V T L – 2

A Very Tiny Language

V T L–2 for the Altair 8800

A VERY TINY LANGUAGE

INTRODUCTION:

VTL-2 is the second *Very Tiny Language* developed for the Altair 8080 Computer system. VTL-2 represents an enormous improvement over the earlier VTL-1 language, and incorporates some thirty additional features. In spite of these enhancements, it still requires only 768 bytes of Read-Only-Memory, and still fits into the three empty PROM sockets already on the 8080 CPU board.

The statements that may be entered as input to the VTL-2 interpreter are of two types:

1. Direct statements, which have no line number, and are executed immediately after they are entered.

2. Program statements, which are used to build a program, and are not executed until the program is run. Program statements must have line-numbers identifying their location in the program.

VTL-2 is simple enough for the beginner to use easily, and yet powerful enough to serve the needs of the most advanced users. The subscripted memory reference commands, and full input-output format control, make VTL-2 a versatile language suitable for solving a wide range of computer problems.

PRELIMINARY CONCEPTS:

Line-numbers must precede each program statement, and must be separated from that statement by a single blank space. These numbers must be in the range of 1-65535. Line-number zero is not permitted. Each line ends with a carriage-return and must be less than 73 characters long.

It is recommended that lines be numbered in steps of ten (10, 20, 30, etc.) so that new statements may be inserted if necessary.

Variables may be represented by any single alphabetic (capitals only) or special character (punctuation mark). `!"#$%&&'()=-+*:;?/>.<,[]` Most of these are available for the user to define as he wishes. A few of the variable names however, have been reserved for special purposes. These so-called "system variables" will be discussed in detail later.

The value assigned to a variable may be either a numeric value in the range of 0-65535, or a single ASCII character (including control-characters). Numeric and string values may be freely interchanged, in which case, the characters are equivalent to the decimal value of their ASCII code representation. Thus, it becomes possible to add 1 to the letter "**A**", giving as a result, the letter "**B**".

The arithmetic operations permitted for use in expressions are:

+ (addition)
-(subtraction)
*** (multiplication)**
/ (division)
= (test for equality)
> (test for greater than or equal to)
< (test for less than)

The test operations, equal to, greater than, or equal to, and less than, all return a value of zero if the test fails, and a value of one if the test is successful.

Expressions in VTL-2 may contain any number of variables or numeric values (literal constants) connected by any of the above operation symbols. Parentheses may be used to alter the order of execution of the operations. If no parentheses are included, the operations proceed in strictly left to right order.

The value resulting from the expression must be assigned to some variable name. This is done with the equal sign. Note that the symbol has two meanings, depending on where it occurs in the expression. The expression: `A=B=C` means test **B** and **C** for equality; if they are equal, put a one in **A**; if they are unequal, put a zero in **A**.

Some examples of valid arithmetic expressions would be:

`Y=A*(X*X)+B*X+C` With left to right execution, this is equivalent to:

Y=(A*X*X+B)*X+C

`Y=(A*X*X)+(B*X)+C` Which is equivalent to: **Y=AX^2+BX+C**

Notice how the absence of parentheses around the quantity B*X in the first expression has completely altered its meaning. Keep the left to right order in mind, and when in doubt, use parentheses to control the order of evaluation.

SYSTEM VARIABLES:

In order to conserve space, and to provide a more consistent syntax, VTL-2 uses "system variables" to accomplish functions usually done with special key words in other languages. This convention is probably the single most important reason for its tiny size.

The system variable "number" or "pound sign" '**#**' represents the line number of the line being executed. Until the statement has been completed, it will contain the current line number; so that the statement:

```
100 A=#
```

is equivalent to simply writing:

```
100 A=100
```

After completion of a line, this variable will contain the number of the next line to be executed. If nothing is done to the variable, this will be the next sequential line in the program text. If a statement changes #, however, the next line executed will be the line with the number that matches the value of #. Thus the variable # may be used to transfer control of execution to a different part of the program.

This is the VTL-2 equivalent of the BASIC "GOTO" statement. For example:

`#=300` means "GOTO 300"

If the # variable should ever be set to zero by some statement, this value will be ignored, and the program will proceed as if no change had taken place. This allows us to write IF statements in VTL-2. Consider the following example:

```
10 X=1  Set X equal to 1
20 #=(X=25)*50  If X is equal to 25 then goto 50
30 X=X+1  Add 1 to X
```

```
40 #=20 Goto 20
50 ..... Continue
```

Notice that the quantity (**X=25**) will have the value one, if it is true that X is equal to 25, and the value zero if it is false. When this logical value is multiplied times 50, the result will be either zero, or 50. If it is 50, the statement causes a "GOTO 50" to occur. If the value is zero, a "GOTO 0" which is a dummy operation, causes the next statement down (number 30) to be executed.

Taking advantage of left to right evaluation, two bytes of memory could be saved by writing:

```
20 #=X=25*50
```

Each and every time the value of # is changed by a program statement, the old-value+1 is saved in the system variable "exclamation point" '**!**'. In other words, after executing a GOTO, the line number of the line that follows the GOTO is saved so that a subroutine will know which program statement called it; and will know where to return when finished. Thus the # variable is used for both GOTO and GOSUB operations. For example:

```
10 X=1
20 #=100
30 X=2
40 #=100
50 X=3
60 #=100
....
100 X=X*X
110 #=! (GOTO back where you came from)
```

In this example, control proceeds from line 20 to line 100. After that, line 110 causes control to return to line 30. When line 40 is executed, the subroutine at 100 will return to line 50.

The actual value stored in the **!** variable is (old-line-number+1) But, VTL-2, if it does not find the exact line number it is searching for, will take the next higher existing line number. Therefore, if a program statement says "**#=52**" and there are lines numbered 50 and 60 with nothing in between, control passes to the next higher line number, which is line 60.

The system variable "question mark" '**?**' represents the user's terminal. It can be either an input, or an output, depending on which side of the equal sign it appears.

The statement "**?=A**" is interpreted as "PRINT **A**", and the statement "**X=?**"is interpreted as "INPUT **X**". Note that the "**?**" may be included anywhere in the expression. For example, the program:

```
10 ?="ENTER THREE VALUES"
20 A=(?+?+?)/3
30 ?="THE AVERAGE IS ";
40 ?=A
```

will request three inputs while executing line 20.

