I have ... to update my disk catalog, write several documents (including this one), update my household inventory, prepare my 1987 tax return (thanks, Earl), prepare a graphic/sound display for a presentation I'll be making next month, play a few games, update my budget (home AND office), organize an equipment inventory for a friend, and prepare AND translate a draft and field program for school use! (puff, puff, puff ... whew!)

So, now that my HUMILITY index has been established, what exactly IS my point?

"Well, Sonny", here the wizened old man, stooped and bent over his cane, rubbed his stubbled chin as he reminisced, "I re-call how in them old days, when that warn't so much spectaculor software, us users actuallly had to WRITE OUR OWN BY OUR OWNSELVES!"

"Gawsh!", breathed the young wide-eyed innocents gathered at his feet, clutching their 64's and "SKATE OR DIE" game disk to their bosoms, "You mean you REALLY ACTUALLY write ... uh, wrote programs that made the computer WORK!!!"

The old fellow shifted his gaze down into their innocent, wondering faces. "Yup, an' they even worked, an' if they didn't work like we wanted WE CHANGED 'EM, MADE 'EM WORK PROPER-LIKE!"

"NOW, no kiddin'!", they murmured in awe as the camera pans away from the scene to focus on the setting sun on the horizon.

Ahem! Sorry, I got carried away. But it's true, there was a time when we (the users) CONTROLLED and USED THE COMPUTER. WOW, too often, we're simply CONSUMERS of computer software. I often make comparisons to music, because that's where my heart lies. I know so many people who CONSUME musical products and I KNOW HOW MUCH THEY'RE MISSING BY NOT KNOWING how to create the music themselves. I know that mass marketing is turning everyone it can into computer product CONSUMERS, I'm one myself - and it's a nice and comfortable thing to be - no fuss, muss, or bother as long as you can find a program that DOES what you want, or one which ALMOST does what you want. (And that's NOT as easy as you might believe.)

Now, I'm no fool, and my ma didn't raise no dummy. I don't believe that everyone can be a super-programmer. Nor do I believe that everyone would ENJOY programming, but the only way to really make full use of your machine AND your software is to learn a bit about making the machine YOUR tool following YOUR instructions. Knowing how one goes about making your computer store and sort information will help you select software to suit your needs more effectively.

One more comment on programming. I know that the vast majority of computer users (including our club members) spend most of their time using OTHER people's software, but that's no sin - like I said, I do it myself! But sometimes we spend a lot for software, and invest a lot of time LEARNING to use the software, when a simple, self-made program would do what we want, could change to suit our own changing circumstances, cost little (especially if programmed in BASIC) and take NO time to learn! A case in point: the extent of most people's need for a home inventory program is so minimal that writing your own (with the knowledge gained from reading Richard's or other columns in this noble journal) or altering a FREE public domain program to meet your needs can be a doubly rewarding adventure.

So that's my soap-box for this month. I hope you'll consider the beast you sit before so much each month. Learn about it and its inner workings! You'll only learn to love it more, 'specifically if it's a 64! (Hey, there's a poet inside me!)

See you USERS next month!
Maze on (the) March!

Computerfest is over and I would like to take this opportunity to thank the club members who so kindly gave of their time and energy to help make our club's presentation so successful. Our main objective in setting up a display at Computerfest was to build awareness of CUGS and judging by the number of people we talked with and explained the operation of our club to, we easily met this objective. We sold four new memberships at Computerfest and judging from the number of enquirers we had, I am sure we will have a number of new members joining CUGS as a result of our display.

Special thanks to Earl and Ken for the disks they prepared which we had running on a C64 and 128. These programs brought many people over for a closer look and to talk with us.

I also would like to thank all the club members who so kindly gave of their time to man the booth and answer the numerous questions we had. Thank you to all the club members who dropped by to say hello. Your comments and support were appreciated.

