CSUB$^{(R)}$

Release VI

Common **SUB**routines

developed for use with **baZic**$^{(R)}$

a high-level computer language

and the

CP/M,$^{(R)}$

NorthStar^{TM} **DOS**

and

MicroDoZ$^{(R)}$

Operating Systems

Revisions Ø1/Ø3
June 8, 1981

developed by

Micro Mike's, Inc.
3Ø15 Plains Blvd.
Amarillo, TX 791Ø2 USA

telephone: 8Ø6-372-3633

making technology uncomplicated ... for People$^{(SM)}$

NOTICE

Before CSUB or any of the programs that execute under CSUB can be run, **MicroDoZbaZic** (or **baZic** for the DOS and CP/M versions of CSUB) must be configured for your CRT. The program CONFIG (or CRT for the DOS and CP/M versions) is used for this purpose. CONFIG is automatically chained to the first time the CSUB disk is booted. See Section 9.5 for details on using the CONFIG program. CRT must be LOADed and RUN to configure the CRT for the DOS and CP/M versions of CSUB.

After this page and the preceding page have been read, they may be moved to the back of this manual to make the table of contents more accessable. This manual is designed so that information can be easily found by using the table of contents.

TABLE OF CONTENTS

INTRODUCTION

NOTE: This program package requires Micro Mike's, Incorporated **uDoZbaZic** computer language to operate as described in this manual. Specifically, Sections 9.5 (CONFIG) and 9.6 (DUP) do not operate under the DOS and CP/M versions of CSUB.

The purpose of this programming package is to allow easy, rapid creation of application software and data bases that run under Micro Mike's, Incorporated **uDoZ** and **baZic.** CSUB establishes a Common set of SUBroutines that eliminate a programmer's need to "re-invent the wheel" for every program written. All functions generally required in an application program are contained in the Common SUBroutine package.

CSUB requires a 24 line by 80 column addressable cursor CRT. We have found that the use of the addressable cursor is the most efficient method of data entry, particularly for inexperienced or first time computer users.

baZic differs from many other BASICs in that string length is limited only by available memory. Most other BASICs allow string arrays with a maximum of 255 characters in a string, but in **baZic,** strings can be as long as available memory and string arrays are simulated by **baZic** programs. This feature allows "string arrays" to be created with string variables that are different lengths.

Most BASIC programmers use a different string variable for each field they need to store. As an example, if they want to store the name, address, and city/state, they will use three strings. One string would be used for the name, one for the address, and one for the city/state.

This method of string use is very ineffecient, since **baZic** allows virtually unlimited string length. CSUB encourages the programmer to use one string for all string information. CSUB supports this feature by keeping track of the position in the string of all information that is stored in the string.

CSUB takes advantage of the string capabilities of **baZic** in several important ways. All string information (variables) in application programs written under CSUB are stored in a single string (B$). This gives the programmer rapid random access to any portion of the string, resulting in quick and easy accessing and printing of string information.

The programmer merely references the part of B$ that contains the string information to be dealt with (B$(1Ø1,11Ø)). This means many separate strings are stored in one string and individual strings can be variable in length.

For storing strings on disk, **baZic** requires 2 BYTES of overhead for each string stored that is less than 255 characters, and 3 BYTES for each string more than 255. By storing all string information in one string, a minimum of disk space is used.

In the same way that all string information is stored in one string, all numeric information is stored in one numeric variable (B(SUBS)) array. Fields 6 and 7 of the DATA statements table for strings, and field 8 for all other variables, represent the location in the string or array where the particular variable is located. The DATA statements table will be discussed in Sections 5 or 6, but generally it uniquely defines each variable which is used for input and/or display on the CRT by the parameters that are set in the DATA statement for each variable.

CSUB Input routines are accessed by calling the appropriate function. Each of the two input functions (FNE and FNF) act by first RESTOREing to the appropriate set of data statements and then taking the input based upon the parameters that are defined in the DATA table. Each line of data statements completely and uniquely defines all parameters associated with every variable input from or displayed on the CRT.

Virtually any application program needs a data base for storage of information. Therefore, the first program needed by an application programmer is one that allows the entering of the data base. One program that accomplishes this task is the demonstration file maintenance program (F/M) "BONES" provided on the CSUB disk.

CSUB supports three file accessing functions: FNG, FNG1, and FNH. FNG is the sequential and random file function. It works on a structured type file specifically set up for FNG. It can read or write the specified file in a random mode and can write sequentially to FNG files.

FNG1 is used in conjunction with FNG for special cases, usually a sequential read where the file channel must remain open. Normally FNG OPENs and CLOSEs the file every time a function call is made. FNG1 allows the programmer to specify a channel number and keep it open for the duration of the FNG function calls.

FNH is the keyed file accessing function. FNH allows indexes to the main file (FNG file) to be created, deleted, renamed, or located for use in conjunction with FNG. Multiple indexes are supported, although these involve special programming considerations.

The basic functions of the file maintenance program (BONES) are: (1) allow new records to be placed into the data base; (2) allow all fields within every record to be viewed, changed, and copied back into the file; (3) allow old records to be deleted (4) allow the index to be renamed, and (5) allow one or all records to be printed on the print device.

Use of a table-driven package for programming offers numerous advantages over programming in **baZic** by itself. The routines of CSUB define standards for both the programmer and the operator. The operator, after becoming familiar with the operation of the subroutines, learns standard formats and operating procedures which follow through each and every program, reducing training time and operator fatigue.

The programmer has similar benefits. Program creation is standardized by CSUB, allowing the programmer to concentrate on how the data base is to be manipulated, rather than writing seemingly endless input and file accessing statements. CSUB's tested routines are very dependable, reducing program errors to only the specific lines added for any particular program.

Typically only one line of **baZic** code will accomplish a complete input with full parameter checking and display formatting. Also one line of **baZic** code can result in a complete disk access, including all error trapping.

CSUB makes BASIC a much more structured language, resulting in easier program construction. Programs become more understandable among different programmers, allowing one person to more quickly and easily understand another's program. Fewer errors result, programs are more easily modified and program construction is simplified. If the programmer uses the CSUB worksheets and appropriate REM statements, the program is documented quickly and easily.

Section 8 (WRITING CSUB PROGRAMS) will provide more information on the actual writing of programs under CSUB.

NOTE: **uDoZbaZic** (or **baZic** for the DOS and CP/M versions of CSUB) must be modified for cursor addressing before CSUB can be used. This procedure is accomplished by the **baZic** program CONFIG for the **uDoZ** version and CRT for the DOS and CP/M versions of CSUB provided on this disk. See Section 9.5 for details on using the program CONFIG and the **baZic** manual for the program CRT.

FUNCTION DESCRIPTIONS

To better understand the functions of CSUB, they are broken into several catagories. The first catagory provides functions for controlling the passage of information between the CRT and internal memory (specifically B$ for alpha-numeric information and B array for numeric information).

The second class of functions operates mostly in the internal memory of the computer. These routines generally move or change information directly in RAM operating on B$ and B array (B()).

The third class of functions control the passage of information between the disk storage devices and internal memory (again B$ and B()).

CSUB should be used to control the passage of information in virtually all of the three situations mentioned. To "picture" the relationship more easily the following "chart" is presented:

```
              CRT
               |
       FNC     |
       FNC1    |
       FND     |
       FNE     |    Passes information between CRT and B$,B()
       FNF     |
       FNM     |
               |
               |
       FND$    |    B$(__________), B(__________)
       FNL$    |    Acts internally on B$ and B()
               |
               |
       FNG     |
       FNG1    |    Passes information between files and B$,B()
       FNH     |
               |
             FILES
```

2.1 CRT Functions

The following functions are all associated with the CRT and its displays. The functions handle all cursor addressing, displaying of variables, and input of variables, including formatting and parameter checking.

2.1.1 FNA (Cursor Addressing)

This function is called by passing the row and column coordinates of the position on the CRT that you want the cursor to appear. The function call would appear as follows:

```
X=FNA(6,1Ø)
```

"X" is a dummy variable (no useful information is returned to "X" by the function call). "6" is the row (line number), and "1Ø" is the column (position on the line) coordinates to which the cursor is to be addressed. In this example the cursor would appear on the sixth line down from the top of the CRT and ten character positions over from the left side.

Version 5 of CSUB supported FNA because it was the only means of cursor addressing. However version VI is designed to run under **baZic** which has its own cursor addressing statement (PRINT@ or !@) which is much faster and more versatile. FNA has been left in CSUB for compatabilit~ with older CSUB programs, but it is recommended that all users of CSUB use the PRINT@ statement. FNA is now defined by the use of the PRINT@ statement. FNA can be deleted from CSUB if you are writing new programs and have no need for FNA. FNA is defined on lines 154 and 156 of CSUB.

2.1.2 FNC (Display Error Message)

This function displays an error message or bulletin. A string of up to 8Ø characters is passed to this function which then displays the message on the 4th line of the CRT. This function flashes the error message on the CRT and rings the bell to alert the operator that an error message or bulletin is being displayed.

The number of times an error message or bulletin is displayed is controlled by a variable (C6) at the beginning of CSUB. This variable may be modified so that you can control the number of times the error message is displayed. If you do not want the messages displayed you may set this variable to "Ø" so that the error message will be displayed zero times. This variable can be changed within a program written under CSUB so that different messages are displayed different numbers of times.

A sample call of the function would appear as follows:

```
X=FNC("THIS IS AN ERROR MESSAGE TO BE DISPLAYED")
```

X is a dummy variable for this function. The message to be displayed can be passed as a string with quotes or as a variable:

```
X=FNC(A$)
```

2.1.3 FNC1 (Display Error Message)

FNC1 is very similar to FNC in that an error message is displayed to inform the user of an error condition. FNC1 differs from FNC in that FNC1 displays the message, rings the bell to alert the operator, and then waits for the operator to press any key before the message is erased and the program continues.

2.1.4 FND (Display N Items)

This function call is generally preceded by a RESTORE statement to the line number that contains the DATA statements that correspond to the variables you want to display. Normally the program would RESTORE to line number 51ØØØ (or any other line number) where the FNE DATA table is located. Items can be displayed from the "middle" of the table by restoring to the proper line number and calling FND to display the correct number of items.

The set of parameters for each variable concerned should be on a different line number. Each should also be REMarked appropriately so that easy variable identification is possible. For more information on the DATA statements see sections 5 and 6 (FNE DATA TABLES and FNF DATA TABLES).

A sample call of this function would appear as follows:

```
RESTORE 51ØØØ\X=FND(1Ø)\REM DISPLAY 1Ø ITEMS
```

This call of the function would result in 1Ø items being displayed on the CRT. This display would take place with full formatting and cursor addressing as implied by the DATA statements located at 51ØØØ in this example.

By setting the DATA statements' pointer to the beginning of the DATA tables and specifying the appropriate number, all the variables can be printed with a single call. By setting the pointer within the table and specifying the number "1" (or any other number), one or more items can be displayed.

2.1.5 FNE (Input the Nth Variable)

The DATA statements associated with this function are located at line number 51ØØØ. This function does an automatic RESTORE 51ØØØ and then an INPUT on the variable described by the Nth DATA statement in the table. This function call will only accept responses which fall within the parameters described in the DATA statement. The screen position, prompt characters, and variable name are also contained in the DATA statement. The parameters of the INPUT are discussed in section 5 (FNE DATA TABLES).

A sample call of the function would appear as follows:

```
X=FNE(1)
```

In this case the input would be on the first value in the table. If the number is greater than 1, the function skips over all other entries and does only the entry requested. If several items are to be input in order, the program would appear as follows:

```
FOR N=1 TO 1Ø
X=FNE(N)
NEXT N
```

On any input function (FNE or FNF) the possibility may exist that the user has made a mistake and wants to "back up". CSUB is set up as delivered to respond to a Control B or the escape key (ESC) as a signal to back up. If either of these keys has been pressed, the function call will return a "2" signifying the user wants to backup.

NOTE: Many "older" CSUB programs used the value of A9 to determine if the user wanted to "back up". A9 is the value used in the function itself and is returned from the function call. Therefore, A9 can be examined to determine if the user wants to back up.

The following program shows how to evaluate the variable returned from the function call in the situation where multiple inputs are to take place yet we want the user to have the ability to back up at any time.

```
1Ø X=FNF(1)\IF X=2 THEN END\REM FIRST INPUT
2Ø FOR N=1 TO 1Ø\REM FOR EACH OF TEN VARIABLES DO INPUT
3Ø X=FNE(N)\REM INPUT NTH VARIABLE
4Ø IF N=1 AND X=2 THEN EXIT 1Ø\REM BACK UP TO FIRST INPUT
5Ø IF X<>2 THEN 6Ø ELSE N=N-1\GOTO 3Ø\REM BACKUP
6Ø NEXT N\REM DO NEXT VARIABLE
```

Line number 1Ø of this example is an FNF input to be discussed in section 2.1.6. If a Control B (^B) or ESCape is entered in response to this input, X will equal 2 and the program will END. In line 2Ø the FOR NEXT loop is established for the number of inputs we want to do. In line 3Ø the function always RESTOREs to the top of the table and does the input as controlled by the FOR NEXT loop.

In line 4Ø we check for the special case where we are on the first FNE input and the user wants to back up to the previous input. We must EXIT from the FOR NEXT loop in this case. Line 5Ø then looks for the condition where the user wants to back up from any of the other inputs. The value of the loop is decremented by one and we return to do the input again.

This is a simple program but we can see the flexibility and power of the subroutines from this example. Remember that this sequence is all that is needed (after writing the DATA statements) to do the input of many variables with full cursor addressing and input parameter checking as well as the redisplay of the variables with full formatting.

2.1.6 FNF (Display the Nth Prompt and Input Variable)

Upon calling this function, the Nth prompt is displayed at the beginning of line 2 and the INPUT is executed on line 3 according to the parameters of the DATA statement. The data table associated with this input should start at line 50000.

This function works very similarly to FNE except that this function displays a prompt string on line 2 of the CRT. The prompt is contained in the DATA statement describing the input.

The input from this function normally takes place on the third line of the CRT. This position can be offset by certain variables so that the input occurs at any position on the CRT although the prompt will always be displayed on the second line. See Section 7 (VARIABLE RESERVATIONS) for more information on offsetting the input of FNF.

A sample call of this function would appear as follows:

```
     X=FNF(5)
```

This call would result in the fifth (5th) prompt in the table being displayed and an input accomplished using the remaining values in the DATA table.

2.1.7 FNM (Display a Mask)

This function is used to display a mask. Each mask contains the row and column coordinates for each entry in the mask. The mask is displayed by calling the function with the starting position of the mask within B$ and the mask number (1 for the first mask). Upon calling the function, the mask is displayed on the CRT by reading the row and column values, positioning the cursor through PRINT@, and printing the strings in their proper places.

The mask system has the ability of handling multiple masks. Consult Section 9.3 for more information on how masks are created and displayed.

A sample call of the function would appear as follow:

```
     X=FNM(1Ø5,1)\REM DISPLAY A MASK
```

The argument 1Ø5 is the beginning position of the mask in B$ and 1 is the number of the mask.

If the INTeger portion of the first argument (1Ø5 in the example) is equal to the argument (no fractional portion), the function will execute an automatic CLS (clear screen) before the mask is printed. If you do not want the screen to be cleared before the mask is printed, add .1 to the first argument (1Ø5.1).

If A9$="Z", the FNM function will assume that you have a CRT that is capable of reverse video and line drawing and will display the reverse video and line drawing characters within the mask. If A9$ is not equal to "Z", the function will display only the text portion of the mask.

Presently, CSUB and its masks are set up to do reverse video and line drawing on the Zenith Z-19 CRT. If you have another CRT that has these features, you will have to change the MASKEDIT program to fit the requirements of your CRT.

2.2 Miscellaneous Functions

These functions involve internal operations within the computer. FNX allows an orderly CHAIN (or APPEND) from one CSUB program to another. FND$ and FNL$ manipulate data in RAM.

2.2.1 FNX (CHAIN to Another Program)

This function clears the screen and prints a message indicating which program is being CHAINed from and to. If the drive number parameter is a whole number the function will APPEND 1ØØØ, which causes CSUB (or any other program below line number 1ØØØ) to be preserved in RAM and the specified program APPENDed starting at line number 1ØØØ. This APPENDed program will completely replace all line numbers from 1ØØØ on up. If there is a fractional portion to the drive number then the function CHAINs to the appropriate program, completely replacing CSUB or any program that is in internal memory.

A typical call of this function would appear as follows:

```
    X=FNX("THIS PROGRAM CHAINS TO","THIS PROGRAM","NAME",DRV.CMD)
```

or:

```
     X=FNX($1,$2,$3,N.M)
```

where the parameters passed are:

```
     $1 = Descriptive Name of Program That has Terminated
     $2 = Descriptive Name of Program CHAINing to
     $3 = Actual file name of program to CHAIN to
      N = Disk drive number of program to CHAIN to
      M = Ø means the specified program is to be APPENDed
      M = 1 means the specified program is to be CHAINed
```

The parameters passed to the function can be variables. Using variables can be especially useful in a program such as a menu program that may branch to many different programs.

The value returned from the function is a dummy variable.

2.2.2 FND$ (Change a Numeric Date into a String)

All dates under CSUB are stored as numeric variables. This was first implemented when CSUB was only supported under 8 digit precision BASIC to save file space. It does not save space under 1Ø or more digits of precision but has been left this way because it does offer convienence to the programmer. Since dates are normally stored as numerics this function is called to convert a numeric date into the familiar string date with slashes between the month and day and month and year.

This function is generally called when printing a date on paper. A numeric argument is passed and the function returns a string. A typical call of the function would appear as follows:

```
!#1,FND$(B(1))
```

where the variable B(1) contains the numeric version of the date. If B(1) contains the value 8268Ø and function D$ is called with B(1) as the argument, the printed value would appear as follows:

```
8/26/8Ø
```

The value returned from the function can be printed directly as in the example or a string can be set to the value as in the following example:

```
X$=FND$(N)
```

2.2.3 FNL$ (Left Justify a String)

Originally (release 5.2 and earlier), all string information input using FNE or FNF is right justified within B$ where it is stored. In other words, if a string was input and the user did not enter a character in all of the positions available, the information entered would have been justified (moved) to the right and leading spaces would have been inserted in that portion of the string not filled.

The FNL$ function reverses this process (usually for printing) so that the information is moved left and trailing spaces are entered. The values passed to FNL$ refer to the Left$ and Right$ positions within B$ which you want left justified. B$ is the only string the arguments of FNL$ act upon.

As an example, if:

```
B$(1,1Ø)="       DOG"
```

and we call function L$ with the following call:

```
     X$=FNL$(1,10)
```

then X$ would now appear as follows:

```
     X$="DOG       "
```

A typical function call would appear as follows:

```
     !#1,FNL$(1,10)
```

This function may be called to fill a variable as in the DOG example or may be called to directly print the results as in the previous example.

A much better and faster way is now provided by CSUB for left justifying a string. FNE and FNF can now be passed an additional command that causes the input to be left justified, right justified, or centered. This feature should eliminate most of the need for FNL$. This function is left in the CSUB package to remain compatible with older CSUB based programs. It may be removed to recover space if you do not need it.

2.3 File Accessing Functions

The following functions were designed to handle most of the disk accesses required by a programmer. The G function is designed to stand by itself to write sequential or random records or work in conjunction with function H, the key file accessing function, for single or multikeyed access capability.

