Now that you know the commands we'll be dealing with, it's time to write some programs. I'll guide you through the first few, but you'll get less and less help as we move forward. **IMPORTANT: The sample routines presented here are for the 89. Numbers related to the screen need to be changed to run on a 92. Otherwise, the code is the same.** A. Moving text

We'll start small. Suppose we had a block of text that we wanted to scroll from right to left across the screen. We could do this in the IO screen, but I'm going to use the graph screen, because you are much more likely to be using it. We'll also use PXLTEXT. You could use PTTEXT, but then you would have to set the window and that's a nuisance. Let's try this:

```
TextScr()
Prgm
For A,158,0,-1
PxlText "This text is scrolling",30,A
EndFor
Endprgm
```

Now that has a few obvious flaws. First of all, if there was something on the screen, the axes were on, or something was set to graph, they're in our way. Secondly, after the word scrolling clears the right edge of the screen, it leaves a big black box behind. By the way, if you're using a 92, you won't use 158 in the FOR statement. Instead, use whatever the last column of pixels is. (I don't have a 92, sorry.) Let's fix the problems

```
TextScr()
Prgm
Clrdraw
FnOff
Setgraph "Axes","Off"
For A,158,0,-1
PxlText "This text is scrolling ",30,A
EndFor
Endprgm
```

That did it, but our text has to stop at the left edge of the screen. Suppose we want it to keep going off the left side. Try it.

If you tried making the FOR go into negatives, it didn't work. Why? PXL values cannot be negative. Actually, we can't do this using just what we've learned so far, because using PXLTEXT we can't make it display just the right side of a character with the left side offscreen. We could set up a black box that the characters go into on the left of the screen, but that doesn't look very good. There is a solution: PTTEXT. As you should have noticed by now, there are some commands which we will almost never use except in a few instances. This is one of them. Of course, since we're using PTTEXT, we have to define the window. Here's how you do it.

```
TextScr()
Prgm
Clrdraw
ZoomDec
FnOff
Setgraph "Axes","Off"
For A,8,-22,-.1
PtText "This text is scrolling ",A,0
EndFor
Endprgm
```

How cool. There is a much more versatile solution to this problem which also lets us scroll graphics. I will explain this in the graphics section.

Let's try another problem with the scrolling text. Try to make the text bounce. Go ahead, try it. You probably got something like this.

```
TextScr()
Prgm
Clrdraw
FnOff
Setgraph "Axes","Off"
Loop
For A,158,0,-1
PxlText "This text is scrolling ",30,A
EndFor
For A,0,158
PxlText "This text is scrolling ",30,A
EndFor
Endloop
Endprgm
```

That will run indefinitely and we have to stop it with on. See if you can make it stop if you press any key.

```
TextScr()
Prgm
Clrdraw
FnOff

Setgraph "Axes","Off"
Loop
For A,158,0,-1
PxlText "This text is scrolling ",30,A
If getkey<>0
goto out
EndFor
For A,0,158
PxlText "This text is scrolling ",30,A
if getkey<>0
goto out
EndFor
Endloop
lbl out
Endprgm
```

We could also set this up similarly with WHILE loops in place of the FOR, but that is more confusing and won't run any faster anyway. But the program doesn't put things back as they should be. Do you remember the syntax?

```
TextScr()
Prgm
Clrdraw
FnOff
Setgraph "Axes","Off"
Loop
For A,158,0,-1
PxlText "This text is scrolling ",30,A
If getkey<>0
goto out
EndFor
For A,0,158
PxlText "This text is scrolling ",30,A
if getkey<>0
goto out
EndFor
Endloop
lbl out
Clrdraw
Setgraph "Axes","On"
DispHome
FnOn
Endprgm
```

Just undo what we did in the beginning. Easy. Who wants graphics?

B. Graphics

Okay, here's a routine that I especially like (so much so that I used as the opening screen to Risk 89). See if you can figure out what it does and how it works.

```
Graphic()
Prgm
Clrdraw
FnOff
Setgraph "Axes","Off"
RplcPic ts
Loop
For B,1,4
For E,158,1,-1
StoPic d,0,1,E,77
RplcPic d,0,0
RplcPic #({"O1","TS","O2","TS"}[B]),0,E
If Getkey()=264
Goto Out
Endfor
Endfor
Endloop
lbl Out
Clrdraw
Setgraph "Axes","On"
DispHome
FnOn
Endprgm
```

Where TS, O1, and O2 are full screen pictures. You were right if you said that it scrolled a series of pictures. Risk goes a step further and scrolls text with the pictures, but that gets very confusing, so don't worry about it if you can't figure out how to do that. Let's go through and figure out what each line does.

