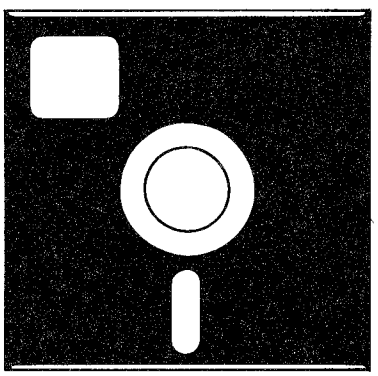
Gee, Pascal's not bad...

In recent years, EF Schumacher's statement that 'small is beautiful' has resonated throughout the computer world. In the third quarter of the seventies, a rash of Tiny BASICs cropped up in the US and, recently, many have been turning their eyes to useful tiny versions of other languages. Les Bell reports on an Australian 'Tiny' Pascal...

your computer



SOFTWARE REVIEW

THERE IS A great deal to be said for languages which are small enough to do sensible things in an efficient and elegant way. Just as the energy crisis convinced Americans that compact cars aren't so bad after all, so the microcomputer has led to a realisation amongst computer professionals that big isn't always better.

While the earliest BASIC interpreters for microcomputers were small, they rapidly grew into the Microsoft and similar software packages we know today. Other language systems followed a similar trend, and today's FORTRAN, Pascal, COBOL and C compilers are as big as, if not bigger than, many minicomputer packages.

On the other hand, one area which has given a lot of people a lot of pleasure and education is the implementation of so-called 'tiny' languages. Doctor Dobbs' Journal started it all in 1976 with its series of Tiny BASIC articles and listings (all around 2K in size), and subsequently tiny packages for other languages, such as Pascal and C, have appeared.

Gammon & Gobbett's G-Pascal follows in the tradition of being an interesting and extremely useable small language package, in this case for the Apple II computer. Its originator, Nick Gammon, was inspired by the original Tiny Pascal published in Byte magazine, but not having a North Star computer (for which it was written), he decided to use the articles as the basis for his own Apple-based Pascal compiler.

The result is a compiler which implements a useful subset of the Pascal language. While it does not have all the bells and whistles of a full Pascal compiler (user-defined types, for example, are missing), it is still a good tool for writing system utilities.

Simplicity Itself

G-Pascal is simplicity itself to install, particularly if you have an Apple II with 48K and DOS 3.3 — then it's just a matter of inserting the disk and booting. RAMCard users will find a larger version of the Pascal on the disk, and this can be set up to automatically load and run by making a loader program the 'HELLO' program on the disk.

The system is also available on cassette, and by changing only one location in memory, can be switched between cassette and DOS I/O. G-Pascal actually consists of a compiler, p-code interpreter and a text editor, all combined into one program which occupies 12K of memory.

Once the program has loaded, it will ask the user 'DISPLAY LOWER CASE? Y/N'. This allows use of lower case adapters. After this, the system is operational, and will sign on with a short copyright message and the main menu.

The main menu offers the user access to a number of subsystems, in much the same way as Apple Pascal 'proper':

(E)dit, (C)ompile, (D)ebug, (F)iles,
(R)un, (S)yntax, (T)race, (Q)uit ?

The Files subsystem allows source files to be loaded, saved or merged. It also offers control of the printer, so that a session can be recorded on hard copy, and allows DOS commands to be executed.

The editor is a fairly simple-minded but effective line-oriented type. Its commands are Compile, Delete line number range, Insert line number, List line number range, Modify line number range, Quit and Syntax, where line number is what you would expect and line number range is a pair of line numbers.

The compile option is the same as entering (C) from the main menu. Delete removes one or more lines from your program, with a query for more than five lines to prevent accidental grief. Insert adds lines after the line specified.

List works like you would expect, with the ability to 'freeze' and release the listing by hitting the space bar. Modify works by listing the lines requested, then deleting them and entering insert mode to allow replacement. Minor changes can be made by using the Apple's cursor control codes to edit erroneous lines.

Quit returns the user to the main menu, while the Syntax option works the same way as from the main menu.

The compiler first asks the user if he/she wants a p-code listing; normally, this is not required. Next it asks if a listing is required. After this, it will go ahead and compile the program.

If the user does not want to generate code, but simply check for errors (useful when first writing a program) he can specify the 'Syntax' command as an alternative. This is identical to the compiler, but generates no p-codes.

Catching Sloppy Programming

If an error is found, the compiler will halt with an arrow pointing to the symbol being processed when the error was discovered, the word 'Error' and an error message. It then returns you to the editor to fix up your sloppy programming!

The output of the compiler is p-code (pseudo-code) which is then interpreted at run time. Each p-code is up to four bytes long, and the manual contains an explanation of what the p-codes mean. This, together with a trace mode of execution and a listing which has p-code addresses, means that it is possible to track down particularly recalcitrant bugs without too much difficulty.

The run-time interpreter also supports a debug mode, which shows the p-code being executed and its operands, the top eight bytes of the stack and the stack frame linkage data used to manage the data on the stack as procedures are called and exited.

