MMike's

MICRO MIKE'S, INCORPORATED 905 South Buchanan Amarillo, Texas 79101 U.S.A. 806/372-3633

DOCUMENTATION FOR CSUB VERSION 5.1

(Preliminary)

Micro Mike's Table-Driven Application Software Package

The basic purpose of this package is to allow easy, rapid creation of application software and data bases that run under North Star DOS and BASIC. 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. These routines require an addressable cursor CRT, since use of the addressable cursor is the most efficient method of data entry, particularly for inexperienced users.

North Start Basic 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 North Star BASIC, string arrays are handled differently. CSUB takes advantage of the string capabilities of North Star BASIC 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$(101,110)). This means many separate strings are stored in one string and individual strings can be variable in length. For storing strings on disk, North Star BASIC requires 2 BYTES of overhead for each string stored that is under 255 characters, and 3 BYTES for each string over 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 elsewhere, but generally it uniquely defines each variable that is used for input and/or display by the parameters that are set in each DATA statement for each variable.

CSUB Input routines are accessed by first RESTOREing to the appropriate set of data statements the programmer is concerned with and then calling the particular function that performs the desired task. 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 creation of the data base. This program is the file maintenance program (BONES) for keyed access files or the

sequential input program (INBONES) for sequential files.

At the most primitive level, there are two classes of files. One class is a temporary file for input of data in a rapid and easy manner. Normally, sequential entries are made into this file for a period of time and then the information in this file is processed and either passed to another file or printed. This input process is accomplished by INBONES.

The other primitive class of file is the permanent file which holds information that will be kept for a long period of time. Individual fields and records within the file may change, but the total file remains fairly constant. BONES is a keyed access program which represents one way of accomplishing such a purpose.

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

Use of a table-driven package offers numerous advantages over BASIC. 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 that follow through each and every program. This reduces 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. CSUB makes BASIC a much more structured language, resulting in easier program construction. Programs become more understandable between 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. With appropriate REM statements, the program becomes essentially self-documenting.

Basic routines in CSUB are:

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

FUNCTION DEFINITIONS

X=FNA (Line #, Position on Line) : Non destructive cursor positioning
X=A dummy variable for this function

This function positions the cusor non-destructively anywhere on the CRT by passing the coordinators of where you want the cursor to appear.

X=FNB(N) Flash the Nth Error Message
X=Dummy Variable

This function is generally preceded by a RESTORE statement to reference the DATA statements' pointer to the appropriate set of DATA statements (i.e., the error messages). The pointer is read down the errors messages until the Nth one is found. This function then rings the "bell" and flashes the error message (or bulletin) on the 4th line of the CRT. The most used messages should be placed closest to the "front" of the DATA table in order to insure the most rapid execution of the function.

X=FNC($): Displays error message or bulletin.
X=Dummy Variable

A string of up to 80 charactors is passed to this function which then displays the message on the 4th line of the CRT.

X=FND(N): Displays N items on CRT
X=Dummy Variable

Generally preceded by a RESTORE statement to the Variable Parameters DATA Statement (VPDS) line number. The set of parameters for each variable concerned should be on a separate line number. Each should also be REMarked appropriately so that easy variable identification is possible. By setting the DATA statements' pointer to the beginning of the Variable Parameters table and specifying the appropriate N, all the variables can be printed with a single call. By setting the pointer within the table and specifying N=1 (or any other number), one or more items can be displayed. Structure of the DATA tables is covered elsewhere.

X=FNE(N): INPUT the Nth item on the CRT
X=Dummy Variable

After RESTOREing to the VPDS, this function does INPUT on the Nth DATA statement. The parameters of the INPUT are discussed elsewhere.

X=FNF(N): Displays the Nth prompt on line 2 and does 3rd line Input
X=Dummy Variable

After RESTOREing, the Nth prompt is displayed at the beginning of line 2 and the standard INPUT is executed according to the parameters of the DATA statement on line 3 of the CRT. Used for all 3rd line inputs that require a prompt (i.e., "Is Information Correct?".)

