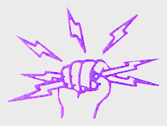
Foenix Rising

A newsletter for developers, users, and enthusiasts of Foenix Retro Systems products

(DRAT !)

This issue chronicles the lessons learned and hardship suffered during my attempt to port a 1979-1980 Commodore PET two-player game to the Foenix F256 platform.

Many have characterized the F256K as a spiritual successor to the Commodore 64 and the similarities are easy to spot (6502 CPU, built in BASIC, SID audio ICs, and familiar and easy to use Sprites, Bitmaps, and redefined characters). Of course, the Commodore PET was absent many of these features and used a somewhat vanilla version of Microsoft BASIC. How difficult could it be to port 195 lines of BASIC code to SuperBASIC and add a few features? Hold onto your hat while I eat mine.



November
2024
Issue F18
(rev b.)

"Angry Planet" - 45 years after *Ouranos*, Weather War lands on Foenix, with a bang

"Porting this will be easy" …

Famous words. I recall playing Ouranos as a kid. It was one of the few two-player *graphically* oriented titles available for the Commodore PET and it was glorious. Distributed as part of "Cursor" volume 21, it quickly made its way into the hands of the first generation of computer kids. Cassette tape distribution was the medium, and thanks to Mr. Caggiano's "Computer Math" class, students in my school had at least one tape; it was a requirement for in-class assignments. Home computers were not yet common, but that was about to change.

Commodore's 1977 blue-faced PET was RAM constrained and shipped with 4k or 8k of memory. The $799 price tag included a 9" monitor, keyboard, and a datasette; seemingly inexpensive as compared to the Apple 1 ($666), but still too expensive for most homes.

For a time, 3rd parties and OEMs provided RAM upgrade paths for Apples, Commodores, and TRS-80s, but they were expensive and not every family had the wherewithal to install ICs into system boards. Five short years later, Commodore threw down the gauntlet and released a maxed out 8-bit memory map with 64K of RAM and the ability to bank switch ROM and RAM.

Commodore invested early and reused aspects of the PET design across nearly every system*; a set of features so beloved, they found their way into the Commander X16.

Vintage home computer manufacturers had many features in common, but none included as good a screen editor, a character set so broad, or a way to embed characters and cursor motion in strings, as seamlessly as Commodore.

In this article, we take an early PET title and attempt to port it to SuperBASIC on the Foenix F256 platform. We begin by acknowledging the **three** aspects of Commodore's character-based BASIC environment greatness, and then lay out **seven** considerations (show stoppers and work-arounds); some obvious, some surprising, but all were necessary to get a version of Ouranos to run on the Foenix F256.

* PET 2001 and 40xx, CBM 80xx, VIC-20, CBM-II, C64, SuperPET, C16, Plus/4, C128, B128, C65 (not released), and more…

The Commodore character-based ecosystem

1. The Full Screen Editor - Life was tough in the late '70s. The world was transitioning from serial connected ASCII terminals and SBCs to integrated computers with dedicated displays, some with graphic characters bound to keys on custom keyboards.

   Those familiar with the 1976 Apple '1' know that a portion of the system board was dedicated to the 'terminal' section which managed input/output. The Apple was primitive, and could only type forward or 'rubout' a character or full line; outside of the obvious, this was not much more than reimagined paper output.

   Commodore unveiled its original PET six months after the Apple began selling, and the PET screen editor empowered cursor movement in four directions, had upper and lower case text, the ability to clear the screen, insert and delete characters, enable reverse field text, and could move the cursor to the *home* position (the upper left of the screen). This simple feature was critically important (item 'b.' below).

   The screen editor also established behavior standards that had not been considered previously. Moving the cursor to a horizontal edge of the screen would wrap to the next or previous line. Moving below the 25th line, advanced (scrolling the text upward), and moving beyond the top line was not permitted.

   The editor also included a mechanism to insert a screen line, used during editing of BASIC program text to allow up to 80 characters per line number.