The first five lines are the same initialization code we've been using. The next line puts TS on the screen. (This isn't necessary. I just wanted it to start with something on screen.) Now we enter a LOOP and two FOR loops. The first FOR puts the number of the picture that is coming on screen in B. The second FOR puts the pixel column that the picture is at. We then copy everything on the screen up to the incoming picture (with the exception of pixel column 0) into D. When we RPLCPIC D in the column 0, we have effectively slid what remains of the old picture. We then place the incoming picture (using indirection) one column to the left of where it was. Finally we check if esc was pressed and GOTO out if it was. Otherwise we finish off the FOR loops and LOOP. If esc was pressed, we reset the graph screen and go back to the home screen. That wasn't so bad, was it? Make sure you understood this, because it's only going to get worse. See if you can modify this to make the pictures scroll up instead of left. Could this program be used to make pictures scroll down or right? A picture could scroll off screen to the right of down, but a new one could not scroll in from the left or top. For this reason, only use this program to scroll up or left (which are usually more useful anyway).

Aren't graphics fun? All right, let's try something different. (You never want to see something scroll again do you?) Let's try something easy. I'm just going to give you this one. Because of the way PXLCRCL works, it can be used to create one of the coolest effects in BASIC. Unfortunately, the effect is overused, so don't waste it. See if you can figure out what this does, then run it.

```
Circ()
Prgm
Clrdraw
FnOff
Setgraph "Axes","Off"
For A,0,100
PxlCrcl 38,79,A
Endfor
For A,100,0,-1
PxlCrcl 38,79,A,0
Endfor
Clrdraw
Setgraph "Axes","On"
DispHome
FnOn
Endprgm
```

Cool, huh? I've seen it used for everything from magic powers and explosions to a screensaver. Try using it without clearing the screen first and using PXLCRCL 38,79,A,-1 for both of the PXLCRCL's above. Not bad, eh? This will conclude our present discussion on graphics, although I will do some more complicated problems later.

C. Input

Those of you who programmed on other calculators before are probably thinking, "Wow, I couldn't do some of this stuff on my old calculator." It's faster,

too. BASIC doesn't seem all that BASIC anymore, does it? In fact, there are quite a few asm games on other calculators that can actually be ported to BASIC on the 89 or 92. But I don't recommend it because it takes a long time! Complex programs do take a while though, and when you finally finish, you're stuck troubleshooting. So before I get into some more intricate routines, I think now would be a good time to see how you're doing in reading code. In the following routine, I notice that if I leave the request box blank, enter an invalid variable, or press esc, the program errors and quits. Fix it.

```
Oops()
Prgm
Dialog
Title "You're getting good"
Text "Can you tell me what's wrong?"
Request "Pick a number",A
Enddlog
expr(A)-->A
ClrIO
For B,1,A
Disp A
Endfor
Pause
ClrIO
DispHome
Endprgm
```

You should have fixed it like this:

```
Oops()
Prgm
lbl A
Dialog
Title "You're getting good"
Text "Can you tell me what's wrong?"
Request "Pick a number",A
Enddlog
If Ok=0
Goto out
Try
expr(A)-->A
Else
Dialog
Title "Stupid"
Text "You know I can't do that"
Enddlog
goto A
EndTry
ClrIO
For B,1,A
Disp A
Endfor
Pause
ClrIO
lbl Out
DispHome
Endprgm
```

Of course, it doesn't have to jump to the end because I press esc (OK=0), but it most cases, your user is trying to tell you something. If you understood that, let's try some more input routines. By the way, did you figure out what that last program did? I'm not telling you this time.

Input seems like a straightforward task, but it can become a nightmare. The programmer has to be prepared for that one idiot who doesn't read the directions, enters the wrong value, and screws up everything. You have to be prepared to make him fix it. (Feel free to insult him, too.) The previous problem is a good example.

Suppose we prompted the user for a string with six or fewer characters. How do we make sure what he entered was valid? Like this:

```
Under6()
Prgm
Lbl NP
Dialog
Title "Name"
Request "Your Name",N
Text "Six or fewer letters"
Enddlog
```

```
If dim(N)>6
Then
Dialog
Title "PAY ATTENTION"
Text "SIX OR FEWER LETTERS"
Enddlog
Goto NP
EndIf
Endprgm
```

Remember what DIM does. For a string, it gives us the number of characters. This example is pretty straightforward so I'm not going to explain it any further. I think you get the idea. Let's move on to something more interesting: linking.

