"Putting on The Squeeze"
by David Goben
28 Monticello St, Willimantic, CT 06226-1325

SYSTEM REQUIREMENTS:
Model 4/4P/4D
1-Disk, 64K
TRSDOS/LS-DOS 6 BASIC

If you would like to make your Model 4 BASIC programs run faster, more compact, or easier to edit, then this program is for you. It's called PACK4, and it will quickly become one of the most valuable utilities in your BASIC programming toolbox.

Most serious BASIC programmers keep a set of program development utilities that make writing their programs much easier and also make them operate at peak efficiency. Probably the most famous of these utilities is a renumbering program, which is by now as common-place as BASIC itself, and is supplied with most BASIC interpreter packages. Without it, inserting large sections of new information between program lines can sometimes become quite an arduous task. Other popular utilities include programs to move or copy a block of lines from one place to another, or a cross-reference utility to keep track of line references, variables, and BASIC keywords. The Enhanced BASIC included with LSI's LS-DOS 6.3 contains all of these valuable features. But it lacks the much-needed compression, expansion, and packing utilities. PACK4 fills this void.

USING PACK4

To use PACK4/CMD, you must work from the DOS level (TRSDOS Ready). You must also have a BASIC file to work with.

For example, if you saved a program to disk as SAMPLE/BAS via the BASIC command SAVE"SAMPLE/BAS", you could pack it using the DOS command PACK4 SAMPLE/BAS TEST/BAS. Where PACK4 is the main command, SAMPLE/BAS is the file that you wish to have packed, and TEST/BAS is the file that you wish the new format to be written to. You could also use PACK4 SAMPLE/BAS SAMPLE/BAS to write the new format over the top of the source file, but you may want to make a backup copy of SAMPLE/BAS first. You can also add drive specifications and passwords if you desire.

When you press the ENTER key, a sign-on message will be displayed. PACK4 will load SAMPLE/BAS into memory and will display its current byte-length and processing will begin. PACK4 takes approximately 1.2 seconds to process each 1K of program data. When processing is complete a new byte-length will be displayed and the destination file TEST/BAS is written. If all goes well a report of success is given and control is returned to DOS.

You may want to go into BASIC and examine the new format of TEST/BAS against that of the original SAMPLE/BAS program.

NOTE: DO NOT use PACK4 on a program that has been saved in ASCII format, it won't work!

USING PARAMETERS WITH PACK4

You can also use parameters to tell PACK4 exactly what you want done. The parameters are:

P = PACK. This is the default condition, but must be included if you also specify another parameter, and desire the packing feature. Packing combines multiple statements onto single lines. All remarks are removed unless the "*" parameter is also specified. Parameter "C" is assumed.

U = UNPACK (expand). This feature is the opposite of PACK in that it expands the program so that statements occupy individual lines. All remarks are removed unless the "*" parameter is also specified. Parameter "C" is assumed.

C = COMPRESS. This removes extra spaces and linefeeds (except inside strings and remarks) from a program. This is a default condition of the "P", "U" and "*" parameters. All remarks will be removed. This parameter is usually used by itself.

* = Retain remarks. This causes all remarks lines to remain intact. Parameter "C" is assumed.

L=nnnn = Set starting line number. Parameters "P" and "U" will renumber a program starting the new file with line 10 and an increment of 10. This parameter allows you to change the starting line number to a value other than 10, such as 100.

I=nnnn = Set increment. This allows you to change the line increment during "P" or "U" operations to a value other than 10.

Parameters are entered inside parenthesis, following the destination file, in the format PACK4 sourcefile destfile (parameters). Note that all shown spaces are required. Parameters may OPTIONALLY be separated by spaces and/or commas.

If no parameters are given, then "P" (pack) is assumed.

SAMPLE PARAMETERS

(P*)	Compress and pack a file, and retain all remarks lines.
(*)	Compress a file and retain all remarks lines (no pack or unpack).
(U)	Unpack a program and remove all remarks.
(C)	Compress a program and remove remarks (no pack or unpack).
(P,L=100)	Compress and pack a file, remove all remarks, and start renumbering with line 100.
(U,I=1,L=1,*)	Compress and unpack a program, retain remarks, and start line renumbering from line 1, with an increment of 1.

Note that compression is a condition that is used at all times. The "C" parameter was provided so that you could compress a program and also remove remarks without PACK4 re-arranging the program structure (combining or separating program statements). Including the "C" parameter with other parameters is redundant, but is acceptable:

Parameters can be entered in any order.

If by chance you use the "P" and "U" parameters together, "P" will take precedence.

PACK4 DEFAULT FEATURES

PACK4 features program compression as a default service. It does this even during expansion, since the Model 4 BASIC listing feature will automatically display spaces in

the proper places in the monitor/printer listing, thus maintaining a line's readability. The Model I and III compressions via the CMD"C" option will not process listings this way, but will appear to merge one keyword right into the next one.

