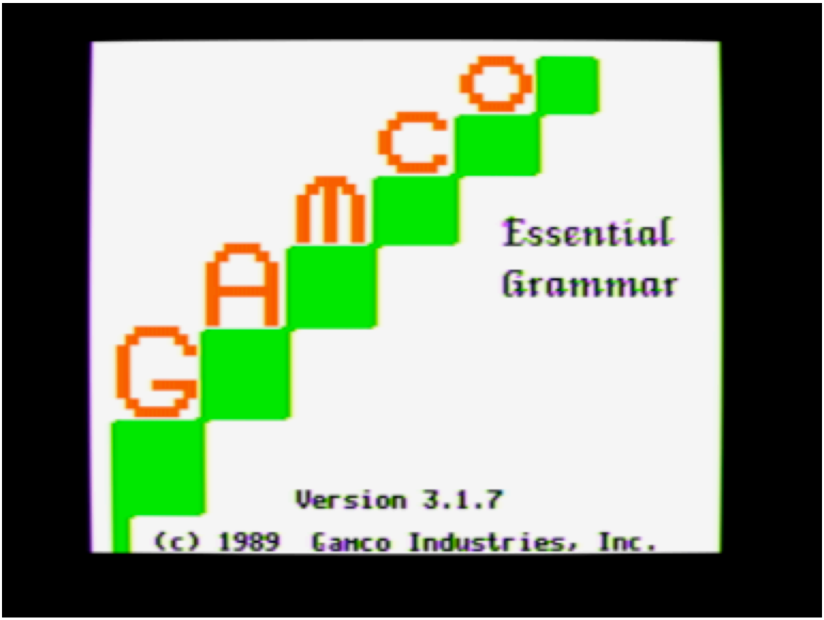
Essential Grammar



2016-10-08

4am
to deprotect
and preserve

Contents

```
-----------Essential Grammar-----------
A 4am crack                  2016-10-08
---------------------------------------

Name: Essential Grammar
Version: 3.1.7
Genre: educational
Year: 1989
Credits:
  Writer: Gayle Briscoe
  Programmer: Doug Pounds
Publisher: Gamco Industries, Inc.
Platform: Apple ][+ or later
Media: double-sided 5.25-inch floppy
OS: ProDOS 1.1.1
Previous cracks: none
Similar cracks:
  #842 Essential Punctuation (MAKE UP
       YOUR MIND IF IT'S ESSENTIAL WE
       CAN'T END IT SILLY RABBIT)
  #841 End Punctuation
  #826 Math Football
  #795 Treasure Dive
  #794 Grammar Baseball
  #686 The Word Problem Game Show
  #685 Eureka: Following Directions
```

Chapter 0
In Which Automation Fails Us
(Does It?)

COPYA
  fails on last pass (both sides)

Locksmith Fast Disk Backup
  unable to read track $22 --
  copy boots ProDOS, prints "BEAGLE
  COMPILER 2.6", then prints "NOT AN
  EXECUTABLE DISK." and hangs

EDD 4 bit copy (no sync, no count)
  works

Copy][+ nibble editor
  track $22 is entirely unformatted

Disk Fixer
  T00 -> looks like standard ProDOS
  no way to read T22

Why didn't COPYA work?
  intentionally unformatted track

Why didn't Locksmith FDB work?
  Probably a runtime protection check
  to ensure track $22 is unreadable

EDD works. What does that tell us?
  The protection check is probably very
  simple, just checking that track $22
  is unreadable.

Next steps:

  1. Search for protection check
  2. Disable it
  3. Declare victory(*)

(*) Go to the gym

Chapter 1
In Which We Strike Out
(Do We?)

```
On the theory that some code on disk is
trying to access track $22, and thus
noticing if it's unexpectedly readable,
let's enumerate some of the ways that
could happen:

- Reading a file that is mapped to the
  unreadable track $22. Copy II+ disk
  map shows there are no files mapped
  to track $22, so let's rule that out.

- Manually seeking to the track and
  looking for a nibble sequence. Given
  that this disk is ProDOS-based, there
  is no explicit support for "seeking
  to a particular track" unless you're
  calling ProDOS internals. (But that's
  always possible, of course!) Without
  calling into ProDOS, this technique
  would require low-level disk access
  (turning on the drive and hitting the
  right stepper motors and whatnot). A
  sector search with Disk Fixer didn't
  find any suspicious instances of

  "BD 89 C0" (LDA $C089,X)    ; drive on
  "AD E9 C0" (LDA $C0E9)      ; drive on
  "BD 80 C0" (LDA $C080,X)    ; stepper

  or any similar variations that would
  point to low-level disk access.
```