When typing in a reply to a request for input, the user may enter any one of three different types of data:

1. A decimal number
2. A variable name
3. Any valid VTL-2 expression

Thus, for example, the user may reply with such things as "1004" or "A+B*(9/X)".
In each case, the expression is completely evaluated before the result is passed to the input statement. The only exception is that you are not allowed to respond with another question mark; as this will mess up the line pointer in the interpreter, causing it to return an improper value.

If a carriage-return, with no value, is typed in response to a request for input, the interpreter will return some undefined value. Therefore, this is not recommended.

When the question-mark is on the left side of the first equal sign, it represents a PRINT statement. When this occurs, either of two different things may be on the right side of the equal sign:

1. Any valid VTL-2 expression (as defined above)

2. A string of characters enclosed in quote marks ("")

When the expression is a numeric one, the value is computed, and printed as a left-adjusted, unsigned, decimal integer, with no leading or trailing blanks. A carriage return never follows the printing of a decimal value.

When the expression is a quoted character string, the actual string of characters is printed, with no leading or trailing blanks. A carriage-return line-feed sequence will follow the printing of a string unless a semicolon follows the closing quote.

This omission of leading and trailing blanks allows complete control of formatting printed output. For example, the program:

```
10 ?=50/2
20 ?=",";
30 ?=265+3
40 ?=".";
50 ?=16
```

will print the line: "25,268.16" with no spaces between the pieces. This feature is most often used in floating point, and multiple precision subroutines. (See "FACTORIALS" in the sample program section.)

If at any time it is desired to have a carriage-return line-feed printed, the statement:

```
? =""
```

will accomplish this.

The system variable "per-cent" '**%**' contains the value of the remainder of the last divide operation. This value will remain the same until the next divide operation.

The system variable "apostrophe" ''' represents a random number. This number will have an unpredictable value in the range of 0-65535. If called twice in the same line, the same value will be returned both times. The value of the variable is scrambled each time any statement is executed. Therefore, for best results it is highly recommended that at least one other computation be performed before the value is again called for. This may even be a simple dummy statement such as: Z=Z+7 For an example of this, see "DON'T LOSE YOUR AT" in the sample programs section.

In addition to decimal numeric input and output, the system variable "dollar sign" '**$**'is used to input and output single characters. As with the question mark variable, "**A=$**" means "input a single ASCII character and place its numeric value in **A**". Similarly, "**$=X**" means "print the single ASCII character whose value is stored in **X**". For example, the program:

```
10 A=65
20 $=A
30 A=A+1
40 #=A<91*20
50 ?=""
```

will print out, as one continuous string, all the letters of the alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ. If you wish to find out what decimal values correspond to which characters, these can be found in your terminal operations manual, or simply computed by typing the direct statement **?=$** and then entering the character whose decimal value is to be found.

The system variable "asterisk" '*****' represents the memory size of your computer. For a 1k system this would be 1024. For a 32k system this would be 32*1024 or 32768. In the 8080 version, this contains random data; so you should initialize it to your actual available memory on your machine.

If the user wishes to allot space for user defined machine-language-subroutines, then the variable * is set equal to the bottom of the first byte required by the user defined routine.

On compiled C versions of VTL-1 and VTL-3, the variable is preset on entry. Setting no memory available "***=0**" exits the program to DOS.

The system-variable "ampersand" '**&**' represents the next available byte of memory in the program buffer. When first calling VTL-2, or when it is desired to erase the present program, this must be initialized to the program start or the value 320 for the 8080 version.. Most other versions of VTL-2, particularly the compiled 'C' versions, come initialized with the program empty (**&=0**);

&=320 Altair 8080 initialization
&=0 Compiled 'C' versions (automatic)

At any given time the user may find out how much of his memory still remains unused by typing

?=*-&

This will cause the system to respond with the number of bytes remaining. A minimum of at least three bytes are needed for any line of VTL-2. The line-numbers are saved in binary and require two bytes regardless of their decimal values. The lines "**1 X=Y**" and "**65000 X=Y**" both take up an identical 7 bytes of memory; and are examples of the normal minimum valid VTL-2 line.

Any memory past the end of a program may be used for array storage. This array storage may be used for saving numeric or string values. The array does not have a name, since there is

only one, but it can be divided up into several pieces by the programmer and used for different groups of data (If the programmer has enough spare memory for the purpose). See the program "CIPHER" in the sample programs section. A subscript expression is identified by a colon '**:**' and a right-parentheses. The colon marks the beginning of the expression; and the right-parentheses marks the end. Thus, for example, "**:1)=0**" places a zero in the first two bytes past the end of the program; and "**:2+7)=A**" places the value of the A variable into the 9th two-byte word past the end of the program.

Subscripts should not be allowed to be less than one (1) as this will point the subscript into the program and could cause it to be wiped out as a result. (Special programs such as the RENUMBER program make use of this feature.)

Subscript expression may be any valid VTL-2 numeric expressions. This example should clarify the use of subscript expressions.

```
10 I=1  Set pointer
20 :I)=$  Input a character to the next word
30 #=:I)=13*60  GOTO 60 if it's a carriage-return char.
40 I=I+1  Point to next array word
50 #=20  Go get another character
60 ?=""  Print a carriage-return/line-feed
70 I=1  Reset pointer
80 $=:I)  print Ith character
90 #=:i)=13*120  If carriage-return, then GOTO 120
100 I=I+1  point to next character
110 #=80  Go get next character
120 ?=""  Print carriage-return/line-feed
```

The above example will read in any string of characters typed by the user, such as a sentence or paragraph, until a carriage-return is typed. It will then echo back the complete string as it was typed in.

For further examples, study the game programs which use character input and those that have arrays representing the playing board. These will be found in the Sample Program section.

Since subscripts are two-byte words, and since values as large as 65535 are allowed as subscripts, it is possible that large values in the subscript expression may "wrap around" the end of memory and reach locations within the program text itself. Therefore, there is a danger that a VTL-2 program using computed subscripts may "clobber" itself. On the other hand, this also means that a VTL-2 program may modify itself (as in the RENUMBER program) although this practice is not recommended.

