OS·ITEMS

VOL. 2 NO. 4 MAY 1980

IN THIS ISSUE...

EDITORIAL

This month's issue of OSI-tems has been put together in somewhat of a rush because last month's meeting was delayed on account of the Transit Strike. The issue has suffered in the volume of articles that you have come to expect of OSI-tems; but, however, I hope the quality of those articles has not suffered.

How many times have you picked up an established magazine in this or any other hobby and lamented the fact that only a few writers and editors seem to get their articles published? Have you ever thought to yourself, "Why don't those @#$%¢&* SOB's print something I'm interested in?" Well, maybe its because only a few authors with necessarily limited interests and talents submit articles.

Maybe our publication is headed down the same path. So far OSI-tems has been supported, very ably so, I might add, on the backs of essentially four people. Although these people are very knowledgeable, they cannot encompass the whole realm of interests of all of our members. Unless we get some input form said members, they will probably be the people who in a little while will be saying; "Why don't those...

BEAT THE STRING BUG

D. SCHWARTZ

Last month's OSI-tems contained an article by Mike Bassman in which he gave one possible approach for getting around the "garbage collection" problem in OSI BASIC. His solution was not to use BASIC strings at all, but to store the ASCII values of the strings numerically and then print them out using the CHR$ function. While this may work, it is quite cumbersome and defeats one of the advantages of using BASIC, which is its string-handling capability. What I will do in this issueis to give a method that will let you use string arrays in almost the straightforward way they were ment to be used, adding only a couple of lines which in effect will collect the string "garbage" for you before it has time to accumulate, and therefore will save the interpeter from trying (and failing) to do this itself.

To do this, you must know where, as well as how, BASIC strings are normally stored. Every string variable in BASIC consists of two parts- a pointer to the string and the string itself. The pointer contains one byte giving the length of the string and two bytes specitying the address in memory at which the actual string begins. When a string is defined within a program by setting it equal to a string which is given explicitly within quotation marks, the pointer will simply point to the program itself, to the first byte after the quote mark. The same thing happens when a string is read from a DATA statement-- the pointer will point to the data statement to the first byte of the string, and no further storage need take place. These kinds of strings cause no trouble.

But when a string is formed in some other way--by a string function, by being INPUT, or by concatenating two other strings together (adding them), the resulting string must be stored somewhere in RAM away from the program itself. BASIC maintains another pointer to tell it where to store these strings. This pointer is stored in decimal locations 129 (low byte) and 130 (high byte) in our version of ROM BASIC. At the start of a program RUN, this pointer is set equal to one more than the address of the highest byte in RAM memory available to BASIC. The first string formed requiring storage outside the program, if it is of length N, will be stored in the N highest bytes of available memory, and the pointer will be decremented by N. The next string formed will be stored in the next M bytes of memory directly below the first string, if M is the length of the new string.

This works fine until you realize what will happen when the value of a string is changed often. Consider the loop:

```
A$="":FOR X=1 TO 20:A$=A$+CHR$(X):NEXT X
```

The first pass through the loop forms a string of length 1, the second pass a new string of length 2, and so on. After executing this loop, the total number of bytes used for string storage will not be 20, as one might guess at first, but rather 1+2+3+....+19+20*10=210 bytes. The first 190 of those bytes now contain "garbage" which we don't need, but they sit there anyway, and the next string formed will start (or rather end) at the 211th byte down from the top of memory. If we continue in this way, we will pretty soon eat up all of the memory available to the computer, and thats where

trouble occurs. When there is no memory left except that which holds the program, the numeric variables, and the pointers to the strings, BASIC is supposed to "collect the garbage"--sort through the strings it has pointers for and re-arrange them so they are neatly packed at the top of memory, and then continue. It is also supposed to do this when you ask it for "FRE(0)", in order to give a accurate accounting of how much memory is really unused. But unfortunately, the routine doesn't work, and so the system crashes if this routine is called. We must therefore insure that this routine is never called.