X=FNX ($1,$2,$3,N): EXIT program and CHAIN to another
X=Dummy Variable

This function FILLs the CR/LF byte in BASIC with its normal value. The function then clears the screen and prints a message indicating which program is Loading which program and CHAINs to the appropriate program. Parameters passed to this function are:

1. $1 - Descriptive Name of Program
2. $2 - Descriptive Name of Calling Program
3. $3 - Actual file name of program to CHAIN to
4. N - Disk drive number of program to CHAIN to

X$=FND$(N) : Change a numeric date into a string
X$=String date of the form MM/DD/YY

To save file space all dates are stored as a numeric. 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 look like this: !#1,FND$(B(1))

X$=FNL$(N,N1): Left Justify a String
X$=The left justified string

All string information input using FNE or FNF is Right justified within B$ where it is stored. In other words, if a string is input and the user does not enter a character in all of the positions available, then the information entered will be justified (moved) to the right and leading spaces will be entered 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$ are the Left$ and Right$ positions within B$ which you want Left justified.

As an example, if:

B$(1,10)= " DOG"
and we call the function
X$=FNL$(1,10)
then:
X$="DOG "
A typical function call would look like this:
!#1,FNL$(1,10)

G=FNG ("File Name", Drive.CMD, LEFT$, SUBS, RECORD): Sequential File Access
G=Number of Records in File or Error Condition

This function reads or writes a sequential access file. The parameters passed to this function are:

1. "File Name" - name of file to be accessed.

2. Drive.CMD - drive number where file is located. The drive number also controls whether this function executes a read or write.

If the INT(Drive)=Drive (1.0, 2.0, 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.

3. LEFT$ - LEFT$ of B$ where read or write to B$ will start.

4. SUBS - the B(subscript) where read or write to B( ) will start.

5. Record - the record number of the file one wishes to read to or write from. Record 0 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.

Structure of Files Accessed by FNG

The sequential file is set up with three numeric values placed at the beginning of the file. The first numeric value contains the number of records written to the file (0 if file is empty). 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 0, then no string information is 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 elsewhere.

Sequential File Format Summary

A, B, C, $'s 1st record, Numeric's 1st record, $'s 2nd record, etc., where:

A = number of records written, 0 if none
B = length of string portions of each record, 0 if none
C = number of numerics per record, 0 if none

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. Programs using this function should check G to make sure it is positive before using the variable and to assure oneself 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 Diminsioned
-10= Attempt to Write Beyond End Mark (Record # passed was too large)

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

Keyed Access File Routine

H=FNH ("File", Drive. CMD, "Index", "Rename", "Keyfile #")
H=Record Number or Error Condition

This function returns a pointer to a main data file 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 0 at the beginning of the file, 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 a KEY
2. Delete a KEY
3. Find a KEY
4. RENAME an Index

Structure of the Keyfile

The keyfile is created with the name of the main file plus an alphanumeric digit from 0-9 or A-Z, which indicates the keyfile for any particular data file that is to be accessed. Thus, for any one data file, up to 36 keyfiles are allowed which lets the programmer access information in the main file by any of 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 0, 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.

Using FNH

H=FNH ("File", Drive.CMD, "Index", "Rename", "Keyfile #") where:

1. "File" = name of the main file being accessed
2. Drive = drive number of file plus:
   .0=read when accessing key file to read the main file

.1=write when accessing keyfile to write the main file
.2 when deleting a record
.3 when creating a record
.4 when renaming an index

3. "Index" = the index which will be compared to the indexes in the keyfile for performing the various keyed access functions

4. "Rename" = a new index that is to replace "Index" in the keyfile.

5. "Keyfile #" = Name of keyfile being accessed. Up to 36 keyfiles are allowed for any data file. This is a string of one character (0-9, A-Z) that is added to the main file name by the routines of CSUB when keyed access is being used.

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 to actually read the appropriate information from the file.

