parameters may be used like any of the standard predefined **VP** numeric parameters.

```
37)  repeat &ncol.
38)    plotyear &count.
39)  end
```

Lines (37)–(39) use another advanced feature of **VP**—the REPEAT-END loop to call the MACRO `plotyear` for each column in the TABDATA array from the data file. Remember that **ncol** is an integer parameter that gives the number of TABDATA columns, so MACRO `plotyear` is called once for each column. The lines within the REPEAT-END loop are executed **ncol** times. Within the loop, the integer parameter **count** takes successive integer values (1, 2, 3, . . .).

## 2.4 Carrying On

If you have followed these three samples, you should have a pretty good feeling for the many ways **VP** can translate text and numbers into graphics. The increasing complications of `vpsamp3.exg` over `vpsamp2.exg` allow `vpsamp3.exg` to produce a publication-quality plot of any data file that meets the simple requirements.

It is useful to consider **VP** as a visual programming and data analysis language. The best way to *learn* **VP** is, perhaps, just to try using it on your own projects. To that end, the remainder of this book may be considered as a reference manual.

Chapter 3 may be quickly browsed for hints on many of the advanced aspects of **VP** programming.

Chapter 4 consists of sample `.exg` files and their graphic output. This is intended as a graphic index. If you want to understand how something can be accomplished in **VP**, find the corresponding picture and examine the source code and explanatory comments.

Chapter 5 gives a detailed reference for each of the **VP** commands, while Chapter 6 discusses the **VP** numeric parameters and character synonyms.

Finally, Chapter 7 discusses the many utility programs for producing hardcopies of your graphics or converting to other formats.

# 3

# Selected Topics

This chapter provides a detailed description of **VP** input, advanced features, and operation. It covers the following topics:

- input format
- numeric calculations
- synonyms for easy mnemonics
- including parameter values and synonyms in text
- text-processing features
- macros, loops, and conditional processing
- graphical input
- understanding coordinate systems
- **VP** arguments, options, and batch processing

## 3.1 Input Format

Depending on its internal state, **VP** will expect either

- command lines
- numeric values or
- text lines

from the keyboard or `.exg` file as its next input. When started, and at the appearance of the **VP** prompt (default: `vp:`), **VP** expects a command line to be entered from the keyboard. When numeric values are expected, **VP** prompts with `data:`. When text is expected, the prompt is changed to reflect the desired input (e.g., `text: Xlabel: Ylabel: Caption:`).

### 3.1.1 Command lines

- Each command line consists of a series of alphanumeric fields delimited by spaces, commas, or = signs. All of these delimiters are equal and consecutive delimiters have the same effect as a single delimiter. A single input line may not contain more than 255 characters.
- If necessary, a field may be delimited by single or double quotes to allow the inclusion of other delimiters (such as embedded spaces).
- If the line begins with an asterisk (*), it is treated as a comment and is echoed on the console. It is not examined further by **VP**.
- Any part of the line after a semicolon (;) is also treated as a comment and is discarded by **VP**. This discarded part is not echoed on the console.
- Before the line is interpreted,
  - Numeric calculation fields delimited by {...} or {...} are evaluated, and the resulting stack values are added as numbers to the input line.
  - Recognized synonym names delimited by & and . (&Sname.) are replaced by their text definitions.
- **VP** examines the resulting fields from left to right, looking for the following characteristics:
  - If the field corresponds to a synonym name (*without* initial & and ending .), the synonym definition is substituted for the field and the line evaluation begins anew.
  - If the field corresponds to a **VP** parameter name *and* the following field is numeric, the parameter value is set to the numeric value. Both the parameter name and its value are removed from the input line. The rules for numeric input are summarized as follows:
  - Comparison between alphanumeric fields and synonym names, parameter names, and commands is made independent of case using a maximum of 8 characters. Thus, the input fields may consist of mixed-case characters. Only enough characters need to be included to distinguish a name from all other **VP** possibilities.
  - Unrecognized fields, numbers, and parameter names *not* followed by numeric fields remain part of the command line. Some names such as TOP can refer to a parameter or a command option.



- The line, which now should correspond to the format given in the reference section, is executed. The command must be the first field and may be followed by options and data.
- Multiple command lines may be included on a single input line using the ^ as a logical end-of-line indicator.