I would especially like to thank all the members of the public who dropped by and got information from us. Those of you who have joined our club have special meaning to the rest of us because you are the reason we set up our booth - to build interest in our computer club.

A final thank you to the Apple II user's group for sponsoring Computerfest and giving us the opportunity to be a part of this great display. Judging by the crowds, there was a definite need for such a show and it was very well received.

The executive have tentatively planned some of the activities and presentations for the upcoming meetings. Our February meeting will be divided into two parts: First, Ken will demonstrate two music programs - "SIDPLAYER" and "MASTER COMPOSER". In the second part, Earl will demonstrate his "Income Tax" calculator. For the April meeting we are planning on examining database programs. May has been scheduled as a "swap night" in which members will get a chance to trade public domain programs which they have written themselves or have obtained from others and would like to share with other members. June is our final meeting before we break for the summer months. Part of the June meeting is planned as a look at the 1581 disk drive. I hope there will be something meaningful for everyone in each of these meetings.

Virtually every computer club in the city was there to hype their group and maybe gain a member or two - including a TI-99A group and an older version ATARI user group. With the removal of the selling table in the middle of the room, the room stayed more comfortable this year. Several local commercial firms had impressive and extensive displays and did a little Sunday sales business besides showing what they had to offer for virtually every machine represented.

The flea market area was moved to a separate room, which made more space all round. Security was much improved (there was someone watching the single entrance door, checking bags and loosely held computer goods. All purchases were heat-seal bagged in order to pass the door inspector. Unfortunately, although the sale was operated most effectively and efficiently, the goods offered were priced rather high for their condition or usefulness. However, that's not to cast a shadow on the sale itself - it's a great idea, and was well managed. The prices were probably a reflection of the inexperience of the persons putting the items up for sale.

All in all, a great time! Well-attended. I hope all the clubs who made the effort to be involved managed to gain a few new members and enjoyed (as I did) browsing the booths, seeing what other clubs had to offer. Hats off and a pat on the back to all the APPLE USER GROUP MEMBERS for a SECOND great event!! You've every right to be proud. See you next year!

SCRATCH 'N' SAVE

For those of you that picked up the CUGS Income Tax Program last month, my humble apologies. There were about half a dozen errors in the two programs - two were in the child exemptions, and one in the old-age exemption. The others were line duplications which did not affect the outcome of your tax calculation. Please remember to have your copy exchanged.

With our additional purchases of public domain or freeware programs, we will be increasing the size of our library by an estimated twenty-five disks in the next couple of months. Seven of these disks are included in this month's MONITOR. After inclusion, the club will be printing a new CLUB DISK LIBRARY listing, which each paid-up member will receive free copy. Additional copies may be purchased for a buck.

One of the items that I picked up at COMPUTERFEST in February was a good-looking cover for my EPSON printer; or so I thought. Although it is identified as being made for an FX80 or FX85, it is much too small. The dimensions of the cover (my measurements) are 14 1/2 inches wide by 12 inches front to back. If you own a printer with those dimensions, you may purchase it from me for five dollars.

One note in closing that I should have mentioned earlier. When you purchase any club library disk and find any kind of problem with it, or with a program on the disk, please let us know about it. We would like to correct any problems as soon as possible so that others don't run into problems.
IN A DARK, DARK ROOM...

There...

There on a disk...

There on a disk, hidden way back....

Way back on the back shelf, under the (legal) copy of PAC-MAN,

was the PROGRAM!

What program?

The one you wrote back in '86 when you needed a quick way to print a bunch of 3-line labels for your disks.

Or the one you wrote in '85 as a math drill for your aspiring 3-year-old Einstein!

Or the one you wrote to keep track of your Zone Rec. Sports Team's progress that year they sucked you into coaching!

So, what about these hidden treasures?

So, they are hidden treasures, and you can cash 'em in!!! Any meeting members are welcome to submit public domain software (theirs or others) to the club disk librarian (Earl or assistant Steve) and have your effort immortalized in our next catalogue!