If an error condition has occurred, then 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
-10 = Hard Disk Error

FNH becomes very convenient if any particular data file exceeds the length of a disk. For example, if one had a data file with 1000 associated keys in the keyfile, and the size of the disks and records are such that only 500 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 500, then use FNG to access the information on the first drive. If the returned value is greater than 500, then subtract 500 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 wanted 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 0 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:

```
10 H=FNH(TEST",D1,"JOE","","K")\REM CALL FUNCTION
20 IF H>0 THEN 30 ELSE GOSUB 55200\GOTO ___ \
30 G=FNG("TEST",D1,1,1,H)\READ RECORD H
40 IF G>0 THEN 50 ELSE GOSUB 55000
50 REM
```

55000 contains optional file error messages that are displayed based upon the ABSolute value of G after its return from FNG. 55200 is the same as 55000 but works on the ABSolute value of H after 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. Thus, the following order would result after sorting:

```
     100
-->  1100
-->  900
     901
     ALPHA
     BETA
     ZETA
     ZFF
```

Care should be taken to study this order, as problems can arise if the programmer isn't aware of the sorting technique used.

BASIC "Memory Map"

Lines 1-4 - Reserved for IFCALL which APPENDS the necessary CSUB onto every program. The BASIC supplied on this disk will now Append without lines 1,2, & 4. This BASIC is a special version of BASIC modified to allow multiple APPENDS.

Line 5 - Always a REMark statement that gives the exact file name under which program is stored. This line may also contain the name of the programmer and a descriptive program name.

Line 6 - Always a GOSUB 65000 to DIMinsion variables always needed by CSUB itself. Variable DIMinsions for variables specific to a particular program are generally DIMinsioned within the program itself.

Line 7 - Always a GOSUB 65100 to set numerous variables needed by CSUB and the programmer. These are generally variables used by all programs running under CSUB.

Lines 10-49999 - These are the unreserved program line numbers

available to all programmers for writing their programs.

Lines 50000-50999 - The beginning 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. Input automatically occurs at position 1 on line 3 while the prompt string specified always appears at position 1 on line 2. This line number is the line number RESTOREd to by a FNF call. The inputs can be moved from line 3 by the use of C2 and C3 offset variables discussed later.

Lines 51000-51999 - The beginning of the DATA table used by FNE inputs. Input or display occurs at the specified position.

52000-58999 - Reserved for subroutines or additional function definitions needed by the programmer.

59000-59999 - Reserved for user-defined print formatting routines. These routines control all printing done on the CRT and only on the CRT. Print formatting for hard copy is presently left up to the programmer.