   Finally, there was a 'quoted' input mode which allowed embedding of cursor commands into strings that could include printable characters, side-by-side. This was handy for character animation, and when combined with the ability to home the cursor to an x/y of 0/0, promoted single-screen apps; programs that dynamically updated portions of the display without scrolling or the need to redraw the entire screen.

   Commodore's screen editor was not just good for editing BASIC commands (interactively or with line numbers), it was also good for immediate mode math, debugging (print the current value of a variable or to perform a test) or marking/prototyping text layout. And thanks to the `CHROUT` kernal routine, all of these controls and features could be leveraged from assembly language as easily as from BASIC.

In summary, the first of these three capabilities surpassed efforts by Apple, ATARI, Texas Instruments, Timex/Sinclair, TRS-80, and scores of other home computer manufactures.

Of course, kids from 8 to 80 really just wanted to play games, so was this really a big deal? Not really. But for the programmer, it made more things possible with less work and because the functionality was common across systems and embedded between the Kernal and BASIC ROM, it was relatively fast.

2. Microsoft BASIC - Simple? (yes) Powerful? (eh) Ubiquitous? (sort of). BASIC was the 'OS' on nearly every system produced from 1978 to 1984 and most equipment manufacturers put their own spin on the language, adding functions and operators to either enhance their product to take advantage of graphics, sound, and text output features of their offering.

   The story behind Commodore CEO Jack Tramiel's licensing of BASIC from Microsoft is legendary; it was an early example of Jack's shrewdness and a close call that almost pushed Microsoft to insolvency.

   BASIC was a product of Dartmouth University's math department in the mid '60s and provided students with a simple, line-number oriented command based language that could be used to solve math problems or create simple input guided programs. Ultimately, output was produced and text, summarizing computed values was displayed, courtesy of the immortal PRINT command.

   BASIC's origin story began with mathematics department personnel from Princeton University, but there is alot more to the story. For a great watch, see this 38 minute clip. It's part of my personal 'top 5' vintage compute videos and might be in yours too!

   In the mid '70s, Microsoft BASIC version 1 was released for the then, new, Altair 8800 system. It began with Dartmouth's 25 commands and was the start of expansion and evolution of the language.

   OEMs added their own spin on matters such as ATARI and Apple adding graphics commands and Commodore building a tape, disk, and print I/O stack. But the real innovation in the PET version led to the joining of their screen editor and tokenizing code, which adeptly managed whitespace (or the absence of it) in Commodore BASIC. See item 'd.' below for the troubles that plagued my efforts. And if you'd like more on how something seemingly simple can grow complex (and a good laugh), watch Robin (*8-Bit Show and Tell*) struggle with Applesoft BASIC as he attempts to key in a straightforward (on the face of it) BASIC program. It's a tragic slog :)

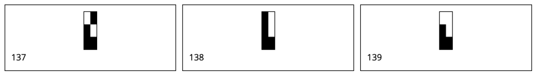
```
MEMSIZ? 8192
WANT SIN-COS-ATN? Y
3712 BYTES FREE

8080 BASIC VR 1.0

READY
█
```

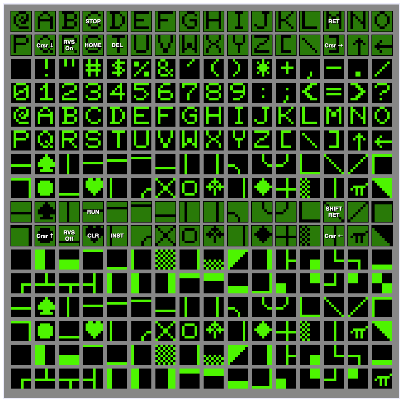
Init of Microsoft BASIC 1.0 from a MITS 8800 (Intel 8080 based)