Soon, (watch for it, now) VERY soon, that program in your past could make you part of a very exciting event.

In a couple of month's time, we'll be holding our FIRST ANNUAL SOFTWARE SWAP NIGHT. Anyone with public domainware willing to share can come and swap your goods for theirs, as easy and often as you please!! The ONLY restrictions are that the work be truly PD, FREE- or SHARE-ware and that it is not (yet) part of our club library. Your "admission" to the event will probably be permitting the club to ADD your treasure to our library for future members (and those to share in person)!!

We hope to make it more than just a bunch of odd-looking people sitting around staring at computer screens, so be thinking about it, and start blowin' the dust of YOUR software. Y'all come, y'hear!

MUSIC WITH YOUR BASIC

Our club disks (and many bulletin boards) offer ready-made music to be played in MASTER COMPOSER or S IDPLAYER format. MASTER COMPOSER is a copyright program originally from ACCESS software which allows easy music writing of files which CONTAIN THEIR OWN ML ROUTINE FOR PLAYING as a part of the music file. SIDPLAYER by Craig Chamberlain originally appeared as a series of articles in COMPUTE! a couple of years ago. It has since been upgraded, and now comes with 200+ page instruction booklet and 128/64 variations on a double-sided disk (available from COMPUTE! Publications for $14.95 U.S.). Both give the user exceptional control over the mighty SID chip.

Both allow the creation of files which may be loaded into the top end of a BASIC or ML program and played in the INTERRUPT CYCLE of the computer. This means that the music plays (apparently) WHILE THE COMPUTER DOES OTHER THINGS (runs your own BASIC or ML program).

As I said, music files in both formats are available readily in the public domain (our library has 2 disks - called JUKEBOX disks of MASTER COMPOSER files), and a newly-submitted disk full (50 files) of SIDPLAYER selections (many of which you heard at our display at COMPUTERFEST). The following interjection is printed to allow you to load and play your a selection in one format or the other and add its playing to your own programming. (N.B. MASTER COMPOSER FILES always end with '.C' and SIDPLAYER FILES [new version] end with '.MUS').

To load (and use) a MASTER COMPOSER music file:

1. Move END OF BASIC (to 28000).
2. LOAD the desired music file.
3. Use appropriate SYS or POKEs to play, etc.

SYS SA = 155 (to start playing)
POKE SA + 1079.4 (to stop playing)
POKE SA + 1079.1 to 32 (stop 1 or more voices).

For UNALTED MC pieces, SA=29965.

MC pieces may be loaded ELSEWHERE in memory, and the SA value is simply changed to reflect the new load address.

To load (and use) a SIDPLAYER music file:

1. The program "SID.OBJ.64" must be on the disk with the selection(s) to be played.
2. Extra lines of code must be added to your program which LOAD SID.OBJ.64 and the actual selection to be played.
3. Use the lines below. The lines from 57000 on MUST NOT BE MOVED OR CHANGED!!
4. Substitute your music file name for the "MUSFILE" in line 200.
5. Each bit of variable SS turns on one of the 264's 3 voices. Thus, POKE SS,1 turns on VOICE 1; POKE SS,4 turns on VOICE 3, and POKE SS,7 turns on all three!

```basic
120 DN%=8:SA=7805:SA=7815:SY=7825:SP=783  
130 PRINT "TUNING INSTRUMNTS...":PRINT:GOSUB 570  
0:REM LOAD SIDPLAYER ML  
200 FS$="MUSFILE";LA$=PEEK(49)+256*PEEK(50)+1000:G  
OSUB 57500:REM LOAD SONG  
210 SYS HK$;REM HOOK (INSTALL)  
220 POKE SX,LO$;POKE SY,HI$;SYS PL$;REM SET FOR PLAYI  
NG  
230 X=PEEK(SX)+256*PEEK(SY);REM GET ADDRESS OF TEX T LINES  
240 IF PEEK(X) THEN CHR$(PEEK(X));K=K+1:GOT  
O 245:REM PRINT UNTIL CHR$(8)  
250 POKE SS,7;REM START PLAYING MUSIC  
260 IF PEEK(SS) AND GOTO 260:REM STILL PLAYING  
270 SYS HU$;REM HUSH  
280 SYS DP$;REM DROP (REMOVE)  
290 END  
57000 POKE SA,1;POKE SX,DX;POKE SY,1;SYS 65466;FS$  
= SID.OBJ.64$;GOSUB 59000  
57010 POKE SA,0;POKE 65493;IF PEEK(SP)AND GOTO 591  
00  
57020 SS=49153;FL$=49153;HK$=49153;PL$=49164;HU$=4998  
T;DP=49985;RETURN  
57050 POKE SA,1;POKE SX,DX;POKE SY,1;SYS 65466;FS$  
= HUS$;GOSUB 59000  
57510 HI=INT(256);LOA=LA-256;MH$=  
57520 POKE SA,0;POKE SX,DX;POKE SY,1;SYS 65493;IF PEEK (SP) AND GOTO 591B  
0  
57530 LA=PEEK(SX)+256*PEEK(SY);RETURN  
59000 FOR K=1 TO LEN(FS$);POKE 584,K;ASC(MIDS(FS$K),NEX  
T  
59100 IF P=4 THEN PRINT "FILE NOT FOUND";END  
59110 P=P+1:IF P=5 THEN PRINT "DEVICE NOT PRESENT";END  
59120 PRINT ST:END
```

UPCOMING CUGS MEETINGS FOR 1988

Wed. Apr. 6
Wed. May 4
Wed. June 1

Write these on your calendar.

All meetings start at 7:00 pm

Meetings at the Northwest Leisure Centre
Room 2
Whoops!

Whoops! I knew it would happen - as soon as you start adding listings to a mag such as ours, the editor's eyes begin to fail and some silly types enter the picture. I hope none's ready to kill over the two small errors below. Remember, I'll happily accept vitriolic criticism from anyone who's ever edited a perfect paper (please attach proof).

In the 128 WINDOW program to display graphic pictures, line 60 of the MULTICOLOR BIT MAP LOADER should have read:

   60 BLOAD"<COLORMEMORY MAP";B15,P55396

(It appears I gave the 128 an extra memory bank (B16) in last month's article.

In the 64 version of the same program, my printer happily omitted a couple of Commodore graphics and replaced them with some unexpected newline numbers erasing line numbers in the process. The lines following line 58 should read ...

   100 PRINT"PRESS 'Y' TO SEE DRAWING"
   110 PRINT"PRESS 'N' TO RETURN TO NORMAL SCREEN"
   120 PRINT"PRESS 'Q' TO QUIT"

Sorry for the goofs. We'll keep trying to get better.
In the last two articles, I examined sequential data files. In this article I will turn to relative data files. A relative file is one which allows data to be accessed in random order—either for READ or WRITE purposes. This is definitely different from sequential files which must be read from and written to starting at the first character in the file. To obtain this random accessing of data some special programming sequences are involved in preparing the file structure. (Note: although relative files could be referred to by the name "random access file", this term is more correctly used to refer to another type of file - user files. You should be careful not to confuse the two as they involve completely different programming techniques.)

Before examining the details of relative data files, the differences between sequential files and relative files should be noted. The main difference between these two is in how data in the file is accessed. Sequential files must have data accessed sequentially, from the start to the data you want. Relative files can have data accessed in any order wanted. Sequential files can have data of variable length while relative files require that all data items must be of the same length. As a result of the two above mentioned differences, relative files usually require much more disk space than sequential files but data can be accessed much faster than sequential files. Relative files require two disk buffers for operation of the file while sequential files only require one buffer. As a result, more sequential files can be accessed simultaneously than relative files. It should also be noted that relative files can not be used on a cassette which must access data sequentially.