Since function G calls do an OPEN and a CLOSE for each call of the function, using this function for sequential reading of a file may be too slow on a floppy disk based system. FNG1 can be used when you want the file OPENed and CLOSEd by your command. In many cases a sequential read is accomplished better by OPENing the file and reading the records without function G but this has the disadvantage of not having any error trapping during the disk access.

When using non CSUB routines to access a file, it may be advantageous to make one call of a CSUB function for the first record because the function contains numerous error trapping features. This will insure that the file is in the proper place and everything is in order before the records are read in a non-CSUB mode.

All writes of CSUB based files should be made through FNG, FNG1, or FNH since they very strictly control the format of data passing into the files. If writes are made into the files without the use of these functions, often the functions will "bomb" at a later time since they recognize that something is wrong with the file structure.

2.3.1 FNG (Sequential File Access)

This function reads or writes a file randomly or sequentially. The parameters passed to this function are:

```
G=FNG("FILE NAME", DRIVE.CMD, LEFT$, SUBS, RECORD.CHANNEL)
```

"FILE NAME" is the name of the file to be accessed. The name must be a legal name as defined by **baZic**. The value passed to the function can be a literal string or a string variable.

DRIVE.CMD is the drive number where the file is located and the command that is to be executed by this call of the function. The integer portion of this numeric value is evaluated to arrive at the drive number. The drive number must be a legal drive as supported by your system hardware.

The fractional or decimal part of this field is the command that the function is to execute. If the INT(Drive)=Drive (1.Ø, 2.Ø, 1, etc.) then this function READs the file. If the INT(Drive) <>Drive (1.1, 2.1, etc.), then this function WRITEs to the file. The integer portion of Drive is the drive number. The commands are summarized as follows:

.Ø = read
.1 = write

LEFT$ is the value within B$ where the read or write to B$ will start. If B$ is several hundred characters (or more) in length and several different file records are contained within B$, each file access must know where to read or write its information within B$.

Function G will know the length of the string information of each file by reading the first three numerics of each FNG file. This information, combined with the LEFT$ value, allows FNG to read or write information from many files directly into or out of B$ without interfering with information contained in B$ from other files.

SUBS is the B(subscript) value where a read or write to the B subscript variable will start. This situation is completely analogous to the previous discussion on LEFT$. The value of SUBS controls the position of numeric data written to or read from the B subscript variable.

RECORD is the record number of the file one wants to read to or write from the file. Record Ø doesn't exist and numbering starts with 1. If record = -1 and the operation is a write, then the record is written onto the end of the file in a sequential manner.

The RECORD number can be passed as a constant or as a variable (recommended).

The CHANNEL argument is added as the fractional part of RECORD if FNG1 has been called to OPEN a file channel. The actual channel number opened should be the value passed as the CHANNEL argument. If FNG1 is called to open channel 7 for a particular file, 7 should be passed as the CHANNEL argument to FNG when it is called to read or write the file.

2.3.1.1 FNG Errors

After calling this function (G=FNG( )), G contains the number of records in the file if everything went correctly. However, if an error condition has occurred, then G will be negative and will indicate what type of error has occurred. Programmers using this function should check G to make sure it is positive before using the variable and to assure themselves that the function performed its job correctly. Function errors returned in G are:

-1 = File Error
-2 = Hard Disk Error
-3 = File Not Found
-4 = Incorrect File TYPE
-5 = File Length Exceeded
-6 = TYPE Error
-7 = File Name Longer than 8 Characters
-8 = Record Doesn't Exist (Record # incorrect)
-9 = B$ Length Not Valid or B( ) Not Diminsione~
-1Ø= Attempt to Write Beyond End Mark (Record # passed was too large)

A subroutine has been implemented at 55ØØØ to handle secondary error messages when a negative value is returned from FNG. The routine expects a negative value in G and prints the appropriate secondary error message. These secondary messages may be used or not at the discretion of the programmer. Function G returns a primary error message when the error is encountered. Additional error messages may be passed to the user to instruct him as to what action to take when the errors occur.

This function contains its own error trapping routines. This means that anyone using the routine must disable his own error trapping before calling the function and re-enable his error trapping upon completion of the function call.

2.3.2 FNG1 (Channel Open and Close)

The purpose of this function is to allow the user to be able to control the file channel to open, as well as the ability to keep the file open for as long as desired. Normally a call to function G would result in the channel being opened and closed by the single function call. By calling FNG1 first, a channel can be specified to be left open for the entire procedure.

By calling FNG1 before the FNG call, significant time advantage can be gained, since the file does not have to be opened every time the file is accessed. When the program is finished using a file, FNG1 should be called again to close the file.

FNG1 will only allow the opening of channels 2 to 7 so as not to interfere with the operation of "regular" FNG function calls. If the function call is accomplished without an error, the value returned from the function call will be the channel number opened. If an error occurs, the value returned from the function call will be a negative number and will correspond to the errors listed under FNG. When closing a channel, FNG1 will return a Ø from the function call.

A sample use of this function would appear as follows:

```
     G1=FNG1("FILE",DRIVE.CMD,CHANNEL)
```

where:

```
     "FILE" is the file name
     DRIVE.CMD is the drive number and command where
          .Ø=open
          .1=close
     CHANNEL is the channel number to be opened or closed
```

In an actual programming situation the function call would be made as follows:

```
     1ØØ G=FNG("FILE",2,1,1,1)\REM GET NUMBER OF RECORDS IN FILE
     11Ø IF G<=Ø THEN STOP\REM CHECK FOR ERRORS
     12Ø N1=G\REM SAVE THE NUMBER OF RECORDS IN N1
     13Ø G=FNG1("FILE",2,3)\REM OPEN CHANNEL NUMBER 3
     14Ø FOR N=1 TO N1\REM FOR THE NUMBER OF RECORDS
     15Ø G=FNG("FILE",2,1,1,N+.3)\REM READ EACH RECORD
     16Ø IF G<=Ø THEN STOP\REM CHECK FOR ERRORS
     17Ø GOSUB XXX\REM SUBROUTINE TO PRINT RECORDS OR WHATEVER
     18Ø NEXT N\REM PROCESS NEXT RECORD
     19Ø G=FNG1("FILE",2.1,3)\REM CLOSE CHANNEL 3
```

Notice that the last field (RECORD #) in the FNG call has the CHANNEL number added as the fractional part of the record number field. The FNG1 call will not cause the CHANNEL to remain open unless the CHANNEL number is added to the FNG call as described.

This technique will allow much faster file accessing than use of FNG only. Write operations can be performed using a similar technique.

2.3.3 FNH (Key File Accessing)

This function returns a pointer to a main data file (or multiple data files) by looking up an index in a keyfile and returning the appropriate information (key) about the location of data in the main file. Multiple keyfiles are allowed for any particular data file. The actual file accessing is accomplished by using the sequential access file routine FNG.

Keyed access differs from sequential access in that the sequential file starts with only three numeric values initialized to Ø at the beginning of the file and one blank record, while keyed access has to have all records in the file completely initialized by setting them to blank records. In keyed access, a large file is divided into records of uniform length so that a record index can be looked up quickly in the keyfile and the main file accessed directly by passing the value returned from FNH to FNG. FNH can perform the following tasks:

1. Create an Index
2. Delete an Index
3. Find an Index
4. RENAME an Index

A typical call of function H would appear as follows:

```
H=FNH ("FILE", DRIVE.CMD, "INDEX", "RENAME", "KEYFILE #")
```

where:

"FILE" is the prefix of the name of the file being accessed. Function H will look for the proper key file by attaching the value in the last field of the function call ("KEYFILE #") to the file name as passed in the first field. If the value of "FILE" is equal to "PART" and the value of "KEYFILE #" is "K" then the keyfile accessed will be named "PARTK".

"DRIVE.CMD" is the drive number of the file to be accessed plus the command (CMD) that is to be executed. The integer portion of DRIVE is the actual drive number while the decimal value is the command. The commands are interpreted by the function as follows:

.Ø=look up the index and read the key from the key file
.1=look up the index and return the value of the key
.2=delete an index
.3=create an index
.4=rename an index

"INDEX" is normally the index name that you want function H to search the keyfile for and perform the indicated operation on. The only exception to this is when you are renaming the index. In this case INDEX will represent the name you want the index to be renamed TO.

"RENAME" is normally just two quote marks to signify a blank string (""). The only time this field will have a value is when you are calling the function to rename the index. In this case the value of RENAME will be the OLD index name that you want changed to the new value that is contained in the field INDEX.

"KEYFILE #" (added to the FILE name) is the name of the keyfile being accessed. Many keyfiles are allowed for any data file. This field is a string of one character (Ø-9, A-Z, etc.) that is added to the main file name by the routines of CSUB when keyed access is being used.

The example under "FILE" shows how a keyfile name is constructed by the function. "KEYFILE #" must always be only a single character in length.

FNH becomes very convenient if any particular data file exceeds the length of a disk. For example, if one has a data file with 1ØØØ associated keys in the keyfile, and the size of the disks and records are such that only 5ØØ records can be placed on a disk, then FNH can be used to access both sections of the file. After calling FNH, simply examine the returned value. If it is less than or equal to 5ØØ, use FNG to access the information on the first drive. If the returned value is greater than 5ØØ, subtract 5ØØ and increment drive # by 1, and access the information on the next drive.

Normally, the fourth argument of FNH is set to "". This argument is only used when renaming an index. If one wants to rename the index "JOE" to "SAM" for the File "TEST" on drive 2, then:

```
     H=FNH("TEST",2.4,"SAM","JOE","K").
```

"SAM" will replace the index "JOE" and "JOE" will no longer be an index to the data file.

When deleting an index from a keyfile, a Ø is returned in H if the deletion was successful.

Programmers should always check the value returned from FNH and FNG to make sure that no error conditions have occurred. One way of accomplishing this is:

```
     1Ø H=FNH("TEST",D1,"JOE","","K")\REM CALL KEY FILE FUNCTION
     2Ø IF H>Ø THEN 3Ø ELSE GOSUB 552ØØ\GOTO ___ \REM ERROR CK
     3Ø G=FNG("TEST",D1,1,1,H)\READ RECORD H
     4Ø IF G>=Ø THEN 5Ø ELSE GOSUB 55ØØØ\REM Ø CHECKS FOR MT FILE
     5Ø REM
```

552ØØ contains optional file error messages which are displayed based upon the ABSolute value of H after its return from FNH. 552ØØ is similar to the routines at 55ØØØ for FNG but works on the ABSolute value of H after the FNH call.

Index strings are processed as follows:

1. Delete leading spaces.
2. Delete trailing spaces.
3. Add spaces to the right until the length of the string equals the index string length.

Index strings are sorted alphanumerically, NOT NUMERICALLY OR ALPHABETICALLY. Thus, the following order would result after sorting:

```
     1ØØ
-->  11
-->  11ØØ
-->  9ØØ
     9Ø1
     ALPHA
     BETA
     ZETA
     ZFF
```

Remember that this file is ordered by the ASCII value starting with the first character. Since an ASCII space " " is 32 and the ASCII value for a "Ø" is 48, a space will always fall before a number. Also, all numbers will come before letters and upper case letters will come before lower case letters. Care should be taken to study this order, as problems can arise if the programmer isn't aware of the sorting technique used.

2.3.3.1 FNH Errors

Upon returning from the function, H contains the record number of the data in the main file if no error conditions were encountered. Thus, H may be passed to FNG in the RECORD field to actually read or write the appropriate information from the file.

If an error condition has occurred, H will have a negative value such that:

-1	=	File Name Too Long
-2	=	Key File Not Found
-3	=	Key File Incorrect Type
-4	=	Blank Index String
-5	=	File is Empty
-6	=	File is Full
-7	=	Duplicate Record Name
-8	=	Record Not Found
-9	=	Index String Too Long
-1Ø	=	Hard Disk Error

2.4 Function Summary

The functions in CSUB are summarized as follows:

```
FNA (N,N) - Non-destructive cursor positioning
FNC($) - Flash String error message
FNC1($) - Display String error message and hold
FND(N) - Display N items on CRT
FNE(N) - Input Nth item
FNF(N) - Display Prompt and Input Nth item
FNM(N,N) - Display a Mask
FNG($,N,N,N) - Sequential file access
FNG1($,N,N) - Opens and Closes FNG files
FNH($,N,$,$,$) - Keyed File access
FNX($,$,$,N) - Exit to another program
FND$(N) - Converts a number into a date
FNL$(N,N) - Left justify a string from B$
```

FILE STRUCTURES

For CSUB to access its data base quickly, the files have been structured so that records are essentially all the same length. This structure imposes some restrictions on the types of applications that CSUB can handle, but these are generally few and far between for an imaginative programmer. The flexibility of the routines means that many different situations can be handled by using the routines in different ways.

Although the structure of CSUB seems to dictate very structured files, this is not the case. Depending upon the application, several things can be done to increase the flexibility. One record can be made to change the contents of other associated records by the use of flags. Linking can be accomplished by letting one or more fields in a particular record "point" to other records or records in separate files so that they are completely linked in as many directions as the programmer wants. Another technique is to let the first record in a file have a different meaning than other records so that the first record can act as a control for that file or others.

3.1 Structure of Files Accessed by FNG

The sequential file is set up with three numeric values and one blank record placed at the beginning of the file. The first numeric value contains the number of records written to the file (Ø if the file is empty, or the maximum number allowed if the file is being used in conjuctio~ with a keyfile).

The second numeric value contains the length of the string containing all string information stored in each record of the file. If this value is Ø, there is not a string variable in each record.

The third numeric value contains the number of numeric variables for each record in the file. C9 must be set to the number of bytes of storage required for any particular precision of BASIC. This is discussed in section 7 (VARIABLE RESERVATIONS).

If we were looking into an FNG file we would see the following:

A, B, C, 1st record's string, 1st record's numerics, 2nd record's string, 2nd record's numerics, etc., where:

A	is the number of records written, Ø if none, or the maximum number if the file has an associated keyfile.
B	is the length of the string variables of each record, Ø if none.

C is the number of numeric variables of each record, Ø if none.

1st record's string is the one string which contains all of the string information of the first record.

1st record's numerics are all the numeric entries for the first record.

etc.

3.2 Structure of the Keyfile

The keyfile is created with the name of the main file plus an alphanumeric digit from Ø-9 or A-Z (etc.), which indicates the keyfile for any particular data file that is to be accessed. Thus, for any one data file, 36 (or more) keyfiles are allowed, permitting the programmer to access information in the main file from any of at least 36 different indexes.

At the first of the keyfile are three numeric entries which contain information about the structure of each particular keyfile. The first numeric value is the number of keys in the file, while the second numeric contains the number of keys in use. The third numeric indicates the length of the index string.

When the keyfile is initially set up, the first numeric is written with the number of keys, the second numeric is set to Ø, and the third is set to the length of the index. The actual indexes and keys start immediately following the three numerics. Each index string is first filled with spaces and written into the file, followed by a numeric key which indicates the record number in the main file.

As entries are made to the keyfile by FNH, the second numeric will be updated to reflect the number of entries in the file. Likewise, on a delete, the second numeric will be decremented. Also upon a deletion, FNH does "housekeeping" on the keyfile so that the space is recovered for use in both the keyfile and the main file (FNG file).

FNH does all necessary maintenance on the keyfile so that records can be added and/or deleted for as long as is necessary without the programmer having to worry about internal ordering and pointers. A deletion causes the space to be recovered from the keyfile, but nothing is done by FNH to the FNG file so that information in the main file is not lost until the key is used by another index.

The key in the keyfile still points to the record number in the FNG file but the index is blank and reusable. This means that the information is still contained in the FNG file, but no index is associated with that record in the keyfile. A subsequent call for a new index by FNH will result in the previous key being assigned the new index.

3.3 Structure of a Mask File

Masks are stored in FNG compatible files in that all mask files contain the first three numerics. Mask file names always end in an ampersand (&) to indicate they are mask files. The mask file name is generally the same as the program that uses the mask except that the ampersand has been added at the end of the file name to form the mask file name.

If the program name were PGM, the mask(s) file name for that program would be named PGM&.

Internally in the file, the first numeric will always be a "1" indicating there is only one string in each record. Even if there are multiple masks, they will all be stored in the single string.

The second numeric is the length of the string (or mask if there is only one mask in the file) including all masks stored within the file. The third numeric should always be zero (Ø), indicating there are no numerics within the file.

The mask string is divided internally so that some bytes are control bytes while others are actually the text bytes of the mask.

LINE NUMBER ASSIGNMENTS

Lines 1 to 999 are reserved for CSUB.

Lines 900-999 are reserved for user-defined print formatting routines. These routines control all printing done on the CRT. Print formatting for hard copy is presently left up to the programmer.

Line 999 is usually a standard FNX function call so that application programs have a standard location to call when they need to change to the next program.

Lines 1000-65535 are the unreserved program line numbers (see exceptions below) available to all programmers for writing their programs. User application programs normally start at 1000 and are APPENDed to CSUB to "RUN".

Lines 50000-50999 mark the area of the DATA table used by FNF inputs. The prompts contained within this data table are what guide the user through the program while it is RUNning. Inputs occur automatically at position 1 on line 3, while the prompt string specified always appears at position 1 on line 2. Line number 50000 is the line number RESTOREd to by an FNF call. The inputs can be moved from line 3 by the use of C2 and C3 offset variables discussed in section 7 (VARIABLE RESERVATIONS).

Lines 51000-51999 mark the area of the DATA table used by FNE inputs. Inputs or displays occur at the position described by the DATA statements themselves.

The previous two sections (50000 and 51000) are normally a part of the application program.

Lines 52000-58999 are reserved for subroutines or additional function definitions needed by the programmer.

User-defined formatting statements should have the following form:

```
     900  ONINT(A5) GOTO 910,920,930,940,950,960
     910  !%$10F2,B(A8)\RETURN\REM 1
     920  !%9F4,B(A8)\RETURN\REM 2
     930  !%2I,B(A8)\RETURN\REM 3
     940  !%1I,B(A8)\RETURN\REM 4
     950  !%6I,B(A8)\RETURN\REM 5
     960  !%9F2,B(A8)\RETURN\REM 6
```

The variable A5 is extracted from the appropriate DATA statements and passed to this user-defined routine by the printing routines of CSUB. The number used is the integer portion of the format variable contained in row five of the NUMERIC (Type 1) DATA statements. A8 is also passed from the appropriate routines in CSUB to this formatting section to indicate which subscript of B is to be printed. String and Date printing on the CRT are handled by other routines contained within CSUB.

If the programmer chooses, additional error trapping can be done in the 900 section. The most probable cause of error in this section would be a FORMAT ERROR. These can be eliminated by checking the range of the variable before it is printed and skipping to another unformatted routine. The variable would be printed and not formatted correctly but the program would not "bomb". An example of this situation would appear as follows:

```
900 ONINT(B(A8)) GOTO 910,920
910 IF INT(A5)<-99999.99 OR INT(A5)>999999.99 THEN 930
915 !%$10F2,B(A8)\RETURN\REM 1
920 IF INT(A5)<-9 OR INT(A5)>99 THEN 930
925 !%2I,B(A8)\RETURN\REM 2
930 !B(A8)\RETURN\REM PRINT UNFORMATTED
```

FNE DATA TABLES

FNE does input on the selected value by "looking up" its characteristics in a table. That table is called the FNE DATA table and is always located at line number 51ØØØ. An FNE input is accomplished by reading down the table until the correct set of parameters is found and then controlling the input so that only values meeting the requirements are accepted.