- Issuing a ProDOS MLI "raw block read" and checking the return code. This is a popular technique under ProDOS, partly because it can be adapted to work on 3.5-inch and 5.25-inch disks. Combined with the knowledge that EDD bit copy produced a working copy, I suspect this is what I'm looking for.

Unfortunately, a sector search for "20 00 BF 80" (JSR $BF00 / [80]) -- the standard opcode sequence for calling the ProDOS MLI with command $80 (block read) -- turned up nothing at all. Which means, perhaps, that this entire chapter was just mental gymnastics.

Or is it? Dun dun DUN...

Chapter 2
In Which Our Tools Prove Useful At Last (Do They?)

After striking out looking for the
protection check, let's change tactics
and look for the error string instead.
My non-working copy prints the message
"NOT AN EXECUTABLE DISK" after booting
ProDOS, launching the Beagle Compiler,
and executing the startup program -- a
delightfully bald-faced lie only a copy
protection developer could love.

Searching for the string "EXECUTABLE"
finds nothing, but wait! Disk Fixer
also supports finding strings with the
low bit clear. Pressing "T" twice to
change from "NORMAL" to "INVERSE" to
"FLASH" mode, then "F"ind "A"SCII and
search for "EXECUTABLE" again. Lo and
behold, it finds a match on track $0A.

```
                 --v--

-------------- DISK EDIT --------------
TRACK $0A/SECTOR $00/VOLUME $FE/BYTE$00
---------------------------------------
$00:>6C<96 0C 60 7C 96 30 BF    l..`|.0?
$08: 6E 96 01 60 4A A4 7C 96    n..`J$|.
$10: 2A 62 94 01 06 1F 02 14    *b......
$18: 0B 94 11 14 0C 94 01 4A    .......J
$20: A4 7C 96 2A 62 94 06 06    $|.*b...
$28: 32 02 14 0B 94 3F 14 0C    2....?..
$30: 94 06 6C 96 0F 60 9C 0B    ..l..`..
$38: 6C 96 10 60 9C 0C 6E 96    l..`..n.
$40: 01 60 4A A4 7C 96 00 60    .`J$|..`
$48: 94 27 09 50 02 0A 51 02    .'.P..Q.
$50: 0E 28 66 94 0A 64 94 09    .(f..d..
$58: 1C 9A 17 4E 4F 54 20 41    ...NOT A
              ^^^^^^^^^^^^^^
                   lies!

$60: 4E 20 45 58 45 43 55 54    N EXECUT
     ^^^^^^^^^^^^^^^^^^^^^^^
           damned lies!

$68: 41 42 4C 45 20 44 49 53    ABLE DIS
     ^^^^^^^^^^^^^^^^^^^^^^^
      (but not statistics)

$70: 4B 2E 24 0A 73 02 6C 94    K.$.s.l.
$78: E6 94 20 6E 96 01 18 0E    f. n....
---------------------------------------
BUFFER 0/SLOT 6/DRIVE 1/MASK OFF/FLASH

---------------------------------------
COMMAND : _

                 --^--
```

Copy II+ disk map tells me that T0A,S00
is part of the file /G/G157/LOGO:

```
                 --v--

DISK MAP                SLOT 6  DRIVE 1
/G/G157/LOGO

   TRACK            1               2
   0123456789ABCDEF0123456789ABCDEF012

S0 ..........*........................
EE ..........*........................
CD ..........*........................
TC ..........*........................
OB ..........*.......*................
RA ..........*.......*................
 9 ...................................
 8 ...................................
 7 ...................................
 6 ...................................
 5 ...................................
 4 ...................................
 3 .........*.........................
 2 .........*.........................
 1 .........*.........................
 F .........*.........................

                 --^--
```

Switching back to Disk Fixer, I can go
into input/output control (press "O")
and switch the DOS type to ProDOS
(press "P"), then go to directory mode
(press "D") and select the "LOGO" file
inside the "G157" directory. Now I can
follow the file with my left and right
arrow keys. A very useful feature!