PACK4 will remove all remarks that are located at the end of lines that contain executable statements. This was done to allow for possible merging of a statement in the next line during a pack operation. If these type of remarks were retained, then the packing feature would have been severely hindered, since executable statements cannot follow a comment on a line.

PACK4 will also, unless told otherwise, remove all remarks lines. If you choose to retain the remarks lines in your programs, then any occurrence of the apostrophy-type remark ("'") will be converted to a REM-type remark. This is done because a REM token will take up only one byte of memory space, whereas the apostrophy actually uses a 3-byte sequence of tokens: a colon (":"), a REM token, and an apostrophy token. However, during a listing only the apostrophy will be displayed (much like the ELSE token mentioned earlier). Changing it to an individual REM token will save the program 2 bytes for each remark line encountered.

Whether you choose to keep or remove your remarks lines, PACK4 will also employ another default feature concerning remarks: it will repoint all line references (from GOTOs, GOSUBs, etc.) that point to a remark line down to an actual executable line. For example, in the following sample 3-line program,

```
10 REM TEST PROGRAM
20 REM TEST TRANSFER
30 GOTO 10
```

the GOTO 10 in line 30 will be changed to a GOTO 30 by PACK4, even if you choose to retain remarks and not alter the statement arrangement in your program. This way if you or PACK4 later delete remark lines 10 and 20, the program will still operate correctly. For many people, this feature alone is worth the trouble of obtaining the entire PACK4 package.

Another default feature during packing is the removal of GOTO tokens after a THEN token. In the statement IF A=B THEN GOTO 40, the GOTO token is not actually necessary, and so the statement will be reduced to IF A=B THEN 40. During an expansion (unpack), the opposite is given: if a line reference follows a THEN token, then a GOTO token will be automatically inserted before the line number if a GOTO token does not already exist there.

Automatic line renumbering after an expansion or pack is also a default feature of PACK4. This way you do not have to worry about insuring that there is an appropriate increment between each line. For example, an expansion of the lines:

```
1 CLS:PRINT"HELLO"
2 END
```

would be expanded to:

```
1 CLS
2 PRINT"HELLO"
3 END
```

The rest of this document describes the features of PACK4 in more detail and the techniques used.

PROGRAM COMPRESSION

Program compression is the process of removing unnecessary remarks, spaces and linefeeds from a program (the same procedure as CMD"C" on most Model I and III disk BASICs). Everyone who has ever written a workable program in BASIC on the Model 4 knows that you must separate all keywords and variables with a space or a syntax-allowed special character. This allows the BASIC interpreter to recognize the individual keywords that it will need to properly execute the program. What many people do not realize is that BASIC stores each of these keywords as one- or two-byte tokens in memory to save both space and execution time. An aspect of this process that most people are not aware of is that the extra spaces are no longer necessary after BASIC has tokenized the program, but are NOT removed during the tokenization process (contrary to what many people have previously claimed). But if you had the capability, they could be removed without affecting the program's operation.

Once these spaces have been removed (and you may be surprised at how many of these spaces actually exist in any given program), your program will execute faster since the interpreter would not have to waste its time skipping over them every time it runs into one. But even with all of these spaces removed, if you afterward listed a line, BASIC would automatically add any needed spaces into the LISTed lines sent to the monitor or printer for the sake of maintaining program readability (The actual program line itself will not be affected by these display-only modifications).

For example, the program statement PRINT ERL is stored in memory as 91 20 D7 in hex (hexidecimal) notation, where 91 is the token for PRINT, 20 is a SPACE character, and D7 is the ERL token. If the SPACE (20 hex) was removed from the statement, creating a shorter 91 D7 byte sequence, a listing of the statement will still SHOW a space between PRINT and ERL, even though the actual line no longer contains it.

There is only one exception to this compression process, which PACK4 will take care of effortlessly. This has to do with the ELSE statement. When ELSE is tokenized, it is stored in tokenized format with a leading statement separator (a colon ":"), which is invisible during a program listing. The problem is that if the space in front of the ELSE token (technically in front of the leading colon) is removed, and the previous 'token' was a BASIC keyword or a non-typed variable (such as variable XY as opposed to XY$), then the previous token and ELSE will APPEAR to be merged. A case in point is the program line IF LP THEN LPRINT XY ELSE PRINT XY. If the space between the variable "XY" and (:)ELSE is removed, a listing will show these tokens as XYELSE, even though the program will still execute properly since internally a colon still exists between XY and ELSE. However, if you edit the line (a literal modification of its listed, ASCII format), or ASCII-save the line (a literal disk-save of its listed, ASCII format) and then reload it, a syntax error will be reported during a subsequent run because XYELSE will have been reinterpreted as a single token (a variable in this case) rather than a separate variable and a token (The BASIC interpreter LITERALLY translates the ASCII file when loading it).

To allow for this possibility, PACK4 will insure that a space will always precede any (:)ELSE token.