The DATA statements contain information that uniquely and completely defines all variables that are to be printed on or input from the CRT. Please check the FNE REFERENCE CHART for summation of the DATA that is contained in the DATA statements.

5.1 FNE Reference Chart

DATA	FIELD #							
TYPE	1	2	3	4	5	6	7	8
NUMERIC	1.X	LINE#.Y	POS	#CHAR	FORMAT.A	LOWLIM	HILIM	B(SUBS)
STRING	2.X	LINE#.Y	POS	#CHAR	Ø	LEFT$	RIGHT$	Z
STR/NUM	3.X	LINE#.Y	POS	#CHAR	Ø	LEFT$	RIGHT$	B(SUBS)
DATE	4.X	LINE#	POS	Ø	Ø	LEFT$ (Ø)	RIGHT$ (Ø)	B(SUBS)

where FIELD # is:

1 X=Ø display and/or input
X=1 display only
X=2 input only

2 LINE# is the number of the line where CSUB is to take input
Y=ASCII value of character to be displayed on input

3 POS is the position on the line where CSUB is to take input

4 #CHAR is the number of prompt characters to be displayed

5 FORMAT is the number of the format statement to use
.A=number of spaces to the right to displace numeric input

6&7 LOWLIM is the lower limit to be accepted for numeric inputs.
HILIM is upper limit to be accepted for numeric inputs.

For Type 2 LEFT$, is the left position in B$ of where the string is to be started.

For Type 2 RIGHT$, is the right position in B$ of where the string will end.

For Type 3 LEFT$, is the left position in C$ of where the input is to be considered valid.

For Type 3 RIGHT$, is the right position in C$ of where the input is to be considered valid.

For Type 4, if the input is to be a numeric date then:

```
LEFT$ = Ø
RIGHT$ = Ø
```

and the date is returned as MMDDYY.

For Type 4, if the input is to be a string date then:

```
LEFT$ = Starting position of date in B$.
RIGHT$ = Ending position of date in B$.
```

and the date is returned as YYMMDD.

8 B(SUBS) is the subscript number assigned to this variable.

Z is the justify command for string inputs such that:

```
Ø = right justify
1 = left justify
2 = center
```

5.2 First Field (FNE)

The DATA statements for input, display, or input and display deal with four basic TYPEs of DATA:

```
TYPE 1 is NUMERIC
TYPE 2 is ALPHA-NUMERIC (STRINGS)
TYPE 3 is STRING-NUMBER
TYPE 4 is DATE
```

This information is entered as the integer portion of the first field in the DATA statements. The fraction portion of this field defines what kind of DATA statement is being used, where:

```
.Ø = Input and Display
.1 = Display Only
.2 = Input Only
```

Examples:

```
1.Ø = Numeric Input and Display
4.1 = Date to be Displayed Only
2.2 = String to be Input Only
```

TYPEs 1, 2, and 4 are mostly self-explanatory to anyone familiar with BASIC, but TYPE 3 may not be readily understood. TYPE 3 (STRING-NUMBERS) was implemented to deal with a specific problem that arises constantly in the writing of application programs. If a particular input is asking a question where only specific alphanumeric responses are allowed, this is the TYPE DATA statement to use. As an example, if a particular input is requesting a yes or no response (Y/N), a problem arises if another character is entered as the programmer has to check for the appropriate response.

TYPE 3 DATA statements do this check automatically. This is done by entering the appropriate DATA into C$ (C$="YNRST") and specifying the appropriate range allowed for each input in fields 6 and 7 of the DATA statement. In this particular instance, if 1 and 2 are specified for fields 6 and 7 respectively, the input will allow only a response of "Y" or "N", but not "R", "S" or "T". If field 7 is a 3, inputs of "Y", "N", and "R" are accepted. If a "Y" is input, the function returns to the specified B subscript (field 8) the value Ø. If an "N" is input, a 1 is returned, and if an "R" is input, a 2 is returned, etc.

If a letter is entered that is not correct, an error message will be displayed that will appear as follows:

YN ONLY (where Y and N represent the acceptable response)

Care must be used in ordering the letters contained in C$ since the error message display could cause unwanted words to be inadvertantly displayed on the CRT.

5.3 Second Field (FNE)

The Second Field passes two pieces of information to the defined functions and uses the form of "LINE#.ASCII character". The integer portion of this field represents the line number on the CRT where the input or display function is to take place for any specific variable. Line numbering starts with one (1) as the first line on the CRT. The fractional part of this field represents the ASCII code of the character to be displayed on the CRT while taking an input. Generally, if money is to be input, a "$" (ASCII=36) is displayed. The standards normally used are:

$	(36)	= Dollars
#	(35)	= Numbers
%	(37)	= Percentage
*	(42)	= String Information

For TYPE 4 - DATE input and display, the integer portion only is entered in field 2, as date input and displays are formatted automatically to MM/DD/YY. The variable C2 works on the integer portion of this field to displace downward the input or display line number by the value of C2. C2 may be positive or negative as long as its value is a position within the limits of the CRT.

5.4 Third Field (FNE)

This field represents the position on the line where the input or display is to take place on the CRT. C3 works on this field by displacing the input or display to the right by the value of C3. If the input is to take place on line 6 at column 4Ø, the value of the third field would be 4Ø. C3 may be positive or negative as long as its value is a position within the limits of the CRT.

5.5 Fourth Field (FNE)

This field represents the number of characters to be displayed upon an input. The character displayed will be the one represented by the fractional part of field 2. If a 1Ø digit string is to be input, field 4 will be 1Ø and field 2 will be X.42, where X is the line number of the input. For TYPE 4 - DATE inputs, this field should be set to Ø, as the routines handle the number of characters displayed automatically.

5.6 Fifth Field (FNE)

This field is used to specify the format of all numeric printing on the CRT. For TYPEs 2, 3, and 4, this field should be set to Ø. For numerics, the value in this field will be converted to A5, where it is passed to the user-defined formatting routines at 9ØØ. Standard formats are provided. The fractional portion of this field represents a displacement to the right, which is used to displace an input from its normal position.

5.7 Sixth and Seventh Fields (FNE)

These fields have different definitions for each of the four TYPEs of DATA.

For TYPE 1 - NUMERIC, field 6 represents the lowest numerical value that this input will accept, and field 7 represents the maximum numerical value that will be accepted. These fields are very important and are used for strict control of all numeric inputs so that only appropriate numeric responses are allowed.

For TYPE 2 - STRING inputs and displays, fields 6 and 7 represent the LEFT$ and RIGHT$, respectively, of the position within B$ from which the string information is to be input or displayed.

For TYPE 3 - STR/NUM, fields 6 and 7 represent the LEFT$ and RIGHT$, respectively, of the range within C$ from which inputs are to be allowed. See the discussion of TYPE 3 - STR/NUM for more explanation on the workings of this TYPE data input.

For TYPE 4 - DATE, if the date is to be numeric, these fields are both set to Ø. If the date is to be a string input, fields 6 and 7 should have the LEFT$ and RIGHT$ position within B$ of where the date is to be input.

5.8 Eighth Field (FNE)

For TYPE 1 this field represents the subscript of B array (B( )) for all numeric variables to be displayed or input.

For TYPE 2 inputs and displays, this field can have one of three meanings:

Ø means to right justify the text within its B$ limits
1 means to left justify the text within its B$ limits
2 means to center the text within its B$ limits

For TYPE 3 or TYPE 4 numeric date inputs, field 8 is the subscript number of the B array for the variable being input.

FNF DATA TABLES

FNF inputs are the inputs which "guide" the user through the programs. These inputs contain the prompts which interface to the user to let the user know what the choices are at any particular point in the program execution. The 8 fields associated with FNF inputs are very similar to FNE fields. The main difference is the addition of a string field (the prompt). Fields 4 to 8 are identical to the FNE fields.

The DATA statements contain information that defines uniquely and completely all variables that are to be printed on or input from the CRT. Please check the FNF REFERENCE CHART for summarization of the DATA that is contained in the DATA statements.

NOTE: Version 5 of CSUB contained an extra field in the FNF data statements table. This field was eliminated in version VI because it was not needed. Older CSUB programs will have to use CSUBEO to execute properly.

6.1 FNF Reference Chart

DATA	FIELD #							
TYPE	1	2	3	4	5	6	7	8
NUMERIC	1.X	"PROMPT"	Y	#CHAR	FORMAT.A	LOWLIM	HILIM	B(SUBS)
STRING	2.X	"PROMPT"	Y	#CHAR	Ø	LEFT$	RIGHT$	Z
STR/NUM	3.X	"PROMPT"	Y	#CHAR	Ø	LEFT$	RIGHT$	B(SUBS)
DATE	4.X	"PROMPT"	Y	Ø	Ø	LEFT$ (Ø)	RIGHT$ (Ø)	B(SUBS)

where FIELD # is:

1 X=Ø display and/or input
X=1 display only
X=2 input only

2 "PROMPT" is the prompt to be displayed by the input

3 Y=ASCII value of character to be displayed on input

4 #CHAR is the number of prompt characters to be displayed

5 FORMAT is the number of the format statement to use
.A=number of spaces to the right to displace numeric input

6&7 LOWLIM is the lower limit to be accepted for numeric inputs.
HILIM is upper limit to be accepted for numeric inputs.

For Type 2, LEFT$ is the left position in B$ of where the string is to be started.

For Type 2 RIGHT$, is the right position in B$ of where the string will end.

For Type 3 LEFT$, is the left position in C$ of where the input is to be considered valid.

For Type 3 RIGHT$, is the right position in C$ of where the input is to be considered valid.

For Type 4, if the input is to be a numeric date then:

```
LEFT$ = Ø
RIGHT$ = Ø
```

For Type 4, if the input is to be a string date then:

```
LEFT$ = Starting position of date in B$.
RIGHT$ = Ending position of date in B$.
```

8 B(SUBS) is the subscript number assigned to this variable.

Z is the justify command for string inputs such that:

```
Ø = right justify
1 = left justify
2 = center
```

6.2 First Field (FNF)

This field is identical to FNE's first field (Section 5.2).

6.3 Second Field (FNF)

This field contains the prompt to be displayed on the 2nd line of the CRT to "prompt" the user as to what information is to be input.

6.4 Third Field (FNF)

This field contains the ASCII value of the prompt you want printed when the input is taken.

6.5 Fourth Field (FNF)

This field represents the number of characters to be displayed upon an input. The character displayed will be the one represented by the ASCII value of field 2. If a 1Ø digit string is to be input, field 4 will be 1Ø and field 2 will be 42. For TYPE 4 - DATE inputs, this field should be set to Ø, as the routines handle automatically the number of characters displayed.

6.6 Fifth Field (FNF)

This field is used to specify the format of all numeric printing on the CRT. For TYPE 2, 3, and 4, this field should be set to Ø. For numerics, the value in this field will be converted to A5, where it is passed to the user-defined formatting routines at 9ØØ. Standard formats are provided. The fractional portion of this field represents a displacement to the right which is used to displace an input from its normal position.

6.7 Sixth and Seventh Fields (FNF)

These fields have different definitions for each of the four TYPEs of DATA.

For TYPE 1 - NUMERIC, field 6 represents the lowest numerical value that this input will accept, and field 7 represents the maximum numerical value that will be accepted. These fields are very important and are used to stricl~ control all numeric inputs so that only appropriate numeric responses are allowed.

For TYPE 2 - STRING inputs and displays, fields 6 and 7 represent the LEFT$ and RIGHT$, respectively, of the position within B$ from where the string information is to be input or displayed.

For TYPE 3 - STR/NUM, fields 6 and 7 represent the LEFT$ and RIGHT$, respectively, of the range within C$ from which inputs are to be allowed. See the discussion of TYPE 3 - STR/NUM in Section 5.2 for more explanation on the workings of this TYPE data input.

For TYPE 4 - DATE, If the date input is to be numeric, these fields are both set to Ø. If the date input is to be a string input, fields 6 and 7 should have the LEFT$ and RIGHT$ position within B$ of where the date is to be input.

6.8 Eighth Field (FNF)

For TYPE 1 this field represents the subscript of B array (B( )) for all numeric variables to be displayed or input.

For TYPE 2 inputs and displays, this field can have one of three meanings:

Ø means to right justify the text within its B$ limits
1 means to left justify the text within its B$ limits
2 means to center the text within its B$ limits

For TYPE 3 or TYPE 4 numeric date inputs field 8 is the subscript number of the B array for the variable being input.

VARIABLE RESERVATIONS

All variables A, B, C, and D are reserved for CSUB functions. These include string variables, numerics, and numeric arrays. These variables must not be used in the course of programming your application programs except in the exact context as presented below.

A9 is the "back up" variable. This variable will equal 2 after any input in which a control B (^B) or ESC has been entered. A9 is returned from the input functions (FNE and FNF) as the value of the function call.

A5$ is defined at the beginning of CSUB to clear one line of the CRT. If your CRT has a clear-to-end-of-line code, define A5$ as this code. If your CRT does not have this feature, define A5$ as 79 or 8Ø spaces (79 if your CRT does an automatic carriage return from the last column position).

A9$ is used as a flag to the FNM function to tell the function if the CRT being used is a Zenith. If the CRT is a Zenith, A9$ will equal a "Z". If the CRT is not a Zenith, A9$ can remain undefined.

B( ) contains ALL numeric file, input and display variables.

B$ contains ALL string file, input, display and mask variables.

B7 is the password protect variable. If B7 is equal to 1, an "X" will be echoed for each character typed by the user in response to an FNE or FNF input. This feature is implemented so that the password is not displayed when entered by the user.

C$ contains ALL string references for TYPE 3 inputs. Use of C$ is explained in the introduction to section 5 (FNE DATA STATEMENTS).

C1 is the subscript offset variable which allows one to input or display multiple records from any particular file or multiple files. For example, if we read a data file consisting of 5 records, each with 1Ø numeric values, into B(1) through B(5Ø), and set C1=Ø, function calls will display or input to the first record. If C1=1Ø, the second record will be displayed or input; if C1=2Ø, the third record will be displayed or input, and so on. Programmers must be sure to reset C1 to the appropriate value after function calls are made.

C2 is the Line Number offset variable that offsets any CRT display or input by the value of the variable. This is a non-destructive offset used primarily for formatting or displays when inputs and displays need to be relocated slightly from their "normal" positions.

C3 is the offset for character position on a line. This variable is similar to C2 except the offset is on the same line. The offset is to the right.

C4 is the string offset variable. This variable is similar to C1 except that this variable controls the offset of B$.

C5 is the STRING-NUMERIC (TYPE 3) offset variable which controls the offset for working with the string-numbers of C$. If C5=Ø, then the string-number functions work as described elsewhere. If C5=1, the first character of C$ is ignored and processing continues in the normal manner, starting with the second character in the string.

C6 controls the number of times that an FNC error message is displayed.

C9 is the precision variable. This variable must be set to the number of bytes required to store a numeric variable (5 for 8 digit precision, 6 for 1Ø digit precision, etc.).

D3$ is the name of the program that called your program originally. This variable is currently set to MENU but can be anything you want. This value will be passed to FNX, the exit function, which will control needs to pass out of this program.

D5$ is the descriptive name of the program you are writing. This variable is printed on the CRT during a program chain by the exit function FNX.

D7$ is the descriptive name of the program that originally called the present program. This name will be printed on the CRT by the exit function FNX.

The following two variables define the drive numbers associated with the program and data files. The assumption is made that all data files will be on the same drive and that the mask will be on the same drive as the programs.

D is the drive number of the programs and the mask.

D1 is the drive number of the keyfile and the main data file associated with this program.

Z1$ is printed when a CSUB program needs to issue a "formfeed" to the printer. The normal value for Z1$ is 12 (Z1$=CHR$(12)).

Z2$ is printed when a CSUB program needs to compress the print on the printer to print a 132 column printout. If your printer uses 14" wide paper, this string should be set to the null value (Ø).

Z3$ is printed when a CSUB program needs to return the printer to normal print (8Ø columns). If your printer does not need this attention, Z3$ should be set to the null value (Ø).

WRITING CSUB PROGRAMS

This section describes some of the uses of CSUB. It would be impossible to describe all the ways that CSUB could be used, but this section will detail the basic procedures involved in writing a File Maintenance (F/M) program under CSUB.

The F/M demonstration program (BONES) allows the user to add records, inquire or change a record, delete records, rename the index, and to print one or all records. This program can be set up completely and be operational in less than one hour of programming time if the programmer is experienced with CSUB and knows what kind of "data base" he wants to establish.

Of course, the F/M program is not the only program that is needed in business applications, but it is the heart of most business application packages. The BONES program should provide enough information to write any CSUB program if the programmer will take the time to study the fully REMarked listing of BONES (APPENDIX A).

The following procedure describes the steps necessary to set up a F/M program. The heart of this procedure is the BONES program. BONES is a general purpose F/M program that can be easily changed to virtually any set of file variables (fields) accessed by a single field (the index).

The BONES program is to be used as an example in demonstrating the use of CSUB. The program listing can be found in Appendix A of this manual and should be consulted while reading this section.

8.1 File Structure

The single most important item for the programmer to decide is the structure of the F/M file. By this, we mean which set of variables are to be stored in this file, which are string variables, and which are numeric variables.

Start by deciding what information is to be stored in the main file. An example might be a list of customers where we would want to store the name, address, city, state, zip, amount owed and a date. The programmer should write down all the different fields he intends to store and how many bytes are to be included in each string field.

Remember that the F/M program will be handling one record at a time. As each record is entered into the data base, the program will prompt the user through a mask to enter each of the fields. Although there are no limits within CSUB or BONES on how many fields can be entered, there shouldn't be more than can be displayed on the CRT. This problem can be easily overcome, but the programmer learning CSUB probably should start with a simple situation for the first CSUB program.

8.1.1 Define Index

After all the fields have been decided upon, the programmer must decide which field is going to be used to access the information. BONES allows only a single index (the field that will be used to access each record). If we are using our hypothetical customer list, we would probably use the customer name as the index. An alternate method would be to assign a number or code to each customer and access the records through this code. This field will become the index.

The index doesn't have to be included in the information being stored in the main file because the index will be stored separately in the keyfile. However, it is generally a good programming practice to keep the index in the main file in case something happens to the keyfile and the records need to be resorted.

The index will be stored in the keyfile that is created using the name of the main file plus some letter or character that distinguishes the keyfile from any other keyfiles which might be associated with the main file. The index will be the name, number, or code that is typed in by the end user when he wants to access a record.

If the user wants to enter a new record, the procedure, from the program point of view, could be illustrated easily. First, the user will select the option that indicates the user wants to create a new record. Next the user will be asked by the program for the index. The program will then (through FNH) search the keyfile and make sure the index is not a duplicate.

If there is no duplicate record, FNH will sort the keyfile and insert the new index in the correct place. Associated with that index entry will be a number pointing to a particular record in the main file. This number is returned from the FNH call and is passed by the programmer to FNG to write the actual record into the main file after all other fields that comprise the record are input. At this point the program will return to the first prompt and await the next command.

The programmer should be able to identify which field would best be used as the index in the situation he wants to program. This decision is very important and affects not only the way the entire program will operate but also the way the user will interact with the program.

The length (number of characters) in the index should be determined at this time. If the index is a four digit number, the index length would be 4. If the index is a name, the length would probably be 2Ø or 3Ø characters. Once the index is chosen the programmer may move on to the next step.

8.1.2 Numeric or String

Each of the fields within a record must be examined to decide if the input should be a string or numeric. Space must be reserved in B$ as well as B() to contain not only the records but also the masks and other transitory information. Most fields will be easily determined.

