Along with pointing out the differences between Tiny BASIC and standard BASIC, Tom offers here some comments and opinions on BASIC and structured programming. Interestingly, his manuscript is one of the few we've received which was prepared using a text editor (a Model 37 TTY driven by a COSMAC 1802 microprocessor). It would seem that more of us (including myself) should be at this stage by now. — John.

Tiny Basic

... a mini-language for your micro

Tom Pittman
PO Box 23189
San Jose CA 95153

If you have an Altair or IMSAI computer or any 8080-based system, you have your choice of several versions of BASIC. There are rumors of BASIC for 6800 and 6502 within the next few months. But these require memory — probably more than you have with your low budget machine.

The alternative is *Tiny BASIC*. The language is a stripped down version of regular BASIC, with integer variables only — no strings, no arrays, and a limited set of statement types. It was first proposed by Bob Albrecht, the "dragon" of Peoples Computer Company (PCC) in Menlo Park, as a language for teaching programming to children. The PCC newspaper ran a series of articles (largely written by Dennis Allison) entitled "Build Your Own BASIC," suggesting how Tiny BASIC might be implemented in a microprocessor. The important portions of these articles have been reprinted in Dr. Dobb's *Journal of Computer Calisthenics and Orthodontia*, published by PCC and available in most computer stores.

BASIC

Before we get into Tiny BASIC, let us look at high level languages in general and BASIC in particular.

When you program in machine language, each command, or statement, represents one operation from the machine's point of view. When we think of a single concept like, "A is the sum of B and C," a machine language program to perform this operation may take several operations, such as:

```
LDA  B
LDA  C
STO  A
```

A high level language, on the other hand, lets you put a single human idea into a single program statement, for instance:

```
LET A = B + C
```

BASIC is one of a class of "algebraic" languages in that it permits the representation of algebraic formulae as part of the language. Other languages in this class are FORTRAN and ALGOL. COBOL does not generally fall in this class (except for the "super" versions).

Of critical importance to all algebraic languages is the concept of an expression. An expression is the programming language notation for what we might think of as "the right-hand side of a formula." Alternatively, we can think of an expression as "a way of expressing the value of some number which the computer is to compute."

An expression may consist of a single number, a single variable name (all variables are referred to by name in high level languages), a single function call (discussed in detail later), or some combination of these, separated by operators and possibly grouped by parentheses. For this discussion, when we refer to an operator, we mean one of the four functions found on a cheap pocket calculator: addition symbolized by " + "; subtraction by " - "; multiplication by " * " (we do not use "X" because that would be confused with the name of the variable "X"); and division by " / ". (The usual symbol for division does not appear on most typewriter and computer keyboards.) Thus,

$$\frac{A-B}{C-D}$$

becomes, in computerese,

```
(A - B) / (C - D)
```

Here the parentheses are used to indicate priority of operations. Normally multiplication and division are performed first, then addition and subtraction. Without the parentheses the expression,

$$\frac{A-B}{C-D}$$

would be understood by the high level language as,

$$a - \frac{B}{C} - d$$

Photo courtesy of Electronic Product Associates, Inc., 1157 Vega Street, San Diego CA 92110.

which is not the same at all.

In BASIC, when an expression is encountered, it is evaluated. That is, the values of the variables are fetched, the numbers are converted (if necessary), the functions are called, and the operations are performed. The evaluation of an expression always results in a number which is defined to be the value of that expression.

The first example which we discussed showed a simple BASIC statement,

```
LET A=B+C
```

This is called an assignment statement, because it assigns the value of the expression "B + C" to the variable A. All algebraic high level languages have some form of assignment statement. They are characterized by the fact that when the computer processes an assignment statement, a single named variable is given a new value. The new value may not necessarily be different from the old; for example:

```
LET A=A
```

This is also a valid assignment statement, even though nothing changes. Assignment statements are also used to put initial values into variables, for instance:

```
LET F=3
```

Control Structures

One of the important characteristics distinguishing different high level languages is the control structure afforded to the programmer. The control structure is determined by the various permitted control statements, which alter the flow of program execution. Normally program execution advances from statement to statement in sequence, although there are however, circumstances in which this sequence is altered. The most common control structure allows one set of operations to be performed if a certain condition is true, and another, if it is false. In "structured programming" this is referred to as the "IF . . . THEN . . . ELSE" construct; its general form is "IF condition is true, THEN do something, ELSE do some other thing." The full generality of this control structure is not directly available in BASIC, but, as we shall see, this is only a minor inconvenience.

Standard BASIC uses the IF . . . THEN construct, and makes it work something like a conditional GOTO:

```
IF A>3 THEN 120
```

If the value of the variable A is greater than three, then (GOTO) line 120, otherwise continue with the next statement in sequence. Actually, the condition to be tested consists of a comparison between two expressions, using any of the comparison operators which are given in Fig. 1.

