PAY NO MORE

$2.00 US

ISSUE 34

1986

Midnite Software Gazette

The First Independent U.S. Magazine for users of Commodore brand computers.

Includes The PAPER

Published by: Micro-PACE Computers,
Robert Wolters.
Editor-In-Chief: Jim Oldfield Jr.
Editor: Tim Sickbert
Assc. Editors: Art Lewis Kimball
Robert Baker
Dr. Immers
Elizabeth Kasper

Address for all correspondence:
PO Box 1747
Champaign, IL 61820

Telephone: 217-356-1885
BBS: 217-356-8056
Punter 300/1200

Issue 34 June 1986

A few notes: This only works if you specify printer type "0" (CBM) at the initial screen (this is the default setting for Easy Script.) You will also notice that there is a lower-case "qo" before each line in your outputted text file. You can get rid of them by reloading the file, using the search/replace feature to delete them, and then resaving the formatted document with the normal (F)ile command. Oh, and never just hit RETURN when prompted for the file name; Easy Script will get real confused. Has anybody else found any other hidden features in this program? -Kevin Hisel

* * * * * * * * * * * * * * *

COMPACTOR-128
and
UNCOMPACTOR-128

By Robert W. Baker

These are updated versions of earlier programs that appeared in several magazine articles and as part of my Commodore-64 Programmer's Library. Originally written for the Commodore PET and C64, I have made several improvements and updated the program to run on the C128 and properly handle BASIC 7.0 commands. These versions should only be run on the C128 in 128 mode.

COMPACTOR 128

The Compactor program reads a BASIC program from disk and creates a new, compacted copy. Compactor-128 deletes all REMarks, unnecessary spaces, and leading colons in a fashion similar to other utility programs.

This program, however, goes one step further. It combines program lines whenever possible to eliminate the link, line number, and line end flags normally associated with each BASIC program line. It makes a BASIC 7.0 program almost as small as possible, and most likely faster running.

I should point out, however, that there was no attempt to play with IF/THEN/ELSE statements to get further improvements, since this was beyond the scope of the program and would make the logic extremely complex.

While creating the original version of this program, I came across a few undocumented "quirks" of Commodore BASIC. Since many people like to experiment with the capabilities of having programs "write" programs on disk, this information may be of interest:

ZERO LENGTH LINES:

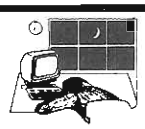
Normally, it is impossible to create a zero length line using the screen editor on any Commodore system. By zero length line, I mean a line with a link, line number, and end of line flag but no BASIC commands or text. If you were to type just a line number using the screen editor, you would actually delete a line instead of entering a zero length line.

When writing a BASIC program on disk as a data file, there is nothing to stop you from entering a zero length line. But if you want the program to run, you cannot have any zero length lines in the program. Most Commodore BASIC's cannot link the program lines correctly whenever there is a zero length line in the program.

LONG LINES:

At the other extreme, you cannot create a BASIC line that is longer than 255 bytes. Again, using the Commodore screen editor you could not create such a line. You are normally limited to a maximum of 78 bytes due to the line wrapping characteristics and at least a one digit line number.

When writing a BASIC program on disk as a data file, be careful not to create a



line greater than 255 bytes. Otherwise the program will usually not load from the disk. If it does load, the program will normally be unuseable.

When running the COMPACTOR program, you have some control over what size program lines will be created. The first input prompt will ask for the maximum line length to create. This must be a positive number between 1 and 255, the default is 255. When entering small numbers, be sure to use spaces to remove unwanted digits from the default number displayed.

After selecting the maximum line length, you're then asked the name of the BASIC program file to be compacted. The program must be on disk; program files cannot be read from tape as data files. If the file is not found or any disk errors are encountered, the error will be reported and the program will abort.

Next you're asked to enter the desired name of the compacted program to be created. This name cannot be the same as the original program name or any other file currently on the disk. If any file already exists with the same name, or if any other disk errors are encountered, the error will be reported and the program will abort.