How? Well, we noted above that 210 bytes of string space has been used in forming our 20 character string. We would like the resulting string to be stored in the first 20 of those bytes, instead of the last 20, because then the locations from the 21 st on could be used for other purposes. So let us set the pointer thatBASIC uses to tell it where to store strings to the point where we want them stored, instead of where BASIC "wants" to store them. Consider the following three lines:

```
10 PL=PEEK(129):PH=PEEK(130)
20 A$="":FOR X=1 TO 20:A$=A$+CHR$(X): NEXT X
30 POKE 129,PL:POKE 130,PH:A$=A$+""
```

In line 10, we save the current value of the string space pointer in the variables PH and PL before forming our string. Line 20 is the same string forming routine as before, and will still use up 210 bytes of stringspace. But then line 30 restores the string space pointer to the value it had in line 10. The next new string will be written over the same bytes as were used by the first "garbage" strings created in line 20. So we "concatenate" a null string to the end of A$! Since this is a concatenated string, it will be written into the area determined by the 129/130 pointer, which is exactly where we want it. We now have our twenty character string neatly packed at the top of memory. If we follow the same procedure whenever we form a string by combining substrings, we can pack all our strings at the high end of memory and never have to worry about collecting "garbage", because there will be no garbage to collect. The following program should convince you of this:

```
5 DIM A$(26)
10 FOR Y=1 TO 26
15 PRINT Y:A$=""
18 PL=PEEK(129):PH=PEEK(130)
20 FOR X=1 TO 24
30 A$=A$+CHR$(Y+X)
40 NEXT Y
50 POKE 129,PL:POKE 130,PH
60 A$(Y)=A$+""
70 NEXT Y
80 FOR Y=1 TO 26:PRINT A$(Y):NEXT Y
```

As written here, the program will work fine, printing out 26 strings of 24 characters each- but if you leave out lines 18 and 50, or the null string from line 60, it won't.

The above is hardly a typical program, it's more of a worst case example of what can sometimes be needed. A typical program may have a small number of strings which are needed throughout the program. If there are many they will probably be in DATA statements, and thus of the kind that cause no garbage. There will then be string data which is input from the user and processed in some way, but which need not be kept from one major loop of the program to the next.

A good example is the "Adventure 65" game from Technical Products Co. There are many strings in the DATA statements of this game, and then there are the commands which are input from the player. These commands must be matched against the program keywords from the DATA, which means a lot of heavy use of MID$. Therefore, if the game runs for more than a few minutes, the screen starts to flash and you must hit the BREAK key to stop it. But all the substrings isolated from one turn's command string are not needed on the next turn, so why not reset the string pointer for each new turn? Adding the line:

```
109 POKE129,0:POKE 130,32
```

to this program will totally eliminate the problem of having the program crash because of the "string bug". Of course, it won't cure the other faults of the program, but that's another story!

TURNING USR(X) ROUTINES INTO DATA STATEMENTS

Thomas Cheng

If you are writing machine language subroutines for BASIC, one of the major problems which you will encounter is in saving the routine. An alternative to a save from the ASSEMBLER or the EXTENDED MONITOR is to turn the routine into a series of DATA statements in BASIC, then to load in by POKEing the values into the proper locations.

The following program does this, first prompting for the addresses of the locations to be saved (these can be entered in either Hex or Decimal. Hex numbers must be preceeded with a $ sign to differentiate from decimal), then the starting line number and increment. Before hitting RETURN after the second inquiry, the user's recorder should be turned on for the following DATA statements and the two lines which will (1) poke the values into the same memory locations, and (2) set the values for the USR(X) call in locations 11 and 12.

