

12 SHELDON ROAD
LONDON NW2 3AL

Dear Lynx User,

Bob Jones has kindly agreed to my starting up the Lynx Newsletter, of which this is the first issue. I will edit and produce a compilation of programs and snippets, and issue it FREE to any member who sends me a stamped addressed envelope. The crunch is that the Newsletter will depend entirely on free contributions from you, and on your SAEs. The more contributions and SAEs I receive, the more interesting and useful it can be.

I intend to produce the next Newsletter for the end of March, so please send me something as soon as possible - that utility you wrote last year, or that hint that would help our programming - from beginners' Basic to advanced Z80 machine code, anything Lynx except advertising will be considered - on paper, tape or disk:
PLEASE SEND IT IN TO ME, WITH YOUR SAE.

Looking forward to hearing from you,

Yours sincerely,

Chris Mathews

PS Please note that I will not accept any liability for contributors' errors or unloadable tapes etc, and that I reserve the usual editorial rights.

PPS Already expected for Newsletter No 2: musicality from Russell Davis, and fast block transfer into video from Phil Kups. Don't let them take over!

This code is similar to 'Fast Work' in Lynx User 2 but has been optimized to respect inks 1-7. PAPER and PROTECT are not respected, but only those colour blocks necessary for ink are written to - a different kind of PROTECT.

Start address A may be set to any suitable value since the code is relocated by the Basic, which may be NEW'd when it has run once.

```
100 LET A=&9E00
110 IF HIMEM>A THEN RESERVE A
120 FOR J=0 TO 154
130   READ I
140   POKE A+J,I
150 NEXT J
160 DATA &F5,&E5,&D5,&C5,&D9,&E5,&D5,&C5
170 DATA &CD,&CE,&00,&22,&C2,&62,&08,&F5
180 DATA &2A,&54,&62,&7D,&6C,&26,&00,&44
190 DATA &29,&29,&29,&29,&29,&0E,&FF,&0C
200 DATA &D6,&03,&30,&FB,&09,&EB,&3A,&5B
210 DATA &62,&CB,&47,&28,&0C,&F5,&D5,&2A
220 DATA &8E,&62,&3E,&03,&CD,&00,&00,&D1
230 DATA &F1,&E6,&06,&2A,&90,&62,&3C,&CD
240 DATA &00,&00,&F1,&08,&C1,&D1,&E1,&D9
250 DATA &C1,&D1,&E1,&F1,&C9,&19,&01,&20
260 DATA &00,&B7,&ED,&42,&D9,&06,&02,&2A
270 DATA &C2,&62,&C5,&46,&23,&4E,&23,&56
280 DATA &23,&5E,&23,&E5,&D5,&D9,&01,&20
290 DATA &00,&09,&E5,&09,&E5,&09,&54,&5D
300 DATA &09,&EB,&D9,&E1,&D1,&D9,&01,&7F
310 DATA &FF,&ED,&79,&C1,&08,&3E,&40,&D3
320 DATA &80,&D6,&20,&D3,&80,&70,&EB,&71
330 DATA &D9,&71,&EB,&70,&AF,&D3,&80,&01
340 DATA &7F,&FF,&ED,&79,&08,&E1,&C1,&10
350 DATA &C1,&D9,&C9
360 REM Set up calls within routine:
370 DPOKE A+53,A+77
380 DPOKE A+64,A+77
390 REM VIDEO vector is at &62B8:
400 DPOKE &62B9,A
410 WINDOW 0,96,0,250
420 DPOKE ALPHA,&0095
430 PROTECT BLACK
440 PAPER BLACK
450 CLS
460 REM Demo:
470 M$="Fast video, ink "
480 FOR J=1 TO 7
490   INK J
500   PRINT M$J
510 NEXT J
520 FOR J=1 TO 7
530   PAUSE 20000
540   INK J
550   PAPER 7-J
560   CLS
570   PRINT M$J
580 NEXT J
```