Dates can be input as a string or as a numeric. Generally, if the date will NOT be used as an index or will not be used at a later date to sort the file, it should be input as a numeric. However, if the date IS to be an index or is to be used as a field to sort records, the date should be input as a string. All dates input as strings are converted automatically to a YYMMDD format so the dates are in a form that is easy for the programmer to manipulate.

8.1.3 File Structure Sheet

The next task is to fill out the File Structure work sheet (See Appendix B). This sheet organizes all the information that has been decided upon, based on the previous discussion. This sheet asks for the name of the main file, a descriptive name, the programmer's name and a revision date. Once this information has been entered, the programmer specifies if the file is to be sequential or keyed access. If the file is keyed access (as in this case), a second file will be created that will be called the keyfile.

The programmer should enter the total number of numerics to be stored in the main file, as well as the string length. The string length is computed by adding all of the lengths of the individual string fields together. The number of records per block is computed by dividing 512 (double density) by the total of the number of numerics times the bytes required to store a single numeric, plus the total string length, plus 2 if the string length is under 255 or 3 if the total string length is greater than 255.

If the file is to be keyed access, enter the length of the index. This is the number of bytes that will be used to store the index. The index should have a name by which it can be referred, such as "account number". The last item in the heading of the File Structure sheet is the key file name, which is generally "K" for the first keyfile and is always a single character in length.

At this point the programmer should be ready to enter the variable descriptions. The numerics should be entered in the order they will appear in the file.

The strings also should be entered. It is generally best to let the index be the first string. Enter the string positions under the label "Strings", such (1,30), and the number of bytes in each string field followed by a description of what the variable is to be called.

After all the variable information is entered, the last items to be addressed are the programs which access the file. This is not a problem when only one program (F/M) is accessing a set of data but becomes very important when many programs access the same data files.

At this point, all the fields of each record should be defined completely. Also, the index should be decided upon and documented in the File Structure work sheet. If this information is clear to the programmer, you are ready to proceed to the next section.

8.2 Masks

The next step is to define and create the mask. Remember that the screen (under a F/M program) will show a record exactly as it "appears" in the file. This means you must draw a mask that describes each field within the records. The mask will be displayed by the program and will remain on the screen even if multiple records are displayed. The records will be displayed in a logical way so that the user can easily determine what information the program is acting upon at any one time. Consult the examples if the purpose of a mask is not clear.

8.2.1 Mask Sheet

The Mask Work Sheet (See Appendix B) is designed to help the programmer lay out the mask before it is created by the MASKEDIT program. The mask sheet asks for the mask name, the program name, the date, a descriptive name of the program and the mask length. The mask length will not be known until the mask has been created and the program MASKEDIT gives the length of the mask. The mask name is ALWAYS made with the program name and an "&" added to the end. Therefore, any file on a CSUB based program that has an "&" on the end of the name is a mask.

The Mask Sheet is divided into 25 lines with 80 characters in a line. This grid allows the programmer to quickly define the position of each field on the CRT. The description of each field should be entered and a place left for the actual information which will be displayed by CSUB. In this way, each field can be fully defined for both the mask and the input field so that no trial and error techniques have to be used.

8.2.2 MASKEDIT

MASKEDIT is the mask edit program. MASKEDIT is used to create or modify a mask. MASKEDIT should be run at this time to create the mask for the example program. The use of the MASKEDIT program is described in Section 9.3. MASKEDIT allows the programmer to type in his mask so that it can be stored as a file. The file will be read by BONES (or whichever program needs it) to be displayed at the appropriate time to aid the end user in entering the information into each of the fields.

8.2.3 Mask Size

The mask size will be very important when the programmer begins writing the program. B$ will have to be DIMensioned large enough to hold not only all the string information, but also the mask or masks concerned with the program.

The mask size is printed when MASKEDIT is given a command to store a mask on the disk or the "L" command is executed. This size should be recorded on the Mask Sheet for future reference.

8.3 FNE Inputs

The FNE inputs are the "mask prompted" inputs. Inputs from FNF are always prompted by the prompt string located in the FNF DATA statements. On the other hand, FNE inputs have no prompt in their DATA statements. FNE inputs rely on a mask having been generated and displayed prior to the input to prompt the user on what information is to be entered at any one certain time.

FNE inputs can be taken at any position on the CRT except lines 2, 3, and 4 which are reserved for FNF inputs and error messages. The position of the inputs and several other factors are controlled by the values in the FNE DATA statement table. The DATA table is programmed from the FNE DATA TABLES work sheet which will be "filled out" in this section.

8.3.1 FNE Resembles File Structure

In the case of a F/M program such as BONES, the FNE DATA table will be essentially a mirror image of the actual file structure. This makes the FNE DATA TABLES work sheet seem redundant for a F/M program. This form should still be filled out completely so that the programmer can remember at a later date how the table was generated. When more than one file is being accessed, this work sheet becomes very important.

This work sheet, upon completion, should look essentially like the file structure sheet completed in a previous section.

8.3.2 Field Characteristics

The most important feature of the FNE DATA TABLES work sheet is that it lets the programmer decide ALL the parameters of each item to be input at this time. The programmer can look at each variable and decide where on the CRT he wants the items to be entered. Here formatting is specified, as well as several other characteristics described in Section 5 (FNE DATA TABLES).

When the programmer begins writing the program, he will not have to be concerned with the validity of the information input since he knows that the information input will conform to the parameters specified by the DATA statements. Another advantage to the programmer is that all the parameters can be decided upon before the programmer even begins to write the programs. When the program is written, the programmer can take a more structured approach to writing, since he/she will only have to decide which variable is to be input and call FNE to do the "work".

8.3.3 FNE DATA TABLES Work Sheet

The work sheet (See Appendix B) can be used to decide and record the parameters of the FNE inputs. The programmer should fill out this form at this time. All of the fields are discussed in section 5 (FNE DATA TABLES). Information to be entered includes the type of the input (numeric, string, string-number, or date), the line number and prompt character, the position on the line, the number of prompt characters to be displayed, the low and high range of the input, the variable subscript number, and the description of the variable.

The information in this work sheet will be tranforme~ directly into a set of DATA statements to be located at line number 51ØØØ. The information will be entered into the program in a later step.

8.4 Display, Input, and File Variable Map

The Display, Input, and File Variable Map (See Appendix B) is used to define and map all variables stored in internal memory (B$ or B()) of a CSUB program. Since B$ and B() can have tens or hundreds of variables from one or more files, this form is filled out to help keep all variables that will be going through B$ and B() separate and distinct from one another.

The Display, Input, and File Variable Map can generally be filled out at this time. The only exception is if the FNF inputs require B$ or B() assignments. Normally, FNF inputs are used to control the flow of the program and don't generate variables that need to be stored. Most FNF inputs appear as follows:

```
Is Information Correct? (1=yes, Ø=no)
```

The B(Ø) variable is generally used for inputs of this nature, since the variable can be used over and over for taking the users response to program flow questions. If other inputs are taken from FNF, this space must be accounted for in the Display, Input, and File Variable Map work sheet.

In the program BONES, B(Ø) is used for transitory inputs concerned with program flow. The only B$ reference in BONES is the three positions to take the delete code (DEL) from the user when the user wants to delete a record. A delete code is used so that the record won't be deleted by accident.

Care should be used when using variables more than one time (such as B(Ø)). If B(Ø) contains a date from a previous input (12318Ø), and the current prompt is a request for a small number (INPUT 1=YES, Ø=NO), a format error may result when CSUB tries to redisplay the date as a single digit integer.

In some programs it may be more logical for the programmer to advance to the next section and define the FNF prompts and inputs before filling out the Display, Input and File Variable Map. If the program is to be a simple F/M program, the form can be filled out at this time. FNF requires much thought on the programmer's part, because the FNF prompts and inputs are very important to the end user and how they interreact to the program.

8.4.1 FNE and FNF Variables

The Display, Input, and File Variable Map sheet can be filled out at this time. Start by entering the heading information, which includes the program name, the descriptive name, the programmer's name and date of last revision.

At this point the variables can be defined. Remember, every variable that is used internally in the program and is input from the CRT or is to be displayed on the CRT or that is coming from or going to a file must be recorded on this sheet. Start with the numeric array variables (B()) and fill in the subscript number and the description of the variable. B(Ø) should be the first variable and is generally used for non-specific inputs as described in the previous section.

If you have more than 11 numeric variables (B(Ø)) through B(1Ø)), you will have to remember to DIMension the B array when you write the actual program. After you finish with the numeric variables proceed to the string variables. Here you will define the position within B$ of where the variable will be, the number of bytes each variable takes, and the description of the variable.

Any other variables used should be assigned at this time, such as the delete code variable previously discussed. These variables are generally placed at the end of the file variables and before the mask variables start.

8.4.2 Mask Variables

The mask or masks are generally placed at the end of B$. Define the position of your mask or masks in the Display, Input and File Variable Map work sheet. You must remember to DIMension B$ for the total number of characters needed, including all string variables, miscellaneous variables (delete code), and all masks. The mask or masks will be read into B$ by FNG and will be displayed by FNM.

8.5 Program Flow (FNF)

This section describes how program flow is controlled. The FNF function is normally used to control the flow of the program but there is nothing to keep the programmer from using FNE if he/she desires. The FNF inputs have the advantage of always displaying a prompt for input. Also, because they always display the prompt on line 2 of the CRT and the input on line 3, they position themselves in a "familiar" location that helps the end user figure out what is "going on" in the program.

8.5.1 ESC or ^B

The ESC key and a control B (^B) have been tied together in CSUB so that the function is the same when either key has been pressed. The purpose of these keys is to allow the user a chance to back up from one entry to another.

The programmer must examine every input to determine if the user wants to "back up". This is done by checking to see if the value returned from a FNF or FNE input is a 2. If the value is a 2, the programmer should make a decision about where would be the logical place to "back up" to. This feature is extremely useful to the end user, since he has the ability to change mistakes even after he has made an entry.

An example of how to determine if ESC has been entered would appear as follows:

```
10 X=FNF(1)\REM DO THE FIRST PROMPT
20 IF X=2 THEN END\REM END IF ESC OR ^B WAS ENTERED
```

8.5.2 ADD, INQUIRE, DELETE, RENAME, PRINT

This section gives an overview of the sections that will follow and guides you through the BONES program and describes how BONES controls the flow of processing. This should give the programmer enough information to design his own program.

The first prompt in the BONES program appears as follows:

```
ENTER 1=ADD, 2=INQUIRE OR CHANGE, 3=DELETE, 4=RENAME, 5=PRINT
```

This prompt is the major branching prompt that allows the user access to the 5 main sections of the BONES program. If the user presses the ESCape key or a ^B the effect would be to "back out" of the program or CHAIN to the program that originally branched to BONES. Each of the options will now be discussed individually.

8.5.3 ADD

This option allows a record to be added to the main file. The program asks for an index and then continues by asking for each of the additional fields associated with this record. Upon entering the index, the index is passed to FNH to assign a new key which points to a record in the main file. This record is then written using FNG. The index is stored in the keyfile as well as the main file.

When the option is selected the cursor immediately goes to the first field (the index) and waits for the input. Once that field is input the cursor goes to the next field and so on until all fields are entered. At anytime the user can back up from one input to the previous one. When all the entries are completed, the program prompts the user with a FNF input that says:

```
ENTER FIELD NUMBER TO CHANGE (Ø TO RECORD)
```

The user now has the opportunity to correct any field that may not be correct. If everything is correct the user can record the information by entering a "Ø". Once a "Ø" is entered the program records the data and branches to the first prompt.

The program flow in this section is controlled initially by the FNE inputs for the selected data fields. If the user presses "ESC" on the first input (the index), the program assumes the user changed his mind or didn't want this option and branches back to the first prompt.

Once all of the entries are made, program control is passed to the FNF prompt which allows the user to change any field. Again, if an ESC is detected, the program assumes the user wants to back out of this option and control is passed to the first prompt.

8.5.4 INQUIRE OR CHANGE

When this option is selected the program immediately calls a subroutine that is used to recall one record. The first prompt in this subroutine is:

```
ENTER INDEX
```

If the ESC key is pressed the program assumes the user doesn't want to continue with this option and RETURNs from the subroutine where control is transferred immediately back to the first prompt. If the user does not enter a valid index, an error is returned from FNH, X is set to "2" and the program RETURNs from the subroutine and branches to the first prompt.

If the user does enter an index that is found by FNH, the record number is passed to FNG to read that record into B$. At this point the subroutine RESTOREs to 51ØØØ and displays the record on the CRT using FND and RETURNs to the INQUIRE OR CHANGE section.

The program now calls FNF to do the second prompt:

```
     ENTER FIELD NUMBER TO CHANGE (Ø TO RECORD)
```

Notice that this is the same input that was executed under the ADD section. The program checks B(Ø) for the field that was input and if B(Ø) is a "Ø" then the program branches to the record routine. If a field number was input the program allows that field to be changed. If an ESC was detected for this input the program branches to the previous input.

There is one special case when the program is allowing the user to change any field. The index cannot be changed without disrupting the key file so the program assumes that the index is the first field and doesn't allow the user to change the first field.

Once the program is instructed to record the information, FNG is called to write the information back. FNH doesn't need to be called at this point since the value in variable H hasn't been changed and still points to the appropriate record.

The INQUIRE OR CHANGE section is exited by pressing the ESC key when the prompt is requesting the index name.

8.5.5 DELETE

The delete sequence starts the same as the inquire or change section by calling a subroutine asking for the index name and returns the record in B$ if the record is found. The same error conditions apply for this section as for the inquire or change section.

After a valid record has been returned, the fourth prompt is displayed which asks for the delete code:

```
     ENTER DELETE CODE (DEL)
```

If an ESC is detected at this input the program branches back to the first prompt, otherwise a comparison is made between the three positions in B$ immediately preceding the mask location with the letters "DEL". If the comparison is false and doesn't match, the program DOESN'T do anything to the record and branches back to the first prompt.

If a match does occur, FNH is called to delete the index. In this case the information is still in the main file under the record number but the index is no longer in the key file and the program "thinks" the record is gone.

If a disk error occurs during this activity, the program assumes something is wrong because the program had already accessed the record to get to its present state. If a disk error occurs, the program aborts. This logic could be changed easily if the programmer doesn't like the present way of handling the error.

8.5.6 RENAME INDEX

The beginning of this routine is exactly like the two previous routines in that a subroutine is called to get the record in question. The index is saved in D8$ and the fifth FNF prompt is called to get the new index name. If an ESC is detected at this point, the program assumes the user has changed his mind and branches to the first prompt.

Otherwise the program calls FNH to rename the index then calls FNG to read the record in, changes the index that is stored in the main file and stores that record back on the disk. A message is then printed that tells the user the index was changed from "X" to "Y".

8.5.7 PRINT

If this option is selected from the first prompt, an FNF prompt will be printed that asks:

```
     PRINT O)NE OR A)LL
```

The purpose of this input is to show the use of the TYPE 3 input which allows the user to enter only an "O" or an "A" in response to the input. Of course, the user can still elect to "back up" by pressing the ESC key and control will pass back to the first prompt. If a valid response was entered the program branches to one of two locations.

If "O" was entered then B(Ø) will contain a "Ø". "1" is added to this value so the ON GOTO statement can be used. This response will cause the program to call the same subroutine to retrieve one record. Once the record is passed successfully to B$ a subroutine is called to print a heading on print device #1 and then the record is printed followed by a formfeed character. At this point the program passes control back to the first prompt of the program.

If "A" was entered, B(Ø) will contain a "1" and the program will branch to the routine to print all of the records. The key file is OPENed to read the first three numerics. The second numeric tells us the number of records in the keyfile. At this point FNG1 is called to OPEN a channel for the upcoming FNG calls. The heading is printed by a subroutine and a FOR-NEXT loop is established from 1 to the number of records.

The keyfile is now read to get the first record number. This value is then passed to FNG to recall the record into B$. The record is displayed on the CRT and a subroutine is called to print the desired parts of the record. Anytime the line counter (F) reaches the desired number of lines per page, a formfeed is output to the printer and a new heading is printed. The page number is incremented as well.

All the records are processed in this manner. The program then calls FNG1 to CLOSE the channel previously OPENed. Another formfeed is output to the printer and the program again branches to the first prompt.

8.6 Changing BONES

If you are changing BONES to be a F/M program for a different file structure than as delivered, you should have filled out all the work sheets and should be ready to make the programming changes. You should keep the work sheets handy and make a paper listing of the BONES program. Start by changing the variables defined at the first of BONES. These varible~ are REMarked fully but will be described here to reinforce their meanings.

B$ and B() should be DIMensioned to fit your requirements. If there are fewer than 1Ø numerics in your program, B() will not have to be dimensioned. DIMension any other variables you need at this time.

Next follow the file reference variables. The description of these follow:

D2$ is the name of the main data file. This name will have the name of the keyfile added to it to determine the name of the keyfile.

D3$ is the name of the keyfile. This variable should be only one character in length. Remember this variable will be added to D2$ to arrive at a keyfile name.

D4$ is the name of the program. In this case the name is BONES but you will probably want to give your program a different name.

D6$ is the name of the program that called BONES originally. This variable is currently set to MENU but can be anything you want. This value will be passed to FNX, the exit function, to assure an orderly transition from one program to another.

D5$ is the descriptive name of the program you are writing. In this case, BONES is described as a "GENERAL PURPOSE FILE MAINTENANCE" program. This title will be printed on the CRT by the exit function FNX.

D7$ is the descriptive name of the program that originally called the present program. This name will be printed on the CRT by the exit function FNX.

The next set of variables definitions affect where items are placed in either B$ or B().

D3 is the beginning position of the index within B$. This position will be passed to all functions requiring the position of the index in B$.

D4 is the ending position of the index within B$.

D8 is the beginning position of the mask in B$. In this example there is only one mask.

D9 is the mask number. Additional masks would each be assigned a separate mask number.

D6 is the position in B$ to read in the string data from the file. If more than one file is to be read into B$, additional starting positions will have to be defined so that the information doesn't overlap.

D7 is the position within B() to read in the numeric file data. Again, if more than one file is to be read into B(), additional positions will have to be defined for each file read into B().

D5 is equal to the number of fields in each record.

The following two variables define the drive numbers associated with the program and data files. These assume that all data files will be on the same drive and that the mask will be on the same drive as the programs.

D is the drive number of the programs and the mask.

D1 is the drive number of the keyfile and the main data file associated with this program.

The following are miscellaneous variable definitions.

P is defined as the print device number. This is generally device number 1.

F1 is a flag that is set to the number of records to be printed on one page on the print device.

F2 is the page counter and is initially set to 1.

C$ is the definition of characters allowed under the type 3 input that occurs when the print option is selected.

Once these variables have been defined, the only other changes required are in the FNF and FNE DATA tables. If BONES is to be left as strictly a F/M program, probably no changes will be required for the FNF DATA statements, since they are general in nature. Often you may want to change the prompt that asks for the index to one that asks for your index by name such as:

```
ENTER CUSTOMER NUMBER
```

The FNE DATA statements can be written directly from your FNE work sheet. DELETE the FNE statements in BONES and replace them with your own. Once these changes have been made and the program SAVEd, you will be almost ready to try your new program. The only thing left to do will be to create the files needed. This is accomplished by selecting option 1 from the FILEWORK program provided on the CSUB diskette. Refer to section 9.4 (FILEWORK) for information on how to use this program.