3. Graphic character design, entry, and output - Every vintage platform has strong suits and questionable design decisions, but one of the more peculiar early entrants was the TRS-80 Model 1, part of the 1977 trinity. It used 'printer grade', 5 x 8 pixel characters inside of a 6 x 12 pixel envelope yielding a 64 x 16 character screen. One one hand, it felt more business-like than 22, 32, or even 40 characters per line (c.), but on the other, 16 lines with so much whitespace between was bizarre, it felt double spaced. The first TRS-80 also spent 64 characters on oddly oriented blocks; here are a few:

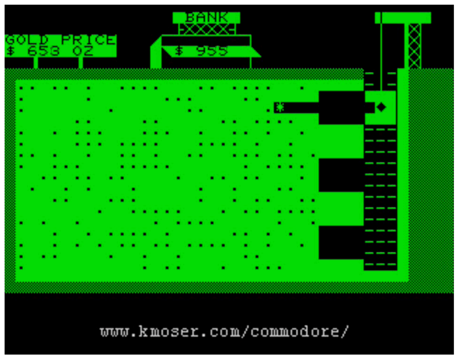
   

   The Apple II (and II+), on the other hand, had upper case only, 7 x 8 pixel characters with no graphic chars whatsoever. They did, however, include a flash attribute in addition to reverse field.

   Commodore overachieved in this area. They created a masterful set covering all of the bases, and dared to print the character set on keycaps, something that they had to be mindful of as keyboards evolved.

   As easily typed as they were, the glyphs also rendered neatly within quoted PRINT statements (as opposed to being coded behind opaque CHR$ functions). They were literally as easy to type and edit as standard alpha-numerics, and this meant games and other programs would quickly appear, starting with conversion from DEC and Unix systems, to be followed by a new generation of original titles. For a look at a few dozen originals, many leveraging PETSCII graphics heavily, click here. This site is owned by Kim Moser and links to a series of YouTube videos.

   Commodore PET 'Miner!'

Intro to the original "Weather War" aka Ouranos

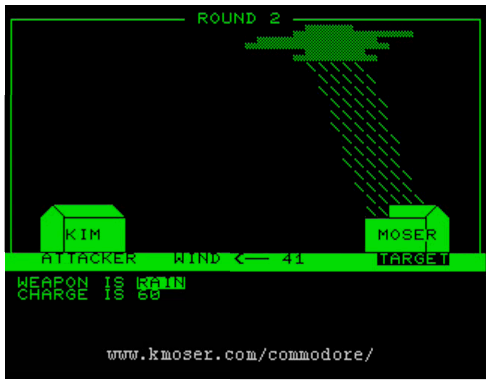
Written by Kathy Rigby for Cursor #21, Ouranos pits two players against each other, leveraging a familiar theme: crush your competitor, or in this case, their home.

Ouranos is 95% PET BASIC, except for a small portion of 6502 code which is loaded into the cassette buffer and called when lightning strikes, to flash the screen. Character graphics are used tastefully; you'll know them when you see them (the lightning strike 'weapon' is particularly well done). Audio is vintage PET, drawing upon the 6522 IC for a drone frequency. Most PETs had a built in speaker, but the older/original variety were often equipped with a DIY circuit and a speaker.

After identification of each player by name, the screen is rendered. Truncated reverse field names adorn the side of simple Monopoly Hotel shaped houses, built from 28 characters. Players take turns formulating strikes on their opponent with each player turn starting with a new wind direction and speed to be accounted for.

Weapon choices include 'hail', 'rain', 'tornado', and 'lightning', and each inflicts differing levels of damage based on wind strength and 'charge'. After selecting a weapon, the attacking player chooses a charge value, which is limited from -150 to 150.

After each player takes their turn in a 'round', the status line is updated and the game progresses. This continues until one player destroys the other player's house or your mom calls you for dinner and you quit the game. At this point, the house percentage remaining is calculated and a winner is declared.



At random points between turns, an "act of nature" can strike, inflicting damage on either player, each labeled "TARGET". The completeness of the game, and competitive nature (similar to *Battleship*), make it an enjoyable excursion.