In this article I will use BASIC 2 when looking at relative files. If you have a 128, BASIC 7.0 contains a number of commands (OPEN#, RECORD#, DCLOSE) which make working with relative data files much simpler than the BASIC 2 version. BASIC 2 commands will work however on any upgraded version of BASIC.

The preparation and use of relative data files usually involves THREE distinct steps. (and usually a minimum of TWO different programs.) These steps are:

1) Reserve space on the diskette for the file.
2) Fill up the space with some character. (Called padding the file.)
3) Use the file by writing data to it, reading data from it, or changing data values.

The first two steps are often done in one program which has the function of initializing the file. A separate program is usually then used to access the data from the file.

1) Reserving space on the diskette for the file.

Before looking at the process of preparing the diskette to store a relative file, the structure of a diskette should be reviewed. A diskette is divided into BLOCKS. Since a block contains 256 bytes of storage and 2 bytes are required to link to the next block, the maximum length of an individual unit of data is 254 bytes. Each individual unit of data is called a RECORD.

To create a relative file there are two values that must be determined before the program to create the file can be written. The first value is the number of records. The number of records is the number of units of data that are to be stored. The number of records can be set larger than actually needed and the maximum is only limited by the space available on the diskette.

A second value needed is the total of the lengths of all the FIELDS within the record. A field is an individual unit of data. Fields of data are best written using carriage returns (ASCII code 13) to separate each field. Doing this makes it much easier to access individual fields later. The total length is determined by adding the lengths of each field plus 1 for a carriage return for each field. This total length CANNOT EXCEED 254.

2) Padding the file.

This process involves putting some initial data in the space that has been reserved so that the main program has something to read from the disk. The padding of the file can be done with any character but be careful that the character chosen doesn't make it hard to work with data later.

3) Main program.

This program is the one that actually uses the relative file. It is used to enter, change, or delete any data on the relative file.

CREATING A RELATIVE DATA FILE

As mentioned above, this usually involves two steps. The first step is to initialize the file - set up on the diskette the space to store the data. The second step is to fill the file with some character - called padding the file. Although this second step is not strictly required, it is good programming to include it and it also gives a quick check for errors in the file formation. These two steps are usually included in one program whose only function is to create the file. This program is completely separate from another program which uses the file to actually store data.

Before creating a relative data file, two pieces of information must be determined. These are: 1) record length - total of the lengths of all fields plus 1 for the carriage return for each field; and 2) number of records - how many records are to be stored, this should be slightly larger than the suspected number needed. A great deal of thought should be put into the determination of these because once the file has been created, it cannot be changed without losing any data that has been entered.

To make it easier to follow, I will use an example of an address book in working with relative data files. My address book will contain the following fields (length of each in parentheses): last name (15), first name (12), street (25), city (15), province (4), postal code (7). The total record length is 83

(15+12+1+25+1+15+4+7=83). I am going to make my file hold 200 records. The explanation of the process follows. You may want to follow the sample program at the end as you read through the remainder of this article.

There are THREE steps involved in creating a relative data file with BASIC 2.
A) OPEN A LINE OF COMMUNICATION.

In actual fact opening a line of communication involves two parts. First a line of communication must be set up to the disk command channel (secondary address 15). Second, a file must be opened over regular channels to pass the data.

OPEN 15,8,15  
OPEN 1fn,8,sa,"file name,L," + CHR$(rl)

The first statement opens the line of communication to the disk command channel.

The second statement opens the relative file.

1fn - logical file number used in opening the file. (I will use 2)
sa - secondary address (2 - 14). (I will use 2 here also)
L - length (note comma before and after)
rl - length of each record in file (0 < rl < 255)

(ours = 83).

B) MARK THE LAST RECORD IN THE FILE.

This is done with one statement.