COMPACTOR reads the program to be compacted as a sequential disk data file, and the file is read twice. The first pass checks for line numbers within the subject program that are the targets of: GOTO/GO TO, GOSUB, RUN, LIST, RESTORE, RESUME, TRAP, COLLISION, or IF...THEN/ELSE...(line#) statements. The first time a target line number is found it is saved into matrix TL. A check is also made for multiple target lines in ON..GOTO and ON..GOSUB statements.

As the first pass progresses, the current line number being processed in the input file is displayed on the screen. This helps give some indication of the scanning progress since it can be rather slow. By the way, when each target line number is found and added to the TL matrix, existing entries are moved to allow inserting the new entry in the proper position and keep all entries in numeric order. This helps speed up later processing.

During the second pass, each line is copied, deleted, or compacted as appropriate. Again, the current line number is displayed as each line of the original program is processed to let you know how the program is progressing. The rules followed by the COMPACTOR are:

Any leading colons and/or spaces on a line are deleted.

A line that has only REMarks is deleted if it is not a target line. The remark will be replaced with a single colon if the line is a target line and must be retained. This may produce a leading colon if the next line is not a target line and is combined with this line. The line cannot be reduced to a zero length line since BASIC cannot link a program correctly with a zero length line, as mentioned earlier.

If any line contains a GOTO/GO TO, RESUME, EXIT, MONITOR, RUN, END, NEW or IF...THEN/ELSE statement, another line cannot or should not be combined with this line. A flag is set in variable F whenever a line containing one of these tokens is found. This flag forces the current line to be written to disk and the next line to be read without combining the two. Any line combined after these BASIC commands might never be executed, thus the compacted program would not function properly.

Any spaces within a line, not enclosed in quotes, are deleted.

Any REMarks at the end of a BASIC line are deleted to the end of the line.

Anything within quotes is copied untouched. If an ending quote is missing from the line, one is added if another line could be combined with the current line.

When a colon is found within a BASIC line and not within quotes, the next non-space character is checked before copying the colon. If a REMark follows the colon, the colon and the rest of the line is deleted. Otherwise, the colon is copied and processing continues as normal.

At the end of each BASIC line, a check is made to see if the next line can be combined with this line. If the flag in the F variable was not set to indicate the line cannot be compacted, and the next line is not a target line, and the length of the next line would not exceed the limit if compacted, the lines are combined. When combining lines, the link and line number are discarded, a colon is written, and the next line is processed as part of the previous line.

If the next line cannot be combined with the current line, the currently constructed line is written on disk with the correct link to the next line. Compactor uses a line buffer to construct the entire line before it is actually written to disk. This allows creating the correct link value to write at the front of each line.

When the end of the program is found, the last line is written to disk along with the ending zero link and all files are properly closed.

As mentioned earlier, any BASIC program line cannot exceed 255 bytes in length. If it does, the program may not load from disk or it may be totally unuseable. Compactor can take full advantage of the longest possible BASIC lines since it reads the next entire program line and calculates whether or not there is enough room to concatenate it with the current line. Actually, it makes this calculation prior to removing unneeded characters from the next line so it usually doesn't quite get to the maximum limit.

The maximum line length you specify at the start of the program can limit the compacted program's line size to some extent. It will keep Compactor from combining lines that would exceed the specified limit. However, any lines read that are already greater than the limit will be copied without combining with other lines. If you select a small limit, then most lines will be copied without combining lines but each line will be compacted by removing spaces and remarks or leading colons.

Keep in mind that the newly created compacted program may have lines that cannot be edited with the screen editor. Any program line that exceeds two screen lines on the Commodore-128 cannot be edited. The companion program, UNCOMPACTOR-128, allows you to uncompact program lines. This allows you to get program lines back to a reasonable size you can edit.

UNCOMPACTOR-128

The Uncompactor program is meant to be a companion to the Compactor-128 program. In short, this program reads a C128 Basic 7.0 program that was saved on disk and creates a new, uncompacted copy. Uncompactor takes any multi statement lines (statements separated by colons) and breaks them into separate program lines with new line numbers. Long lines created by Compactor-128 can now be edited and the program re-compacted.