"Angry Planet" for the Foenix F256

I started thinking about an F256 version for the October 2024 Game Jam. It seemed right-sized for a three day hackathon style effort and I planned on enhancing it slightly for release by the deadline. (I am fond of this corner of retro and history, and I wanted to put it in the hands of more people).

Ideally, I would meet this objective without opening up pandoras box, but it didn't work out that way. I was reminded of family commitments during 2 of the 3 days of the jam, so had to bow out. But I vowed to complete the job and having made some progress, I'm here to talk about it, and more importantly, talk about the challenges faced working on the port between BASIC versions.

Enhancements and Foenix features leveraged

From a gameplay perspective, the only new feature involves the addition of two new "acts"; one that affects permanent change to the house layout on the landscape, and a second which starts a time-based element that creates a 'rising tide', thus throwing an unexpected dynamic into the top-down strategy of striking your opponent. But there are plenty of changes within.

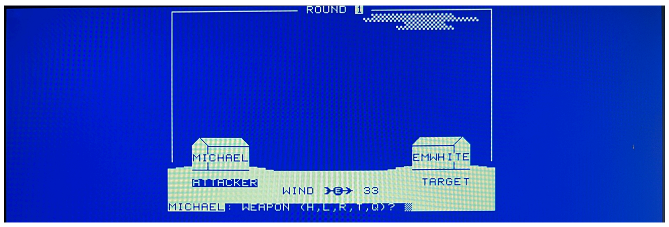
If you list the programs side-by-side, it will appear that the codebase has been rewritten from the ground up, and much of it has been. Many of the PRINT statements had to be split across multiple lines, primarily because of the way SuperBASIC deals with control characters and string literals (graphic chars, specifically). Elsewhere, spaces had to be inserted between arguments, values, and variables to prevent errors from being raised. Finally, logic/decision statements had to be refactored, and the only way to do this was to convert to modern IF / THEN / ELSE blocks, as we will discuss below.

From a gameplay perspective, the flow and objective of the game, and the math and formulas are all true to the original, though var types had to be altered (e.) and typecast (aka, forcing a float to an integer with the INT() function and similar actions).

Those with F256 Jr. platforms, soldering irons, and the desire to live dangerously, may wish to wire up CB2 of the WDC65C22, and using the PET schematic, relive 1977. Otherwise, PSG sound is leveraged and modeled to be low budget, just like the original.

Redefined characters are used to more closely match some of the PET graphic characters and to create reverse field video. You will also notice something unrelated to gameplay: a CALL to a machine language intro that provides window dressing for Angry Planet using the F256 Tile system and a set of converted renderings from a photograph of my own personal PET 4032 machine. This is augmented by a large compound sprite, styled in wireframe like the protagonist in Nintendo's *Punch Out* arcade game. Artwork is courtesy of my daughter; she is still working on it.

As always, the last 20% is taking 80% of the time, but I'm shooting to distribute this by year-end. I am sacrificing the year-end 32 page edition of Foenix Rising in order to tidy up and distribute this title, a working version of SIDlab, and a version of MIDI Retro-scope for the F256K2 (with some help).



Angry Planet, mid-development

Porting notes, gotchas, and work-arounds

(a.) Moving code to the Foenix platform - The first two challenges to porting, involve getting code from the vintage platform to a modern platform. Some BASIC languages (Commodore is one) store data in tokenized format, with line numbers encoded in binary, memory pointers linked, and single byte tokens representing BASIC keywords.

The F256, on the other hand, employs an ASCII text format when stored on disk, and optimizes once loaded into memory.

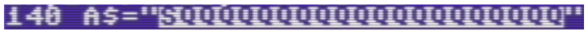
Commodore's internal format is well documented and there are programs to convert to text. For small programs, however, I found it just as easy to scrape from the VICE emulator and move text to a modern editor. I then used `fnxmgr` to push the code through USB into $2:8000 and SuperBASIC's XLOAD command to push it down to main memory where it could be run and saved.