This routine will print out 16 bytes per line of DATA statement, but if this is wished to be changed, the increment of C=B+15 in line 80 can be changed to the desired number of bytes.

```
LIST

 5 REM MACHINE LANGUAGE SAVE
 7 REM ***THOMAS CHENG***
 10 INPUT"START, END";B$,C$:INPUT"LINENO, INC";ST,IN
 20 IFLEFT$(B$,1)="$"THENGOSUB140:P=A:GOTO40
 30 B=VAL(B$):REM IT ALWAYS ENDS UP IN B
 40 B$=C$:IFLEFT$(B$,1)="$"THENGOSUB140:GOTO55
 50 A=VAL(C$):REM-A IS SECOND VALUE
 55 SAVE:PRINT:PRINT:PRINTST;"READN,N2:FORK=NTON2:READQ:POKEK,Q:NEXTK"
 60 ST=ST+IN:PRINTST;"Q=INT(N/256):POKE12,Q:POKE11,N-Q*256"
 70 ST=ST+IN:PRINTST;"DATA";MID$(STR$(B),2);",";MID$(STR$(A),2)
 80 ST=ST+IN:C=B+15:PRINTST;"DATA";:IFC>ATHENC=A
 90 FORK=BTOC:LO=PEEK(K):GOSUB170:PRINTA$;:IFK<CTHENPRINT",";
 110 NEXTK
 120 IFC<>ATHENB=C+1:PRINT:GOTO30
 130 PRINT:PRINT:PRINT:POKE517,0:END
 140 A=0:A$="0123456789ABCDEF":FORK=1TOLEN(B$):FORL=1TO16
 150 IFMID$(B$,K,1)=MID$(A$,L,1)THENA=A+(16↑(LEN(B$)-K)*(L-1))
 160 NEXTL:NEXTK:RETURN
 170 A$=MID$(STR$(LO),2):RETURN
OK
```

LIFE

Salomon Lederman

This program is a C1 implementation of Conway's simulation of LIFE. LIFE is a biological simulation of the life of cell cultures. The program creates each generation and outputs to the video.

The program is mainly in machine language. Lines 1000-1250 POKE in the machine language code, through DATA statements. The rest of the program just does the input routine. The commands are: 1) GO left; 2) GO right; 3) Go up; 4) Go down; 5) Puts a * under the cursor; 6) Executes the program; 7) Deletes the star.

```
5 POKE11,0:POKE12,24:FORX=6016TO6090:READQ:POKEX,Q:NEXTX
7 FORX=6144TO6353:READQ:POKEX,Q:NEXTX
10 FORX=6400TO7424:POKEX,0:POKEX+46800,32:NEXTX
20 C=53775:POKEC,187:POKE530,1:K=57083:POKEK,127:T=32
30 IFPEEK(K)=127THENPOKEC,T:C=C-1:GOTO100
40 IFPEEK(57088)=191THENPOKEC,T:C=C+1:GOTO100
50 IFPEEK(57088)=223THENPOKEC,T:C=C-32:GOTO100
60 IFPEEK(57088)=239THENPOKEC,T:C=C+32:GOTO100
70 IFPEEK(57088)=247THENPOKEC,42:GOTO100
75 IFPEEK(K)=253THENPOKEC,32:GOTO100
80 IFPEEK(57088)=251THENPOKEC,T:X=USR(X)
90 GOTO30
100 T=PEEK(C):POKEC,187:FORX=1TO75:NEXTX:GOTO30
1000 DATA173,0,0,96,141,0,0,96,173,0,0,96,141,0,0,96,233,0,0,233,0,0
1010 DATA238,0,0,238,0,0,233,0,0,238,0,0,233,0,0,233,0,0,96,32,32
1015 DATA0,42,32,32,32,32,32,238,223,24,233,224,24,238,225,24,233,255
1020 DATA24,238,1,25,238,31,25,233,32,25,238,33,25,96,169,0,141,129
1030 DATA23,141,133
1040 DATA23,169,208,141,130,23,141,134,23,162,0,189,173,23,157,144,23
1050 DATA232,224,25,208,245,32,128,23,201,42,208,3,32,144,23,173
1090 DATA 130, 23, 201, 212, 240, 52, 173, 129, 23, 24
1100 DATA 105, 1, 141, 129, 23, 141, 133, 23, 173, 130
1110 DATA 23, 105, 0, 141, 130, 23, 141, 134, 23, 152
1120 DATA 0, 189, 145, 23, 24, 105, 1, 157, 145, 23
1130 DATA 189, 146, 23, 105, 0, 157, 146, 23, 232, 232
1140 DATA 232, 224, 24, 208, 232, 76, 29, 24, 169, 0
1150 DATA 141, 129, 23, 141, 133, 23, 169, 208, 141, 130
1160 DATA 23, 141, 134, 23, 169, 0, 141, 137, 23, 141
1170 DATA141,23,169,25,141,138,23,141,142,23
1180 DATA 32, 136, 23, 201, 2, 240, 7, 170, 189, 163
1190 DATA 23, 32, 132, 23, 169, 0, 32, 140, 23, 173
1200 DATA 129, 23, 24, 105, 1, 141, 129, 23, 141, 133
1210 DATA 23, 173, 130, 23, 105, 0, 141, 130, 23, 141
1220 DATA 134, 23, 173, 137, 23, 24, 105, 1, 141, 137
1230 DATA 23, 141, 141, 23, 173, 138, 23, 105, 0, 141
1240 DATA 138, 23, 141, 142, 23, 173, 130, 23, 201, 212
1250 DATA 240, 3, 76, 130, 24, 76, 0, 24, 36, 36
OK
```