G-Pascal is a subset of standard Pascal. While it omits some of the most powerful features of Pascal (for example, run-time range error checking), what remains is still powerful enough to be useful and to demonstrate the principles of structured program design.

Integers are stored as three-byte signed values and range from -8388608 to 8388607; constants may be expressed in either decimal or hexadecimal format. Char variables occupy a single byte, so that arrays of char are much more space-efficient than arrays of int. Other data types are not supported, nor is the **type** statement.

However, with some ingenuity, types can be simulated, as in the following example.

Reprinted with permission *Your Computer* magazine, August 1982.

In full Pascal:

```
type
  colour = (red, green, blue);

var
  fred : colour;

begin
  fred := green
end.
```

In G-Pascal:

```
const
  red = 0; green = 1; blue = 2;

var
  fred : integer;

begin
  fred := green
end.
```

Both of these will have the same effect. String constants are supported, up to three letters in length, basically by storing them in an int variable. Of course, arrays of char are possible, and these would be the best way of dealing with strings where more than three letters are required.

Arrays can only be single-dimensional, and there is no run-time checking of subscript bounds. That's not really as serious a problem as it sounds, as it is possible to write checks into a program (good defensive programming!), and there are ways to simulate two dimensional arrays when only a single dimension is available (a computer's memory is only a single dimension anyway; all languages have to 'fake' multiple dimensions).

Identifiers can be any length, with all characters significant. One up for sensible programming. At last I can have a variable called hereisalongvariable and know the compiler knows it's different from hereisalongvariabul!

In order to support recursion, G-Pascal passes all arguments by value on the stack. This has the side effect of preventing a procedure returning a result by altering one of its arguments, as is possible in standard Pascal. On the other hand, it allows recursive calling of functions indefinitely or at least until the stack overflows.

G-Pascal supports a default extension to the case statement, through the use of an **else** clause at the end of the list of case selectors.

A Useful Extension

A useful extension is the MEM array, which is a pre-declared array of memory locations starting at zero. By using this array in assignment statements, peeking and poking are effectively emulated. A similar array, MEMC, is an array of char.

G-Pascal can read and write disk files using the same conventions as Applesoft BASIC and DOS. For example, the statement *WRITE (4, "CATALOG", 13);* will do a catalogue of the disk.

Other extensions include the predeclared integer RANDOM, the CURSOR statement, which is equivalent to the Apple Pascal GoToXY procedure, and the function ADDRESS(identifier) which returns the address of a variable.

G-Pascal also includes three built-in procedures for high-res graphics: HGR, which sets graphics mode and clears the screen, HPLOT(colour, column, row), which plots a point, and TEXT, which returns to text mode. The low-resolution graphics modes in the Apple monitor can be called from G-Pascal to do low-res plotting.

There is also a built-in procedure MUSIC(pitch, duration) which will generate tones.

The G-Pascal manual gives some useful addresses inside the system which allow the user to customise it. For example, normally the compiler generates code starting immediately after the source code in memory. This makes for quick recompilation when bugs are found.

With large programs, however, there might not be enough memory left over after the source has been loaded for the program to be compiled into. By changing two memory locations, the compiler can be set up to allow the p-code to overlay the source, thus permitting much larger programs to be compiled.

Similarly, the symbol table size can be altered, the right and left brackets changed from '( )' to '[ ]', the printer slot altered and so on.

Software Report Card

Unit:	G-Pascal
Made By:	Gammon & Gobbett Computer Services
Useful for:	System utilities, games, education, Pascal
Hardware Reqd:	Apple II, disk drive

Ratings:	**excellent**	**very good**	**good**	**poor**
Documentation:		✓		
Ease of Use:		✓		
Speed:	✓			
Functionality		✓		
Support:		✓		
Value-for-money:	✓			

Extras Included:	N/A
Price:	$40
Review Unit from:	Gammon & Gobbett Computer Services, PO Box 124, Ivanhoe, VIC 3079

The manual is well-written and informative, with plenty of examples of programs. Special sections outline the unique G-Pascal statements such as the high-res graphics functions, and there is particularly useful information on converting from other Pascals and debugging.

The reference information towards the back of the manual covers configuring the system, machine language subroutines and meanings of p-codes.

Amazing Value

G-Pascal is a compact but particularly potent package. It is especially useful for writing system utilities, as well as games, where its performance would be considerably faster than BASIC.

As an example of the kind of work that can be done in G-Pascal, Nick Gammon has written a complete Adventure game, and we hope to feature this in a future issue. After seeing the program, we can only say that once you know how it's done, you'll be surprised how simple it is (the same could not be said of a BASIC Adventure, however!).

We played around extensively with G-Pascal and found no bugs — however, Gammon & Gobbett did issue a bug fix sheet for early versions of G-Pascal (the fixes have been applied to later versions).

Not only does G-Pascal work well, it also represents amazing value for money, at only $40 for the diskette and 58-page user manual. Contact Gammon & Gobbett Computer Services, PO Box 124, Ivanhoe VIC 3079. □