In each case, if the comparison of the two expressions evaluates as true, the implied GOTO is taken; otherwise the next statement in sequence is executed. In Tiny BASIC the syntax is slightly different. Instead of a statement number, a whole statement follows the THEN part of the IF . . . THEN. The comparison above, in Tiny BASIC, would be:

```
IF A>3 THEN GOTO 120
```

But we could also validly write:

```
IF A<=3 THEN LET A=A+10
```

or some such. Note that this is *not* valid in standard BASIC.

The GOTO construct has been the subject of controversy in the last few years. A strong case has been made for "GOTO-less programming" which uses only certain other control structures to achieve structured programs which are more readable and less

=	Equality (the comparison is true if the two expressions are equal)
>	Greater than
<	Less than
< =	Less or Equal (not Greater)
> =	Greater or Equal
< >	Not Equal

Fig. 1. Comparison Operators.

prone to errors. I believe that both good and incomprehensible programs are possible regardless of the control structures used or not used, but I seem to be in a minority at this time. Suffice to say that BASIC is not conducive to structured programming in the technical sense of the term.

Standard BASIC has one control structure which has been omitted from Tiny BASIC. This is the FOR . . . NEXT loop. Normally, if a program requires some sequence to be performed thirteen times, the following program steps might be used:

```
10 FOR I=1 TO 13
20 ...
30 NEXT I
```

Statement 20 would be executed 13 times, with the variable I containing successively the values, 1, 2, 3 . . . 12, 13. In Tiny BASIC the same operation is a little more verbose:

```
10 LET I=1
20 ...
30 LET I=I+1
40 IF I<=13 THEN GOTO 20
```

but, as you can see, nothing is lost in program capability.

Data Structures

Standard BASIC also has some data structures which have not been carried over into Tiny BASIC. The only data structure in Tiny BASIC is the integer number, which is further limited to 16 binary bits for a value in the range of -32768 to +32767. Compare this precision with the six digit precision in standard BASIC, which also gives you fractional numbers (sometimes called "floating point"). Regular BASIC allows arrays, or variables with multiple values distinguished by "subscripts," and strings, which are variables with text information for values instead of numbers. We will see presently how these deficiencies in Tiny BASIC can be overcome.

Input/Output

Thus far we have said nothing about input and output, how to see the answers the computer has calculated, or how to put in starting values. These needs are accommodated in BASIC by the PRINT and INPUT statements. Numbers are printed (in decimal, for us humans to read) at the user terminal by the PRINT statement:

```
PRINT A, B + C
```

This prints two numbers; the first is the value of the variable A, and the second is the value of the expression B+C. In general, the PRINT statement evaluates and prints expressions. It is perfectly valid to write

```
PRINT 1, 123, 0-0
```

although we know in advance what will be displayed on the terminal. To make our output more readable, BASIC permits the program to print out text labels on the data.

```
PRINT "THE SUM OF 1 + 2 IS", 3 + 2
```

will display the line:

```
THE SUM OF 1 + 2 IS 5
```

To feed new numbers from the terminal to the program the INPUT statement is used.

```
INPUT A, B, C
```

will request three numbers from the input keyboard. The more popular versions of Tiny BASIC have an extra capability here beyond standard BASIC, in that the operator can type in numbers *and* whole expressions. Thus, if in response to the INPUT request above, the operator types

```
1+2, 3*(4+5), B-A
```

the variable A will receive the value 3, B will receive the value 27, and C will receive the value 24 = 27-3. Therefore, a program in Tiny BASIC, which permits no text strings, can display and accept as input limited text information:

```
10 LET Y=1
20 LET N=0
30 PRINT "PLEASE ANSWER Y OR N";
40 INPUT A
50 IF A=Y THEN GOTO 100
60 IF A=N THEN GOTO 120
70 GOTO 30
```

This little program asks for an answer, which should be either the letter "Y" or the letter "N" (or their equivalents, the numbers 1 or 0, respectively). If the operator types anything else, the request is repeated. Obviously, this technique will not work for something like a person's name where any letters of the alphabet in any sequence must be expected, but it is certainly an improvement over no alphabetic input at all.

A generalized text output capability in Tiny BASIC depends on another characteristic peculiar to Tiny BASIC and not shared by standard. That is the fact that the line number in a GOTO or GOSUB statement is not limited to numbers only, but may itself be any valid expression which evaluates to a line number. The program which is shown in Fig. 2 prints A, B, or C, depending on whether the variable N has the value 1, 2, or 3. Note that, if N is out of range, nothing is printed.

The USR Function

What about the fact that there are no arrays? Let us turn to the USR function for a way to store and retrieve blocks of data. The remarks which follow apply only to my version of Tiny BASIC and are unique in that respect.