The system-variable "greater-than" '>' is used to pass a value to a machine-language program. When encountered on the left side of the equal sign, the expression is evaluated, the

value placed as a 16-bit integer into two registers, and control passed to the user routine. In the 8080 version, a software-interrupt (SWI) is generated, with the values passed in the A and

B registers. Make sure you have provided handler routines. The returned-value in the same two registers is placed in the system-variable >.

On the compiled 'C' versions of VTL-2 (and VTL-3) the ">" symbol is used for output to a file. The syntax is:

```
>="filename"
```

The > variable contains the status of whether the write was successful or not.

Similarly, on the same systems, the "<" symbol is used for input from a file. Again the syntax is:

```
<=:"filename"
```

The < variable contains the status of whether the read was successful or not.

Output or input continues in each case until a Control-C character is encountered. This can either be from the console or found in the input or output. Make sure programs that output using this function include the end-character.

There is no "END" statement in VTL-2. The interpreter simply continues sequentially through the program until it runs out of lines to execute or until a statement is encountered which will try to transfer control to a line that is greater in number than any in the program.

Besides the tightly assembled versions of VTL-2 for the 8080 and 6800, there have been other versions made for the 68000 and other processors including DATA-100 terminals. Since I didn't write the 68000 variations, I can't tell you where to find them; only that they exist.

I have since compiled (rather huge actually) two variations of VTL to run under DOS (and that run in a DOS window under Windows). VTL2.COM and VLT-3.COM. The VTL-3 version just being a 32-bit version of VTL-2. Both are free, just for the asking. All I want there is credit for making the programs. VTL2.COM is straight-up VTL-2, with the addition of file-handling. You can enter the command from DOS:

VTL2 HURKLE.VTL

which will find the program HURKLE.VTL load it, and run it.

Write me at:

"Frank McCoy" <mccoyf@millcomm.com>

for more information; or about anything to do with VTL-2

OPERATIONAL CHARACTERISTICS

When the 8800 is first turned on the following things must be done before any VLT-2 program may be entered. First, lift the Stop and Reset switches simultaneously. Then release the Reset switch and then the Run/Stop switch. Next set the address switches to 174000 octal (F800 Hex) A15–A11 up, A10-A0 down. Raise Examine. Lower switches A15-A10.

Finally set the terminal option on the sense switches. Raise A11 if terminal is on port 0. If the terminal is on port 1, lower switch A11. Press and release Run/Stop switch.

Once VTL-2 is in control, the message "**OK**" will be printed. The next step is to set your memory-size. This is done by directly entering

***=1024** for a 1K memory system,
***=1024*62** for a 62K system (64K -2K ROM)

Next, set the end-of-program pointer. This is done by typing

```
&=320
```

for an 8080 system regardless of memory size.

VTL-2 is now ready to begin accepting programs and commands. If at any time it is desired to erase the program in memory, repeat the last step above. This will re-initialize the VTL-2 program buffer space.

When a program line is entered, it will be inserted into its proper space in the program text. If the line just entered has the same line-number as a line already in the text, the old line will be replaced by the new line. If the line-number only is typed, followed immediately by a carriage-return, the line with that number will be deleted.

While typing in program lines, VTL-2 should single-space,
and make no replies to lines entered. If, after typing a line, the system double-spaces down and prints, "OK", that indicates that there was not enough memory available to insert the new line just typed.

The user may check to see how much memory remains for program entry at any time by typing the direct statement (no line number) **?=*-&**. The system will respond with the number unused bytes remaining.

While typing in a line, the back-arrow key (Shift-0 on some terminals, Underline on others) will cause the last character typed to be deleted from the input buffer. The character will still appear on the screen, terminal, or printout; but will no longer be in memory. Thus the line: "**A=B*C__+N**" goes in as "**A=B+N**", where the "***C**" was deleted by the two back-arrow (or

underline) characters.

At any time before hitting Return, the entire line may be erased by typing the At-sign character '**@**' (Shift-P or "Cancel" on some terminals.)

Typing the single character zero '**0**' followed by a carriage-return causes VTL-2 to print out a complete listing of the program.

While printing is taking place, whether as a program listing or as output from a program, the operation can be cancelled and control returned to the operator by pressing Control-C. When this is done, VTL-2 completes its current print statement, and then prints "**OK**" to acknowledge the interruption.

In addition to this, any other key (preferably a non-printing control-character such as Control-A) may be pressed. This will cause VTL-2 to temporarily suspend operation and wait for another key to be pressed. (Again, preferably another non-printing character.)

This feature allows users with video terminals to list their programs a section at a time; hitting Control-A to stop the listing; and hitting it again to resume listing.

Note that these characters also affect printing being done by a program. You may temporarily halt your program with a Control-A, and start it up again with another Control-A. These keys only work during printing which uses the question-mark '?' system-variable. String printing with the dollar-sign '$' variable will NOT interrupt in this manner. This allows the user the option of making his program interruptible or non-interruptible.

Should an uninterruptible program become "locked up" in a loop, the only way out is with the front-panel reset, as described above when starting. You can then jump to VTl-2 in the same manner you did when starting; just don't re-initialize the program-pointer, and your program should still be in memory for you to edit or run again.

To run a program once entered, the user simply types the direct command:

```
#=1
```
(GOTO 1)

This causes VTL-2 to find the lowest numbered line and start executing there. If it is desired to begin execution at some other line, say line 1000, simply type:

```
#=1000
```

or whatever line is desired to start at.

Comments may be inserted on any line by preceding them with a right-parentheses ')' character. This symbol must follow the expression on the line immediately, with no blanks in between. This causes VTL-2 to stop evaluating the line and go on to the next line. If a line is to

contain only a comment, then the first character on the line should be a right-parentheses. Examples of commented lines:

```
100 A=2)  Initialize variable
110 )  Just a comment alone.
```

There are no error-messages in VTL-2. If an expression is wrong, the results of evaluating that expression will also be wrong. ALL expressions, even if improperly formed, WILL be evaluated! In other words: VTL-2 assumes that you know what you are doing; and will do its best to execute any statement that you give it. This leaves WIDE latitude for trying various program "tricks"; but also leaves the complete responsibility for verifying program accuracy with the programmer. Thus you will have ample opportunity to "shoot yourself in the foot" if not careful.