User-defined formatting statements should have the following form:

```
59000  ONINT(A5)  GOTO  59010,59020,59030,59040,59050,59060
59010 !%$10F2,B(A8)\RETURN\REM 1
59020 !%9F4,B(A8)\RETURN\REM 2
59030 !%2I,B(A8)\RETURN\REM 3
59040 !%1I,B(A8)\RETURN\REM 4
59050 !%6I,B(A8)\RETURN\REM 5
59060 !%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. This 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.

60000-65535 - These lines are reserved for CSUB itself.

65000 - DIMinsion for CSUB variables

65100 - Defines variables needed by CSUB.

Organization of Data in the DATA Tables

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 REFERENCE CHART FOR DATA STATEMENTS for rapid summation of the DATA that is contained in the DATA statements.

REFERENCE CHART FOR FNE DATA STATEMENTS

	FIELD #							
TYPE	1	2	3	4	5	6	7	8
NUMERIC	1.X	Line#.Y	POS	#CHAR	FORMAT.Z	LOWLIM	HILIM	B(SUBS)
STRING	2.X	Line#.Y	POS	#CHAR	0	LEFT$	RIGHT$	0
STR/NUM	3.X	Line#.Y	POS	#CHAR	0	LEFT$	RIGHT$	B(SUBS)
DATE	4.X	Line#	POS	0	0	0	0	B(SUBS)

where

X=0 then display and input
X=1 then display only
X=2 then input only
Y=ASCII value of character to be displayed on input
Z=number of spaces to right displace numeric input

FIELDS OF DATA STATEMENTS

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 DATES

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:

.0 = Input and Display
.1 = Display Only
.2 = Input Only

EXAMPLES:
1.0 = 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 constantly arises in the writing of application programs. If a particular input is asking a question where only specific alphanumeric responses are allowed, then this is the TYPE DATA statement to use. As an example, if a particular input is requesting a yes or no response (Y/N), then 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, then the input will allow only a response of "Y" or "N", but not "R", "S" or "T". If field 7 is a 3, then inputs of "Y", "N", and "R" are accepted. If a "Y" is input, then the function returns to the specified B subscript (field 8) the value 0. If an "N" is input, then a 1 is returned, and if an "R" is input, a 2 is returned, etc.

First Field (FNF)

This field is identical to FNE's 1st field except that a minus sign is placed in front of the type specification.

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:

1. $ (36) = Dollars
2. # (35) = Numbers
3. % (37) = Percentage
4. * (42) = String Info

For TYPE 4 - DATE input and display, the integer portion only is entered in Field 2, as date input and displays are automatically formatted 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.

Second Field (FNF)

This field contains the prompt that is to be displayed on the 2nd line of the CRT for the input.

Third Field (FNF) - This field represents the display character that is the fractional portion of Field 2 for FNE DATA statements.

Third Field (FNE) - Fourth Field (FNF)

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.

Fourth Field (FNE) - Fifth 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 fractional part of field 2. If a 10 digit string is to be input, then

field 4 will be 10 and field 2 will be X.42, where X=line number of the input. For TYPE 4 - DATE inputs, this field should be set to 0, as the routines handle the number of characters displayed automatically.

Fifth Field (FNE) - Sixth 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 0. For numerics, the value in this field will be converted to A5, where it is passed to the user-defined formatting routines at 59000. Standard formats are provided to assist programmers. The fractional portion of this field represents a displacement to the right which is used to displace an input from its normal display position.

Sixth and Seventh Fields (FNE) - Seventh & Eight (FNF)

These fields have different definitions for each of the four TYPEs of DATA. For TYPE 4 - DATE, these fields are both set to 0. 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 strictly 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$, respectively, of the position within B$ where the string information is to be input to or displayed from. 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.

Eighth Field (FNE) - Ninth Field (FNF)

This field represents the subscript of B( ) for all numeric variables to be displayed or input. For TYPE 2 - STRING inputs and displays, this field is set to 0 because fields 6 and 7 define the storage parameters for the string variables. The number used in field 8 is the variable reference number to be used when accessing any particular numeric variable during programming.

Variable Reservations and Definitions

All variables A, B, C, and D are reserved. This includes string variables, numerics, and numeric assays.

A9 - "back up" variable. This variable will equal 2 after any input in which a control B (^B) has been typed. A9 should be checked after every input to determine of the user wants to "Back Up".

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

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

C$ - contains ALL string references for TYPE-3 inputs . Use of C$ is explained elsewhere.

C1 - 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 10 numeric values, into B(1) through B(50), and set C1=0, then function calls will display or input to the first record. If C1=10, then the second record will be displayed or input; if C1=20, 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 - Line Number offset variable that offsets any CRT display or input by the value of the variable. This is a non-destructive offset and is used primarily for formatting or displays when inputs and displays need to be slightly different.