When splitting multi statement lines, the new line numbers are created from the original line number with an increment of one between each new line generated. This procedure is followed for however many statements exist in the line, as long as

new line numbers can be generated without reaching the line number of the next line in the original program. If that point is reached, the remainder of the original line is then copied as part of the last line generated, with any appropriate separating colons.

The program must take into account certain Basic 7.0 tokens or keywords since they effect whether or not a particular line can be broken into separate lines. Thus, any data following a GOTO, GO TO, END, RUN, RETURN, REM, NEW, RESUME, EXIT or MONITOR token is copied unchanged to the end of the current program line. Also, once a quote is detected, the line must be copied until another quote or the end of the program line is reached. Special consideration must be given to IF/THEN/ELSE blocks with or without BEGIN and BEND statements, since they sometimes can be split.

When running the Uncompactor program, you have some control over what size program lines will be uncompacted. The first input prompt will ask for the minimum line length to try uncompacting. This should be a positive number between 1 and 255, but there is no check of the value entered. If you just hit return when prompted, the default is one, which will force every line to be uncompacted. Selecting a number like 20 will cause smaller lines to be left untouched while longer lines are uncompacted.

After selecting the minimum line length, you're then asked the name of the Basic program file to be uncompacted. The program must be on disk, program files cannot be read from tape as data files. If the file is not found or any disk errors are encountered, they will be reported and the program will terminate.

Next you're asked to enter the desired name of the uncompacted program to be created. This name cannot be the same as the original program name or any other file currently on the disk. If any file all ready exists with the same name or any disk errors are encountered, they will be reported and the program will terminate.

Uncompactor reads the program to be uncompacted as a sequential disk data file, and the file is only read once. As the original program is read, each line number is displayed on the screen. This helps give some indication of how things are progressing as Uncompactor runs, since it can be rather slow.

After copying the original program line number, the actual program line is read into the C matrix and the next link and line number are read. When the zero link is found at the end of the program, the next line number is forced to 64000. This number exceeds any possible Basic program line number, thus forcing proper handling of the last line of the program read.

Once the entire line has been read, it is scanned for colons and certain Basic tokens (as mentioned earlier) if the line is longer than the limit selected. If the line is shorter than the specified limit, it's copied untouched. If one of the special Basic tokens is found, the remainder of the line is copied untouched.

When a colon is found, the line is split if the current line number plus one is less than the next line number. The current line is written to disk with the proper link and ending flag. Single leading colons at the start of any line are retained, while spaces or extra colons following any colon in the middle of a line are deleted.

As mentioned earlier, lines containing IF-THEN-ELSE statements must be treated specially. If the line contains an IF statement without a BEGIN command, it cannot be split or the program logic would be changed. If a BEGIN command is found, then anything following on that line can

be split into separate lines.

If a BEND is found with an ELSE command immediately following it, that line cannot be split unless it contains a BEGIN command after the ELSE. Thus, lines containing an IF or ELSE cannot be split following those statements unless they contain a BEGIN command on the same line.

Whenever a quote is encountered, the remainder of the line is copied untouched until the next quote or end of line is found. Special handling of two byte tokens unique to Basic 7.0 on the C128 are also included. At the end of the program, a zero link is written to disk to properly terminate the new program and all files are closed.

Newly created, compacted or uncompacted programs are fully linked and ready to run programs. Both programs can be used on any standard Basic 7.0 program created on the C128 that do not contain imbedded assembly language routines.

As with any new utility, if entering from the printed listings be sure to save backup copies of all programs before running these utilities the first few times. To be extra safe, save your backups on separate disketts from the one being used to test the programs. Be extremely careful until the programs are fully tested and proven to work reliabily, to avoid destroying files or entire disks.

If you have any questions or problems with these or any other of my programs, you can reach me via email addressed to 'RBAKER' on Quantum Link. Download copies of these programs are also available in my program library on Quantum Link or you can send $5 for copies on disk.