For those who would like to experiment with VTL-2 on a DOS based machine, I have copies of VTL2.COM and VTL3.COM that I distribute freely. VTL-3 being a 32-bit version of VTL-2,

Simply send me an email at:

"Frank McCoy" <mccoyf@millcomm.com>

asking for a copy, and I'll send you one. The only limit I place on the usage is recognition of authorship.

I also appreciate hearing of any interesting VTL2 programs other people have created. Listings *much* appreciated!

(Note from Grant, I would also appreciate hearing about VTL-2 programs! Send me an e-mail at grant@stockly.com or share your program on the http://www.stockly.com forums.)

Sample of Programming

(8800 version)

When starting: A. Turn the computer on.
C. Lift the Run/Stop and Reset switches
D. Examine 174000 (F800 Hex)
E. Press and release Run/Stop switch
F. VTL-2 should reply with "**OK**"

```
OK                  VTL-2 prompt
*=1024*62           Set memory-size to 62K bytes

OK                  VTL-2 prompt
&=320               Reset program pointer

OK                  VTL-2 prompt

10 A=0  Set A equal to zero
20 B=1  Set B equal to one
30 ?=A  Print the factorial number
40 "=”! = ";  Print “Factorial equals”
50 ?=B  Print the value of the factorial
60 ?=”” Print a carriage-return/line-feed
70 A=A+1  Increment factorial number
80 B=B*A  Calculate next by multiplication
90 #=A<9*30  If less than !9 then GOTO step 30
#=10  Execute program
0! = 1              (Program Output)
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320

OK                  VTL-2 prompt saying it's done.
```

List of new features of VTL-2 Compared to VTL-1

#1 11 more variables
#2 1 array
#3 Computed return-address
#4 End-of-program pointer
#5 End-of-memory pointer
#6 Random number generator
#7 Single character string input
#8 Single character string output
#9 Machine-language subroutines
#10 Computed line inputs
#11 Computed array
#12 Faster operation
#13 Input/output compatibility on LIST
#14 Faster line insertion
#15 No double carriage-return on line inputs
#16 “OK” prompt
#17 No prompt on line-insertion
#18 Nulls and control-characters equal or less than carriage-return are not inserted into lines
#19 Control-C stops execution when printing
#20 Any key pressed suspends operation when printing
Any key pressed resumes operation when printing
#21 All interrupt vectors are preserved
#22 On initialization, setting the line-number equal to zero is no longer needed
#23 Multiple operations are allowed on a value on a line
#24 Single and multiple parentheses are allowed
#25 Print statements may be with or without a following carriage-return/line-feed
#26 No leading or trailing spaces are added to a printed number
#27 # stands for present line number, not present line-number plus one
#28 More understandable system variables. EI: # stands for line-number
#29 33 more bytes are allotted for program space
#30 Programs takes up 10% less space in memory
#31 Up to 72 characters are allowed per line

List of Features

Variables:

Variable:	Meaning
A-Z	Common Variables
	Use freely for storing values

System Variables:

!	Return-Address
	Points to line# after the last #= statement
“	Pointer for literal print statements
#	Current line-number
$	Single-character input or output
%	Remainder after last divide operation
&	Points to last byte of program
’	Random number
(	Sets start of parenthesized expression
)	End:
	Sets end of line
	Sets end of parenthesized expression
	Sets end of array description Used also for REMARK statements
*	Points to end of memory
>	Machine-language subroutine (680 or 8800)
	Output to disk-file (Compiled C versions)
<	Input from disk-file (Compiled C versions)
?	Print statement when left of equal-sign
	Input statement when right of equal-sign
:	Defines start of array description
;	When following a literal print-statement,
	says do not print carriage-return/line-feed

.-=;+,/^[]	May be used as standard variables;
	But use not recommended for legibility reasons

Operators:

+	Add to previous value
-	Subtract from previous value
*	Multiply times previous value
/	Divide previous value by
=	Is previous value equal to (Yes = 1, No = 0)
<	Is previous value less than (Yes = 1, No = 0)
>	Is previous value equal-to or greater than (Yes = 1, No = 0)

The default operator is the less-than test, if symbol for operator is unrecognized.

Control-Code Cross-Reference
For some common video terminals

All numeric values in decimal
X and Y are decimal values

Usage	Info-Term	SWTP-TVT	VT-100/ANSI
Home-Up	18	19	<Esc>[H
Erase-Screen	28	22	<Esc>[J
Goto X,Y			<Esc>[X;Y]H
Form-Feed (Next page on TVT.)	12	12	12
Carriage-Return	13	13	13
Back-Space	8	8	8
Line-Feed	10	10	10
Back-Cursor	19	18	<Esc>[D
Up-Cursor		21	<Esc>[A
Down-Cursor	10	10	<Esc>[B

These codes can usually be translated one-for-one, from one program to another. For example, where you see:

100 $=22 **For the SWTP TVT, You'd enter:**

100 $=27 **For a VT-100 or computer running ANSI.SYS**
101 ?="[J";

The remainder of this manual contains sample programs. You do not need to retype all of these programs, they can be downloaded from the forums at http://www.stockly.com. Programs may be "pasted" into the Altair 8080 using a serial terminal program like HyperTerm or TeraTerm. See the forums for more information!