POTATO CHIP INVASION

Mike Bassman

Here is a new game for your C1 by Mike. Save the Earth from the invading alien Potato Chips. The game is a cross between 'Alien Invaders' and 'Hectic'(except this one works). Commands are: 1) Left; 2) Right; 7) Fire.

```
1 REM***POTATO CHIP INVESION***
2 REM***MIKE BASSMAN***
5 POKE605,0
6 FORK=611TO625:POKEK,32:NEXTK:POKEK,255
10 DATA546,604
20 DATA174,59,211,24,173,35,2,105,32,141,55,2,173,35,2,105,0
30 DATA141,56,2,142,195,208,24,173,35,2,201,195,208,3,173,36,2
40 DATA201,208,208,1,96,56,173,35,2,233,1,141,35,2,173,35,2
50 DATA233,0,141,36,2,76,34,2
60 FORX=546TO571:I=PEEK(X+64490):POKEX,I:NEXT:POKE572,96
70 POKE11,34:POKE12,2:X=USR(X)
80 PRINT"      Potato Chips"
90 PRINT"      ------------"
100 FORK=1TO10:PRINT:NEXT
110 PRINT"Hit return to start game";:POKE54117,32
120 POKE11,0:POKE12,253:X=USR(X):POKE11,34:POKE12,2:X=USR(X)
121 K=611
123 IFPEEK(K)=255THEN130
124 X=PEEK(K)
125 POKE53413+K-611,X:K=K+1:GOTO123
130 READA,B:FORK=ATOB:READX:POKEK,X:NEXT
135 READA,B:FORK=ATOB:READX:POKEK,X:NEXT
140 POKE530,1:POKE57088,127:GL=54127:F=2
150 G1(1)=146:G1(2)=149:G1(3)=147
160 POKEGL,G1(F)
170 P=57088:T=9
190 FORZ=1TO10
```

```
200 IFPEEK(P)<>127THEN250
210 IFGL=54117ANDF=1THEN1000
220 POKEGL,32:F=F-1:IFF=0THENF=3:GL=GL-1
230 POKEGL,G1(F):GOTO1000
250 IFPEEK(P)<>191THEN300
260 IFGL=54139ANDF=3THEN1000
270 POKEGL,32:F=F+1:IFF=4THENF=1:GL=GL+1
280 POKEGL,G1(F):GOTO1000
300 IFPEEK(P)<>253THEN1000
305 BL=GL-64:Z=Z+3
310 IFPEEK(BL)=32THEN400
315 POKEBL+32,32
320 IFPEEK(BL)=235THENBL=BL-1
330 FORK=99TO32STEP-1:POKEBL,K:POKEBL+1,K:NEXTK:S=S+1:NG=NG-1:GOTO1000
400 POKEBL,G1(F):POKEBL+32,32:BL=BL-32:IFBL>53443THEN310
410 POKEBL+32,32
1000 NEXTZ:T=T+1:X=USR(X):TG=INT(T/10):IFNG>TGTHEN1050
1010 X=INT(RND(1)*21+1)+53477
1030 POKEX,234:POKEX+1,235:NG=NG+1
1050 POKE11,0:POKE12,28:X=USR(X)
1060 IFPEEK(610)=0THEN190
1070 FORK=255TO32STEP-1:POKEGL,K:NEXTK
1080 FORK=1TO35:PRINT:NEXT
1090 PRINT"Your final score was";S
1095 PRINT:PRINT:PRINT"The high score is";PEEK(605)
1096 PRINT:PRINT
1097 IFS>PEEK(605)THEN1150
1100 PRINT:PRINT:INPUT"try again";Y$
1110 IFLEFT$(Y$,1)="Y"THENRESTORE:CLEAR:GOTO10
1120 END
1150 PRINT"You have beaten it.":PRINT:PRINT
1155 POKE605,S
1160 INPUT"Your name";S$
1170 FORK=611TO611+LEN(S$)-1:POKEK,ASC(MID$(S$,K-610,1)):NEXTK
1171 POKEK,32:POKEK+1,32
1172 S$=STR$(S):FORG=2TOLEN(S$):POKEK+G,ASC(MID$(S$,G,1))
1173 NEXTG:POKEK+G,255
1180 PRINT:PRINT:GOTO1100
1500 DATA7168,7223