[You may have noticed that we have, in this issue, deviated from our general practice of publishing generic programs that can be used on most, if not all, of Commodore's 8 bit machines. The two programs published in this issue will run only on the C128. It had to be. If anybody knows the Midnite Software Gazette and would like to write an article, please call or mail us a note and we will give you our pay rates. JO]

```
100 rem         compactor 128
110 rem      by: robert w. baker
120 rem 15 windsor dr, atco, nj 08004
130 :
140 goto 390
150 gosub 160: v1=v
160 get#5,c$: if ds=0 then v=asc(c$): return: else 1
110
170 if l$="" then return
180 la=la+len(l$)+3: a1=int(la/256): a2=la-(a1*256)
190 print#6,chr$(a2);chr$(a1);l$;chr$(0);: l$="": re
turn
200 p=0:gosub 150:ln=v1+(256*v):l1=v1:l2=v
210 print"[UP]";tab(10);ln;"          ": rem <-- 10
spaces
220 do:gosub 160:p=p+1:c(p)=v:loop until v=0:f=0:pl=
p:p=1:return
230 z=0: if n=0 then return
240 for x=xx to n-1: if ln>tl(x) then next x: n=0: r
eturn
250 if tl(x)=ln then z=1
260 xx=x:x=n:next x:return
270 l=-1:do:v=c(p):p=p+1:loop until v<>32
280 do while v>47 and v<58:if l<0 then l=0
290 l=(10*l)+v-48:v=c(p):p=p+1:loop
300 if v=46 then if l<0 then l=0: v=c(p):p=p+1
310 if l<0 then return:else y=n:if n=0 then 360
320 for x=0 to n-1:if l>tl(x) then next x:goto 350
330 if tl(x)=l then y=-1:else for y=n to x+1 step -1
:tl(y)=tl(y-1):next y:y=x
340 x=n:next x
350 if y<0 then return
360 tl(y)=l: n=n+1: if n<=mx then return
370 print"[DOWN]table overflow, too many target line
s!": goto 1120
380 print"[UP]";spc(20);"( done )[DOWN]": return
390 print"[CLR]       c o m p a c t o r - 1 2 8[DOWN
]"
400 print"this program will attempt to eliminate"
410 print"remarks and spaces, plus combine"
420 print"program lines whenever possible"
430 print"in a c128 basic program saved on disk,"
440 print"creating a smaller overall program.[DOWN]"
450 print"----------------------------------------[DO
WN]"
460 read mx,mt: dim tl(mx),t(mt),c(256)
470 read nt: for x=1 to nt: read t(x): next x
480 input"max line length to create:   255[LEFT][LEF
T][LEFT][LEFT][LEFT]";ml:if ml<1 or ml>255 then run
490 input"[DOWN] input filename:   .[LEFT][LEFT][LEF
T]";f1$: if f1$="." then end
500 close 5: open 5,8,5,"0:"+f1$+",p,r": if ds>0 the
n 1110
510 input"[DOWN]output filename:   .[LEFT][LEFT][LEF
T]";f2$: if f2$="." then close 5: end
520 close 6: open 6,8,6,"0:"+f2$+",p,w": if ds>0 the
n 1110
530 print"[DOWN]ok, starting first pass"
540 print"scanning program for referenced line#'s"
550 print"[DOWN]at line:" :gosub 150
560 gosub 150: if v+v1>0 then gosub 200:else 880
570 v=c(p):p=p+1
580 if v=32 then 570
590 if v=0 or v=143 then 560
600 if v<>34 then 640: rem skip inside quotes
610 v=c(p):p=p+1: if v=34 then 570
620 if v>0 then 610:else 560
630 rem scan standard tokens in tbl
640 z=0:for x=1 to nt: if v<=t(x) then z=t(x): x=nt