HURKLE

```
100 ?=""
110 ?="A HURKLE IS HIDING ON A"
120 ?="10 BY 10 GRID. HOMEBASE"
130 ?="ON THE GRID IS POINT 00"
140 ?="AND A GRIDPOINT IS ANY"
150 ?="PAIR OF WHOLE NUMBERS"
160 ?="TRY TO GUESS THE HURKLE'S"
170 ?="GRIDPOINT. YOU GET 5 GUESSES"
180 ?=""
190 R='/100*0+%
200 A=R/10
210 B=%
220 K=1
230 ?="GUESS #";
240 ?=K
250 ?=" ?";
260 X=?/10
270 Y=%
280 ?=""
290 #=X*10+Y=R*540
300 K=K+1
310 #=K=6*440
320 ?="GO ";
330 #=Y=B*370+(Y<B*360)
340 ?="SOUTH";
350 #=370
360 ?="NORTH";
370 #=X=A*410+(X<A*400)
380 ?="WEST";
390 #=410
400 ?="EAST";
410 ?=""
420 ?=""
430 #=230
440 ?=""
450 ?="SORRY THAT'S 5 GUESSES"
460 ?="THE HURKLE IS AT ";
470 ?=A
480 ?=B
490 ?=""
500 ?=""
510 ?="LETS PLAY AGAIN."
520 ?="HURKLE IS HIDING"
530 #=180
540 ?="YOU FOUND HIM IN ";
550 ?=K
560 ?=" GUESSES"
570 #=490
```

TIME OF DAY DIGITAL CLOCK

FOR 300 BAUD TERMINALS:

```
10 ?="HOUR ?";
20 H=?
30 ?="MINUTE ?";
40 M=?
50 ?="SECOND ?";
60 S=?
70 ?="READY"
80 A=$
90 S=S+1
100 M=S/60+M
110 S=%
120 H=M/60+H
130 M=%
140 H=H/24*0+%
150 ?="TIME: ";
160 ?=H/10
170 ?=%
180 ?=":";
190 ?=M/10
200 ?=%
210 ?=":";
220 ?=S/10
230 ?=%
240 $=13
250 A=B
260 T=31
270 T=T-1
280 #=T=0*90
290 #=270
```

FOR 110 BAUD TERMINALS:

```
10 ?="HOUR ?";
20 H=?
30 ?="MINUTE ?";
40 M=?
50 ?="SECOND ?";
60 S=?
70 ?="READY"
80 A=$
90 S=S+1
100 M=S/60+M
110 S=%
120 H=M/60+H
130 M=%
140 H=H/24*0+%
150 ?=H/10
160 ?=%
170 ?=":";
180 ?=M/10
190 ?=%
200 ?=":";
210 ?=S/10
220 ?=%
230 $=13
240 A=B
250 A=B
260 A=B
270 A=B+B
280 T=14
290 T=T-1
300 #=T=0*90
310 #=290
```

FACTORIALS
CALCULATES FACTORIALS UNTIL IT RUNS OUT OF MEMORY
FOR 1K OF MEMORY THIS IS ABOUT 208!

```
10 A=1
20 L=2
30 :1)=1
40 I=2
50 :I)=0
60 I=I+1
70 #=L>I*50
80 ?=""
90 ?=""
100 ?=A
110 ?="! ="
120 ?=""
130 I=L+1
140 I=I-1
150 #=:I)=0*140
160 ?=:I)
170 I=I-1
180 #=I=0*220
190 ?=:I)/10
200 ?=%
210 #=170
220 A=A+1
230 I=1
240 C=0
250 X=:I)
260 :I)=A*X
270 #=:I)<X*320
280 :I)=:I)+C
290 C=:I)/100
300 :I)=%
310 I=I+1
320 #=L>I*250
330 #=C=0*80
340 L=L+1
350 #=*-&/2<L*380
360 :I)=C
370 #=290
```

WEEKDAY

```
10 #=440
20 ?="DAY OF THE WEEK"
30 ?=""
40 ?="MONTH? ";
50 M=?
60 #=M>13*40
70 #=M=0*40
80 ?=DAY OF MONTH? ";
90 D=?
100 ?="YEAR? "
110 Y=?
120 #=Y>1800*230
130 #=Y<100*150
140 #=70
150 ?=""
160 ?="IS THAT 19";
170 ?=Y
180 ?="? ";
190 K=$
200 #=K=89=0*70
210 ?="ES"
220 Y=Y+1900
230 C=Y/100
240 Y=%
250 #=Y/4*0+%=0*280
260 :1)=6
270 :2)=2
280 W=Y/4+Y+D+:M)+(2*(C=18))/7*0+%
290 #=300+(20*W)
300 ?="SUN";
310 #=430
320 ?="MON";
330 #=430
340 ?="TUES";
350 #=430
360 ?="WEDNES";
370 #=340
380 ?="THURS";
390 #=430
400 ?="FRI";
410 #=430
420 ?="SATUR";
430 ?="DAY"
440 :1)=0
450 :2)=3
460 :3)=3
470 :4)=6
480 :5)=1
490 :6)=4
500 :7)=6
510 :8)=2
520 :9)=5
530 :10)=0
540 :11)=3
550 :12)=5
560 #=20
```

STARSHOOTER

```
10 I=0
20 I=I+1
30 :I)=46
40 #=I<41*20
50 :25)=42
60 I=8
70 J=1
80 $=I-1/7+64
90 ?=" - ";
100 S=I+J
110 $=:S)
120 J=J+1
130 #=J=6*160
140 ?=" ";
150 #=100
160 I=I+7
170 ?=""
180 ?=""
190 #=I<43*70
200 ?=""
210 ?="    1  2  3  4  5"
220 ?=""
230 ?="YOUR MOVE --";
240 I=42
250 I=I+1
260 :I)=$
270 #=:I)=13*320
280 #=:I)=3*580
290 #=:I)=95=0*250
300 I=I-1
310 #=260
320 A=:43)-64
330 ?=""
340 #=A>6*230
350 B=:44)-48
360 #=B>6*230
370 S=A*7+1+B
380 ?=""
390 #=:S)=42*420
400 ?="THAT'S NOT A STAR!"
410 #=230
420 :S)=46
430 C=S-7
440 #=520
450 C=S-1
460 #=520
470 C=S+1
480 #=520
490 C=S+7
500 #=520
510 #=60
520 ^=!
530 #=:C)=42*560
540 :C)=42
550 #=^
560 :C)=46
570 #=^
```

Object of the game is to change this:

```
          A - . . . . .

          B - . . . . .

          C - . . * . .

          D - . . . . .

          E - . . . . .

             1 2 3 4 5
```

To this:

```
A - * * * * *

B - * . . . *

C - * . . . *

D - * . . . *

E - * * * * *

   1 2 3 4 5
```