1510 DATA160,35,162,211,140,12,23,142,13,23,24,173,35,211,201,234,240
1520 DATA9,24,192,59,240,9,200,76,4,23,169,255,75,34,23,159,0
1530 DATA141,98,2,169,2,133,12,169,34,133,11,169,59,141,35,2
1540 DATA169,211,141,36,2,96
OK
```

THE INS AND OUTS OF SOFTWARE COPYRIGHT

Terry Terrance

Software copyright is something one does not see much written on in the hobbyist literature. This is probably due to two effects; first, software copyright is a difficult subject which has even legal experts perplexed and second, the major hobby-computer magazines are also software publishers and so they have an interest in keeping this information surpressed.

On the way to becomming a software publisher, I have found out something about copyright as it applies to computer programs; and, while I do not have a complete understanding on the subject, (and nobody--courts, lawyers or legislators really does) I will try to pass along what I have been able to learn.

Why should you copyright? Well for several reasons. First, it will force anyone seeking to reprint what you have written (in, say, OSI-tems) to get your permission first and give you full credit. Second, you never know when what you have written may have tremendous commercial potential; potential that you might not see at first. Copyrighting will protect your interests as well as those of your buyer. Lastly, securing protection under the copyright law costs you nothing and is easy to do.

Under the new (1978) copyright law, copyright is just that, a right, inherent in the act of creation; and is not something granted by the government. As such, you do not have to register or file anything to get copyright protection (although in some cases it is a good idea). All you have to do is abide by certain rules to protect the copyright that you already have as the original author.

The first event in the life of a 'work' is its completion date. This is the date the program is put into readable form (either machine readable or human readable) in a more or less completed version. This date will be the day on which your copyright protection starts. It will be asked for if you decide to file for a registration number. When you decide a program is complete is totally up to you. Declaring a work complete on a certain date does not preclude protection for revisions or additions that follow later; these come under the title of 'derivative works' for which you are protected by the original copyright.