Some programmers have claimed that a space is mandatory after the "AS" token in a FIELD statement, such as in the statement FIELD 1,2 AS A$. This was true for earlier versions of BASIC, like that for the Model I and III, but not with the Model 4 version. This version

will automatically take this compression factor into account, and so the space requirement is no longer a necessary rule. Therefore compressing the statement to FIELD 1,2ASA$ will NOT interpret ASA$ as a single variable, but as "A$" preceded by the "AS" token. Of course this allowance is only provided for in the FIELD statement, and nowhere else.

PROGRAM EXPANSION

Program expansion is an often highly desired capability, especially in the course of the debugging process. This feature involves placing each individual statement onto a separate line. For example, the line 10 CLS:PRINT"HELLO":END, would be expanded (and accordingly renumbered) by PACK4 into:

```
10 CLS
20 PRINT"HELLO"
30 END
```

By expanding a program and then running it, if an error occurs during the course of the program's operation, the programmer can instantly narrow the problem down to the program statement on the reported error line.

Another advantage of program expansion is when you need to insert additional program statements in the middle of a multi-statement line (a line with more than one command on it). By first expanding the program, you can separate each statement onto separate lines and therefore place your new statements in their proper places with ease and without having to worry about moving other statements to other lines to make room for the new logic. Once the new insertions are completed, you can pack the program back together by letting PACK4 do the work for you by utilizing its packing feature.

An exception to the expansion process, which PACK4 also handles effortlessly, is when program logic requires multiple statements to remain on an individual line, such as is the case for IF-THEN and IF-THEN-ELSE constructs. And example is with the line: 10 IF A=B THEN X=Y:Z=T ELSE Z=Y:X=T. Placing each of these statements onto separate lines will cause chaos without the insertion of additional program logic, such as GOTOs after the Z=T and X=T statements, so that they will leapfrog to the correct trailing statements for the continuation of the proper logic. It is usually best to leave this type of structure intact and unaltered.

PROGRAM PACKING

Program packing is the opposite of expansion. It is also the main feature of the package, of which compression and expansion are only consequences (a packing procedure first involves expanding the program, and then packing and compressing it back together).

Program packing can be exemplified thus: the three lines unpacked previously in the expansion example will once again be combined into one line by the packing feature. What a packer does is to basically join each consecutive program statement together with the previous statement until one of four situations are encountered:

1) The line length limit is reached. If the appending of a new statement will cause the 255-byte line length (expanded, listing-type display length) of the current line to be exceeded, then the building of the current line will be ended and a new line will be initiated.

2) The new statement's line number is referenced by a GOTO, GOSUB, RESUME, etc. When a line such as this is encountered, then a new line is automatically begun to prevent the program's logic-flow from being violated.

3) An IF-THEN program construct is encountered. If this is found, the line will, if it will fit, be the final information appended onto the current line. If it will not fit, then the current line will be closed up and the new IF-THEN data will occupy its own line following it.

4) The current line is a remark line. A new line will immediately be initiated which will contain the remark. However, if the remarks are to be removed, then the remark line will be completely ignored.

A situation that would normally spell trouble for a packer is when a line ends with a string that does not contain a closing quotation mark. This is normally a legal and acceptable practice when the string is the last statement on a line. However, this would wreck havoc on a program if subsequent statements are later appended to the line. Therefore PACK4 will insure that all quoted strings are terminated with a quote by automatically appending one to any that do not already contain one.

TECHNICAL NOTES

The compression and packing process of PACK4 will save you hundreds of bytes on even relatively short programs, and thousands on large ones. For example, if you have a BASIC program that is too large to run under LS-DOS 6.3 BASIC (this BASIC's buffer is smaller than that of TRSDOS 6.2's), but will run under TRSDOS 6.2 BASIC, then PACK4 will make the program small enough to operate in it.

When a program is packed, it is FIRST unpacked (just like using the "U" parameter). It is also renumbered before the actual packing process is begun, to insure that all new lines will contain valid line numbers. It is then re-packed into a tighter space and renumbered again.

When a program is renumbered, if the new line number exceeds BASIC upper number limit of 65529, then this error will be reported, and a program abort will occur.

When you use PACK4, the source BASIC program cannot be an ASCII-saved file (saved with the ",A" option) or a protected-saved file (saved with the ",P" option), but must be a file saved by the standard BASIC SAVE procedure.

Please note that if you ASCII-save a previously packed file, the spaces that will be displayed during a LIST from DOS will also be sent to the disk file (an ASCII-save is simply a normal listing routed to a disk file). Thus when you reload it, as the BASIC interpreter re-tokenizes the file from ASCII into a workable BASIC program, the extra spaces will also be inserted into the new tokenized code. This means you must re--pack the file if you do not want the spaces.

If you edit a line, the spaces that will be displayed in a listing of the edited line will also be inserted into the actual line once the edit is completed, since an edited line is reinterpreted just like a line from an ASCII-saved file; as though it were freshly keyed in.

David Goben is a programming consultant, and an associate editor to 80 Micro.