DESIGN NOTES FOR TINY BASIC

by Dennis Allison, happy Lady, & friends
(reprinted from *People's Computer Company* Vol. 4, No. 2)

SOME MOTIVATIONS

A lot of people have just gotten into having their own computer. Often they don't know too much about software and particularly systems software, but would like to be able to program in something other than machine language. The TINY BASIC project is aimed at you if you are one of these people. Our goals are very limited--- to provide a minimal BASIC-like language for writing simple programs. Later we may make it more complicated, but now the name of the game is **keep it simple**. That translates to a limited language (no floating point, no sines and cosines, no arrays, etc.) and even this is a pretty difficult undertaking.

Originally we had planned to limit ourselves to the 8080, but with a variety of new machines appearing at very low prices, we have decided to try to make a portable TINY BASIC system even at the cost of some efficiency. Most of the language processor will be written in a pseudo language which is good for writing interpreters like TINY BASIC. This pseudo language (which interprets TINY BASIC) will then itself be implemented interpretively. To implement TINY BASIC on a new machine, one simply writes a simple interpreter for this pseudo language and not a whole interpreter for TINY BASIC.

We'd like this to be a participatory design project. This sequence of design notes follows the project which we are doing here at PCC. There may well be errors in content and concept. If you're making a BASIC along with us, we'd appreciate your help and your corrections.

Incidentally, were we building a production interpreter or compiler, we would probably structure the whole system quite differently. We chose this scheme because it is easy for people to change without access to specialized tools like parser generator programs.

THE TINY BASIC LANGUAGE

There isn't much to it. TINY BASIC looks like BASIC but all variables are integers There are no functions yet (we plan to add RND, TAB, and some others later). Statement numbers must be between 1 and 255 so we can store them in a single byte. LIST only works on the whole program. There is no FOR-NEXT statement. We've tried to simplify the language to the point where it will fit into a very small memory so impecunious. tyros can use the system.

The boxes shown define the language. The guide gives a quick reference to what we will include. The formal grammar defines **exactly** what is a legal TINY BASIC statement. The grammar is important because our interpreter design will be based upon it.

IT'S ALL DONE WITH MIRRORS------ OR HOW TINY BASIC WORKS

All the variables in TINY BASIC: the control information as to which statement is presently being executed and how the next statement is to be found, the return addresses of active GOSUBS-----all this information constitutes the **state** of the TINY BASIC interpreter.

There are several procedures which act upon this state. One procedure knows how to execute any TINY BASIC statement. Given the starting point in memory of a TINY BASIC statement, it will execute it changing the state of the machine as required. For example,

```
100 LET S = A+6 (cr)
```

would change the value of S to the sum of the contents of the variable A and the interger 6, and sets the next line counter to whatever line follows 100, if the line exists.

A second procedure really controls the interpretation process by telling the line interpreter what to do. When TINY BASIC is loaded, this control routine performs some initialization, and then attempts to read a line of information from the console. The characters typed in are saved in a buffer, LBUF. It first checks to see if there is a leading line number. If there is, it incorporates the line into the program by first deleting the line with the same line number (if it is present) then inserting the new line if it is of nonzero length. If there is no line number present, it attempts to execute the line directly. With this strategy, all possible commands, even LIST and CLEAR and RUN are possible inside programs. Suicidal programs are also certainly possible.

TINY BASIC GRAMMAR

The things in **bold face** stand for themselves. The names in lower case represent classes of things. '::=' is read 'is defined as'. The asterisk denotes zero or more occurances of the object to its immediate left. Parenthesis group objects. ε is the empty set. | denotes the alternative (the exclusive-or).

```
line::= number statement (cr) | statement (cr)
statement::= PRINT expr-list
             IF expression relop expression THEN statement
             GOTO expression
             INPUT var-list
             LET var = expression
             GOSUB expression
             RETURN
             CLEAR
             LIST
             RUN
             END
expr-list::= (string | expression) ( , (string | expression) *)
var-list::= var (, var)*
expression::= (+ | - | ε) term ( (+ | -) term)*
term::= factor ( (* | /) factor)*
factor::= var | number | (expression)
var::= A | B |C ... |Y | Z
number::= digit digit*
digit::= 0| 1 | 2 |... | 8 | 9
relop::= < ( > | = | ε) | > ( < | = | ε) | =
```

A **BREAK** from the console will interrupt execution of the program.