For small and less complex programs, the F256 screen editor is just fine, but in this case, I found myself frustrated by looking back and forth between a version in the emulator, and the code in SuperBASIC so I altered my workflow to use "VS Code" where I could leverage robust editing, search and replace, split screen, and other tools.

(b.) Requirement for a HOME command - To update screen contents in BASIC, programmers must POKE characters directly, or position the cursor to PRINT.

Some languages offer an x/y coordinate based PLOT command, but less sophisticated efforts fall far short of this, providing limited control beyond the simple clear screen (form feed) and carriage return (and/or linefeed), common on previous generation TTYs.

Applesoft BASIC had HTAB and VTAB commands in addition to the ability to 'home' and clear the screen, and Commodore BASIC, from inception, provided home and full cursor control tokens, expressed within their PRINT command. Here, a <HOME> is followed by 20 <CURSOR DOWN> characters. This is used in Ouranos to positions the cursor on a lower status line:



```
140 A$="SQQQQQQQQQQQQQQQQQQQQ"
```

Unfortunately, SuperBASIC falls short in this regard. It has four-direction cursor control via ctrl chars and a CLS BASIC command to clear the screen (also bound to ctrl-L / form feed), but no ability to HOME the cursor.

The F256 hardware *does* support an x/y system via registers at `$D014` and `$D016`, however SuperBASIC does not consistently reference them. It also fails to adapt its screen editor for 40 x 30 or other modes.

But if you look into the `cls.asm` file of the SuperBASIC GitHub repo here, you'll see that CLS indeed calls its own HOME function. Unfortunately, the secondary function is not bound to a control character so there is no way to PRINT CHR$(x) and no documented address vector to reliably call.

SuperBASIC does use the read/write VICKY registers for cursor positioning, but it only writes to them. It maintains its own memory variables for *x* and *y* cursor positions (`EXTColumn` and `EXTrow`) but the bad news is, memory addresses are mixed within a block of variables that are subject to change between versions. And there is no way to identify the SuperBASIC version number, so determining SuperBASIC's variables for this purpose is difficult.

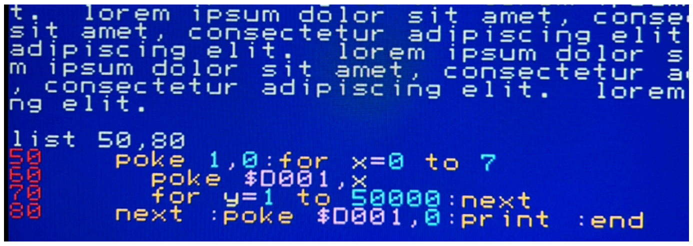
What I was able to do was to clone the assembly language from the latest GitHub push and write a short initialization routine that attempts to scrape the version number from the power-on screen; failing this, it examines memory after a built-in test to find the variable in memory. The user may override the *learned* settings by uncommenting line #1 of Angry Planet and enter their own values. (a table is maintained with several options based on my systems and others)

(c.) Double-x / double-y text modes - While there were exceptions, the first generation of popular vintage computers predominantly featured 40 character line lengths and 24 or 25 screen lines. This was driven primarily by the capabilities of NTSC and PAL televisions of the day, and the need to play well with RF modulators, which degraded signal quality considerably.

Of course the first PET and some CP/M based machines used hard wired CRTs and were crystal clear. However, screen memory was also a limiting factor and 1k - 2k seemed about right for a machine with only 8k-16k of memory. Thus, a 40 x 25 line standard was born.

Foenix F256 machines have registers to enable double-x, double-y, and 70 Hz. refresh and when combined, provide 8 different character modes. Turning all bits on (value = 7) will present a 40 x 25 screen; perfect!