8.7 Converting From CSUB 5.2 to VI

Every attempt was made to keep CSUB VI as close as possible to version 5.2. However, there are several changes that must be made to take advantage of the new features and increased speed of version VI.

Under CSUB VI, CSUB has been renumbered so that all functions are located in low memory (line numbers 1 to 999) instead of high memory (line numbers 55,ØØØ and up). This eliminates the need for CSUB to be APPENDed to each program and therefore saves one LOAD each time an option is selected from a menu.

Now CSUB VI remains in the memory and APPENDs the needed program at location 1ØØØ. This change allows a 4Ø% increase in speed of CHAINing from one menu option to another.

In addition, the masks have been given a new format. Under CSUB 5.2, all masks contained the cursor addressing information needed to position the mask. This caused a great deal of trouble when masks had to be converted from one CRT to another. Under VI, the masks contain only the relative position to print each field of the mask and a new function is defined (FNM) to position the cursor using the PRINT@ statement of **uDoZbaZic.**

For each and every program written under CSUB 5.2, the following changes must be made to run under release VI:

1. Load the program and delete all line numbers except the DATA statements located at line numbers 50000 and 51000.

2. Save the DATA statements in a temporary file.

3. LOAD the program into internal memory again.

4. Delete the following items from the program:

   ```
   APPEND CSUB
   GOSUB 65000 and GOSUB 65100
   DATA statements at 50000 and 51000
   All FILL D2,X statements
   ```

5. Optionally, you can change all !Z$ (clear the screen) to the CLS statement and all FNA function calls (position the cursor) to PRINT@ or !@ statements.

6. RENumber the program to start at line number 1000. The command would appear as follows:

   ```
   REN 1000,10
   ```

7. APPEND the DATA statements (at 50000 and 51000) previously saved to the temporary file.

8. Optionally, if you are using CSUBEN (with the new FNF format), remove the minus signs from the first field of the DATA statement and remove field 3 (always a zero under CSUB 5.2). If you do not want to modify these lines, use CSUBEO (with the old format) so that your programs will continue to work properly.

9. If you use the secondary error messages at 55000 and 55200, APPEND HGERROR from the CSUB disk on to your program.

10. SAVE the program under the original name.

11. Change all mask displays to use FNM (the mask display function). Run the program CONVRT on all old CSUB mask files to convert them from the old mask format to the new mask format.

For the CSUB program itself, you will have to modify the first line so that A$ is equal to the name of the program you want to branch to when CSUB is loaded and executed.

PROGRAM DESCRIPTIONS

This section contains the program descriptions for CSUB and its associated utility and demonstration programs.

9.1 CSUB

CSUB is the set of functions (actually not subroutines) which accomplish the actions described by this manual. The CSUB file on the disk may contain any one of the four CSUB files included on the CSUB disk. Any one of these files can be LOADed into RAM and then SAVEd into the CSUB file. Each of the four CSUBs has its own characteristics and special purposes as described below.

9.1.1 CSUBEO

CSUBEO is the version containing the line editor (E) but uses the old (O) format for FNF inputs. When CSUB was first devised, an attempt was made to include all DATA statements for FNE and FNF at the same location (50000). However, after further development, the FNE and FNF data statements were separated (FNF at 50000 and FNE at 51000).

Two remnants of the original CSUB became illogical for the new format that remained in CSUB through Release 5.2. The data type (first field of the DATA table) for FNF required a minus sign in front of the data type number. In addition, the FNF DATA statements table required an extra field that was not used and was always set to zero (0). This field was the position field, but under FNF the position of the input is always on the third line, except if the position is modified using the C2 or C3 variables. CSUBEO still supports those programs written with these features.

9.1.2 CSUBEN

CSUBEN is the version that contains the line editor (E) but uses the new (N) format for FNF inputs. This version is the primary version of CSUB and should be used in most situations. CSUBEN occupies the most memory of any of the three CSUBs.

Under CSUBEN, FNF DATA statements no longer need to have a negative sign before the first field and the position field has been eliminated.

The only difference between CSUBEO and CSUBEN can be found at line numbers 148 and 236. In line 148 the READ A2 has been removed from the list of variables read in the READ statement. In line 236, the statement taking the INT(A4) has been modified to ABS(INT(A4)).

9.1.3 CSUBW

CSUBW is the version of CSUB that does not contain a line editor. This version is used when the program is so large that CSUBE can not be used. This version of CSUB uses the new format for FNF inputs.

9.1.4 CSUBI

CSUBI contains only the Input section of CSUB. This version is to be used when there is no file accessing to be done under the CSUB file accessing routines. This version of CSUB uses the new format for FNF inputs.

9.2 BONES

BONES is a demonstration keyed access file maintenance program. This program can be used to very quickly establish a F/M program for any new data base by merely changing a few parameters. Program and file names should be changed at the beginning of the program. A new set of DATA statements (FNE at 51ØØØ) should be written to accurately reflect your file structure. The FNF DATA statements may need to be changed if they do not reflect what you want the program to do.

See Section 8 for more information on the BONES program.

9.2.1 BONE

The file BONE is not a program but the main (FNG) data file for the demonstration program BONES. This file is in place on the CSUB disk so that the user can quickly try the BONES program once the cursor addressing features have been established by the CONFIG program. You can use FILEWORK to examine the contents of the BONE file if you have questions concerning how the file is set up.

9.2.2 BONEK

Again, BONEK is not a program but the key file (FNH) for the BONE file and the BONES program. This file (as well as BONE) was created using option 1 (Create Key File) from the FILEWORK menu. The contents of the file can be printed using option 4 (Print Any File) from the same program.

9.2.3 BONES&

This file contains the mask that is displayed when BONES program is run. Notice this file follows the rule that the mask file should be identical to the program file name except that an ampersand (&) is added to the program file name. To examine the contents of this file, use the MASKEDIT program. The file name is BONES& and the mask number is 1.

9.3 MASKEDIT

The MASKEDIT program is used to create, edit, store on disk, retrieve from disk, and print masks. Three modes to MASKEDIT are; command, enter text, and enter lines. In the command mode, any of several commands may be given MASKEDIT on which to operate. The command mode is used to initiate moving the cursor, adding to the mask, deleting fields from the mask, moving fields within the mask, sorting the mask for proper displaying, and printing the mask on the printer.

In the enter mode, text may be entered as part of the mask. After the command is executed, all printable characters on the keyboard may be entered into the mask at the current cursor position.

The enter line mode is used to draw lines and/or boxes on the CRT (if your CRT has this capability). Once in the line drawing mode, the cursor is moved horizontally or vertically by the cursor control keys and a line or box is drawn following the cursor movement.

Either text or lines may be entered into the mask in normal or reverse video (again, if your CRT supports this feature).

Line two is used to keep the user informed of the present cursor position and the mode of operation. The cursor position and mode message will always display in reverse video when the "R" command is executed and in normal video when the "N" command is executed to indicate the current status of the program. Since line two is reserved for FNF prompts, fields should never be placed there.

When the MASKEDIT program is RUN, the first prompt to appear on the CRT reads as follows:

```
    MASK NAME, DRIVE NUMBER:
```

The user should enter the name of the mask, a comma and the drive number. The comma and drive number may be omitted if the file is on the default drive. If the file does not exist, the program will branch immediately into the command mode, clear the screen, and print the following message on the CRT:

```
    1, 1   ENTER COMMAND
```

If the mask name is an existing file name on the specified drive, the program will respond with the following prompt:

```
    MASK NUMBER:
```

Since this program can handle multiple masks in the same file, this prompt is requesting the number of the mask you want. If you have only one or want the first mask, enter a "1".

9.3.1 Commands

MASKEDIT provides commands to perform several functions. The movement of the cursor is controlled by several commands allowing the cursor to be moved to any position on the CRT. The commands are summarized as follows:

1	Move cursor down and left.
2	Move cursor down.
3	Move cursor down and right.
4	Move cursor left.
5	Enter text.
6	Move cursor right.
7	Move cursor up and left.
8	Move cursor up.
9	Move cursor up and right.
Ø	Save the mask on disk.
.	Enter lines.
D	Delete a field.
R	Reverse video.
N	Normal video.
P	Print a mask.
S	Sort a mask.
M	Move a mask field.
E	Edit a mask field.
Q	Quit to **baZic.**
J	Jump to a screen position.
L	Length of mask (# of bytes in mask string).

9.3.2 Cursor Commands (1-4, 6-9)

The cursor commands are arranged so that a numeric keypad may be used to enter the direction of cursor motion. Using the center of the keypad as a reference, you may move the cursor up, down, to both sides, and to all four diagonals by pressing the key that corresponds to that direction. If your CRT does not have a numeric keypad, you may use the number keys on the keyboard. In this situation the command numbers won't be as easy to use.

The following number keys cause the cursor to move in the direction indicated:

KEY	DIRECTION OF MOVEMENT
1	down and left
2	down
3	down and right
4	left
6	right
7	up and left
8	up
9	up and right

9.3.3 Enter text (5)

The cursor commands are used to move the cursor to the proper position on the CRT. To enter text at the cursor position, press the "5" key. Upon pressing the "5" key the program will enter the text mode whereby all succeeding characters pressed are entered as text. The enter mode is terminated by a carriage return. The enter text mode is shown on the second line of the CRT by the following message:

```
     1, 1  ENTER TEXT
```

9.3.4 Line Drawing (.)

The "." key is used to begin and end lines. When a line or box is to be drawn on the screen, position the cursor at the beginning of the line or box and press the "." key. At this time you should see the following message on line 2 of the CRT:

```
     #, #  LINES
```

Now move the cursor horizontally or vertically and a line will follow the cursor. Another "." will terminate the line drawing mode. Remember, the line will be drawn only where the cursor has been and not at its current position. Since there are so many decisions to be made in the line drawing mode by the program, the cursor should be moved slowly, so that the program can keep up with the user.

If you want a line to be drawn in reverse video, first enter the line drawing mode by using the "." command. Next press the "R" command for reverse video. Draw the line or lines as you want using the cursor movement keys. Before terminating the line drawing mode, use the "N" command to return to normal video. Now the line drawing mode can be exited by the "." command.

If a mistake is made while drawing lines, push the space bar and use the cursor motion keys to "back up" over the incorrect lines. This will cause the line to be erased. When the erasing is finished, push the space bar again to exit the space drawing mode. If you are drawing reverse video lines, remember to place the program in the normal video mode (N key) before printing spaces or else the spaces will print in reverse video.

To draw a box, position the cursor to one corner of where the box is to be drawn. Enter the line drawing mode by pressing the "." key. Using the cursor movement keys, draw the box. For the last corner, you will have to "turn the corner" of the box before the second "." command is issued to terminate the line drawing mode.

9.3.5 Record the Mask (Ø)

The "Ø" key is used to record the present mask on disk. Before saving a mask on the disk, the user should generally perform a sort using the "S" command to optimize the display of the mask.

9.3.6 Delete a Field (D)

The "D" (DELETE) command is used to delete fields from the mask. When the "D" key is pressed, the program responds with a prompt on line 2 that appears as follows:

```
     1, 1   'B' TO SPACE, 'D' TO DELETE, 'Q' TO EXIT
```

Pressing the "B" key will space you from one field to the next in the order in which they are stored internally. Use the "B" key to advance to the field which you want to delete. The "D" key is used at this time to delete the field following the cursor. If you decide you do not want to delete the field after all, use the "Q" key to quit.

9.3.7 Move a Field (M)

The "M" (MOVE) command is used to move a field from one place on the CRT to another. Once the "M" command is pressed the second line will have the following message:

```
     1, 1   'B' TO SPACE, 'M' AT FIELD TO MOVE, 'Q' TO EXIT
```

The "B" key is used to space to the field to be moved. The M key is again pressed to specify the field when the cursor is at the beginning of the field to be moved. Once the proper field has been reached and the "M" key has been pressed, the following will appear on line 2 of the CRT:

```
     MOVE CURSOR TO NEW POSITION, 'M' TO MOVE, 'Q' TO EXIT
```

Now move the cursor (using the numeric keypad) to the position on the CRT to which the field is to be moved. Once at this position the "M" key can be used to move the previously defined field to the present location of the cursor. The "Q" key may be pressed to quit the command and return to the command mode if a decision is made to not move the field.

9.3.8 Edit a Field (E)

The "E" (EDIT) command may be used to edit a field either during text entry or at anytime while using the MASKEDIT program. When the edit command is executed, the following display will be on line 2 of the CRT:

```
     1, 1   'B' TO SPACE, 'E' AT FIELD TO EDIT, 'Q' TO EXIT
```

The edit command works similarly to several of the other commands in that the "B" key is pressed to move the cursor to the beginning of the field to be edited. Once at the proper field the "E" key is again pressed to signify that the present field is to be edited. The following display is shown on line 2 when the "E" key is pressed and the program is in the edit mode:

EDIT

The edit commands are similar to the **baZic** line editor commands. These commands are all executed by holding the control key down at the same time the command key is pressed. All control keys are represented by the "up arrow", followed by the specific letter (^A). The control keys and their meanings are summarized in the following chart:

^A	Move cursor right.
^H	(backspace) Move cursor left.
^Z	Delete character at cursor position.
^Y	Toggle insert mode on/off.
^N	Display to end of line and return cursor to beginning.
^B	Quit edit and leave field unchanged.
^G	Copy to end of line and terminate edit mode.
<CR>	(carriage return) End edit at present cursor position.

9.3.9 Jump to a Screen Position (J)

The "J" command is used to allow the programmer to specify the exact position on the CRT of where he/she would like the cursor to be. Upon entering the "J" command, the following display is shown on line 2:

```
JUMP      ENTER   ROW,COLUMN:
```

At this time, enter the row and column coordinates of the position on the CRT of where you want the cursor. Be sure to enter the row value first and to separate the row and column values with a comma. When the carriage return is pressed, the cursor will advance to the position specified.

9.3.1Ø Reverse Video Mode (R)

The "R" (REVERSE VIDEO) command is used to cause the program to enter into reverse video mode. In this mode, both text and lines entered will be in reverse video. When you are finished with reverse video, use the "N" (NORMAL) command to cause the program to return to the normal mode.

To draw lines in reverse video, first enter the line drawing mode using the "." command. Enter reverse video by using the "R" command. When you are finished drawing the lines, exit the reverse video mode by the "N" command and then exit the line drawing mode by pressing the "." key again. See Section 9.3.15 for more information on the "R" command.

9.3.11 Sort a Field (S)

The "S" (SORT) command is used to sort the mask. When fields are entered into the mask out of their normal "top-down" order, the mask will display in the same order the mask was entered. By using the sort command, the mask can be arranged so that it always displays in a top-down, left-to-right manner.

As a general practice, the mask should always be sorted prior to saving on disk with the "Ø" command and after every deletion. Two sort commands may not be issued one after another.

9.3.12 Print a Mask (P)

The "P" (PRINT) command is used to print the mask on the printer. Upon pressing the "P" key, the program asks for the device number with the following prompt:

```
PRINT ON DEVICE NUMBER:
```

The response should be a legally defined device on your system but is generally device 1. After answering the previous question, the program will ask if you want the grid suppressed on the printout with the following prompt:

```
SUPPRESS GRID (YN)
```

If you answer "Y" for yes, the mask will be printed as it appears on the CRT. If you answer "N" for no, the mask will be printed with a grid showing the location of all mask fields.

9.3.13 Quit to baZic (Q)

The "Q" (QUIT) command is used to quit the program and return to **baZic.** Upon entering the "Q" command, the program will ask if you want to quit by displaying the following prompt:

```
DO YOU WANT TO QUIT (YN)
```

If you press the "Y" key for yes, the program will terminate and you will see the READY message of **baZic.** If you press the "N" key for no, the program will return to the command mode.

9.3.14 Length of Mask (L)

At any time during the editing of a mask (while the program is in the command mode) the "L" command can be executed to determine the length of the present mask. The value returned will be the number of bytes that the mask occupies. The value will be displayed and the following prompt will be given:

```
PRESS RETURN TO CONTINUE
```

To continue editing, press the return key. When a save command is executed (Ø), the program will report the length of the mask automatically without having to use the "L" command.

9.3.15 Miscellaneous

If you want to reverse the video on a field already created, position the cursor to the beginning of the field and press the R key. Move the cursor to the end of the field and press the N key. Now use the sort command (S) to sort the mask and the selected field will be converted to reverse video.

9.4 FILEWORK

The FILEWORK program is used to manipulate CSUB type files. The program is menu driven, allowing the user to select the option from the menu. The program is not written under CSUB itself but appears very similar to a CSUB program. Upon LOADing and RUNning this program the following menu will be displayed:

```
                    FILE UTILITY PROGRAM
     Input Option Number
     #

     1. Create Key File
     2. Create Sequential File
     3. Transfer Any File
     4. Print Any File
     5. Add Records To Key File
```

Option 1 is used to create a keyed access file pair (a file for FNG and FNH). Option 2 is used to create a single FNG type sequential file. Option 3 is used to transfer information from one file to another. Option 4 is used to print one or more records of any CSUB type file on the CRT or printer. Option 5 is used to expand a keyed access file and its keyfile when the number of records has expanded past the length of the file.

9.4.1 Create Key File

When this option is selected, the following display will appear on the CRT:

```
                       CREATE KEY FILE

     File Name
     Key File Name
     String Length
     Number of Numerics
     Key Index Length
     Number of Records
     Drive Number
```

The cursor will move to the first input and will proceed down the list as each item is entered. A control B (^B) can be entered at any time to back up to the previous input.

The File Name should have seven characters or fewer and is the name of the main (FNG) file you want to create. The Key File Name is the single letter that will be added to the main file name to arrive at the key file name.

The String Length is the length of all the string information to be stored in each record of the file. The program automatically adds the overhead bytes required to store a string, so the user of this program should enter only the number of characters required in the string.

The Number of Numerics is the number of numeric entries to be stored in each record of the file.

The Key Index Length is the size of the string used as the index in the keyed access file (key file).

The Number of Records is the number of records that you want in the file. The program will automatically create both files (key file and main file) large enough to hold the number of records specified.

The Drive Number is the number of the drive on which you want the files created.

As the files are being created the program prints the names and sizes of the files being created on line 2 of the CRT.

9.4.2 Create Sequential File

This option is used to create a standard FNG type sequential file. When this option is selected, the following display will result:

```
                    CREATE SEQUENTIAL FILE

     File Name
     String Length
     Number of Numerics
     Number of Records
     Drive Number
```

The File Name is the name of the file to be created. It, of course, must follow the standard rules for file names.

The String Length is the length of the string portion of each record in the file.

The Number of Numerics is the number of numeric fields within each record of the file.

The Number of Records is the maximum number of records to be entered in the file. This value controls the size of the file to be created.

The Drive Number is the drive number on which the file is to be created.

As the file is being created, the program prints the name and size of the file being created on line 2 of the CRT.

9.4.3 Transfer Any File

This option is used to transfer information from one file to another. This option works on all CSUB type files, including key files. When the option is selected, the screen is cleared and the following display is printed:

```
                    TRANSFER ANY FILE


   Source File Name , Drive #
   Destination File Name , Drive #
```

The cursor will again move to the first input and then to the second when the first is completed. The Source File Name and drive number should be entered. The Source File is the file from which you want the information transferred. Don't forget to separate the file name and the drive number by a comma.

The Destination File Name is the name of the file that you want the source file transferred to. If the Destination File Name does not exist, the program will automatically create the specified file. This name and the drive number should follow the standard file name and drive number rules.

