ALT_README

4 bpNichol DISKS

/ôlˈtərnədiv/ README

experimental readme file

a style of document that introduces poetry and programming hand in hand.

document author:
onebigear

production time:
2022.02.10

disk information

author: bpNichol

production year:
1983 - 1984

platform: Apple IIe

language: Apple BASIC

content: computer poems

edition: 100 numbered and signed copies distributed on 5.25" floppies along with printed matter.

remark: the yellow box contains manuscript versions, given to the MAL by Lionel Kearns, a poet and a friend of bpNichol.

LIST
RUN
CATALOG

You are reading this alternative” README” placed next to the Apple IIe computer. In the yellow box next to the computer labeled as “bpNichol”, there are multiple floppy disks containing the computer poems written by the eponymous poet. This short document is written for sharing, navigation and appreciation. I will share how I read the poems, understood the source program, and appreciated the works. This is written out of my endeavor to apply the methods of “critical code studies” to the lab. It is a methodology, a theory, and a doing that understands code as contextual. Poetry is for everyone, you don’t need to know all about CCS to like computer poetry. This document intends to introduce computer poems in a playful and accessible manner.

For transparency, I did have introductory knowledge of computer programming, and the lab manager, libi, demonstrated to me how to use the computer, load disk, and use commands. After that I fuzzed around and referred to the manuals in the lab for help.

Commands and methods

You need to load BASIC into the computer so that it can understand BASIC. You can find the DOS 3.3 system disk nearby the Apple IIe. Insert the disk into the disk drive before turning up the computer. Take out the disk after the system is loaded, and insert any disks titled as “First Screening” from the yellow box.

Use the CATALOG command to show the working directory. There is a HELLO program. It’s just there.

Use the RUN command to run the program:
RUN FIRST SCREENING.

Use the LIST command to read the source program. The syntax is: LIST <line number 1> <line number 2>, for example LIST 1 - 100 to display line 1 to 100. So that the interface is not displaying an overflow of text and you can examine selected sections of code.

Depending on which copy you picked, the line numbers might not be exactly the same as I am referring to them as follows. By using the LIST command, I made a chart that specifies what sections of the code responds to which poem. We are looking at one program that contains multiple computer poems.

GOSUB &
RETURN

GOSUB 300 -> ISLAND
GOSUB 360 -> SELF REFLEXIVE NO.1
GOSUB 430 -> SELF REFLEXIVE NO.2
GOSUB 500 -> TIDAL POOL
GOSUB 550 -> Dedication (not poem)
GOSUB 1000 -> CONSTRUCTION ONE

I felt that GOSUB is arguably the most important structural command in the poemgram. Yes this is a portmanteau that I made up, referring to computer programs that are poetic and poems that are computational at the same time.

The C-64 Wiki offers a concise description:

The BASIC command GOSUB jumps to a subroutine at the indicated line number. This call is placed onto the stack. The subroutine finalizes using a RETURN command. Program execution continues at the command following the initial GOSUB command.

The Petit Computer Wiki offers a more elaborate description. You don’t need to read the entirety of the documentation to understand what the command is doing, although the theory behind it is fascinating.

GOSUB and RETURN are commands which allow use of subroutines in Petit Computer BASIC.

GOSUB must be followed by a label. When a GOSUB is encountered while running the program, that point in the program is pushed onto a 'call stack', then execution of the program continues from the label (like GOTO). When a RETURN is encountered, the last value added to the 'call stack' is removed, and execution continues from that point again (i.e. the location of the corresponding GOSUB).

Essentially, the idea is that you can create a small program (a "subroutine"), put a label at the beginning of it, and put RETURN at the end of it. You can execute this subroutine by using GOSUB with that label. If you have a task that needs to be performed many times, this can save a lot of typing and make your code more robust to change. Even if a section of code is used only once, using subroutines to separate different tasks can help make your program tidy and more easily understood and maintained.

The 'call stack' has a maximum depth of 255. This means that from the start of the program you can run 255 consecutive GOSUBs, but the next one will generate an Out of memory error. If the system encounters a RETURN when the call stack is empty, it generates a RETURN without GOSUB error.

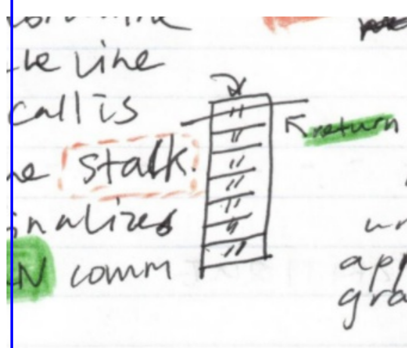

Illustration of the stack

The GOSUB stack memory is shared with the FOR stack memory... so, if the very start of a program is a FOR loop, you can only run 254 consecutive GOSUBs within that loop before getting an Out of memory error. Any additions to the FOR stack made in a subroutine, that are not removed in the normal way by the execution of a NEXT ending the loop, are removed by the RETURN. So, a single RETURN can free up two or more spaces in the stack memory. In contrast, additions to the call stack made during a FOR loop are not removed by a NEXT.