D. Linking

If you plan on writing a multiplayer game, you need to be able to link. Unfortunately, the link cables do not transmit very rapidly (actually it isn't the port's fault, it's the processor's, but that's irrelevant), but they usually work quickly enough. We will use SENDCALC and GETCALC, but if you are communicating between an 89 and 92, you would use SENDCHAT and GETCHAT. First of all, GETCALC will sit and wait...and wait...and wait, until it receives something, so be sure that there will always be something to receive (even if it is just a signal that nothing is happening) whenever the calculator reaches that point. Secondly, SENDCALC will not wait forever, it will error and quit your program. Isn't linking fun? Fortunately there is a solution. Come on, I know you know what it is. The TRY block (wow, that really is useful). I used this routine in Risk instead of SENDCALC.

```
Linkup(S)
Prgm
Local S
Lbl A
Try
SendCalc S
Else
Dialog
Title "Link Error!"
Text "The link cable is not properly"
Text "inserted.  Please fix it"
Enddlog
Goto A
EndTry
EndPrgm
```

So typing Linkup(A) would attempt to send A, and if the other calculator isn't ready, Linkup gets mad. You can still receive with GetCalc. By the way, just because a value is stored in A on one calculator doesn't mean the other calculator can't receive it as B. If one calculator runs SENDCALC(A) while the other runs GETCALC(B), A on the first will be stored as B in the second, so be careful. That's about all there is to say about linking.

E. Lists, matrices, and data sets

I have said very little of lists, matrices, and data sets. This is not to suggest that they are not important. They are one of your greatest tools. Lets try some routines using lists and matrices (data sets are just sets of lists).

Lets try writing a routine that inputs a list in the form *{text, x, y}* and writes the test at *x,y*. I'm using a list instead of multiple inputs for a reason.

```
Write(L)
Func
local l
Try
PtText L[1],L[2],L[3]
Else
Dialog
Title "Invalid"
Text "Invalid input"
Enddlog
Endtry
EndFunc
```

That was a useless routine because we could have just used PTTEXT, but it is still useful for learning. L[N] calls the Nth element of L. This is very important. Let's try a variation.

```
Write(M)
Func
local l,a
For A,1,dim(m)[1]
Try
PtText M[A,1],M[A,2],M[A,3]
Else
Dialog
Title "Invalid"
Text "Invalid input"
Enddlog
Endtry
EndFor
EndFunc
```

This inputs a matrix in the form [[text,x,y][text,x,y]...]. Dim(M) returns a list in the form {numberofrows,numberofcols}, so Dim(m)[1] returns the number of rows. M[R,C] returns the element in the Rth row and Cth column, so M[A,1] returns the Ath row and 1st column (one of the texts). If you understood these two simple examples, you're all set with lists and matrices. They aren't complicated, just useful. If you want to parse lists or matrices, you can use the same commands we've been using for strings (i.e.. Left, RIGHT, MID, Augment, etc.). I'm not going to get into the more complicated task of parsing. If you want more information, e-mail me KeysDezes@aol.com) and I'll help you on a case-by-case basis.

F. Gaming Graphics

I know this is why you downloaded this file. You want to write games in BASIC. You want to show the world that Mario will run in BASIC. There's a problem there. It can't. No matter how hard you try. But that's okay. Plenty of complex programs can run in BASIC on the 89 and 92. Our website (www.crosswinds.net/~proganon and soon to be pa.ticalc.org I hope) has a couple of interesting examples of gaming on the 89 in BASIC. Two of the more interesting are Dragon Warrior (By Robert Vollrath and Matt Scanlan) and May Day (By Grant Elliott - that's me). Neither is finished, but we can still find some examples of good graphics code. This is from Dragon Warrior:

```
DragDraw()
Prgm
Clrdraw
FnOff
Setgraph "Axes","Off"For  x,0,14
For  y,0,7
RplcPic  #("pic"&string(board[1+y,1+x])),12+8*y,39+8*x
EndFor
EndFor
Endprgm
```