9.4.4 Print Any File

This option is used to print the contents of a CSUB type file on the CRT or printer. This option can print one or all of the records in the specified file. Files with byte accessing cannot be printed with this option since they are not standard CSUB type files. When this option is selected, the following display results:

```
                    PRINT ANY FILE

   File Name , Drive #
   1-Printer  Ø-CRT
   1-One Record  Ø-List
```

The File Name and Drive # is the name and disk drive number of the file that you want printed. The second option is used to tell the program whether you want the information printed on the CRT or printer. Enter the appropriate number as shown in the mask. The last prompt is to indicate if you want only one record or the entire file (List) to be printed.

If you decide to print only one record, the program will prompt you for the record number.

When the requested information has been entered, the program will print the name of the file and the first three numerics on the second line of the CRT. The record or records specified will be printed under the file name. If the file is printed on the CRT the program will print about 1Ø records before stopping and printing a "PRESS RETURN TO CONTINUE" message. At this time you may press the RETURN key to see more records or the ^B key to return to the first menu.

9.4.5 Add Records To Key File

This option supports the expansion of a key file pair, if the number of records in the file has grown beyond the size of the file. This option does not work on sequential files because they are very easy to expand by just creating the file larger and using the file transfer option to move the records from the old file to the new file. The new file and key file must be on a separate drive than the files being expanded.

When the option is selected, the following display results:

```
                    ADD RECORDS TO KEY FILE


     File Name
     Key File Name
     Source Drive #
     Destination Drive #
     Number of Records
```

Enter the File Name and the single letter Key File Name. The Source Drive # is the drive you want to copy from (old file that is now too small). The Destination Drive # is the drive you want to create the new files on and must be different than the Source Drive #.

The Number of Records is the number of records you want the file expanded to hold. This number should be larger than the number of records in the file to be expanded.

9.5 CONFIG

This program is not supplied nor does it work with the DOS and CP/M versions of CSUB.

This program is designed to allow changing the features of **uDoZbaZic**, which are often different from one system to another. This program is generally the first program run on a system when it is initially installed. If the user does not know the correct response to any of the questions, the default conditions can be specified by simply entering a carriage return in response to the questions.

The program CCOPY is used to make the initial copy of the master diskette. This program must be on the same diskette as the program CONFIG.

The most important item to be considered is the configuration for the proper CRT. If the CRT you are using is listed, merely select the number of that CRT in response to the prompt under the CRT selection menu and the program will set up the system for your CRT automatically. If your CRT is not listed and you are a programmer, you should be able to set up **uDoZbaZic** for your CRT by following the instructions in the program.

In response to all requests for information by this program, the carriage return must be pressed either by itself to signify the default condition, or after making any entry in response to any input for the program to "know" that you are finished with your entry.

The first prompt to appear on the CRT will inquire if you need to make a copy of the program disk before you continue. The prompt will appear as follows:

```
DO YOU NEED TO MAKE A BACKUP COPY OF THE PROGRAM DISK? (Y OR N)
```

You must enter a "Y" if you want to make a copy. Any other response will allow you to continue with the program without making a copy. If you are making a copy, the next prompt will instruct you to place a blank disk in drive two of your computer. The program disk should remain write protected throughout the entire procedure. Your program disk should still be in drive one. The prompt will appear as follows:

```
    TO MAKE A COPY OF THE PROGRAM DISK, LEAVE THE PROGRAM DISK IN
    DRIVE ONE (1) AND PLACE A BLANK DISK IN DRIVE TWO (2).

    THE DISK IN DRIVE 1 SHOULD BE WRITE PROTECTED AND THE DISK
    IN DRIVE 2 SHOULD NOT BE WRITE PROTECTED.

    PRESS RETURN TO CONTINUE
```

When the return key is pressed, the program will first initialize the diskette in drive two and then copy the contents of the disk in drive one to drive two. The following message will be displayed while the copy is in progress:

```
THE COPY WILL TAKE BETWEEN 1 AND 2 MINUTES.  PLEASE BE PATIENT.
```

When the process is complete, the first prompt will appear again as follows:

```
DO YOU NEED TO MAKE A BACKUP COPY OF THE PROGRAM DISK? (Y OR N)
```

This time enter an "N" for no. The program will now tell you to store the original disk from drive one, and to change the disk in drive two to drive one. This procedure is followed so as to save the original disk and to place the copy made in drive two into drive one where it can be modified to work with your hardware. The message will appear as follows:

```
IF YOU HAVE JUST MADE A COPY OF YOUR MASTER DISKETTE, YOU SHOULD
REMOVE THE MASTER DISKETTE FROM DRIVE 1 AND STORE THIS DISKETTE
IN A SAFE LOCATION.  REMOVE THE DISKETTE FROM DRIVE 2 AND PLACE
IT IN DRIVE 1.  THIS DISKETTE WILL BECOME YOUR WORKING DISKETTE.
MAKE SURE THE WRITE PROTECT TAB IS NOT IN PLACE ON THIS DISKETTE.

PRESS RETURN TO CONTINUE
```

After the copy procedure has been completed, you may continue with the rest of the configuration process.

The sign on message is the next item to appear on the screen. This message will remain displayed until you press the carriage return key. Note the message also indicates that the **uDoZ** disk must be in drive one for this program to function properly.

The first prompt of the configuration section of this program will ask for the value of the PAGE byte. The program will first display the current value of the PAGE byte and give a short description of the purpose of this byte. The value of the PAGE byte is the number of lines to be displayed on the CRT before the display is "held" by the PRESS RETURN TO CONTINUE message.

This value is normally only needed by programmers and does not affect anything in the application programs. For most users, the value of 24 lines per page is correct and therefore the user should enter only a carriage return in response to this prompt indicating the default value is to be used.

The next value to be set is the verify byte. This byte controls whether **uDoZ** is to verify all information that it writes to the diskettes. It is recommended that the verify be left "on" as delivered. A verified write takes longer to perform than a non verified write, but the accuracy of information is better if the writes are verified.

If you do not want verified writes, enter a zero (Ø) in response to the verify prompt. If you want all writes verified, enter a one (1) or press only the carriage return to specify the default condition (writes verified).

The CONFIG byte is the next situation to be covered. This value "tells" **uDoZ** which drives in the system are quad capacity drives. The default assumes that there are no quad capacity drives in the system and that all drives are just double density single sided drives.

The program will ask eight questions in regard to the quad capacity drives. If you do not know which drives are quad capacity in your system, enter a return only in response to the eight questions. The system will work even if you have quad drives but specify them as double density only. The reverse situation will not work (if you have double density drives and you specify them as quad capacity).

Two questions are asked about each drive. The first question asks if the drive is a quad capacity drive and the second question asks if the drive has fast stepping capability. Generally, both questions about a drive should be answered with the same response. If the drive is a quad, it should have fast stepping capability. If the drive is not a quad, it should not have fast stepping capability.

In response to the eight questions, enter the correct answer. Enter a "Y" to indicate each drive that is a quad capacity drive. If the drive is not a quad capacity drive, enter a "N" or a carriage return only to signify the default condition.

After the four drives have been specified correctly, the next value to consider is the Display SYStem byte. This byte controls the listing of system level files during a CATalog of a disk. If the system files are to be displayed this byte should be a zero. If the system files are not to be displayed this byte should be a one. If you are unsure how your system is to be set up, select the default condition by pressing only the carriage return key.

The next item to be displayed on the CRT is a short note explaining the setting of the cursor addressing and clear screen information to allow **uDoZ** to communicate properly with your CRT. After reading this message, press return to continue, and you will see a short menu of CRTs for which the program "knows" the correct cursor addressing and clear screen values.

If your CRT is listed, select the number corresponding to your CRT and press the carriage return. The program will take care of all the "details" for your CRT. If your CRT is not listed, the "OTHER" option may be of value to you if you know the cursor addressi~ prefix, the offset values, and the clear screen sequence for your CRT.

If you do not know these values, you should not select the "OTHER" option. You should locate some knowledgeable person to configure your CRT for you.

Assuming your CRT was listed and you selected the correct option, the next item to consider is the turnkey command which is to be given to **uDoZ** whenever it is booted. The default turnkey command for **uDoZ** is to load **baZic** and then for **baZic** to begin executing the program called "CSUB".

This condition should be selected by most users by simply pressing the carriage return key in response to the prompt. If desired, a new turnkey command can be entered before pressing the return key. A new turnkey command should be entered only by a programmer who knows the consequences of the command entered.

At this point, all the necessary information has been gathered by the program and a testing phase is begun. The program will first clear the screen on the CRT and print a message on the first two lines of the CRT. If this has happened, you should enter a "1" and a return to indicate that the clear screen command is working properly. If you enter a "Ø" in response to this prompt, the program will branch back to the CRT configuration section to let you try again.

The program will draw a "looping rope" (prolate cycloid) on the CRT, followed by numbers and asterisks to signify the limit positions on the CRT. The right side and bottom of the CRT will be covered by the asterisks. The top and left side will have a number at every position to represent the rows and columns of the CRT. A message will be printed, explaining the display and a prompt will be shown at the bottom asking if everything is O.K.

If you enter a "1" at this point, the information gathered by this program will be written to the disk in drive one of the computer. If you enter a "Ø", indicating that some problem still exists, the program will branch back to the CRT configuration section to allow you to try again.

Once the correct cursor addressing and clear screen sequences have been established and the user enters a "1" in response to the last prompt, the program writes the updated information on the disk, clears the screen and prints a message indicating the configuration is complete, and reboots the computer. At this time you should be running your application program.

9.6 DUP (Diskette Utility Program)

This program is not supplied, nor does it work with the DOS and CP/M versions of CSUB.

This program is the basic utility program which allows the user to initialize and copy diskettes. The "source" disk or drive is always the drive that you will copy from while the "destination" disk is the drive that will be copied to.

To run this program the COPY option must be selected from the appropriate menu. The utility menu and prompt appear as follows:

```
                           DISKETTE UTILITY MENU
      PICK OPTION NUMBER (1-5)
      #

            1) Initialize
            2) Gross Copy
            3) Intelligent Copy
            4) Copy Files
            5) Compact Disk
```

If you have decided you do not want this program, you may return to the program that called this one by pressing the ESCape key.

At this time you should select the appropriate option number. Option 1 allows the user to initialize a diskette. Option 2 is used for a complete disk to disk copy where all data is transferred, regardless of directory entries or how much actual data is on the disk. Option 3 is a copy by directory where files are copied file by file based on the directory information and the user's choice.

Option 4 allows the user to copy a single file by its file name or a group of files as specified by a programmer. Option 5 is used to compress a disk when file deletions have left "holes" in the usable space.

9.6.1 Initialize

If you select Option 1 (Initialize), the first prompt for this option is:

```
                        *** DISK INITIALIZATION ***
      ENTER DRIVE NUMBER TO INITIALIZE
      #
```

If you decide you do not want to initialize a diskette, you may return to the DISKETTE UTILITY MENU by pressing the ESCape key.

If you are not familiar with the numbering of drives on your computer, consult your hardware manual and/or request help from someone who is familiar with your system. Generally, drive 1 is the left-most drive and drive 2 is the right-most drive. With the optional hard disk drive, drives may be assigned in a relative manner without regard to physical location. Be careful that you know in which drive you want to initialize a diskette.

After you have made your choice of drive, the initialize program will "look" at the disk you wish to initialize and determine if there are any files on the disk. Any files on the disk will be displayed on the CRT so that the user may decide if the initialization is to continue.

After viewing the partial directory of the disk, you will see the following prompt:

```
THESE ARE SOME OF THE FILES ON THE DISK. 1=INITIALIZE, Ø=DO NOT
#
```

If you enter "ESC" the program will return to the first prompt in the initialization section (ENTER DRIVE NUMBER TO INITIALIZE) and the disk will not be initialized. If you enter "Ø", the program will branch to the DISKETTE UTILITY MENU and the diskette will not be initialized. If you enter "1" in response to this prompt, the program will initialize the disk, thereby erasing all information that is on the disk.

While the diskette is being initialized, the program will print the following message on line 4 of the CRT:

```
Initializing Drive X
```

Upon completion of the initialization, the program will ask if you would like to initialize another disk by displaying the following prompt:

```
INITIALIZE ANOTHER DISK ? (1=YES, Ø=NO)
#
```

To initialize more diskettes simply respond with a "1". If you respond with a "Ø" or any other value, the program will return to the DISKETTE UTILITY MENU.

9.6.2 Gross Copy

Gross copy is used when you want to copy every byte of information from one diskette to another. This copy pays no attention to the directory and copies all information from one disk to another. **Always write protect the disk you are copying from.**

The first prompt displayed upon selecting this option is:

```
                  GROSS COPY PROGRAM
INPUT SOURCE DRIVE NUMBER
#
```

The user should respond with the drive number of the disk that is to be copied. Once this information has been entered the program will respond with the following prompt:

```
INPUT DESTINATION DRIVE NUMBER
#
```

The user should respond with the drive number of the disk that is to be copied to. If one drive specified is a quad (two sided) drive and the other drive is a single sided drive, the program will copy only one side and will inform the user with the following message:

```
     QUAD AND SINGLE SIDED DRIVES MIXED CAN ONLY COPY ONE SIDE
```

If the same drive is selected as the SOURCE and DESTINATION disk, the program will initiate a one drive copy. The following message will appear on the screen to inform the user that the one drive copy is in the "on" mode:

```
     ONECOPY ON
```

In this situation, the program will ask you to insert the source disk in drive 1 with the following prompt:

```
     INSERT SOURCE DISK IN DRIVE 1 AND PRESS RETURN
```

The bell is sounded to alert the user of each change of diskette. Information is read from the source disk and "held" in the internal memory of the computer. Now the program will ask you to place the destination disk in drive 1 with the following prompt:

```
     INSERT DESTINATION DISK IN DRIVE 1 AND PRESS RETURN
```

This process will continue until the entire diskette is copied or until an error condition occurs.

If the source and destination drives are specified as different drives, the program will attempt an automatic copy of information from the source diskette to the destination diskette. If the destination disk was initialized properly, the program will copy all information on the source disk to the destination.

If the destination diskette is not initialized, the program will display the first six entries of the destination diskette and will ask if you want to continue with the copy. The prompt will appear as follows:

```
  DESTINATION DISK HAS FILES.  CONTINUE WITH COPY ? (1=YES, 2=NO)
```

Enter a "1" to continue the copy or "2" to branch to the DISKETTE UTILITY MENU.

As the copy is being made, the program will display the following message on the CRT:

```
     COPY IN PROGRESS
```

If a hard disk error occurs during the copy process, a message will be displayed, indicating which drive caused the error to occur. The message will appear as follows:

```
     HARD DISK ERROR ON DRIVE X
```

When this condition (hard disk error) occurs the program will ask if you want to continue with the copy program (try the copy again), or exit back to the DISKETTE UTILITY MENU. The prompt will appear as follows:

```
     PRESS E TO EXIT OR A TO TRY AGAIN
```

If the door was not closed properly or the disk was not seated properly, the chances are excellent that the copy can be completed successfully by merely opening the offending drive door and closing it slowly a time or two, without using force.

When the "E" option is selected, the program will display one last prompt before branching to the DISKETTE UTILITY MENU. This prompt will tell you to insert your utility diskette in the default drive (normally drive 1). The prompt will appear as follows:

```
     PLACE UTILITY DISK IN DEFAULT DRIVE AND PRESS RETURN
```

Press return after placing the proper disk in the proper drive.

9.6.3 Intelligent Copy

This option copies in A)uto or S)ingle step mode. A)uto mode copies file-by-file until the entire disk is copied or will stop when the destination filespace is too small to continue or a duplicate filename is encountered. S)ingle step mode copies file-by-file but prompts the operator as to the action to be taken for each file.

The first prompt is the request to place the source and destination diskettes in the proper drives. The prompt reads as follows:

```
       INTELLIGENT COPY COPYRIGHT (C) 1981, MICRO MIKE'S, INC.
     INSERT DISKETTES THEN PRESS ANY KEY
```

After inserting the correct diskettes in the proper drives, press any key to continue with the copy.

The second prompt is the request for the drive number to copy from:

```
       INTELLIGENT COPY COPYRIGHT (C) 1981, MICRO MIKE'S, INC.
     INPUT SOURCE DRIVE NUMBER
     #
```

followed by the request for the drive to copy to:

```
     INPUT DESTINATION DRIVE NUMBER
     #
```

If the user enters the same response to both drive number requests, the program will print the following message and return to the first prompt:

```
     SOURCE DRIVE CAN'T EQUAL DESTINATION DRIVE
```

The mode is selected in response to the third prompt:

```
     CHOOSE A)UTO OR S)INGLE STEP
     *
```

To select your choice respond with the first letter of the name of the option you want ("A" for auto or "S" for single step). The A)UTO option begins copying from the point where it is invoked to the end of the disk. The S)INGLE STEP option allows the user to view each file before deciding if the file is to be copied.

If the A)UTO mode is selected, the screen will appear as follows:

```
     INTELLIGENT COPY COPYRIGHT (C) 1981, MICRO MIKE'S, INC.
     C

     FILENAME          AF=Ø
```

The file name and the attribute will display on the fourth line of the CRT as the files are being copied. When the copy is complete, the program will respond with the following prompt:

```
     INPUT E)XIT OR C)ONTINUE
```

If the user selects the "E" option, the user is returned to the DISKETTE UTILITY MENU. If the user selects the "C" option, the program will branch to the initial prompt (INPUT THE SOURCE DRIVE NUMBER).

Several conditions will terminate the A)UTO mode. If space on the destination disk (the one being copied TO) is too small, the program will terminate with the message:

```
     DISK IS FULL
```

If the directory space becomes full, the program will issue the following message:

```
     DIRECTORY IS FULL
```

If a hard disk error occurs, the program will display the following message:

```
     HARD DISK ERROR ON DRIVE X
```

If the destination disk is write protected (write protect tab in place), the program will respond with the following message:

```
     FILE IS WRITE PROTECTED
```

The occurrence of any of the these messages will result in the following prompt:

```
     INPUT E)XIT OR C)ONTINUE
```

If the user selects the "E" option, the user is returned to the DISKETTE UTILITY MENU. If the user selects the "C" option, the program will branch to the initial prompt (INPUT THE SOURCE DRIVE NUMBER).

If the program encounters a file that already exits on the destination disk, the program will respond with the following prompt and message:

```
     CHOOSE S)KIP, R)EPLACE, N)EWNAME, OR E)XIT
     *
     DUPLICATE FILE NAME
```

The S)KIP option skips the present file and goes to the next file. The file is NOT copied if you specify S)KIP.

The R)EPLACE option will cause the file on the destination disk to be over-written by the file of the same name on the source disk (the disk being copied from). The program will then continue to the end or will stop if any of the terminating conditions are found again (file space too small or duplicate file name).

The N)EWNAME option will allow you to give the file a new name. In this instance you will still retain the file on the destination drive with the duplicate name but the file will be copied from the source to the destination drive and be named whatever you choose.

If the N)EWNAME option is chosen, the program will ask for the new file name with the following prompt:

```
     INPUT NEW FILE NAME
     ********
```

If the name has fewer than eight characters, enter a RETURN after the name. If the name has eight characters the program will supply the RETURN.

The E)XIT option allows you to exit to the error condition prompt:

```
     INPUT E)XIT OR C)ONTINUE
```

If the user selects the "E" option, the user is returned to the DISKETTE UTILITY MENU. If the user selects the "C" option, the program will branch to the initial prompt (INPUT THE SOURCE DRIVE NUMBER).