650 next x: if z=v then gosub 270: goto 580
660 if v<>145 then 720: rem 'on'
670 do:v=c(p):p=p+1: if v=0 then 560
680 loop until v<>137 and v<>141
690 gosub 270
700 if v=44 then 690
710 if v=32 then v=c(p):p=p+1: goto 700:else 590
720 if v<>203 then 750: rem 'go'
730 do:v=c(p):p=p+1:loop until v<>32:if v=64 then go
sub 270
740 goto 580
750 if v<>155 then 780:else gosub 270: rem 'list'
760 do until v<>32:v=c(p):p=p+1:loop:if v=171 then g
osub 270
770 goto 580
780 if v<>206 then 800: rem 2 byte tokens
790 v=c(p):p=p+1: if v=0 then 560:else 570
800 if v<>254 then 570
810 v=c(p):p=p+1: if v=0 then 560
820 if v<>23 then 570: rem 'collision'
830 v=c(p):p=p+1: if v=0 then 560: else if v=58 then
 570
840 if v=44 then gosub 270: goto 580
850 if v<>40 then 830: rem ignore inside ()
860 v=c(p):p=p+1: if v=0 then 560:else if v=58 then
570
870 if v<>41 then 860:else 830
880 gosub 380: xx=0: print"starting second pass, com
pacting lines[DOWN]"
890 close 5: open 5,8,5,"0:"+f1$+",p,r": if ds>0 the
n 1110
900 gosub 150: print#6,chr$(v1);chr$(v);: la=v1+(256
*v): l$=""
910 read nt: for x=1 to nt: read t(x): next x: print
"at line:"
920 gosub 150: if v+v1>0 then gosub 200: else 1130
930 do:v=c(p):p=p+1:loop until v<>32 and v<>58
940 l$=chr$(l1)+chr$(l2): if v<>0 and v<>143 then 97
0
950 gosub 230: if z=0 then 920:else l$=l$+":": goto
1070
960 l$=l$+chr$(v): do:v=c(p):p=p+1:loop until v<>32:
 if v=58 then 1050
970 z=0: for x=1 to nt: if v<=t(x) then z=t(x): x=nt
980 next x: if z=v then f=1: goto 960
990 if v=0 or v=143 then 1070
1000 if v=206 or v=254 then l$=l$+chr$(v):v=c(p):p=p
+1:goto 960
1010 if v<>34 then 960
1020 do:l$=l$+chr$(v): v=c(p): p=p+1: if v=34 then 9
60
1030 loop while v>0: if f=0 then l$=l$+chr$(34)
1040 goto 1070
1050 do:v=c(p):p=p+1:loop until v<>32 and v<>58
1060 if v<>0 and v<>143 then l$=l$+":": goto 970
1070 if f=1 then gosub 170: goto 920
1080 gosub 150: if v+v1>0 then gosub 200: else 1130
1090 if (len(l$)+pl+4)<=ml then gosub 230: if z=0 th
en 1050
1100 gosub 170: goto 930
1110 print"[DOWN][RVON]disk error": print ds$
1120 close 5: close 6: end
1130 gosub 170: print#6,chr$(0);chr$(0);: close 5: c
lose 6
1140 gosub 380: end
1150 rem ------------------------
1160 data 1000: rem max# referenced lines
1170 data 11:   rem max# tokens in tables

1180 data 8:    rem #standard tokens for scan
1190 rem goto,run,restore,gosub,then,else,resume,tra
p
1200 data 137,138,  140,   141, 167, 213,  214,  215
1210 data 11:   rem #standard tokens for compact
1220 rem  end,goto,run,if,return,new,else,go,resume,
exit,monitor
1230 data 128,137, 138,139, 142, 162,203,213, 214,
237, 250
```

```
100 rem u n c o m p a c t o r - 1 2 8
110 rem      by: robert w. baker
120 rem 15 windsor dr., atco, nj 08004
130 :
140 goto 200
150 gosub 160: v1=v
160 get#5,c$: if ds=0 then v=asc(c$):return:else 670
170 if l$="" then return