C3 - Character Position on Line offset; similar to C2 except that the offset is on the same line. Offset is to the right.

C4 - String offset; similar to C1 except that this variable controls the offset of B$.

C5 - STRING-NUMERIC (TYPE 3) offset variable that controls the offset for working with the string-numbers of C$. If C5=0, then the string-number functions work as described elsewhere. If C5=1, then the first character of C$ is ignored and processing continues in the normal manner, starting with the second character in the string.

C9 - This variable must be set to the number of bytes required to store a numeric variable (5 for 8 digit precision).

D2 - This is a variable that is set by CSUB after determining if the user has a single or double density version of BASIC (Release 4 or 5). This byte is FILLed by most application programs when it is neccessary to turn off the CR & LF of BASIC. This procedure is required when extensive cursor addressing is to be used in an application programs. It keeps unwanted Carriage Returns and Line Feeds from occuring during the cusor addressing.

USING CSUB

Keyed Access File Maintenance (F/M) Program

The BONES program is to be used as an example in demonstrating the use of CSUB. This program should be LOADed and LISTed on hard copy (if available) and consulted while reading this section.

To help the programmer organize his thoughts and programs several work sheets are provided. One should generally start with the FILE STRUCTURE sheet first. In this particular case we are going to make a keyed access file using an account number as the index.

After establishing the file structure, we complete the MASK Sheet to show where our mask as well as variables are to appear on the CRT. All positions are carefully mapped on the mask sheet so that we can

exactly determine our DATA statements for all variables concerned. Next one can generally fill out the FNE DATA TABLES worksheet. See the enclosed example. Check the referance chart or the section on FNE if you have trouble determining the value for any specific field. All fields must have some entry even if it is zero.

At this time we should be ready to do the FNF DATA TABLES chart. This is, in essence, the prompts that will be used during program execution to help guide the user through the program. In this particular case we have four prompts. Sometimes the prompts won't be known until we are actually writing the program. However, the more structured our approach is to the program writing, the more likely we are to know the prompts before we even start writing the program. One should have a good idea of what the program is to do before one starts writing it!

The VARIABLE MAP can now be filled out. Since all variables displayed or input from the CRT and all file variables are contained within B() or B$ we must carefully map out their position within these maps. The more complex the program the more important it is for the programmer to exactly define the variables used. The VARIABLE MAP gives the programmer an overall view of his variable structure.

Multiple files can be accessed by MAPing their variables into different regions of B$ and B(). Remember that all the string or numeric variables for any one file must be consecutive within B$ or B(). The starting position can be anywhere within the limits of the variables and has to be specified in FNG when reading or writing a file. The position of the variable is defined in the DATA tables when imputig or displaying usig FND, FNE, or FNF.

PROGRAMS INCLUDED ON CSUB DISK

1. BONES - A standard 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 51000) should be written to accurately reflect your file structure. FNF DATA statements may need to be changed if they do not reflect what you want to do.

2. INBONES - A standard program for inputing information into a sequential file. The parameters at the first of the program should be checked for proper file & program names, ect. when changing this program. Also one needs to set the FNE and FNF DATA statements to reflect the particulars for your program.

3. PRBONES - This program is a standardized program for printing out the contents of a sequential file. This program features a modular type of printout where records are printed as an entity. The routine at 500 should contain the heading for the top of each "page". The routine at 600 is the one that actually prints a record. To change this program to print a different file both of these routines should be modified to reflect what you are printing. There are generally no FNE DATA statements for this type of print out. Also the definitions

before LINE # 100 should be examined and modified to fit your desired circumstances.

4. CSUB - CSUB is APPENDed onto all application programs. This APPEND is performed at RUN time and is dependant on the CALLs on lines 1,2, and 4. Also. machine code must be in place at the top of the 2900 section in DOS to allow the APPEND command to be an executable statement. CSUB will not RUN by itself.

5. MENU - This program merely allows one to CHAIN to all the programs on this disk by selecting the program from a menu. MENU operates under CSUB.

6. FILEXFER - This is a program that allows one to easily transfer information from one data file (Type 3) to another. If the file specified to receive the data exists, then the information from the source file is written over any data previously entered into the file. If the destination file does not exist it is CREATED the same size as the source file.