PRINT#15,"P"+CHR$(sa+96)+CHR$(lb)+CHR$(hb)+CHR$(pl)

"p" - position or pointer - this tells the disk drive to look for a certain record.

sa - secondary address used in opening the file. This must be added to 96.

lb - low byte of record number

hb - high byte of record number

pl - place in record. This usually is 1 (one) to indicate the start of the record.

To calculate the low and high bytes of a record use the following formula. The high byte is the integer of the record number / 256. The low byte is found by taking the record number - hb*256. If rn = record number; lb = low byte; and hb = high byte; then hb = int(rn/256) and lb = rn - (hb*256). This calculation is a real pain so the easiest way to use it is to put it in a subroutine and let the computer calculate it for you. All you have to do is to send the record number to the subroutine. See lines 10000 - 10050 in the sample program.

Note that you position to a record through the COMMAND CHANNEL (sa = 15). In a program use a statement like: RNW 200:GOSUB 10020 to position to the last record. Now something (like 'end') should be printed on this record which will cause all other records to be created. This is done by using PRINT#2,"END".

C) CLOSE THE FILES.

To close the files, simply use the close statement for all opened files. If you close the command channel at this time, you will get a flashing light on the disk drive which indicates an error. This error, however, is the "record not present error" which is not really an error as the record wasn't present but you just created it.

CLOSE 1fn

MORE...

CHECKING FOR ERRORS

At every step the disk should be checked for errors. This is done using the normal error checking routine except one additional error must be "trapped". This error is number 50 - record not present error. This error condition will exist until after the file has been created. To check for errors and also trap error #50 set a flag to 0 (FL=0) and then gosub to an error subroutine (see lines 10100 - 10150 in the sample program).

PADDING THE FILE

Once the file has been created, it should be padded (filled with characters) to make it easier to put data into the file and to distinguish a filled record from an empty one. The character(s) selected for padding must be chosen with great care as improperly chosen characters could later affect the flexibility of using the data.

The easiest method to pad the file is to create a string which consists of all the padded fields added together with a chr$(13) between each. The actual padding process consists of FOUR steps:

A) CREATING THE PADDING STRING.

Creating the padding string involves making a string of characters which can be stored in each record on the diskette. This is done so that a character will appear to mark the fields to make data entry simpler. The choice of a character (or characters) to pad the file is very important. You must pick a character that will not interfere with data entry. For example - a letter of the alphabet, while easy to use, could cause interference with searches through actual data later.

The procedure is very simple. Pick the longest field and create a string the length of this field and assign it to a variable. By using the left$( function, parts of this string are added together to create one string for the record. After each field (except the last) add a chr$(13) as a field separator. For my example I am going to use a plus sign (+) followed by blanks as my padding string. Let p$ = "  
("' + then 24 spaces). Build the padding string by: p$ = left$(p$,15) + chr$(13). This covers the first field. To add the second field use: p$ = p$ + left$(p$,12) + chr$(13). Continue to add the third field: p$ = p$ + p$ + chr$(13) and the fourth field: p$ = p$ + left$(p$,15) + chr$(13). Continue in the same manner to add the fifth and sixth fields. p$ will now have a length of 83 bytes. Do not put a carriage return after the last field - one is put automatically when the file is printed to the diskette. VERY IMPORTANT: THE LENGTH OF THE PADDING STRING MUST EQUAL THE RECORD LENGTH SPECIFIED WHEN THE FILE WAS OPENED.

B) OPENING THE LINE OF COMMUNICATION.

A relative file is simultaneously opened for both reading and writing.  
OPEN 1fn,8,sa, "file name"

C) PRINTING THE PADDING STRING TO THE RECORDS.

There are two steps involved in printing the padding string. First, the read/write head must be positioned to the record and then second, the padding string can be written to the disk. Both of these steps are usually repeated within a loop so that every record in the file will be padded. BASIC has a problem with the positioning to a record - the read/write head sometimes gets lost. To prevent this from happening you must reposition the pointer after each record was printed.