Exactly how you prove that a program was completed on a certain date, without formally registering the copyright, is unclear to me. Possibly REM statements imbedded in the program giving completion dates and revision dates would be sufficient. A better way might be having a copy of the program noterized.

Now comes the day you want to 'publish' the program. 'Publish' has been interpeted to mean any distribution of the program. Even giving a single copy to a friend, or a review copy to a magazine or publisher constitutes publishing. If you do not abide by some simple

rules here, you will lose your copyright and your program will enter the 'public domain'. If you have so lost your copyright, it is recoverable, but not without expendature of effort and money to file a formal registration.

All you have to do when you are ready to 'publish' is to include a copyright notice on each and every copy of the program that you give out. The copyright notice has three parts; to quote the copyright office: "(1) the symbol © , or the word "Copyright," or the abbreviation "Copr."; (2) the year of publication; and (3) the name of the owner of copyright. For example "© 1978 Constance Porter." There is a further requirement, again to quote, "The notice is to be affixed to the copies in such a manner and location as to give reasonable notice of the claim of copyright." This last requirement may give some problems in a program. To be absolutely sure of copyright notice I would imbed the notice in REM statements so it would be visable in the program listing; include it in PRINT statements so it will show when the program is run; and write it on the casette or disk and any documentation that you send out with it.

If you do all of this, your copyright is fully protected and it has really cost you nothing. If you have not done this you may still be okay, because filing a formal registration of copyright, within five years, can recover your copyright. Also, there are some cases where a distribution can be made, without the notice, and still maintain copyright. One such instance, which will interest us, is sending out review copies. If you have sent out copies for review, and in your cover letter stated that they were such, and made provisions for their return, such as including a SSAE, you may well still have copyright.

Lets say you want to file an application for copyright registration. Remember, filing an application for registration gets you, in theory, no more protection; but it does make your copyright less assailable by fixing the date of creation and form of your work. Write to the "Register of Copyrights, Library of Congress, Washington, D.C. 20559." Ask for form "TX-Application for Copyright Registration for a Non-Dramatic Literary Work." Why this form? I don't know its the one my lawyer says to use. Follow the instructions, fill it out and include $10. (check or money order). That's all there is to it, and its cheap insurance for any program that you have serious intentions for.

A word about derivative works. Your original copyright, filed or not, protects any derivative works based on the original. This means revisions and upgrades are protected without having to file again. It also prohibits someone form changing your BASIC syntax to get your program to run on another system and then relesing it themselves.

In light of all of the above, why do magazine and software publishers insist that all submitted materials be without copyright? Well, for one thing, their claim that dealing with copyrighted works is too difficult absolutely does not hold water. Since, in the event of a copyright infringement suit, author, publisher, and seller are all liable to the same prosecution and penalties; it protects all parties to have a valid copyright assured. The best way to do this is to have the author secure a strong copyright form the beginning. And there are no additional legal hassles in transfering an author's copyright.

So why does 'Popular Creative kilo/Computing' want unprotected software? Well; I'm sure you can draw your own conclusions.

As programmers become more sophisticated and as reputeble software houses spring up, I think this situation will change. Publishers will probably demand a valid copyright in effect before they will handle software, for the protection of all concerned. Such publishers will probably also ask authors to assign their copyright to them. There are commercial reasons for this but also the publishers will be in a better position to persue copyright violators either by legal means or by other means (like cutting off dealers who pirate software).

I have painted a rosey picture of copyright, but there are thorns amoung the roses. The copyright laws are less than perfect, and even moreso when they are applied to computer programs. Judges, with no computer experience, have made some bad decisions recently. So even 'full protection under the law' can be less than total protection. Spurious claims can be made against your copyright and tracking down and prosecuting infringers of your copyright can be long, difficult and expensive.

In all though, I think copyrighting, since it can be easy and inexpensive, is something every programmer should consider any time he relinquishes a copy of a program for any reason.