CLEAR SCREEN; RV=REVERSE
213Ø REM RVOFF=REVERSE OFF
```