180 la=la+len(l$)+2: l1=int(la/256): l2=la-(l1*256)
190 print#6,chr$(l2);chr$(l1);l$;: l$="": return
200 print"[CLR][DOWN][DOWN]     u n c o m p a c t -
1 2 8[DOWN][DOWN]"
210 print"this program will break up lines in a"
220 print"c128 basic program saved on disk,"
230 print"creating more but smaller program lines.[D
OWN]"
240 read n: dim t(n),c(256):for y=1 to n:read t(y):n
ext y
250 print"minimum line length to try uncompacting":
input"   l[LEFT][LEFT][LEFT]";xl
260 input"[DOWN] input filename:    .[LEFT][LEFT][LEF
T]";f1$: if f1$="." then end
270 close 5: open 5,8,5,"0:"+f1$+",p,r": if ds>0 the
n 670
280 input"[DOWN]output filename:    .[LEFT][LEFT][LEF
T]";f2$: if f2$="." then close 5: end
290 close 6: open 6,8,6,"0:"+f2$+",p,w": if ds>0 the
n 670
300 print"[DOWN]ok, working on line# .....[DOWN][DOW
N]"
310 gosub 150: print#6,chr$(v1);c$;: la=v1+(256*v):
l$="": goto 350
320 gosub 170: if lk=0 then 680
330 ln=nl: print "[UP]";ln;"       ": l$=chr$(ll)+chr
$(lh)
340 x=0:do:gosub 160:x=x+1:c(x)=v:loop until v=0
350 gosub 150: lk=v+v1
360 if lk>0 then gosub 150:nl=v1+(256*v):ll=v1:lh=v:
else nl=64000
370 if l$="" then 320
380 v=x: x=1: if v<xl then 640
390 if c(x)<>58 then 450: rem colon
400 if x=1 then l$=l$+chr$(c(x)): goto 440
410 ln=ln+1: if ln>=nl then 640: rem don't split
420 h=int(ln/256): l=ln-(256*h): rem ok to split
430 l$=l$+chr$(0): gosub 170: l$=chr$(l)+chr$(h)
440 do:x=x+1:loop until c(x)<>32 and c(x)<>58
450 y=x: if c(x)<>139 then 510: rem 'if'
460 y=y+1: if c(y)=0 then 640
470 if c(y)=206 then y=y+1: goto 460
480 if c(y)=34 then do:y=y+1:loop until c(y)=0 or c(
y)=34:if c(y)=0 then 640
490 if c(y)<>254 then 460
500 y=y+1: if c(y)=24 then 580:else 460: rem begin
510 if c(x)<>254 then 600: rem bend
520 if c(x+1)=25 then l$=l$+chr$(c(x))+chr$(c(x+1)):
 x=x+2: y=x:else 610
530 do until c(y)<>32:y=y+1:loop:z=y:if c(y)<>58 the
n 640
540 do:y=y+1:loop until c(y)<>32 and c(y)<>58
550 if c(y)<>213 then x=z: goto 410: rem else
560 do:y=y+1:loop until c(y)<>32: if c(y)<>254 then
640
570 y=y+1: if c(y)<>24 then 640
580 for z=x to y: l$=l$+chr$(c(z)): next z: x=y
590 do:y=y+1:loop until c(x)<>32:if c(x)=58 then 410
:else 640
600 if c(x)<>206 then 620: rem 2-byte token
610 l$=l$+chr$(c(x))+chr$(c(x+1)): x=x+2: goto 390
620 z=0:for y=1 to n: if c(x)<=t(y) then z=t(y):y=n
630 next y: if c(x)<>z then 650
640 do:l$=l$+chr$(c(x)):x=x+1:loop untilc(x-1)=0:got
o 320
650 if c(x)=34 then do:l$=l$+chr$(c(x)):x=x+1:loop u
ntil c(x)=34 or c(x)=0
660 l$=l$+chr$(c(x)): if c(x)>0 then x=x+1: goto 390
:else 320
670 print"[DOWN][RVON]disk error": print ds$: close
5: close 6: end
680 print#6,chr$(0);chr$(0);: close 5: close 6
690 print"[UP]";spc(10);"( done )[DOWN]"
700 rem * tokens that don't allow split
710 rem * end,goto,run,return,rem,new
720 rem * go (to),resume,exit,monitor
730 data 10,128,137,138,142,143,162,203,214,237,250
```