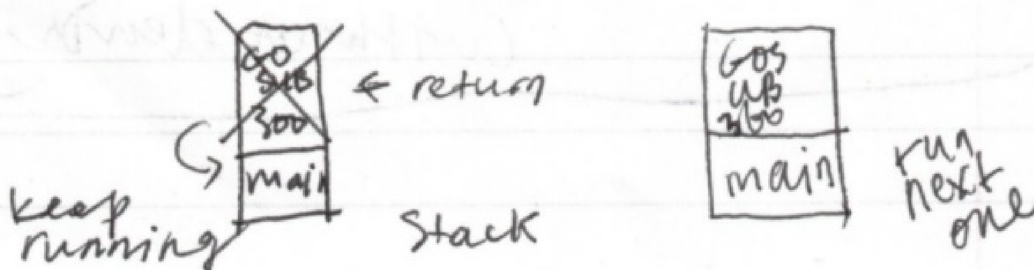

Illustration of the stack while the poemgrams are running

POEMGRAM 1: ISLAND

Critical Code Critique

As specified in the chart, you can use the GOSUB command to directly jump to the poem. I guess that when bpNichol worked on the disk, he imagined that as the poems play, it's like a screening. With GOSUB we as readers are interactive, afforded to jump to sections that we select. Although, this is not like flipping through book pages, as we can only GOSUB to the sections that are marked with GOSUB and end with RETURN. I've read the thesis of Katherine Wooler and found out the handy usage of GOSUB. I found out about Katerine Wooler's master thesis through the blog of Lori Emerson, the director of the Media Archaeology Lab. The blog posts and Katherine Wooler's thesis are linked in the appendix in the end of this document.

Code Snippet

```
300  HOME
 301  FOR PAUSE = 1 TO
2000: NEXT
 304  VTAB  8: HTAB 18:
PRINT "______"
 305  VTAB  10: HTAB 18:
PRINT "ISLAND"
 306  VTAB  11: HTAB 18:
PRINT "______"
 310  FOR PAUSE = 1 TO
4000: NEXT
 315  HOME
 320  FOR PAUSE = 1 TO
1200: NEXT
 325  FOR H = 1 TO 120
 330  PRINT " WAVE
WAVE WAVE","ROCK
WAVE WAVE WAVE","
","ROCK"
 335  NEXT H
 340  HOME
 345  FOR PAUSE = 1 TO
1000: NEXT
 350  RETURN
```

Page 97 of INSTANT BASIC gives a good explanation of the FOR NEXT loops.

To take a closer look at inside the for loop between line 325 and 335.

You may also want to refer to page 10 to 12 on the particularities of the PRINT statement in BASIC, as the use of commas and semicolons will produce different print layouts.

Poemgrammatic Ontology

I have to use this phrase "poemgramatic ontology", if you are going to read through this document you have to bear with me remixing techno-philosophy into computer poems. The "ontology of the poemgram" provides me a thinking framework, through which the relationships between the poetic and the computational qualities of the writing emerge and unify, into the "poemgrammatic"!

Please experience bpNichol's poetry outside of his computer poems. For example The Alphabet Game: A bpNichol Reader. After reading bpNichol's poems addressing directly the paper and typography as the media and the interface, one may see how the computer poems continue to embody a concern, a sensitive eye for the media and the interface.

The poemgram makes a simple and creative use of the for loop logical structure. The for loop is one of the first things to encounter while learning programming, maybe bpNichol thought of the creative use of for loop when he learned BASIC? The for loop of 120 iterations produces the flowing "WAVE". {insert what the rock statement is doing}

POEMGRAM 2: SELF REFLEXIVE NO.1

Logic wise, it's not all that different from the one above: for loop, empty spaces and all.

POEMGRAM 3: SELF REFLEXIVE NO.2

The poemgram reminds me of File Not Found, a BASIC Poem for Ron Padgett. It is written by the computational artist Nick Montfort and he discussed it in A Platform Poetics: Computational Art, Material and Formal Specificities, and 101 BASIC Poems (2013–). Let's read the poem, understand the structure, and dwell into the similarities and uniqueness of the poemgrams.

File Not Found, a BASIC Poem for Ron Padgett

```
10 PRINT "Nothing in that
drawer."
20 FOR I = 0 TO 999
30 NEXT I
40 GOTO 10
```

You can look for how Nick interpreted the poem in his own words in the article. It can be found in issue 1 of the digital review (published on the internet). He is saying that, as the poemgram loops through the print statement, it is referring to a different drawer at each time. And as each drawer is pulled, nothing can be found in that drawer, and the poemgram keeps on looping, until the 999th time.

SELF REFLEXIVE NO.2 also makes use of the for loop, but the attention is not at the drawer, but at the bottom line of the screen. Similar to the command interfaces we use today, a new print statement will appear at the bottom of an old one. Both poemgrams reflect the computer screen as a space. One leads us to understand the computer screen to a wall of 999, empty drawers; and the other one leads us to the bottom where change is persistent. While both poems both use a for loop structure, they taste different, if you take a bit of time to "taste" them you will be sure what sentiments I am referring to.

POEMGRAM 4: TIDAL POOL

TIDAL POOL is an inverse of THE ISLAND, or, a "relative motion" to THE ISLAND. Can you imagine how it is implemented?

POEMGRAM 5: CONSTRUCTION ONE

```
1015  HOME
1018  FOR PAUSE = 1 TO
2500: NEXT
1020  FOR F = 1 TO 24
1025  PRINT " "
1030  NEXT F
1035  PRINT " ","TOWER"
1040  FOR PAUSE = 1 TO
1200: NEXT
1045  GOSUB 1520
1050  FOR PAUSE = 1 TO
1150: NEXT
1055  GOSUB 1520
1060  FOR PAUSE = 1 TO
1100: NEXT
1065  GOSUB 1520

                ...

1520  PRINT " ","TOWER"
```

This is my favorite one. Not that it is my favorite poemgram from the disk, but my favorite interpretation, because it used the GOSUB command extensively. The poemgram uses more lines than the ones before. From line 1040 to line 1065, the poemgram is taking shorter and shorter pauses to call GOSUB 1520. The statement in line 1520 is to print the word "TOWER". Poemgramatically, GOSUB has advantages over using a print statement each time. Can we think of other ways to write this section of the poemgram with less lines? *Hint: decreasing intervals.*