MEMORY TEST

```
10 A=0
20 ?="MEMORY SIZE? ";
30 B=?
40 A=A+257
50 C=A
60 I=0
70 I=I+1
80 :I)=C
90 C=C+769
95 ?="";
100 #=I*2+&<B*70
110 C=A
120 I=0
130 I=I+1
140 D=:I)
145 #=D=C=0*170
150 C=C+769
155 ?="";
160 #=130
170 E=C/256
180 F=%
190 G=D/256
200 H=%
210 J=E=G
220 P=I*2+&+J
230 #=P<B*300
240 ?="TEST # ";
250 ?=A/256
260 ?=" COMPLETE"
270 #=A<65535*40
280 ?="DONE"
290 #=999
300 ?="ERROR AT BYTE #";
305 ?=P
310 ?=""
320 #=J*342
330 F=E
340 H=G
342 ?=" TEST BYTE WAS ";
344 ?=F
345 ?=""
346 ?="ERROR BYTE WAS ";
348 ?=H
349 ?=""
350 L=0
360 F=F/2
370 M=%
380 H=H/2
390 N=%
400 #=M=N=0*430
410 L=L+1
420 #=L<8*360
430 ?="BIT IN ERROR IS #";
440 ?=L
450 ?=""
```

FACTORS

Calculates factors of a number

```
10 ?="NUMBER? ";
20 N=?
30 X=N
40 $=27
45 ?="[2J";
50 ?=N
60 ?=" IS ";
70 #=N/2*0+%=0*140
80 D=3
90 Q=N/D
100 #=%=0*160
110 #=D>Q*300
120 D=D+2
130 #=90
140 ?="EVEN."
150 #=10
160 ?=""
170 ?=D
180 N=Q
190 Q=N/D
200 #=%=0*220
210 #=120
220 ?="^";
230 P=1
240 N=Q
250 Q=N/D
260 P=P+1
270 #=%=0*240
280 ?=P
290 #=120
300 #=N=1*340
310 #=N=X*390
320 ?=""
330 ?=N
340 ?=""
350 ?="DONE"
370 ?=""
380 #=10
390 ?="PRIME."
400 ?=""
410 #=340
```

DON'T LOSE YOUR AT!

BY

Ed Verner

Adapted to VTL-2 BY Gary Shannon

(A game similar to "BAGLES")

The object of the game is to guess the secret number picked by the computer. The number has three digits, no zeroes, and no digit is repeated. After you type in your guess, the computer will print an “IT” for every correct digit in the wrong position, and an “AT” for every correct digit in the right position. You win when you get three “AT”s. Each time that you guess incorrectly, you lose 5% of the points you have left

```
10 T=0
20 L=0
30 ?="DON'T LOSE YOUR 'AT'"
40 X='/9*0+%+1
50 Y='/9*0+%+1
60 #=X=Y*40
70 Z='/9*0+%+1
80 #=X=Z*40
90 #=Y=Z*40
100 ?="I'VE GOT A NUMBER."
105 L=L+1
110 P=10000
120 ?=""
130 ?="YOU HAVE ";
140 ?=P/100
150 ?=".";
160 ?=%/10
170 ?=%
180 ?=" POINTS LEFT"
190 ?=""
200 ?="WHAT'S YOUR GUESS? -- ";
210 G=?
220 A=G/100
230 B=%/10
240 C=%
260 S=600
270 #=A=Y*S
280 #=A=Z*S
290 #=B=X*S
300 #=B=Z*S
310 #=C=X*S
320 #=C=Y*S
330 K=0
340 S=620
350 #=A=X*S
360 #=B=Y*S
370 #=C=Z*S
380 #=K<3*580
390 ?=""
400 ?="YOU WIN ";
410 ?=P/100
420 ?=".";
430 ?=%/10
440 ?=%
450 ?=" POINTS FOR A TOTAL OF ";
460 T=T+P
490 ?=T/100
500 ?=".";
510 ?=%/10
520 ?=%
540 ?=" POINTS IN ";
550 ?=L
560 ?=" GAMES."
570 #=30
580 P=P/20*19
590 #=120
600 ?="IT ";
610 #=!
620 ?="AT ";
630 K=K+1
640 #=!
```

```
****************************** HAVE FUN! ****************************
```

CRAPS!

```
10 T=100
20 $=22
30 ?="CRAPS!"
40 ?=""
50 ?="HOW MUCH DO YOU BET? - ";
60 B=?
70 #=B=0*90
80 ?="GOOD LUCK!"
90 #=B=0*480
100 #=T>B*160
110 ?="TOO MUCH!"
120 ?="YOU HAVE $";
130 ?=T
140 ?=" LEFT."
150 #=40
160 ?=""
170 ?="ROLL-";
180 A=?
190 $=22
200 ?="FIRST ROLL: ";
210 #=500
220 #=R=7*360
230 #=R=11*360
240 #=R<4*390
250 #=R=12*390
260 ?=""
270 ?=R
280 ?=" IS YOUR POINT"
290 P=R
300 ?="ROLL-";
310 A=$
320 #=500
330 #=R=7*390
340 #=R=P*360
350 #=300
360 ?="YOU WIN"
370 T=T+B
380 #=120
390 T=T-B
400 ?="YOU LOSE"
410 #=T=0*430
420 #=120
430 ?="YOU ARE BUSTED!"
440 ?="MOVE OVER AND LET THE NEXT"
450 ?="SUCKER TRY."
460 ?=""
470 #=10
480 ?="BE SERIOUS"
490 #=40
500 R='/6*0+%+1
510 ?=R
520 X=X+11213
530 ?=" AND ";
540 S='/6*0+%+1
550 X=X*56001
560 ?=S
570 ?=" (";
580 R=R+S
590 ?=R
600 ?=")"
610 #=!
```