ASSORTED SORTS by Cloudwalker (Russel Davis)

```
100 CLS
110 INPUT "Number of items ";N
120 DIM I(N-1)
130 FOR L=0 TO N-1
140   INPUT "Next item ";n
150   LET I(L)=n
160 NEXT L
170 PRINT "Choose which sort:"
180 PRINT "      B = Bubble"
190 PRINT "      R = Ripple"
200 PRINT "      S = Selection"
210 PRINT "      H = Shell"
220 INPUT S$
230 LET S#=UPC$(S$)
240 IF S#="R" THEN GOTO 350
250 IF S#="S" THEN GOTO 450
260 IF S#="H" THEN GOTO 520
270 REM ***** BUBBLE SORT. *****
280 FOR L=1 TO N-1
290   FOR K=N-1 TO L STEP -1
300     IF I(K)<I(K-1) THEN SWAP I(K),I(K-1)
310   NEXT K
320 NEXT L
330 GOTO 650
340 REM ***** RIPPLE SORT. *****
350 FOR L=0 TO N-2
360   LET M=0
370   FOR K=0 TO N-L-2
380     IF I(K)>I(K+1) THEN LET M=1
390     IF I(K)>I(K+1) THEN SWAP I(K),I(K+1)
400   NEXT K
410   IF M=0 THEN LET L=N
420 NEXT L
430 GOTO 650
440 REM ***** SELECTION SORT. *****
450 FOR L=0 TO N-2
460   FOR O=L+1 TO N-1
470     IF I(O)<I(L) THEN SWAP I(O),I(L)
480   NEXT O
490 NEXT L
500 GOTO 650
510 REM ***** SHELL SORT. *****
520 IF N=1 THEN GOTO 650
530 LET G=N
540 LET G=INT(G/2)
550 FOR J=G TO N-1
560   LET I=J-G,T=I(J)
570   IF I<0 THEN GOTO 610
580   IF T>=I(I) THEN GOTO 610
590   LET I(I+G)=I(I),I=I-G
600   GOTO 570
610   LET I(I+G)=T
620 NEXT J
630 IF G>1 THEN GOTO 540
640 REM *****
650 FOR L=0 TO N-1
660   PRINT L+1,I(L)
670 NEXT L
```

INSTRING by Chris Mathews

The INSTRING or INSTR# function, where one string is matched into another, is a very useful searching function. For example, to find the first instance of the sub-string V# within string X#,

```
I=INSTR#(V#,X#)
```

I then holds the character index into X# at which V# starts; if V# was not found in X# then I=0.

The following FOR loop in Basic does this fairly efficiently (example included):

```
100 LET X#="INTELLIGENT"
110 LET V#="GEN"
120 LET X=LEN(X#),V=LEN(V#),D=X-V+1,I=0
130 FOR J=1 TO D
140   IF V#=MID$(X#,J,V) THEN LET I=J,J=D
150 NEXT J
160 PRINT I
```

Note that if V# is longer than X#, or if X# is the null string "", then I=0; if V# is the null string then I=1. Also, in the code given here only the first instance of the substring is searched for - if you want to find all instances of the substring then you will have to search repeatedly, using RIGHT# each time a match is found so that only the remainder of the string is searched next time.

USING THE WINDOW COMMAND by Chris Mathews

Using the Lynx's WINDOW command precisely can be important for neat presentation. Unfortunately the way it works has not been fully documented.

WINDOW 45,50,90,100 defines the window 90,101,90,99 in pixel coordinates. Generally, WINDOW a,b,c,d defines the window a%2,b%2+1,c,d-1 in pixels. I call the value (b-a) the declared width of the window, and the value (d-c) its declared height.

Bearing in mind that the size of a Lynx character is 6 by 10 pixels, or 3 by 10 WINDOW units, a text window should have a declared width which is a multiple of 3 and a declared height a multiple of 10. For example, WINDOW 3,123,5,25 allows exactly 2 lines of text, but WINDOW 3,123,5,26 allows 3 lines, even though all of the 3rd line except its top pixel lies below the window. WINDOW 3,121,5,245 or WINDOW 3,122,5,245 causes characters to be split around the right end of the line.