The following 4-line SuperBASIC program will give you a quick 1 minute tour through the 8 character modes before returning *safely* to the default 80 x 60 screen. Be sure to have some text on the screen prior to running this so you can see the difference as the modes change:



```
t.  lorem ipsum dolor sit amet, conse
sit amet, consectetur adipiscing elit
adipiscing elit.  lorem ipsum dolor s
m ipsum dolor sit amet, consectetur a
, consectetur adipiscing elit.  lorem
ng elit.

list 50,80
50      poke 1,0:for x=0 to 7
60        poke $D001,x
70        for y=1 to 50000:next
80      next :poke $D001,0:print :end
```

Notice, I said it would 'safely' return to the default. The key message here is, the current version of SuperBASIC only supports 80 x 60, and while the program above works and looks good in mode = 7, SuperBASIC breaks. Cursor movement, program listings, and character wrap all depend on 80 x 60, without exception.

Pro-tip: If you find yourself flying blind; try this:

```
POKE 1,0:POKE $D001,0:POKE $D000,1:CURSOR ON
```

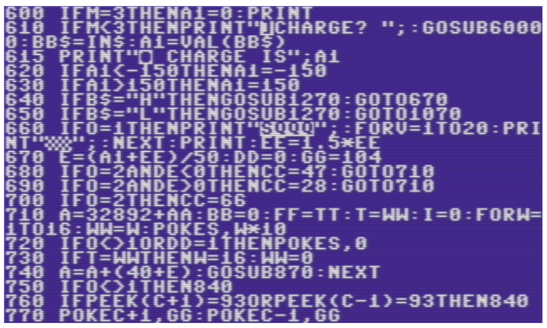
It will give you a chance to save your work !!

To port Ouranos, I opted to center the original 40 cols. within 80 and used the free space for other things.

For a deep dive discussion and a demo of F256 character modes, you might be interested in this video from my YouTube channel.

(d.) Treatment of whitespace - The BASIC language is known to be a bad student in the school of structured programming. It is common to cram as much code into as few lines as possible. In fact, there are contests and videos dedicated to this dark art (see my "Five line F256 terminal" for an example).

To illustrate, here is a look at the original code:



```
600 IFM=3THENA1=0:PRINT
610 IFM<3THENPRINT"▣CHARGE? ";:GOSUB6000
0:BB$=IN$:A1=VAL(BB$)
615 PRINT"▢ CHARGE IS";A1
620 IFA1<-150THENA1=-150
630 IFA1>150THENA1=150
640 IFB$="H"THENGOSUB1270:GOTO670
650 IFB$="L"THENGOSUB1270:GOTO1070
660 IFO=1THENPRINT"S000";:FORV=1TO20:PRI
NT"▒";:NEXT:PRINT:EE=1.5*EE
670 E=(A1+EE)/50:DD=0:GG=104
680 IFO=2ANDE<0THENCC=47:GOTO710
690 IFO=2ANDE>0THENCC=28:GOTO710
700 IFO=2THENCC=66
710 A=32892+AA:BB=0:FF=TT:T=WW:I=0:FORW=
1TO16:WW=W:POKES,W*10
720 IFO<>1ORDD=1THENPOKES,0
730 IFT=WWTHENW=16:WW=0
740 A=A+(40+E):GOSUB870:NEXT
750 IFO<>1THEN840
760 IFPEEK(C+1)=93ORPEEK(C-1)=93THEN840
770 POKEC+1,GG:POKEC-1,GG
```

Of course, the next person that comes along has their work cut out for them. NONE of this will run will run on a F256, as is.

SuperBASIC demands whitespace and order, and does not have a sense of humor when encountering code formatted like this. Here is a simple (bad) example:

```
ifichar=7then30
```

… here is the minimum of what is acceptable:

```
if ichar=7 then goto 30
```

Whatever you do, DO NOT omit the space between 'then' and 'goto'. Doing so will raise the infamous "Open Structure" error which fails to identify the line number of the offending line, leaving you guessing.

Of course, it would be wrong to blame SuperBASIC for requiring well ordered code, but if you start with tightly packed Commodore-style code, you're going to be busy!