CIPHER GAME

```
10 I=0
20 I=I+1
30 :I)=I+64
40 #=I<26*20
50 I=1
60 ?=""
70 M='/26*0+%+1
80 H=:M)
90 :M)=:I)
100 :I)=H
110 I=I+1
120 #=I<27*70
130 ?="TEXT?"
140 ?=""
150 I=27
160 :I)=$
170 #=:I)=13*220
180 #=:I)=95=0*200
190 I=I-2
200 I=I+1
210 #=160
220 ?=""
230 I=27
240 #=:I)<64*270
250 T=:I)-64
260 :I)=:T)
270 I=I+1
280 #=:I)>14*240
290 ?=""
300 ?="CODE:"
310 ?=""
320 I=27
330 $=:I)
340 #=:I)=13*370
350 I=I+1
360 #=330
370 ?=""
380 ?="SWITCH? - ";
390 A=$
400 B=$
410 #=B=64*370
420 I=27
430 #=:I)=A*490
440 #=:I)=B=0*460
450 :I)=A
460 I=I+1
470 #=:I)=13*290
480 #=430
490 :I)=B
500 #=460
```

PHRASE SORT

```
10 $=22
20 I=0
30 I=I+1
40 :I)=$
50 L=:I)=95*2
60 I=I-L
70 #=:I)>14*30
80 ?=""
90 I=1
100 K=I
110 J=K
120 #=:K)=32*160
130 #=:J)=32*150
140 #=:K)>:J)*160
150 J=K
160 K=K+1
170 #=:K)>14*120
180 H=:I)
190 :I)=:J)
200 :J)=H
210 I=I+1
220 #=:I)>14*100
230 I=0
240 I=I+1
250 $=:I)
260 #=:I)>14*240
270 ?=""
```

LUNAR LANDER

```
10 $=26
20 ?=" LUNAR LANDER"
30 )BY SU-MING WU
40 )MAR-VISTA CA.
50 )MAY 1977
55 )CORRECTED BY FRANK MCCOY
56 )JUNE 6 1977
60 ?=""
90 F=120
110 V=50
120 D=500
130 ?="FUEL SPEED DISTANCE BURN"
140 T=F
141 #=410
142 ?=F
150 ?=" ";
151 T=V<10000*V+(V>10000*(0-V))
152 #=410
153 $=T=V*32+(T<V*45)
154 ?=T
160 T=D
161 ?=" ";
162 #=410
163 ?=D
170 ?=" ? ";
220 B=?
230 B=B<F*B+(B>F*F)
240 F=F-B
250 #=B<5*280
260 Z=B-5/2
270 #=290
280 Z=0-(0-B+5/2)
290 #=Z+D-V=0+(V-B+5=0)=2*470
291 #=Z+D-V-1>10000*490
293 D=Z+D-V
294 V=V-B+5
300 B=0
305 #=F=0*490
310 #=140
410 #=T>1000*!
420 ?=" ";
430 #=T>100*!
440 ?=" ";
450 #=T>10*!
460 ?=" ";
465 #=!
470 ?="PERFECT LANDING!"
480 #=600
490 L=V*V+(5-B*D*2)
491 #=L=0*499
493  X=L*4
494 W=L
495 L=X/L+L/2
496 #=L<W*494
497 L=L/2
500 #=L<2*470
510 ?="YOU HIT THE GROUND WITH A VELOCITY OF ";
520 ?=L
530 ?=" FEET PER SECOND"
540 #=L>10*580
      550 #=L>5*562
      555 ?="A VERY GOOD LANDING"
      560 #=600
      562 #=L>8*570
      563 ?="A FAIR LANDING, BUT YOU DAMAGED"
      564 ?="PART OF YOUR LANDING GEAR"
      565 #=600
570 ?="A POOR LANDING---YOU'LL PROBABLY BE STRANDED HERE"
575 ?="FOR THE REST OF YOUR SHORT LIFE"
      576 #=600
      580 ?="NO SURVIVERS-----REST IN PEACE"
      600 ?=""
      610 ?=""
      620 ?=""
      630 ?=""
      640 ?="DO YOU WANT TO PLAY AGAIN? ";
      650 A=$
      660 #=A=89*10
      670 #=A=78=0*630
      680 ?="O";
```

LIFE
FAST VERSION

```
10 #=370
20 S=Y<F*Y+(Y=0*E)+(Y=F)-1*O+(X<Q*X+(X=0*O)+(X=Q))
25 :S)=:S)+2
30 X=X+1-(J<X*3)+(J-1=X*(Y=I))
40 Y=J-1=X+Y
50 #=I+1>Y*20
70 #=90
80 #=:I-1*O+J)/2*0+%*20
90 J=J+1-(O=J*O)
100 I=J=1+I
110 ?="";
120 X=J-1
130 Y=I-1
140 #=I<F*80
150 I=1
160 J=1
180 ?=""
190 P=0
200 K=I-1*O+J
210 :K)=:K)<5+(:K)>8)=0
220 P=P+:K)
230 $=:K)*10+32
240 J=J+1-(J=O*O)
250 #=1<J*200
260 ?=""
270 I=I+1
280 #=I<F*200
290 ?="GEN = ";
300 ?=G
310 G=G+1
320 ?=" POP = ";
330 ?=P
340 I=1
350 J=1
360 #=0<P*110+(P=0*650)
370 I=1
380 G=0
390 ?="SIZE? ";
400 O=?
410 Q=O+1
420 ?="BY? ";
430 E=?
440 F=E+1
450 J=O*E+2
460 #=J*2+&>**390
470 :I)=0
480 I=I+1
490 #=J>I*470
495 #=631
500 I=1
510 ?=""
520 J=1
530 #=I>10*550
540 ?=" ";
550 ?=I
560 ?=" ";
570 L=$
580 :I-1*O+J)=L=32+(L=13)+(L=95)+(L=64)=0*6
590 J=J+1-(L=95*2)
600 #=L=13*620+(L=64*510)
610 #=J<Q*570
620 I=I+1
625 #=I<F*510
626 #=631
627 #=150
631 $=22
632 $=18
633 $=32
634 $=18
635 $=22
636 $=18
637 $=32
638 $=18
640 #=!
```

This program takes at least 2K to operate properly.
This version was written for the SWTP TVT; but will run on any normal terminal. For best results (on the TVT) try for a 31 by 15 matrix.