7. MEDIT - This program is the Mask EDIT program. It's function is the creation and editing of masks. Masks are created by embedding cursor addressing information within a string followed by the ASCII characters that one wants to appear at that specific location. The cursor addressing and clear screen information is set at the first of the program and must be defined properly for the specific CRT one is using. Masks are "built" with this program by specifying the INSERT command, entering the proper coordinates taken from the mask worksheet, and then typing in the actual information that is to appear on the screen. Other commands are provided for modifying the mask in different ways.

When making a mask the first line should contain the program name. Lines 2,3, and 4 on the CRT are reserved for the prompt, Input, and error messages respectively. Nothing should be entered on these three lines because the routines of CSUB erase these lines periodically during their execution. Mask names are generally made by adding an ampersand (&) sign to the program name that is using the mask.

8. PMASK - This program will print a mask on paper. Device #2 is assumed to be the printer and CRT together. If much editing of the mask has occured, this program will often not print the mask correctly.

9. PNEFILE - This program allows one to Print any (NE) data (Type 3) FILE on the CRT or Printer. After execution, this program CHAIN's back to the MENU.

10. FKCREATE - This program allows one to CREATE and initialize a keyed access file and its associated KEY file. After taking the parameters of your files, this program CREATEs the appropriate files and writes all the records as blanks. The key file is set so that all records are blank and available for use.

11. FCREATE - This program CREATEs the appropriately sized sequential

file. The three beginning numerics are written into the file so that the file is avaialbe for immediate use by a FNG function call.

Several additional programs are provided on this disk to demonstrate other features of CSUB. These programs are not documented at the present time but should offer help to anyone studying them.

Trading Type I Programs for Library Membership

One of the most important aspects of this CSUB package is the standardization it brings to writing and running programs under North Star BASIC. We believe this aspect is very important if large numbers of programs are to be made available to work for large numbers of people. Micro Mike's Program Library has been created to facilitate the cataloging and storing of large numbers of programs. We are currently interfacing a 50 megabyte hard disk to our Horizon computer to store the programs of the Library. This computer will be equipped with a modem that will allow members to call 24 hours a day and recieve programs that are in the library.

In order to build the library as quickly as possible we are soliciting programs from interested parties to be included in the library. Membership in the library will be traded for programs that are submitted and excepted by Micro Mike's, Inc. We are looking for people that have specific knowledge of problems that they would like solved by application software. A typical "program" submitted should be a complete turnkey package. These packages will generally have from 5 to 20 actual programs written under CSUB. Each program should be a module that performs a specific task. Programs should be connected by a MENU program and should CHAIN to the appropriate program.

The CSUB disk as provided shows only the most elementary techniques of using CSUB. Additional information can be obtained by purchasing any of the application programs now available from Micro Mike's, Inc. and studying them. Additional Documentation on using the more advanced techniques of CSUB will be available in the future for a nominal charge.

Programs to be traded should be submitted on diskette with all of the worksheets filled out for all programs included. Programs will not be rejected arbitrarily. Rejections will occur only when in our estimation sufficient time and energy has not been expended to make the programs useable by lay people.

Packettes of CSUB worksheets are available for a nominal fee from Micro Mike's, INC.

MMike's

MICRO MIKE'S, INCORPORATED 905 South Buchanan Amarillo, Texas 79101 U.S.A. 806/372-3633

NOTE FOR PROGRAMS USING CSUB

Before programs that use CSUB can function properly, two important items must be considered: cursor addressing and clear screen. These items are extremely machine dependent (CRTs) and have to be set specifically for most users' systems.

Cursor addressing is normally conducted by using an escape sequence. With the ADM-3A, the sequence is Escape = R,C (27,61,R,C), where R and C represent the Row and Column coordinates. There is generally an offset amount that must be added to the actual position so that the cursor is positioned correctly. This function is handled by Function A (FNA) within CSUB. FNA is called by passing the Row and Column position where the user wants the cursor to appear (X=FNA(R,C)). FNA is DEFined at 60110-60120 in CSUB and must be correct for CSUB to function. As supplied, CSUB supports the Escape Equals sequence which will work on the ADM-3A, Soroc IQ-120, and the Intertec Intertube Version 1.5X. For all other terminals or video display units the appropriate lines of CSUB must be modified to nondestructively move the cursor to the correct location.

Z$ is used in all Application Programs to clear the screen on the CRT. Z$ is defined in line 65120, and must be set to the proper value or sequence needed to clear the screen on your particular CRT or video display. For the ADM-3A, Z$ must equal CHR$(26). For the Soroc the proper definition of Z$ is the sequence ESCape, CHR$(12). The Intertec Intertube uses a CHR$(12) to clear its screen.