The USR function is invoked with one, two, or three arguments (expressions separated by commas within the parentheses). The first (or only) argument is evaluated to the binary address of a machine language subroutine somewhere in the computer memory. The USR function does a machine language subroutine call (JSR instruction) to that address. The user is obliged to be sure that there is in fact a subroutine at that address. If there is not, Tiny BASIC (and thus your computer) will execute whatever is there. The second and third arguments, if present, will be loaded into the CPU registers before jumping to this subroutine. On exit, any answer the subroutine produces may be left in the CPU accumulator, and it becomes the value of the function. Two machine language routines are already provided with the BASIC Interpreter; if S is the address of the beginning of the interpreter,

```
USR(S + 20, M)
```

has as its value the byte stored in memory at the address in the variable M (that is, the contents of the second argument is evaluated to a memory address). Also,

```
USR(S + 24, M, B)
```

stores the low order 8 bits of the value of B into the memory location addressed by M. The return value of this function is meaningless.

Consider the standard BASIC program in Fig. 3(a) to input ten numbers and print the largest as compared to the Tiny BASIC program in Fig. 3(b).

I have used this example for two reasons: First, it shows how the USR function may be used to simulate the operation of arrays. Second, it is typical of many of the applications commonly ad-

```
10 IF N>0 THEN IF N<4 THEN GOSUB 20+(N * 10)
20 RETURN
30 PRINT "A"
35 RETURN
40 PRINT "B"
45 RETURN
50 PRINT "C"
55 RETURN
```

Fig. 2. Program to Print A, B, or C, depending on the value of N.

to argue for arrays; however, neither real nor simulated arrays are required for this program! Here is the same program, with no arrays:

```
10 LET I=1
20 LET L=0
30 INPUT V
40 IF L<V THEN LET L=V
50 LET I=I+1
60 IF I<=10 THEN GOTO 30
90 PRINT L
```

Summary

Tiny BASIC is not a super language. But, it also does not require a super computer to run. I've given here only a cursory examination of the power of Tiny BASIC. A full description of Tiny BASIC may be found in the Itty Bitty Computers *Tiny BASIC User's Manual.* This comes with a hex paper tape of the program and is available for $5 from: Itty Bitty Computers, PO Box 23189, San Jose CA 95153.

There are different versions for each of the following systems, so be sure to specify which system you are running:

M6800 with MIKBUG, EXBUG, or home brew (Executes in 0100-08FF); AMI Proto board (Executes in E000-E7FF); SPHERE (Executes in 0200-09FF); 6502 with KIM, TIM or homebrew (Executes in 0200-0AFF); JOLT (Executes in 1000-18FF); APPLE (Executes in 0300-0BFF); KIM-2 4K RAM (executes in 2000-28FF).

Although few people have paper tape systems, we are unable to provide the program on audio cassette. But if you request it, we will supply a hexadecimal listing of the program instead of tape which you can key in and then can save on cassette for future use.

If you have a small 8080 system, there are several widely differing versions of Tiny BASIC in the public domain. Most of them have been published in Dr. Dobb's Journal, which is $10 per year from: People's Computer Company, PO Box 310, Menlo Park CA 94025. This journal has also published a number of games which run in Tiny BASIC.

One final comment. Tiny BASIC was originally conceived as "free software" by the people at PCC. The 6800 and 6502 versions described in this article are not free; they are proprietary and copyrighted. Software is my only source of income, and, if I cannot make it from programs like Tiny BASIC, I won't write them. Please respect the labor of those of us who are trying to make quality software available to you: pay for the programs you use. ■

```
                          10 LET I=1
10 FOR I=1 TO 10          20 INPUT V
20 INPUT V(I)             25 LET V=USR(S=24,I,V)
30 NEXT I                 30 LET I=I+1
40 LET L=V(1)             35 IF I<=10 THEN GOTO 20
50 FOR I=2 TO 10          40 LET L=USR (S+20, 1)
60 IF L>=V(I) THEN 80     50 LET I=2
70 LET L=V(I)             60 IF L<USR(S+20,I) THEN LET L=USR(S+20,I)
80 NEXT I                 80 LET I=I+1
90 PRINT L                90 PRINT L
```

Fig. 3. Programs to input ten numbers and print the largest. (a) Standard BASIC; (b) Tiny BASIC.

Rainbow Computing, Inc.

(formerly Rainbow Enterprises)
10723 White Oak Avenue
Granada Hills, CA 91344
(213) 360-2171

"The computer store featuring software support."

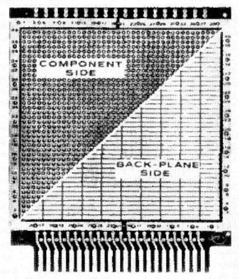
Customized Hardware – Software Packages

Program Conversions & Original Programming

Expert Consulting, Tutoring & Research Services

Authorized Distributor for WAVE MATE Microcomputer Systems

ByteBoard -3



4 common connections per backplane pad MINIMIZES/ELIMINATES interconnection wiring 1/16 FR4-tinned.
BB-3 $15.50 ea. $140/doz.

AFS
A. F. STAHLER CO.
P.O. BOX 354
CUPERTINO CA 95014
(408) 252-4219