I hope this makes WINDOWS clearer!

The Lynx Printer Manual describes how to set or reset bits in the printer control block &61BC-&61BF in order to modify which control bytes are sent to the printer. However there are quite a few printers around for which this is not enough to obtain sensible printing, especially if the printer does not use standard ASCII at all. The routine below works in addition to the Lynx control block to translate each character value output by the Lynx to the corresponding value required by the printer. It works for both parallel and serial printing (PPRINT and SPRINT), and will also work for terminal communication using the printer ports.

In the example data, the space character, which has the ASCII value 32 on the Lynx, has the value 64 on the printer. The Basic, which relocates the machine code and translation table to start at address C, should be deleted once it has run, so the last line is a NEW. Take care not to overwrite the code and table once you have run the Basic!

```
100 LET C=&9E00
110 IF HIMEM>C THEN  RESERVE C
120 FOR J=0 TO 14
130   READ I
140   POKE C+J,I
150 NEXT J
160 LET D=C+J
170 DPOKE C+3,D
180 FOR J=0 TO 127
190   READ I
200   POKE D+J,I
210 NEXT J
220 REM Machine code data:
230 DATA &E5,&C5,&21,&0F,&9E,&4F,&06,&00
240 DATA &09,&7E,&C1,&E1,&C3,&CB,&48
250 REM
260 REM Example translation data:
270 DATA 0,1,2,3,4,5,6,7
280 DATA 8,9,10,11,12,13,14,15
290 DATA 16,17,18,19,20,21,22,23
300 DATA 24,25,26,27,28,29,30,31
310 DATA 64,33,34,35,36,37,38,39
320 DATA 40,41,42,43,44,45,46,47
330 DATA 48,49,50,51,52,53,54,55
340 DATA 56,57,58,59,60,61,62,63
350 DATA 64,65,66,67,68,69,70,71
360 DATA 72,73,74,75,76,77,78,79
370 DATA 80,81,82,83,84,85,86,87
380 DATA 88,89,90,91,92,93,94,95
390 DATA 96,97,98,99,100,101,102,103
400 DATA 104,105,106,107,108,109,110,111
410 DATA 112,113,114,115,116,117,118,119
420 DATA 120,121,122,123,124,125,126,127
430 REM
440 REM Change address of print driver:
450 LET X=DPEEK(&6202)
460 DPOKE C+13,X
470 DPOKE &6202,C
480 NEW
```

2-DIMENSIONAL ARRAYS by Chris Mathews

The way the Lynx handles arrays has confounded several members wanting to use them with 2 or more dimensions.

In most languages you would dimension a 2-D array A of 3 rows by 4 columns with `DIM A(3,4)`. Then, to access the element in the 2nd row and 3rd column you would use `A(2,3)`.

On the Lynx you use `DIM A(3*4)`, which gives you a 1-dimensional array of 13 elements, indexed 0-12 (you get the zeroth element 'for free'). If you want to treat this array as 2-D, you have to calculate the start of the row you want and then add on the offset in that row. So instead of `A(2,3)` you use `A(1*4+3)`, because element (2,3) is 1 whole row along plus 3 elements.

	1	2	3	4
1				
2			X	
3				

This generalises to `A(i,j)` being accessed as `A((i-1)*n+j)`, where n is the number of columns in the DIM statement.

If you want to use the zeroth element, ie to index the array from 0, you can use the same `DIM A(3*4)`, and then to access `A(1,2)` you use `A(1*4+2)`; which is `A(i,j)` accessed as `A(i*n+j)`. This is perhaps easier to remember - it's the same as the last explanation in Lynx User 1.

	0	1	2	3
0				
1			X	
2				

Notice that both the above methods are essentially the same: only the numbers used to label the array elements are different. The 1st explanation in Lynx User 1 is a different method, which looks at each row of the array 'from right to left' using subtractions, and I do not recommend it. The important thing however is that when you access Lynx arrays dimensionally, stick to the same method throughout your program!