They're using indirection to call pictures (BG1, BG8, etc.) out of a matrix (Board) of numbers (1, 8, etc.). This bring pictures into view one column at a time. Try this one from May Day.

```
MayDraw()
Prgm
Clrdraw
FnOff
Setgraph "Axes","Off"
For A,1,64
{3,3,4,4,4,3,2,2,2,2,3,4,5,5,5,5,5,4,3,2,1,1,1,1,1,2,3,4,5,6,6,6,6,6,6,6,5,4,3,2,1,0,0,0,0,0,0,0,1,2,3,4,5,6,7,7,7
,7,7,7,7,7}[a]-->d
{3,4,4,3,2,2,2,3,4,5,5,5,5,4,3,2,1,1,1,1,1,2,3,4,5,6,6,6,6,6,6,5,4,3,2,1,0,0,0,0,0,0,0,1,2,3,4,5,6,7,7,7,7,7,7,7,6
,5,4,3,2,1,0}[a]-->e
Try
RplcPic #("P"&mid(map[y+e],x+d,1)),e*8,d*8+46
Else
RplcPic P0,e*8,d*8+46
Endtry
EndFor
Endprgm
```

I'm not going to get into what line 9 does, because it involves explaining the system I'm using to encode the levels. Just know that this loads a picture (P1, P2, etc.) based on what it reads off a list (Map) of strings. It loads P0 (a black box) if part of our view is off the edge of the map. X and Y are the present coordinates for our field of vision. Now, what's with those lists at the top? Well, they load appropriate coordinates in a spiral shape. This program loads the screen radiating out from the center. In my opinion, this is much more entertaining than watching columns of sprites load. The strings are smaller than the matrices and slightly faster in execution, but also require more code to get the correct information. So, it's a tradeoff. Robert and Matt use matrices so they can rapidly edit their levels.

Some people talk about grayscale in BASIC. To be honest, it is possible, but not great. You use CYCLEPIC *string, numberorpics, wait[, numberofcycles][, direction]*, but you can't do anything while it's cycling. I prefer writing my own routine:

```
PicCyc()
Prgm
Loop
Rplcpic Pic1
...
Rplcpic Pic2
Endloop
Endprgm
```

It works remarkably well. Of course, you could cycle more than two pictures for higher degrees of grayscale, but I don't recommend it. It simply looks bad, especially if there's anything executing between pictures. And, of course, part of the code should include a way to get out of the loop. You could use this to wait for enter (assuming the loop runs quickly, which it should to make convincing gray).

```
PicCyc()
Prgm
While getkey<>13
Rplcpic Pic1
...
Rplcpic Pic2
Endwhile
Endprgm
```

Unfortunately, GETKEY is not fast (even when returning 0) and this will slow down the loop, lowering the quality. However, it will work. I used it for the maps in Risk, and they look fairly convincing. **Important: the quality of the gray in this program relies heavily on battery level.**

G. Problems

There isn't too much more I can teach you. As with all programming languages, BASIC cannot be entirely taught: it must be learned. I've taught you many tricks that I use on a day-to-day basis. (I cannot stress the importance of indirection enough.) The time has come for you to go off on your own and experiment. It's very difficult to crash your calculator in BASIC (although it can be done), so feel free to try something strange. I recommend backing things up frequently, just in case. I wish you much luck in your future endeavors. If you ever need help, try posting on the TI-BASIC mailing list (TI-BASIC@lists.ticalc.org) or send one of us a question at KeysDezes@aol.com (me), UFGator584@aol.com (Robert), and HSolo2@aol.com (Matt). If it's possible, we'll tell you how. For now, why don't you try writing some of these programs. (* represents a more difficult routine.) Remember, all complex programs are built from simpler code. I bid you farewell, and hope you've had as much fun reading this as I had writing it.

Make a program to check if an expression is an element of a list.
Make a program that moves small sprite around the screen, controlled by the arrow keys.
Make a program to parse an equation without using PART. (Hint: Convert the equation into a string)
Make a program to create another program based on parameters set by the user.* (Hint: You need DEFINE and EXPR)
Make a program to shuffle a deck of cards.
Make a program to play a simple card game.*
Make a program to play a simple bowling game (graphical) with scoring.*

If you can make all of these simpler programs, you won't have any trouble making complex programs (like games). I hope to see your programs on ticalc.org very soon.

I do ask that you don't steal any of the routines from our programs without giving us credit. A simple, "Thank you Programmers Anonymous for the nice picture scroller," is plenty. Keep in mind that we, like you, are here to get our names out by releasing some of the most complex BASIC programs out there. This guide is provided for your learning use and some of the routines have not even been released yet. Thank you and goodbye.

Grant Elliott
Programmers Anonymous

<<Previous - Contents