TIC-TAC-TOE

This version for Dec VT-100
Or compatibles (ANSI.SYS)

```
1000 Q=0
1010 H=0
1020 J=0
1030 I=0
1035 $=27
1037 ?="[J";
1040 U=0
1050 S=1
1054 ?=""
1056 ?=""
1060 I=I+1
1070 :I)=I+48
1080 #=I<9*1060
1090 #=1680
1100 $=27
1101 ?="[H";
1110 ?=""
1120 ?=" YOUR MOVE - ";
1130 U=1
1140 M=$-48
1150 #=3-48=M*2040
1160 ?=""
1170 #=M=0*1030
1180 #=9<M*1211
1190 #=:M)<65*1240
1191 $=27
1192 ?="[H";
1200 ?=" SOMEBODY ALREADY THERE"
1210 #=1120
1211 $=27
1212 ?="[H";
1220 ?=" ILLEGAL MOVE! "
1230 #=1120
1240 :M)=88
1250 #=1680
1260 X=1
1270 L=0
1280 K=0
1290 N=1
1300 A=N
1310 B=X+N
1320 C=2*X+N
1330 #=:A)=:B)+(:A)=:C))=2*1880
1340 #=:A)=:B)+(:C)<65)=2*1410
1350 D=A
1360 A=B
1370 B=C
1380 C=D
1390 #=A=N*1440
1400 #=1340
1410 #=K>1*(:L)=79)*1440
1420 L=A
1430 K=C
1440 #=X=4+(2*X+N=9+(X=2))*30+#
1450 N=X=1*2+1+N
```

```
1460 #=1300
1470 X=X+1
1480 N=X=2*2+1
1490 #=1300
1500 #=K>1*1620
1510 I=0
1520 P=0
1530 I=I+1
1540 P=:I)>65+P
1550 #=I<9*1530
1560 #=P=9*1950
1570 K=M
1575 X=0
1580 K=:5)>65*(K+S/2*0+%+('/16384*2+1)+K/10*0+%/9*0+%+1)
1590 K=:5)<65*5+K
1600 X=X+1
1605 S=X>9+S
1610 #=:K)>65*1580
1620 :K)=79
1625 S=:5)=79*M+S/2*0+%
1630 #=L>1*(:L)=79)*1910
1640 !=1100
1650 $=27
1655 ?="[H";
1660 ?=" MY MOVE - ";
1670 ?=K
1675 ?=" "
1677 ?=""
1680 ?=""
1690 ?=""
1700 R=!
1710 I=7
1720 ?=" | |"
1725 ?=" ";
1730 ?=" ";
1740 #=U*1770
1750 $=:I)
1760 #=1780
1770 $=:I)<65*32+(:I)>65*:I))
1780 P=:I)>65+P
1790 I=I+1
1800 #=I/3*0+%=1*1830
1810 ?=" |";
1820 #=1730
1830 ?=""
1840 ?=" | |"
1850 #=I=4*R
1855 I=I-6
1860 ?=" -----+-----+-----"
1870 #=1720
1880 ?=" !!!YOU WIN!!!"
1890 H=H+1
1900 #=1970
1910 #=1650
1920 ?=" YOU LOSE."
1930 J=J+1
1940 #=1970
1950 ?=" CAT GOT THIS ONE."
```

```
1960 Q=Q+1
1970 $=13
1975 ?=" PLAY AGAIN? ";
1977 $=8
1980 S=$
1990 #=S=89+(S=121)=0*2020
2000 #=S=89*2006
2002 ?="es"
2004 #=1030
2006 ?="ES"
2010 #=1030
2020 #=S=78+(S=110)=0*1970
2030 #=S=78*2036
2032 ?="o"
2034 #=2040
2036 ?="O"
2040 ?=""
2050 ?=" I WON ";
2060 ?=J
2070 ?=" GAME";
2072 #=J=1*2076
2074 ?="S";
2076 ?=""
2080 ?=" YOU WON ";
2090 ?=H
2100 ?=" GAME";
2102 #=H=1*2106
2104 ?="S";
2106 ?=""
2110 ?=" WE TIED ";
2120 ?=Q
2130 ?=" GAME";
2132 #=Q=1*2136
2134 ?="S";
2136 ?=""
2137 ?="HIT ANY KEY TO EXIT"
2138 Z=$
2140 *=0
```

Note-1: Those last 3 lines are *only* used on compiled programs to make it exit to DOS. You don't need to include them on terminal-based programs.

Note-2: This particular program requires SLIGHTLY *OVER* 2K of memory to run. It could be (and has been in the past) trimmed down to just barely run with 2048 bytes on the 680B; but I no longer have a copy of that version.

Using single-character home-up and clear-screen gets most of the needed space; but not quite enough. Some serious editing would have to be done to save the needed space.

It's included as an example for those with more memory or using the compiled version of VTL-2.

RENUMBER

680B version

```
64000 A=#
64010 C=#
64020 B=&
64030 &=B
64040 ?="STARTING #? ";
64050 D=?
64060 ?="STEP SIZE? ";
64070 E=?
64080 &=1
64090 G=131
64100 J=0
64110 I=&
64120 H=#+1
64130 &=I
64140 G=&+1/2+G
64150 &=%
64160 #=:G)>A*15*(C-A)+#
64170 #=D-1>(A-1)+(J>D)>1*C
64180 :G)=D
64190 &=&+1
64200 J=D
64210 D=D+E
64220 I=&
64230 &=B
64240 K=#+1
64250 &=I+1
64260 X=:G)/256*0+%
64270 I=&
64280 &=B
64290 #=%>1*K
64300 #=H
64310 ?="DONE"
64320 ?="TO REMOVE RENUMBER FROM PROGRAM, TYPE: &=";
64330 ?=G*2+&
64340 ?=""
64350 &=B
```

8800 version

```
64090 G=159
```

Compiled version

```
64090 G=0-1
```

Note-1: The only difference between the three variations is line number 64090; which is half the size of & minus-one, on an empty program.

Note-2: The program itself is relocatable. I.e., it, itself can be renumbered and it will still run. However, the step-size between program steps must remain constant in the source, or line 64160 will not work right. Also, the largest number of the program to be renumbered must be less than the first number of the renumber program itself. That's why the numbers are large.

Note-3: This version of the renumbering program ONLY renumbers the basic line-numbers, not computed goto statements. Therefore a renumbered program will have to be edited manually to make all the "gotos" match the original source.

Note-4: To renumber a program, simply type-in or load RENUMBER.VTL on top of an existing program, type: #=64000 and let it run.