Looking at the code above, lines 610, 660, and 710 are likely to be too long once whitespace is inserted. Lines 680, 690, 720, and 760 will need to be refactored, and other code which depends on this code (especially where variable `EE` must now be raised to `EE#`, a floating point) will demand other changes. We will cover some of this complexity in items (e.) and (f.) below.

SuperBASIC gets high marks for its **PROC** and blocked **IF** / **ENDIF**, but is unforgiving otherwise. The lesson here is to embrace modern features and methods whenever possible. If converting old code, give in early, and take the time to refactor. There is no other way.

(e.) Var data type oddities - Despite being 50 years old, the 'C' language set the bar for *modern* typed languages, establishing a comprehensive set of types (such as `int`, `float`, `char`, `double`) and establishing conventions that would ensure portability across system architectures, allowing for new types as technology evolved. And they have.

But BASIC was not as ambitious, but of course, predated the C Language by 8 years. BASIC prescribed a simple, single character type that **only** supported floating point numbers.

Dartmouth basic allowed string literals to be printed, but there was no string variable supported until a subsequent release.

Microsoft BASIC for 6502 supported the string '`$`' data type right out of the box. Microsoft's view (based on work at Dartmouth) was, numeric vars were of type float unless declared as an integer ('`%`' suffix). Of course, the processing required to compute values with mantissa and exponents is expensive, so `ints` have always been speedy by comparison. There is a funny story, or folklore at a minimum, about Bill Gates complaining that Steve Wozniak's Integer BASIC outperformed Microsoft's version. (this was prior to Apple licensing it for Applesoft, To me any story that includes "Bill Gates" and "unhappy" is true, so is worth repeating :)

The key point is, SuperBASIC bucks convention, and does the complete opposite; it defaults to integer and requires a '`#`' var suffix when a float is desired.

So if you enter the following in immediate mode:

```
pi=3.1415927
```

You are bestowed with an "`Out of Range`" error. This example was discussed on the Foenix Discord earlier this month.

So be aware of this gotcha. Ouranos code has multiple calcs that use decimal math, so an add of "#" to a handful of variables was required. And then (of course), accompanying **INT()** functions had to be inserted elsewhere to avoid "`Illegal argument`" errors across references and assigns.

(f.) Complex / compound expressions - Here's a head scratcher: Item '(d.)' above notwithstanding, what is wrong with the following expression:

```
s$=spc(int(3.5+(7-len(t$))/2))
```

The answer should be "nothing"; given a name such as "Bartholomew", you might expect `s$` to equate to a single space ' '.

SuperBASIC returns (as above in example 'd.') an "`Illegal argument`" error. But why?

If we split this into two steps:

```
sz=int(3.5+(7-len(t$))/2)
s$=spc(sz)
```

… we arrive at the correct result. Chock it up to a parser problem/bug (or feature?). Be aware that in many cases, you will find a remedy by splitting expressions into multiple parts.

There is other general weirdness to ponder such as:

```
print int(4/2)       … which yields the expected:
2
```

meanwhile…

```
print int (4/2)      … returns something unexpected:
02.00000
```

It would seem that the whitespace between the `int` function and the expression of (`4/2`) provoked a padded '10s place float. Not so fast…

What is actually happening: whitespace is causing eval of `int` as an uninitialized var (returns `0`). But why does `print` output `2.00000`?

We invoked floating point division inadvertently (versus Integer division?!). Scribbled on the bottom of page 29 of the SuperBASIC manual are the words:

"*Signed division. An error occurs if the divisor is zero. Backslash is integer division, forward slash returns a floating point value.*"

The message is, be intentional in your use of operators and bound check your work in immediate mode; else, unexpected results will visit you in the future.

(g.) If / then conditional form - The SuperBASIC manual differentiates the standard form of `if / then` from its preferred form. It also suggests something between, which is downright ugly.