In S)INGLE STEP mode, the program works very similarly to the auto mode except each file is displayed on the CRT one at a time, allowing the operator to make decisions based on the file name. Under S)INGLE STEP the program will display the name of the first file on the source drive and issue the following prompt:

```
CHOOSE C)OPY, S)KIP, A)UTO, OR E)XIT
*
FILENAME          AF= Ø
```

The C)OPY option performs a copy of the file and then displays the next file. If a duplicate record is encountered, the program performs exactly as stated in the A)UTO mode description of duplicate file names.

The S)KIP option skips the currently displayed file and brings up the next file for the operator to act upon.

The A)UTO option allows the user to A)UTO copy all files on the disk. A)UTO may be invoked anytime under S)ingle Step to copy to the end of the disk. The previous description of A)UTO applies anytime it is invoked.

The E)XIT option works exactly as previously described.

All files are compacted automatically as they are copied. Using ICOPY, files are transferred onto the "tail end" of any data already on the disk being copied to. Filenames are displayed on the CRT screen as they are being copied. ICOPY employs read-after-write verification of data being transferred.

Anytime the user is at the INPUT E)XIT OR C)ONTINUE prompt and the user specifies the "E" option and ICOPY was called from another program, ICOPY will prompt the user to insert the utility diskette in the default drive. (This diskette must have the files **baZic** and DUP on it for proper functioning.) When this situation occurs, the program will respond with the following prompt and will wait for you to place the correct disk in the default drive and press return to continue:

```
INSERT SYSTEM DISK AND PRESS ANY KEY
```

9.6.4 File Copy

The copy file option allows the user to copy a single file or multiple files from the user's choice of drives. This program will not normally be used by the end user because of the complexity of the program. This program is "programmable", in that all the questions asked by this program can be answered by another program. In this mode, the program will answer its own prompts automatically and no operator intervention is required.

The program first prompts the user to insert the proper diskettes by the following prompt:

```
          COPY FILE COPYRIGHT (C) 1981, MICRO MIKE'S, INC.
     INSERT DISKETTES AND PRESS ANY KEY
```

The program next prompts the user for the drive number containing the file(s) to be copied. The prompt appears as follows:

```
          COPY FILE COPYRIGHT (C) 1981, MICRO MIKE'S, INC.
     INPUT SOURCE DRIVE NUMBER
     #
```

After entering the source drive number, the program asks for the drive number to copy the file(s) to:

```
     INPUT DESTINATION DRIVE NUMBER
     #
```

If the source drive number is equal to the destination drive number, the program will consider the file copy a special case. This situation will be discussed later. The "normal" flow of the program is for the source and destination drives to be different. For this situation (drives different), the program begins a series of prompts requesting information on how you want the file copies to be made. The first prompt in this series is:

```
     DO YOU WANT TO RENAME FILES? Y OR N
```

Many times the file you want to copy from has already been created on the destination disk. The previous prompt asks if you want to automatically rename each file encountered on the destination disk having the same name as the file being copied from the source disk. Answer "Y" if you want all duplicate files renamed automatically or "N" if you do not want duplicate files renamed.

The next prompt is:

```
     DO YOU WANT ALL DUPLICATE FILES REPLACED? Y OR N
```

If you answer "Y" for this prompt, any duplicate files on the destination disk will be replaced automatically by the file of the same name on the source disk. An "N" response will allow you to decide individually what action is taken for each duplicate file name encountered.

The next prompt is:

```
     DO YOU WANT TO USE A FILE OF FILE NAMES? Y OR N
```

The "N" option specifies that the user will enter the names of each file to be copied. Alternately, the user may have a file established which contains the names of all the files that are to be copied. Selecting the "Y" option places the file copy program in a mode allowing the program to "look" at the specified file and perform a copy of all files listed in the file of file names. If the "Y" option is selected the next prompt will be:

```
INPUT FILE NAME
********
```

The file name of the file of file names should be entered. If the file is not 8 characters long, a carriage return will have to be entered by the user, otherwise the program will supply the carriage return. After entering the file name, the program will prompt you for the drive number of the file of file names.

The next prompt reads as follows:

```
DO YOU WANT TO SET INDIVIDUAL FILE DENSITIES? Y OR N
```

This option allows the user to set the densities (single or double) for each file encountered in the current copy session. If the "Y" option is selected, the user will be prompted for the file density for each file copied. If the "Y" option is selected, the following prompt will be displayed for each file to be copied:

```
INPUT DESTINATION FILE DENSITY.  S OR D
```

The user may specify the density for each file copied. If the "N" option is selected to the DENSITY prompt, the program will set automatically the density of each file copied to the density of the directory on the destination disk. The density of the destination disk is determined when the disk is initialized. Normally only double density is used.

The next prompt to be displayed is:

```
ALL FILE ATTRIBUTES=Ø DO YOU WANT TO SET THE FILE ATTRIBUTES?
*
```

Each file on the disk can have one of 64 attributes (Ø to 63). This prompt is asking if you want to be able to select the attributes of each file that is to be copied. If you select "N" for no, the program will copy each file to the destination disk and preserve the attribute of the file so that the destination file attribute is the same as the source file attribute. If you select the "Y" option, the program will respond with the following two prompts for each file to be copied:

```
INPUT THE SOURCE FILE ATTRIBUTE.  Ø..63
##
```

```
INPUT THE DESTINATION FILE ATTRIBUTE.  Ø..63
##
```

At this point, the program is ready to take in the file name to be copied. The prompt appears as follows:

```
INPUT FILE NAME TO COPY
********
```

Enter the name of the file to be copied. If the file name is not 8 characters long, press the carriage return after the file name is entered. If the file name is 8 characters in length, the program will supply the carriage return.

If the file name selected is not on the destination drive, the program will create a file the same size as the file on the source disk and will copy the file to the destination disk. If the file already exists on the destination disk, the program will check the responses to the previous questions to determine what action it is to take.

If the answer to the auto replace prompt was positive, the file on the destination disk will be replaced automatically by the file of the same name on the source disk. The program will make the replacement and print the following message on the CRT to inform the user the auto replace mode is in effect:

```
AUTO REPLACE
```

If the new name option is selected as positive, the program will automatically prompt the user for the new name the file is to be given when it is copied to the destination disk. The prompt appears as follows:

```
INPUT NEW FILE NAME
********
```

Once the copy is completed, the program will ask the user if he wants to exit the program, copy another file using the answers to all previous questions, or start over with the program and answer all questions again. The prompt appears as follows:

```
CHOOSE E)XIT, M)ORE FILES, OR R)ESTART COPY
```

The "E" option always takes the user to the program that originally called the COPYFILE program. The "M" option allows the user to copy more files. In this mode, the user is not asked for the source and destination drive number nor the other questions pertaining to how the copy is to be made. Selecting the "R" option takes the user back to the original prompt of the program (INPUT SOURCE DRIVE NUMBER).

If, at anytime during the copy, the user encounters a duplicate file and the user specified that the program was NOT to replace automatically the destination file with the source file, the program will display the following prompt and message:

```
CHOOSE E)XIT, S)KIP, R)EPLACE, OR N)EWNAME
*
DUPLICATE FILE NAME
```

If the user selects the "E" option, the program displays the following prompt:

```
INPUT E)XIT OR C)ONTINUE
```

The "E" option in this case takes the user back to the calling program. The "C" option allows the user to start over at the original prompt (INPUT SOURCE DRIVE NUMBER).

If the user selects the "S" option, the file is skipped and is not copied from the source disk to the destination disk.

The "R" option specifies that the destination file is to be overwritten by the source file.

The "N" option allows the file to be copied from the source disk to the destination disk and be given a new name on the destination disk. The program issues the following prompt when the "N" option is selected:

```
INPUT NEW FILE NAME
```

If the initial two prompts (INPUT SOURCE AND DESTINATION DRIVE NUMBERS) result in the same drive number being input by the user, the program will continue to function but two prompts will not be issued as in the normal sequence. These two prompts are:

```
DO YOU WANT TO RENAME FILES? Y OR N
DO YOU WANT ALL DUPLICATE FILES REPLACED? Y OR N
```

The program assumes that duplicate files will be encountered since the source disk and destination disk are the same. The general procedure in this situation is to rename the file as described previously. The DUPLICATE FILE NAME messages and prompts will also be as previously described.

During the course of a file copy, several error messages may be displayed on the CRT. The following messages are essentially self explanatory:

```
FILE NOT FOUND
DESTINATION DIRECTORY IS FULL
DESTINATION DISK IS FULL
DUPLICATE FILE NAME
```

In the automatic mode, where the program is using a file of files, two additional messages are displayed on the CRT to inform the user of proper operation of the program:

```
READING FILE NAMES FROM FILE
END OF FILE NAME FILE
```

Any time the "E" option is selected from the "INPUT E)XIT OR C)ONTINUE" option, the program will tell you to insert the diskette containing the utility programs into the default drive. The prompt will appear as follows:

```
INSERT SYSTEM DISKETTE AND PRESS ANY KEY
```

9.6.5 Compact

The compact option is used to recover file space lost on a drive because of the deletion of files that were not at the end of the data area. The compact option re-organizes the data on a disk so that all the files are at the beginning of the disk and all of the "blank" room is at the "end" of the disk. **A copy should always be made of a disk before it is compacted.**

The first prompt of this option is:

```
     COMPACT COPYRIGHT (C) 1981, MICRO MIKE'S, INC.
INSERT DISKETTE TO COMPACT AND PRESS ANY KEY
```

After inserting the proper diskette, press any key and the next prompt of this option will display:

```
     COMPACT COPYRIGHT (C) 1981, MICRO MIKE'S, INC.
INPUT DRIVE TO COMPACT
#
```

If a hard disk error occurs during the compaction, the program will print the following message:

```
HARD DISK ERROR ON DRIVE X
```

When the compaction is complete or an error occurs, the following prompt is displayed:

```
INPUT E)XIT OR C)ONTINUE
```

If the user selects the "C" option to continue, the program branches to the first prompt of the compaction (INPUT DRIVE TO COMPACT). The "E" option takes the user to the DISKETTE UTILITY MENU.

Anytime the user is at the INPUT E)XIT OR C)ONTINUE prompt and the user specifies the "E" option and the program has been called from another program, the program will prompt the user to place the diskette containing the system utilities in the proper drive. When this situation occurs, the program will respond with a prompt and will wait for you to place the correct disk in the default drive and press return to continue:

```
     INSERT SYSTEM DISKETTE AND PRESS ANY KEY
```

9.6.6 Interfacing COPY Programs to baZic Programs

All of the initialize, copy, and compact programs can be interfaced very easily to **uDoZbaZic.** The DUP program is written in **baZic** and can be CHAINed to by any other **baZic** program. Pressing the ESCape key or ^B from the DISKETTE UTILITY MENU will cause DUP to CHAIN back to the program that called DUP. For this to work properly, the programmer may have to make some slight changes to the DUP program to set the CHAIN to program if the name of the program is not "CSUB".

DUP actually is only a menu system to call the appropriate programs. Initialize is the only function accomplished in the program DUP. All other functions are used by either CHAINing to the appropriate program (GCOPY) or by using the DOSCMD feature of **baZic** to call a machine language program (ICOPY, COPYFILE, COMPACT).

Since GCOPY is a **baZic** program, it can be CHAINed to like any other program. By listing the program and making the correct modifications, GCOPY can be made to do about anything the programmer may want to do. The program CCOPY is a shortened version of GCOPY that is used to make the initial copy of the master diskette from the program CONFIG.

This program (CCOPY) demonstrates the modification of the copy program so that a **baZic** program tells the user what to do (which drive to place the diskettes, etc.) and CHAINs to the copy program so that no user intervention or special knowledge is required to make the copy.

In a similar manner, ICOPY, FILECOPY, and COMPACT can be used by a **baZic** program. In the case of these three programs, the program is called by using the DOSCMD statement of **baZic.** The DOSCMD statement can be used to execute a GO ICOPY command from **baZic.**

When a DOSCMD statement is used, the contents of the entire command being passed to **uDoZ** is moved to the CCB (Common Command Buffer) located at 8ØH. MicroDoz then acts on the command in the CCB.

The programs ICOPY, FILECOPY, and COMPACT all have the additional capability of using values in the CCB to answer the prompts displayed when the program is executed. When any of these programs are executed, they locate the next separator byte in the CCB. If two separators are together, the programs assume that no commands follow and they use the file name after the second separator as the program to load when the copy program is exited.

If the copy programs do not find a double separate˜ at the current separator byte, the programs assume that what follows in the buffer is additional commands which are to be used by the program to answer its own prompts.

As an example, the following sequence could be sent to **uDoZ** from **baZic** and would result in the files listed in the file FNAME being copied from drive 1 to drive 2 without operator intervention. When FILECOPY has finished executing the copy, the program will return to **baZic** which will LOAD and execute the program "CSUB". The **uDoZ** command appears as follows:

```
10 DOSCMD "COPYFILE\1\2\N\Y\Y\FNAME   \1\N\N\E\\BAZIC\CSUB"
```

The result of this command sequence would be to execute the program named COPYFILE. Once COPYFILE is executed it will respond with the following prompts which are answered by the values in the CCB. Each response in the CCB will be underlined in the series of questions that follow. The sequence of prompts is as follows:

```
COPY FROM DRIVE 1
COPY TO DRIVE 2
AUTO RENAME N
AUTO REPLACE IF DUPLICATE FILE NAME Y
USE A FILE OF FILE NAMES Y
NAME OF FILE NAME TO USE FNAME
FILE IS ON DRIVE 1
PROMPT DENSITY FOR EACH FILE N
PROMPT ATTRIBUTES FOR EACH FILE N
INPUT E)XIT OR C)ONTINUE E
```

(The file-of-file-names file should be written sequentially, with each file name stored as a string of 8 characters or fewer. The list of file names should end with an endmark.)

This versatility of **uDoZbaZic** allows the programmer great power in programming. No longer does the end user have to know the Disk Operating System just to initialize a disk or make a backup copy. All these things can be set up by the programmer and the end user has only to press a key on the keyboard to accomplish these tasks.

9.7 CONVRT

This program is used to convert a mask from the "old" type mask (which contained cursor addressing information) to the "new" style mask (which does not contain cursor addressing information).

The program CONVRT prompts for the file name and changes a single mask. The program CONVRTA requests the name of a diskette directory and changes all masks on the diskette to the new format.

USING CSUB

The following section is the end user's documentation on how to use CSUB and explains the features of CSUB common to all programs written under CSUB.

10.1 Using CSUB Programs

CSUB is a set of programs designed to help keep the users of this application program from making mistakes as the program package is used. CSUB checks each input to determine, to the best of its ability, if the information you have entered is correct and is the information that the application program is requesting.

CSUB is not infallible and cannot catch every mistake a user can make. However, many of the common mistakes can be trapped and "thrown away" by CSUB so that no harm is done to the programs or data. When an answer is thrown away, you will be given another chance to answer correctly.

Certain safety features have been written into the CSUB program to reduce the possibility of errors. The computer will prompt you in entering your data. A "prompt" is a message that the program displays telling you what the computer is looking for in response to its message. You will be expected to enter some information or answer a question each time a prompt is displayed.

Where information or data is demanded for program operation, CSUB will not allow you to continue until you have entered that data. When information is not essential, such as descriptions or comments, you may enter a space or carriage return for alpha-numeric entries or a zero (0) for numeric entries and the program will continue.

When your program is in operation, a white rectangular dot, underline, or some other marker called the CURSOR, will appear on the screen. The cursor indicates where you are in the program's operation at any given time. Each time you press a key the letter associated with that key will appear at the cursor position and the cursor will be advanced one position.

The cursor will appear at the beginning of a block of characters, ranging in number from 1 to 79, called PROMPT CHARACTERS. The CSUB program is so designed that when the prompt character field is filled, the cursor will move automatically to the next block where data are to be inserted. If the prompt character field is not filled completely by your response, simply press RETURN and the cursor will move automatically to the next position.

Typing errors within a block of prompt characters can always be corrected by using the back space key (Control H if your CRT doesn't have a back space key) to back the cursor to the incorrect character where it can be corrected by typing the correct letter over the incorrect one. Some versions of CSUB have a complete line editor that allow the insertion and deletion of characters within the prompt characters block. See Section 1Ø.2 for more information on the line editor.

Pressing the ESCape key will return the cursor to the previous block during an input, or if at the beginning of a program to the Main Menu. If your CRT has no ESCape key, a "Control B" command (simultaneously pressing the CTRL and "B" keys) may be substituted for the ESCape key. (Some CRTs may use special function keys for the purpose of backing up.) ESCape may be considered to be, in many cases, a "back-up key". Pressing the ESCape key will be represented in this manual by "ESC".

When entering numeric values, only the characters "-" and "." can be used. Unfilled spaces have no effect on the value of the number entered. If you entered 345.ØØ into a display field containing eight dollar signs, the screen would appear as follows before you make your entry:

```
$$$$$$$$
```

and as follows after your entry, before you press the RETURN key:

```
345.ØØ$$
```

Once you press the return key to tell the program to "take" the amount, the screen would appear as follows:

```
 $345.ØØ
```

PLEASE NOTE the difference between the small letter "l" and the number "1" and the capital letter "O" and the digit "Ø" (zero). The computer makes a distinction between these characters and they cannot be interchanged.

A few prompts will request a reply that must be entered in ALL CAPITAL LETTERS, i.e.:

```
Enter Password (PASS)
or,
Y=Yes, N=No.
```

You must abide the computer's requests. When the computer calls for numerical data, only numbers can be entered and these must be within the particular range of that part of the program, or the computer will reject the entry and notify you of the input error.

The prompt characters may be examined to determine what kind of input is being requested. The following table demonstrates the common prompt characters and their meanings:

%	A number is to be input that represents a percentage.
$	A number is to be input that represents a dollar value.
#	A number is to be input.
*	Any character on the keyboard is acceptable.

Dates represent a special type of data to be entered. The format for a date input will always appear as follows:

MM/DD/YY

MM represents a two digit month entry, DD represents a two digit day entry, and YY represents a two digit year entry. Any dates that have only one digit possible will advance automatically to the next field within the date input. For example, if you enter a 3 for the month of March, CSUB will know that there are no months that have a value of 3Ø or more and the program will automatically advance the cursor to the day section.

The cursor can be "backed up" from one field within the date input to the previous field by pressing the ESCape key. If the cursor is already on the month field (MM), pressing the ESCape key will cause the cursor to jump to the last logical input.

Some programs will require a response to the question, "Correct Program?" You must respond according to the directions given by the computer before you can proceed. Responses can range from Y or N to 1=Yes and Ø=No. You must follow the computer's instructions.

Basically, a Yes response will permit you to continue in that program. A No response, in the menu programs, will return you to the Main Menu. If the prompt is "1=Printer, Ø=CRT," pressing "1" will activate the printer, if it is turned on and is on line. A "Ø" response will display the information on the screen.

In the File Maintenance (FM) programs there are generally three options:

Enter 1=Add, 2=Change, 3=Delete

Option 1 allows you to enter a new record. Option 2 allows you to make changes in any field in the record except the field that is used to "look up" the record (index). Option 3 allows you to delete the entire record. Remember not to delete a record unless it is no longer needed and there are no valid data in the record.