MORE...
D) CLOSE THE FILE.

Once all work is completed, the files must be closed.

CLOSE 1: CLOSE 13

In the next article I will examine writing a program to write data to the file and access the data from the file.

100 REM RELATIVE FILE ** ADDRESS **
110 : 120 REM RECORD LENGTH = 83 // # RECORDS = 200
130 : 200 REM CREATE RELATIVE FILE
210 : 220 : PRINT "CREATING FILE ** ADDRESS **"
230 : OPEN 15,8,15
240 : OPEN 2,8,2,"ADDRESS.L."+CHR$(3)
250 : FL=0:REM RECORD NOT PRESENT FLAG
260 : GOSUB 10120:REM ERROR?
270 : RN = 200:GOSUB 10020:REM POSITION TO LAST RECORD
280 : PRINT#:"END")REM CREATE FILE
290 : GOSUB 10120:REM ERROR?
300 : CLOSE 2:REM FILE NOW CREATED
310 :
400 REM PAD FILE
410 :
420 : REM CREATE PADDING STRING
430 :
440 : SPACES
450 : PDS=LEFT$(PS,15)+CHR$(13):REM LAST NAME
460 : PS=PS+LEFT$(PS,12)+CHR$(13):REM FIRST NAME
470 : PS=PS+PS+$CHR$(13):REM STREET
480 : PS=PS+PS+LEFT$(PS,15)+CHR$(13):REM CITY
490 : PS=PS+PS+LEFT$(PS,4)+CHR$(13):REM PROV
500 : PS=PS+PS+LEFT$(PS,7):REM POSTAL CODE
510 : IF LEN(POS).<43 THEN PRINT "ERROR":STOP
520 :
530 : REM WRITE PADDING STRING TO RECORDS
540 :
550 : OPEN 2,8,2,"ADDRESS":REM OPEN FILE
560 : FL=1:REM RECORD NOT PRESENT = NOW AN ERROR
570 : GOSUB 10120:REM ERROR CHECK
580 : FOR RN = 1 TO 200:REM PADDNG LOOP
590 : GOSUB 10020:REM POSITION TO RECORD
600 : PRINT#:PS:REM PRINT PADDING STRING TO RECORD
610 : GOSUB 10120:REM ERROR CHECK
620 : PRINT"name PADDNG RECORD":RN:REM MESSAGE
630 : GOSUB 10020:REM REPOSITION TO RECORD
640 : NEXT RN
650 : CLOSE 2:CLOSE 15:REM DONE - CLOSE FILES
660 END
670 :
10000 REM SUBROUTINE TO POSITION TO RECORD
10010 :
10020 : HB = RN/256:LB = RN - (HB*256)
10030 :
10040 :
10050 RETURN
10060 :
10100 REM ERROR CHECK
10110 :
10120 : INPUT#15,ER,ES,T,S
10130 : IF ER < 20 THEN RETURN:REM NO ERROR
10140 : IF ER = 50 AND FL=0 THEN RETURN:REM TRAP ERROR
10150 : ? ER,ES
10160 : CLOSE 2:CLOSE 15:END

Don’t Miss Out!

PRIZE DRAW - CUGS 1988

At each CUGS meeting during 1988 there will be a computer generated draw for a winner of a prize.

RULES:

Paid up members for 1988 only will be eligible.

Draw will be made at the end of each meeting.

The winner must be present at the meeting to claim the prize. If the drawn member is not present, further draws will be made until the prize is distributed.

All prizes must be accepted as is - no substitutions permitted.

The membership list will be updated at break during each meeting so that new members will be included in the draw.

Prize for March draw - disk box

February winner was - Steve Bogues

NEXT MEETING

Wednesday
April 6, 7 pm

Northwest Leisure Centre

Databases in Detail
A careful look at several currently available for the C64/128.