'standard' form:

```
if {conditional test} then {action}
```

Well, it's standard except, if you intend on jumping to a line number, you need to explicitly say **GOTO** after **THEN**

The 'preferred' form, which modern developers will like is:

```
if {conditional test}
  {action}
endif
```

This is all for the good, I suppose, but fails to explain why compound boolean operations within conditionals are not supported. So, gone is the ability to express:

```
if g > 64 and g < 74 then print "D grade"
```

… and no combination of parens will resolve this.

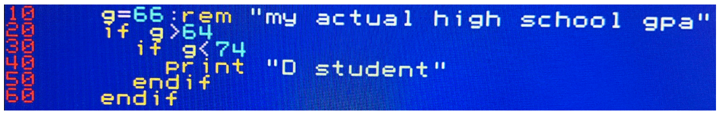
The 'acceptable' form can be single line or multi-line and would read as follows (as mentioned, its 'ugly'):

```
If g>64:if g<75: print "D grade":endif:endif
```

So, now is a good time to embrace the structured programming 'way' of nesting conditionals. If you adopt this form simply:

```
10 g=66 : rem "my high school gpa"
20 if g>64
30 if g<75
40 print "d"
50 endif
60 endif
```

... SuperBASIC will reward you with a pretty print listing as follows (but only when listed from the beginning of the logical block; line 20 in this case):



```
10   g=66:rem "my actual high school gpa"
20   if g>64
30     if g<74
40       print "D student"
50     endif
60   endif
```

This is much better, and if you've developed software in other languages, it's probably how you think about complex conditionals anyway. And yes, '`else`' is supported, but in my version of SuperBASIC, it doesn't format correctly.

In closing, here is the link to the SuperBASIC documentation. `MGR42` added **CALL** to the document in March of 2024, but more edits are warranted. Also, bug fixes, better error messages, work on the expression evaluator, and some quality of life features. If you've been around Foenix for a while, you'll know that some good programs have already been released. Quirks aside, it is powerful and useful. And please share your findings on the SuperBASIC channel.

Other tips and tricks for Microsoft BASIC converts

- There is no immediate mode **RUN** *x* form, but you can **GOTO** *x* to begin execution at a given line number.
- **LIST** uses a comma not a hyphen and it is acceptable to say "`,100`" to express "to line 100" or to say "`100,`" to express "list lines 100 onward". There is no way to slow down or pause a program listing and no continuous, cursor guided scrolling up or down. It is also possible to **LIST** *procname*(). This will list a procedure in its entirety. (the parens are required whether or not variables are passed).
- There is no **STEP** (commonly available as an optional parameter in **FOR** loops), however you can use a **WHILE / DO** and increment the iterator as needed. NOTE that there is a **DOWNTO** modifier, valid in **FOR**, but only for consecutive values. **WHILE** is more flexible.
- Pg. 29 of the SuperBASIC documentation cites "**and**" but should mention "/" (float division) and "\" (integer division); the latter is equiv. to **DIV** in BBC BASIC.
- It is possible to **BLOAD** binary data into memory at a specific address, including memory above `$00:FFFF` however, you cannot load FONT data directly into the `$C000` range. A work around for this is to "two-step it", by loading data into RAM, then copy as follows:

  ```
  bload "cbm.font",$2000 : poke 1,1
  for x = 0 to 2047:poke $c000+x,peek($2000+x):next
  ```
- The difference between **PRINT** and **CPRINT** is, the original will 'action' cursor commands (see sample* below), while the latter prints glyph characters for the set. Try:

  ```
  for x = 1 to 255 : cprint chr$(x);: next
  ```

*Useful F256 **CHR$** codes (use with standard **PRINT** only):

key	val	action	key	val	action
`ctrl-l`	12	clear screen	`ctrl-a`	1	start of line
`ctrl-b`	2	cursor left	`ctrl-e`	5	end of line
`ctrl-f`	6	cursor right	`ctrl-k`	11	kill to EOL
`ctrl-n`	14	cursor down	F3	131	green text
`ctrl-p`	16	cursor up	F4	132	lurid text