Chapter 3
In Which The Truth Is In The Numbers (Is It?)

The "LOGO" file is in some sort of Beagle Compiler-specific compressed format. I see strings (including some BLOAD commands and the aforementioned "NOT AN EXECUTABLE DISK" error string), but I can't LIST it. I also do not have a Beagle Decompiler. Is there such a thing? I scoured the documentation from Beagle Bros. and found no reference for this file format. That would be a worthwhile side project!

However, on the last sector of the file (T12,S04 -- slightly confusing because Disk Fixer is numbering the sectors in ProDOS order, while Copy II+ disk map numbered them in DOS order), I see an interesting "string":

```
                 --v--

-------------- DISK EDIT --------------
TRACK $12/SECTOR $04/VOLUME $FE/BYTE$00
---------------------------------------
$00:>FF<09 13 08 4A A4 9C 16   ....J$..
$08: AC 9C 15 94 21 0F 13 08   ,...!...
$10: 0A 31 08 4A A4 7C 96 1C   .1.J$|..
$18: 18 94 FF 06 2E 08 4A A4   ......J$
$20: 9C 16 AC 9C 15 96 DB 01   ..,...[.
$28: 0F 2E 08 0A 31 08 0A F2   ....1..r
$30: 07 0A 84 07 12 02 33 32   ......32
                       ^^^^^

$38: 01 30 03 31 39 31 03 31   .0.191.1
     ^^^^^^^^^^^^^^^^^^^^^^^

                                  [...]
```

```
$40: 32 38 02 31 31 02 39 36   28.11.96
     ^^^^^^^^^^^^^^^^^^^^^^^

$48: 03 31 34 31 01 30 02 39   .141.0.9
     ^^^^^^^^^^^^^^^^^^^^^^^

$50: 36 02 39 36 01 33 02 39   6.96.3.9
     ^^^^^^^^^^^^^^^^^^^^^^^

$58: 36 01 30 02 39 38 01 32   6.0.98.2
     ^^^^^^^^^^^^^^^^^^^^^^^

$60: 01 30 02 31 34 02 31 38   .0.14.18
$68: 01 32 01 37 02 31 30 02   .2.7.10.
$70: 31 34 01 38 02 31 32 01   14.8.12.
$78: 37 02 31 30 02 31 33 02   7.10.13.
---------------------------------------
BUFFER 0/SLOT 6/DRIVE 1/MASK OFF/NORMAL
PRODOS:LOGO                        /$04
---------------------------------------
COMMAND : _
```

--^--

Starting at byte $36, I spy with my little eye a sequence of numbers written out as a string of ASCII characters:

"32 0 191 128 11 96 141 0 96 96 3 96 0 98 2 0"

...and so on. Each is preceded by a hex byte #$01, #$02, or #$03, which appears to be the length in character of the number as a string. "32" is always preceded by #$02 because it's a 2-digit number -- er, string -- while "191" is preceded by #$03 because it's a 3-digit number string.

So what is this sequence? The numbers are all decimal (base 10) numbers between 0 and 255. Converted them to hexademical (base 16), they turn out to be quite interesting indeed:

```
 32 -> $20
  0 -> $00
191 -> $BF
128 -> $80
 11 -> $0B
 96 -> $60
141 -> $8D
  0 -> $00
 96 -> $60
 96 -> $60
  3 -> $03
  0 -> $00
 98 -> $62
  2 -> $02
  0 -> $00
```

"20 00 BF" looks suspiciously like 6502 assembly code. (It's a JSR to $BF00, which is the ProDOS MLI subroutine!) The rest of it fits my theory that this ends up as executable code. I like data that fits my theory. That's so much easier than changing my theory to fit the data. Boring!

