Create Smaller, Smarter QuickB[illegible]IC Programs

PC World's programming whiz and editor of Star-Dot-Star demonstrates some state-of-the-art tricks with QuickBASIC.

What you need: QuickBASIC 4.5

users have long relied on BASIC's straightforward syntax to obtain quick results. But basic BASIC is short on structured programming capabilities, which can conserve memory and make program logic easy to follow. Enter Microsoft *QuickBASIC 4.5*.

QuickBASIC's structured programming aids and built-in intelligence form a software development tool that rivals the powers of C and Pascal. To highlight some of these new abilities, let's explore a sample utility that reformats text into newsletter-style columns.

Text In, Software Out

First, enter COLUMNS.BAS (see **LISTING 1**) into *QuickBASIC*'s syntax-checking editor. As you type in the function and submodules, *QuickBASIC* automatically opens a new window and generates the DECLARE statements at the top of the program listing, so you can skip typing them in. Press <F2> to move between program modules.

Next, compile the program using the Run menu's Make EXE File command, exit *QuickBASIC*, then run the program from the DOS prompt. Or, simply run the program from within *QuickBASIC* by pressing <Shift>-<F5>. Once running, COLUMNS prompts you to enter the name of a text file to process (for the purposes of this tutorial, an unformatted ASCII file of several 20-character lines is best). At the subsequent prompts, press <Enter> to display output on the screen, then specify 3 columns, 15 lines per column, 25 characters per column, and 4 blank lines between pages. You should see your formatted text zip by at breakneck speed. To save the output to disk, rerun COLUMNS and supply an output file name (or LPT1:, to print the listing) at the second prompt.

COLUMNS.BAS imposes no limits on file length, number of columns, or spacing. Be prepared to experiment, though, as some values can produce strange results. For example, a column width narrower than the longest text-file line creates gibberish. To be safe, always keep a backup copy of your original text.

```
DECLARE SUB ReadLines (MaxLines%, NumLines%)
DECLARE SUB WriteLines (NumLines%)
DECLARE FUNCTION GetOptions% ()
DECLARE SUB OpenFiles ()

COMMON SHARED NumColumns AS INTEGER
COMMON SHARED LinesPerColumn AS INTEGER
COMMON SHARED ColumnWidth AS INTEGER
COMMON SHARED BlankLines AS INTEGER

CALL OpenFiles
MaxLines% = GetOptions%

DIM SHARED TextLines$(1 TO MaxLines%)

DO UNTIL EOF(1)
   CALL ReadLines(MaxLines%, NumLines%)
   CALL WriteLines(NumLines%)
LOOP

CLOSE #1          ' Close input file
CLOSE #2          ' Close output file
END               ' End of main module

FUNCTION GetOptions%
   INPUT "Number of columns"; NumColumns
   INPUT "Number of lines per column"; LinesPerColumn
   INPUT "Number of characters per column"; ColumnWidth
   INPUT "Number of blank lines between pages"; BlankLines
   GetOptions% = NumColumns * LinesPerColumn
END FUNCTION

SUB OpenFiles
   INPUT "Read from what file"; InFileName$
   IF InFileName$ = "" THEN END
   OPEN InFileName$ FOR INPUT AS #1
   INPUT "Write to what file (Enter=screen)"; OutFileName$
   IF OutFileName$ = "" THEN OutFileName$ = "scrn:"
   OPEN OutFileName$ FOR OUTPUT AS #2
END SUB

SUB ReadLines (MaxLines%, NumLines%)
' Read up to MaxLines strings or to end of file
   NumLines% = 0
   DO UNTIL EOF(1) OR NumLines% = MaxLines%
      NumLines% = NumLines% + 1
      LINE INPUT #1, TextLines$(NumLines%)
```

(continues)

LISTING 1: COLUMNS.BAS, a text-formatting utility, demonstrates many structured programming techniques now available to QuickBASIC programmers.

Inside COLUMNS.BAS

COLUMNS.BAS exploits four advanced *QuickBASIC* features: the CALL statement used with SUB procedures, structured DO loops, local variables, and FUNCTION procedures.

The program's 18-line main module (from the first DECLARE SUB statement to the 'End of main module' comment) contains CALLs to three subprograms—OpenFiles, ReadLines, and WriteLines. CALL statements transfer control to subprograms, optionally passing parameter values (in parentheses, as shown in the calls to ReadLines and WriteLines). This ability to pass variables to subprograms is one reason for *QuickBASIC*'s newfound popularity among developers.

Structured loops make program logic easier to understand

```
    LOOP
  ' Erase any left over strings in TextLines$()
    FOR i% = NumLines% + 1 TO MaxLines%
       TextLines$(i%) = ""
    NEXT i%
 END SUB

 SUB WriteLines (NumLines%)
 ' Write multiple-column text
    FOR i% = 1 TO LinesPerColumn
       FOR j% = 0 TO NumColumns - 1
          s$ = TextLines$(i% + j% * LinesPerColumn)
          PRINT #2, s$;
          IF j% < NumColumns - 1 THEN
             PRINT #2, SPC(ColumnWidth - LEN(s$));
          END IF
       NEXT j%
       PRINT #2, ' Start new output line
    NEXT i%
 ' Add blank lines between pages
    FOR i% = 1 TO BlankLines
       PRINT #2,
    NEXT i%
 END SUB
```

LISTING 1: ***(continued)***

by moving the bulk of the code into subprograms. For instance, the DO loop in the main module repeatedly calls ReadLines and WriteLines until it reaches the end of the input text file. The DO loop in ReadLines loads up to the specified maximum number of strings (MaxLines%) into an array. Such statements are far simpler to understand than nonstructured, GOTO-ridden BASIC code.

QuickBASIC's use of local variables helps eliminate bugs, including those caused by variable-name conflict, a common logic error. Because local variables are "visible" only to statements within the subprogram where the variables are defined, conflicts with identically named variables elsewhere can't occur. And local variables conserve memory because values in multiple modules can share the same RAM locations.

The variables i% and j% in WriteLines are local and therefore invisible to the rest of COLUMNS.BAS. Similarly, MaxLines% and NumLines% are local to the main module. The values of these variables are available to ReadLines because the variable names are passed to the submodule as parameters by the CALL ReadLines statement. Only global variables, which are declared with COMMON SHARED statements, are visible to every program module and function.

FUNCTION procedures are subprograms that return a value when called; they work similarly to the way variables do in BASIC expressions. For example, the GetOptions% function returns the number of strings required for one output page, a value the program's main module uses to dimension string array TextLines$.

Of course, there's more to *QuickBASIC* than one article can cover. But even a few highlights should convince anyone who wants quick, clean code to give *QuickBASIC* a closer look.
—Tom Swan

Alfred Glossbrenner's many books cover communications and personal computing. His latest is Alfred Glossbrenner's Master Guide to Free Software for IBMs and Compatibles *(St. Martin's Press, New York, 1989). Richard Jantz wrote* Ventura Publisher 2.0 for the IBM PC: Mastering Desktop Publishing *(John Wiley & Sons, New York, 1989) and* The Complete Scanner Handbook *(Peachpit Press, Berkeley, California, 1989). Tom Swan is a contributing editor for* PC World *and the author of* Mastering Turbo Pascal 4.0 *(Howard W. Sams & Co., Indianapolis, Indiana, 1988) and* Mastering Turbo Assembler *(Howard W. Sams & Co., 1989).* ●