Your terminal keyboard is similar to the keyboard of an electric typewriter. The touch is set at the factory and cannot be adjusted. Only the appropriate pressure is required. You do not have to "hit" a key for a response. Pressing a key rapidly two or more times, either as a nervous habit or in agitation, may result in the computer "scolding" you for giving it an improper command or of entering too many characters into your response.

10.2 CSUB Line Editor

Most versions of CSUB contain a built in line editor that allows the user to edit any alpha-numeric entry. The line editor is deleted from some versions of CSUB in order to recover more internal memory for use by the application program. All other versions of CSUB incorporate the line editor.

Upon selecting any alpha-numeric field (the prompt character is an asterisk), the contents of the field can be edited by pressing control N (^N). This action places CSUB in the edit mode. Once in the edit mode, the editing control characters become operational.

The following is a list of the editor commands of CSUB:

^N	Enter the edit mode.
^Z	Erase the character at the cursor position.
^A	Move the cursor one character to the right.
^Y	Make a space to insert one character.
^G	Terminate the edit mode.
^B	Back out of the current input and return to the last previous input or program.

To insert a character or characters into a field, move the cursor to the place where you want the character inserted (^A moves the cursor to the right and backspace moves the cursor to the left). Type a ^Y for each character to be inserted. Each time the ^Y is typed, the characters in the field to the right of the cursor will move over one position and a space will be inserted at the cursor position.

Any character within the field may be changed by simply positioning the cursor over the character and typing the correct character. As characters are inserted, all characters to the right of the insert position are moved to the right. The last character on the right hand side of the field is "dropped" off the end of the line.

To back up (move to the left) one character, use the backspace key, or ^H, or ^Q. These codes cause the cursor to back up one space without destroying any information.

The ^Z key is used to delete a character. To delete a character, position the cursor over the character and type ^Z. The character will be deleted and all characters to the right of the character deleted will be moved to the left one space to fill in the space left by the deleted character.

1Ø.3 Error Messages and Bulletins

Although there are several types of bulletins, they all fall into one general class. There are three types of error messages, including: CSUB Non-fatal, CSUB Fatal, and **baZic** Fatal. When bulletins or CSUB Non-fatal errors occur, no harm is done to programs or data. When either of the fatal errors occur, a programmer should be consulted to determine the cause of the error.

Sometimes the computer will display a message on the screen to the user. A message can be either an error or notification of a computer process. An error message is flashed three or four times across the screen. Bulletins generally remain displayed until the process is complete.

1Ø.3.1 Process Bulletins

The first type of message is the notification of a process. These messages are printed only once and remain on the screen until the process is completed at which time they disappear automatically. Be patient and wait for the process to finish before attempting to type on the keyboard. The computer will usually ignore any keystrokes that occur during these messages. Examples of these types of messages appear below:

```
File Maintenance Program Loading Main Menu
Computer is Printing--Do Not Interrupt
```

1Ø.3.2 CSUB Non-fatal

This type of error message occurs normally during the course of running an application program. When these errors occur there is no cause for alarm. These errors indicate that you have made some mistake in your entry, but CSUB has discovered the error and rejected your entry. Continue by determining what mistake you made and trying your entry again.

The meanings of several CSUB Non-fatal error messages are given below:

OUT OF RANGE

This error occurs when you enter data out of the range limit of the particular input. If a menu selection has only 5 options and you enter a 6 in response to the prompt of "Enter Option", this message will occur.

NOTHING ENTERED

This error occurs when you do not make an entry but you press the return key, indicating to the computer that you are through with the entry.

YN ONLY

This error occurs when you type in an incorrect entry. If the computer is expecting a "Y" or "N" entry in response to a prompt of "Enter Y for Yes or N for No", and any other entry is made, this message will result. This message can contain any combination of letters.

RECORD NOT FOUND

This message can be displayed in many situations. It generally means you have typed the index (account number, etc.) incorrectly. Normally you will have a chance to re-enter the information that was entered incorrectly.

DUPLICATE RECORD NAME or
DUPLICATE ENTRIES NOT ALLOWED

These error messages occur when you are trying to make a new entry into a file, but a record of the same name or account number already exists. Try again with a different account number or name.

FILE ERROR

This error can be caused by many different situations. It usually indicates an error in a disk file. Check your diskette and make sure it is not write protected.

RECORD DELETED
RECORD CREATED

These are not errors but are displayed in a manner similar to error messages. These messages occur when deleting or creating records in key-access files.

1Ø.3.3 CSUB Fatal

These errors are potentially very damaging although not necessarily so. When one of these errors occurs it means that CSUB has detected an internal error which indicates that something has gone wrong in the program. Occurrences of these errors generally warrant the attention of a programmer as there is usually nothing the end user can do to correct the situation.

Examples of CSUB Fatal errors follow:

FILE IS FULL

When this error occurs it is time to call your programmer. This error means that your files were not created large enough for the data you have to store. A programmer is needed to "fix" this error. Sometimes it is possible to delete entries to regain extra file space.

The following is a list of other errors that can occur:

B$ to spaces initially
Check Disk Drives
Hard Disk Error
File Not Found
Incorrect File Type
Key File Not Found
Key File Type Incorrect
File Name Too Long
Can't Write Beyond Endmark
^C Use During XXXX File Use
baZic error XX
Type error in XXX CK for C9 precision
File Length Exceeded
Blank Index String
Index Too Big

10.3.4 baZic Fatal

The occurrence of these errors means that CSUB has failed to catch and handle properly an error that has occurred. The programmer should be called when any of these errors occur in a program. These errors will be characterized by the **baZic** prompt "READY", preceded by one of the following **baZic** errors:

ARG
DIMENSION
OUT OF BOUNDS
TYPE
FORMAT
LINE NUMBER
FILE
HARD DISK
DIVIDE BY ZERO
SYNTAX
READ
INPUT
ARG MISMATCH
NUMERIC OV
STOP IN LINE XX
LENGTH
CONTINUE
CONTROL STACK
DOUBLE DEF

```
FUNCTION
ILLEGAL DIRECT
INTERNAL STACK OV
MEMORY FULL
MISSING NEXT
NO PROGRAM
TOO LARGE OR NO PROGRAM
```

BONES Listing

Lines that are more than 66 columns in the program listing are continued on the next line without a line number in this APPENDIX.

```
1000 REM BONES      *** SINGLE KEY--KEYED ACCESS FILE
     MAINTENANCE DEMO ***
1010REM DIMENSION VARIABLES NEEDED
1020 DIMB$(75+1035)\REM STRING VARIABLES AND MASK
1030 DIM D3$(30),D5$(30),D7$(30)
1040REM FILE NAME DEFINITIONS AND DESCRIPTIONS
1050 D2$="BONE"\REM DATA FILE NAME
1060 D3$="K"\REM KEY FILE NAME
1070 D4$="BONES"\REM PROGRAM NAME
1080 D6$="MENU"\REM  NAME OF CALLING PROGRAM
1090 D5$="GENERAL PURPOSE FILE MAINTENANCE"\REM DESCRIPTIVE PGM
     NAME
1100 D7$="SYSTEM MENU"\REM DESCRIPTIVE NAME OF CALLING PROGRAM
1110REM B$ AND B( ) DELIMITERS
1120 D3=1\REM BEGINNING POSITION OF INDEX IN B$
1130 D4=4\REM ENDING POSITION OF INDEX IN B$
1140 D8=76\REM STARTING POSITION OF MASK IN B$
1150 D9=1\REM NUMBER OF MASK
1160 D6=1\REM POSITION IN B$ TO READ IN STRING DATA
1170 D7=1\REM POSITION IN B( ) TO READ IN NUMERIC DATA
1180 D5=10\REM # FIELDS IN EACH RECORD
1190REM DRIVE # SPECIFICATIONS
1200 D=1\REM PROGRAM DRIVE #
1210 D1=1\REM DATA DRIVE #
1220REM MISC DEFINITIONS
1230 P=1\REM PRINT DEVICE
1240 F1=20\REM NUMBER OF RECORD TO PRINT ON ONE PAGE
1250 F2=1\REM PAGE COUNTER SET TO ONE
1260 C$="OA"\REM ACCEPTABLE ANSWERS TO FNF PROMPT 6
1270 G=FNG(D4$+"&",D,D8,0,1)\REM READ MASK INTO B$ STARTING AT D8
1280 IF G>0 THEN 1300\GOSUB 55000\GOTO 1710\REM IF ERROR HAS
     OCCURED THEN END
1290REM START OF PROGRAM
1300 CLS\X=FNM(D8,D9)\REM DISPLAY MASK
1310X=FNF(1)\REM DO 1ST PROMPT
1320 IF X=2 THEN 1710\REM CHECK FOR CONTROL B (IF ^B THEN END)
1330 ON B(0) GOTO 1340,1500,1610,1720,1850\REM 1=ADD,2=INQ,3=DEL,
     4=REN,5=PRT
1340 REM ADD A RECORD
1350 X=FNE(1)\IF X=2 THEN 1300\REM GET INDEX
1360 H=FNH(D2$,D1+.3,B$(D3,D4),"",D3$)\REM CALL FNH TO CREATE A
     NEW INDEX
1370 IF H>0 THEN 1390\IF H=-7 THEN 1380\GOSUB 55200\GOTO 1300\REM
     CHECK ERROR
1380 X=FNC("INDEX IN USE.  TRY ANOTHER.")\GOTO 1340
```

```
1390 FOR N=2 TO D5\REM FOR N=2 TO NUMBER OF FIELDS
1400 X=FNE(N)\IF X<>2 THEN 1410\N=N-1\IF N=1 THEN EXIT 1300
     ELSE 1400
1410 NEXT N
1420 X=FNF(2)\REM CALL 2ND PROMPT AND INPUT
1430 IF X=2 THEN 1300\REM BACK OUT OF CHANGE SECTION
1440 IF B(0)=0 THEN 1480\REM RECORD DATA
1450 X=FNE(B(0)+1)\REM INPUT ON SELECTED FIELD
1460 IF X=2 THEN 1420\REM BACK OUT OF INPUT
1470 GOTO 1420
1480 G=FNG(D2$,D1+.1,D6,D7,H)\REM NO ERROR SO CALL FNG TO WRITE
     RECORD
1490 IF G>0 THEN 1300\GOSUB 55000\GOTO 1300\REM CHECK FOR ERROR
1500 REM CHANGE OR INQUIRE
1510 GOSUB 2160\REM GET RECORD
1520 IF X=2 THEN 1300\REM ^B CHECK
1530 X=FNF(2)\REM DO 2ND PROMPT & INPUT
1540 IF X=2 THEN 1500\REM ^B CHECK
1550 IF B(0)=0 THEN 1590\REM CHECK FOR 0 TO RECORD
1560 X=FNE(B(0)+1)\REM DO INPUT ON SELECTED FIELD
1570 IF X=2 THEN 1530\REM ^B CHECK
1580 GOTO 1530\REM GO DO 2ND PROMPT AGAIN
1590 G=FNG(D2$,D1+.1,D6,D7,H)\REM WRITE INFO BACK INTO FILE
1600 IF G>0 THEN 1300\GOSUB 55000\GOTO 1300\REM ERROR CHECK & END
1610 REM DELETE-DO 3RD INPUT
1620 GOSUB 2160\REM GET RECORD
1630 IF X=2 THEN 1300\REM ^B CHECK
1640 X=FNF(F)\REM DO 4TH PROMPT
1650 IF B$(D8-3,D8-1)="DEL" THEN 1660\X=FNC("RECORD NOT DELETED")
     \GOTO1300
1660 H=FNH(D2$,D1+.2,B$(D3,D4),"",D3$)\REM DELETE INDEX & RECOVER
     KEY
1670 IF H=0 THEN 1680\GOSUB 55200\GOTO 1300\REM DO ERROR CHECK
1680 G=FNG(D2$,D1+.1,D6,D7,H)\REM WRITE INFO BACK INTO FILE
1690 IF G>0 THEN 1300\GOSUB 55000\GOTO 1300\REM ERROR CHECK & END
1700 X=FNC("DISK ERROR HAS OCCURED--PROGRAM ABORTING!")
1710 X=FNX(D5$,D7$,D6$,D)
1720 REM RENAME THE INEX
1730 GOSUB 2160\REM GET RECORD
1740 IF X=2 THEN 1300
1750 D8$=B$(D3,D4)\REM SAVE OLD INDEX IN D8$
1760 X=FNF(5)\REM NEW INDEX NAME
1770 IF X=2 THEN 1300
1780 REM RENAME AN INDEX
1790 H=FNH(D2$,D1+.4,B$(D3,D4),D8$,D3$)\REM CALL FNH TO RENAME
     INDEX
1800 IF H>0 THEN 1810\GOSUB 55200\GOTO 1700\REM ERROR CONDITION
1810 G=FNG(D2$,D1+.1,D6,D7,H)\REM WRITE INFO BACK INTO FILE
1820 IF G>0 THEN 1830\GOSUB 55000\GOTO 1300\REM ERROR CHECK
1830 X=FNC("INDEX HAS BEEN CHANGED FROM "+D8$+" TO "+B$(D3,D4))
1840 GOTO 1300
1850 REM PRINT SECTION
1860 X=FNF(6)\REM PRINT ONE RECORD OR ALL
1870 IF X=2 THEN 1300
1880 ON B(0)+1 GOTO 1890,1950\REM GOTO PRINT ONE, PRINT ALL
```

```
1890 GOSUB 2160\REM GET RECORD
1900 IF X=2 THEN 1860
1910 GOSUB 2310\REM PRINT THE HEADING
1920 GOSUB 2250\REM PRINT THE RECORD
1930 !#P,Z1$\REM OUTPUT A FORM FEED TO PRINTER
1940 GOTO 1300
1950 D9$=STR$(D1)\D9$(1,1)=","\REM CONVERT DRIVE # TO STRING
1960 OPEN#2,D2$+D3$+D9$\REM OPEN THE KEYFILE
1970 READ #2,N1,N2,N3\REM READ THE 1ST THREE NUMERICS
1980 G1=FNG1(D2$,D1,7)\REM OPEN MAIN FILE THROUGH CHANNEL NUMBER 7
1990 GOSUB 2310\REM PRINT HEADING
2000 FOR N=1 TO N2\REM N2 EQUALS NUMBER OF RECORDS
2010 READ #2,N$,N3\REM READ THE INDEX AND RECORD NUMBER
2020 G=FNG(D2$,D1,D6,D7,N3+.7)\REM READ A RECORD FROM MAIN FILE
2030 IF G>0 THEN 2040\GOSUB 55000\GOTO 1850\REM CHECK FOR ERROR
2040 RESTORE 51000\X=FND(D5)\REM DISPLAY ON CRT
2050 GOSUB 2250\REM PRINT RECORD
2060 IF F<F1 THEN 2110\REM PRINT RECORD IF MORE WILL FIT ON PAGE
2070 GOSUB 2310\REM PRINT HEADING
2080 !#P,Z1$\REM OUTPUT A FORM FEED TO PRINTER
2090 F=0\REM ZERO RECORD COUNTER
2100 F2=F2+1\REM INCREMENT PAGE COUNTER
2110 NEXT N
2120 G1=FNG1(D2$,D1+.1,7)\REM CLOSE FILE
2130 CLOSE#2
2140 !#P,Z1$\REM OUTPUT FORM FEED TO PRINTER
2150 GOTO 1300
2160 REM SECTION TO RECALL A RECORD
2170 X=FNF(3)\REM ENTER INDEX
2180 IF X=2 THEN RETURN\REM ^B CHECK
2190 H=FNH(D2$,D1,B$(D3,D4),"",D3$)\REM CALL FNH TO GET KEY
2200 IF H>0 THEN 2210\GOSUB 55200\X=2\GOTO 2240\REM CHECK FOR
     ERROR
2210 G=FNG(D2$,D1,D6,D7,H)\REM READ A RECORD FROM MAIN FILE
2220 IF G>0 THEN 2230\GOSUB 55000\GOTO 1700\REM CHECK FOR ERROR
2230 RESTORE 51000\X=FND(D5)\REM DISPLAY ITEMS
2240 RETURN
2250 REM SUBROUTINE TO PRINT ONE RECORD
2260 !#P,B$(1,4),TAB(7),B$(25,40),TAB(30),%$CA15F2,B(2),
2270 !#P,TAB(46),%$CA15F2,B(3),TAB(64),%6F3,B(4),TAB(71),FND$(B(5))
2280 !#P
2290 F=F+1
2300 RETURN
2310 REM SUBROUTINE TO PRINT HEADING
2320 !#P\!#P
2330 !#P,TAB(33),"TESTING COMPANY"
2340 !#P
2350 !#P,TAB(37),"PAGE",F2
2360 !#P\!#P
2370 !#P,"INDEX  LEFT JUSTIFIED STRING  POSITIVE DOLLAR  NEGATIVE
     DOLLAR",
2380 !#P,"  PERCENT  DATE"
2390 FOR X=1 TO 78\!#P,"-",\NEXT X\!#P,"-"\REM PRINT SEPARATOR
2400 !#P
2410 RETURN
```

```
50000 DATA1
50010 DATA"ENTER 1=ADD, 2=INQUIRE OR CHANGE, 3=DELETE, 4=RENAME
      INDEX, 5=PRINT"
50020 DATA.35,1,4,1,5,0
50030 DATA1,"ENTER FIELD NUMBER TO CHANGE (0 TO RECORD)"
50040 DATA.35,1,4,0,9,0
50050 DATA2,"ENTER INDEX"
50060 DATA.42,4,0,1,4,0
50070 DATA2,"ENTER DELETE CODE (DEL)"
50080 DATA.42,3,0,73,75,0
50090 DATA2,"ENTER THE NEW INDEX NAME"
50100 DATA.42,4,0,1,4,0
50110 DATA3,"PRINT O)NE OR A)LL
50120 DATA.42,1,0,1,2,0
51000 DATA2,7.42,18,4,0,1,4,0\REM INDEX
51010 DATA2,12.42,32,20,0,5,24,0\REM STRING RIGHT JUSTIFIED
51020 DATA2,13.42,32,20,0,25,44,1\REM STRING LEFT JUSTIFIED
51030 DATA2,14.42,32,20,0,45,64,2\REM STRING CENTERED
51040 DATA1,19.35,34,8,10,1,99999999,1\REM POSITIVE NUMERIC
51050 DATA1,20.35,34,11,1,1,9999999.99,2\REM POSITIVE DOLLAR
51060 DATA1,21.36,34,11,1,-9999999.99,-1,3\REM NEGATIVE DOLLAR
51070 DATA1,22.37,34,6,2,-9.999,99.999,4\REM PERCENT
51080 DATA4,7,45,0,0,0,0,5\REM DATE AS NUMERIC
51090 DATA4,8,45,0,0,65,72,0\REM DATE AS STRING
55000ONABS(G)GOTO55010,55020,55020,55010,55010,55010,55010,55010,
      55010,55010
55010X=FNC("PROGRAM MALFUNCTION--CALL YOUR PROGRAMMER")\RETURN
55020X=FNC("CHECK DRIVES FOR CLOSED DOOR AND CORRECT DISKETTE")
      \RETURN
55200ONABS(H)GOTO55010,55010,55010,55010,55010,55010,55220,55220,
      55010,55020
55220X=FNC("TRY ENTRY AGAIN")\RETURN
```

Sample Work Sheets

The five work sheets are provided to aid the programmer in writing CSUB programs. The work sheets are explained in Section 8 (WRITING CSUB PROGRAMS). These work sheets may be reproduced by any programmer without prior written consent by Micro Mike's, Inc.