Dropping into the monitor, I can see it in its native form:

```
*6000:20 00 BF 80 0B 60 8D 00 60 60 03
 60 00 62 02 00

*6000L

6000-   20 00 BF    JSR   $BF00
6003-  [80]         ; MLI block read
6004-  [0B 60]      ; MLI param address
6006-   8D 00 60    STA   $6000
6009-   60          RTS
600A-  [03]         ; MLI param count
600B-  [60]         ; unit number
600C-  [00 62]      ; buffer address
600E-  [02 00]      ; block number
```

Block $0002 is not on track $22, but perhaps it's being changed dynamically at runtime? (That would also fit my theory that this protection routine can be adapted for 5.25-inch and 3.5-inch disks.)

At any rate, I think I know what I'm looking at here. This (compiled) BASIC program is taking a sequence of numbers and POKE-ing them into memory, then calling that assembly language routine to accomplish what would otherwise impossible from pure BASIC -- calling the ProDOS MLI with a raw block read, and storing the result.

```
According to "Beneath Apple ProDOS"
(p. 6-19), the return codes from a raw
block read (MLI code $80) are

   $00 - no errors
   $04 - incorrect parameter count
   $27 - I/O error or bad block number
   $28 - drive not found
   $56 - buffer already in use

Since the original disk's track $22 is
unformatted and thus unreadable, I'm
guessing the "correct" answer is $27
("I/O error"). The block number seems
to be set at runtime, so I can't just
hard-code a bad block number. (Darn!)
But I can change this entire routine so
it simply puts the expected value in
memory address $6000.

*6000:2C
*6003:EA A9 27
*6000L

6000-   2C 00 BF     BIT   $BF00
6003-   EA           NOP
6004-   A9 27        LDA   #$27
6006-   8D 00 60     STA   $6000
6009-   60           RTS

Converting this hacked routine back to
decimal would yield the sequence

44 0 191 234 169 39 141 0 96 96 3 96
```

Unfortunately, when I converted that sequence into the Beagle Compiler format, it was exactly one byte too long. For visual comparison, using "." placeholders for the length bytes:

Original:
```
".32.0.191.128.11.96.141.0.96.96.3.96"
```

Hacked:
```
".44.0.191.234.169.39.141.0.96.96.3.96"
```

See? Too long.

However, since the BIT instruction is now effectively useless, I can twiddle the other two bytes in that instruction to try to regain a byte. I can't make 0 any shorter, but I can change 191 to any 2-digit number -- which, when it is written out as a string, will be one byte shorter.

New hacked routine:
```
".44.0.10.234.169.39.141.0.96.96.3.96"
       ^^
```

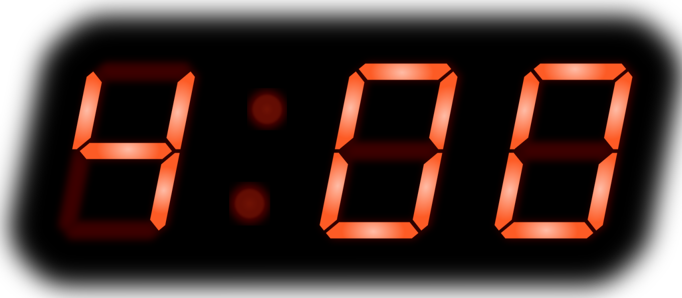
...which fits precisely in the space provided, and has the benefit of always putting the expected value (#$27) into the appropriate memory location ($6000) whether track $22 is readable or not.

```
T12,S02,$36:
"33 32 01 30 03 31 39 31 03 31 32 38 02
 31 31 02 39 36"
 -->
"34 34 01 30 02 31 30 03 32 33 34 03 31
 36 39 02 33 39"

]PR#6
...works, and it is glorious...
```

Side B also has an unreadable track $22 but no second copy of the protection code, er, string. The game works all the way through and again after playing it through, so I'm fairly confident no further patches are required.

Quod erat liberandum.



A 4am crack   No. 869
-------------------EOF-----------------