### 3.1.2 Numeric input

Certain commands (such as XDATA or YDATA) tell **VP** to look for numeric values in the following lines. In this case, successive lines are scanned for numbers until a nonnumeric field is encountered. If this nonnumeric field is the first field on a new line, that line is taken as the next command line. If the nonnumeric field is not the first field, then the remainder of the line is ignored and a new command line is assumed to start on the following line. Numeric input, including parameter values, must obey the following rules:

- Numbers use free format notation without embedded spaces.
- A decimal point may be included but is not required.
- An optional `E` or `D` may be followed by an integer value for standard scientific exponential notation. Thus, 100 may be entered as `1E2` or `10.0d1`.
- Hexadecimal numbers begin with a dollar sign ($) (such as `$13` or `$2EF0`), with an underline (_) (such as `_13` or `_2EF0`), or with C-language notation `0x` (such as `0x13` or `0x2EF0`).
- An arbitrary count of numbers may be included on each input line for the XDATA, YDATA, ZDATA, and XYDATA commands.
- Simple binary operator numeric calculations may be included when numeric input is expected.
  - The allowed binary operations are: + = * /.
  - Binary operators are evaluated from left to right. For example, `"1+3/4"` gives 1, but `"3/4+1"` gives 1.75.
  - The expression string must not include any spaces or parentheses. For example, if `A`,`B`,`C`,`D`, and `E` are numeric character strings, an input of the form `A*B/C+D-E`, will be evaluated properly but `A*B/C+(D-E)` will not. The parentheses will cause the entire field to be treated as non-numeric.

This is particularly useful with synonym and parameter substitution, as discussed later in the chapter. It allows numeric values relative to parameter positions. For example:

```
COMMENT &xsize./2 &ysize./2 inch center
COMMENT &xsize.-0.5 &ysize.-0.5 inch upper right
```

will always place a comment in the middle of the axis box and in the upper right corner.

- The more powerful form of {...} and [...] calculations will be discussed in Section 3.2.

### 3.1.3 Text Input

Other commands (such as COMMENT or LABELS) tell **VP** to accept the following lines as alphanumeric text without breaking them up into individual fields, without converting to uppercase, without recognizing the logical end-of-line character ^, and without recognizing the semicolon (;) as the start of trailing comments.

It is possible to include parameter values and synonym definitions in the text. See Section 3.4 for a summary of this and other special text-processing features.

### 3.1.4 Interrupting execution

During file READs, **VP** execution may be interrupted by hitting the `Ctrl-C` or `Break` characters. At this point **VP** will display the message

```
interrupt: Q=quit all reads, E=exit, otherwise continue:
```

Depending on the character entered, processing will continue, all READs will be halted, or **VP** will exit. With DOS, a check is made for an interrupt only as each new line is read, and not during execution of any command.

## 3.2 Numeric Calculations

Arbitrary numeric calculations may be used to define **VP** synonyms and parameters and data manipulation before plotting. Expressions inside square brackets [...] are evaluated in HSILOP (reverse POLISH) notation, while expressions within curly brackets {...} are evaluated in standard notation with operator priority determined solely by parentheses.

Thus, to define the real parameter PI as 3.1415 . . . , you may use either

```
define pi [ 1 ATAN 4 * ]      ; in HSILOP notation
define pi { 4*ATAN(1)  }      ; in normal notation
```

Such expressions may be used in **VP** command lines whenever numeric input is expected. All expressions are evaluated as 8-byte real numbers. For example,

```
axis xmin { EXP(-1) } xmax { EXP(3) }
```

can be used to define the x-axis range.

Parameter values may be used in numeric calculation simply by including their symbolic name,

```
comment { xmin+SQRT(2) } { ymin**0.5 }
```



The & and . delimiters that are used to include parameter values in text (&Sname.) are not needed inside [...] or {...} calculations; their use causes rounding to NDECPL places. They may be used for synonyms.

Successive numeric calculations do not need individual sets of [...] or {...} brackets, so

```
comment { xmin+SQRT(2)  ymin**0.5 }
```

which places two values on the stack, is the same as the previous line of code.

XDATA, YDATA, ZDATA, and TABDATA values may also be used in calculations. Here, the colon (:) operator acts as subscript notation in either HSILOP [...] or normal {...} calculations.

```
print { 1:X  2:X  10:X }  ; prints 1st, 2nd, and 10th XDATA
print { nmbr:YDATA }      ; prints the last YDATA value
print [ 3:C1  3:C2 ]      ; prints 1st and 2nd columns
                          ;   in TABDATA row 3
```

When used with XDATA, YDATA, ZDATA, and XYDATA column commands the calculation delimiters have an implied loop. It is best to specify this loop size explicitly with the LOOP parameter *on the same line* as the calculations. If LOOP is not given *on the same line*, **VP** will attempt to guess the implied loop length (NMBR or NCOL or NROW). This allows arbitrary redefinition of data such as

```
xdata  { c1 } loop &nrow.          ; same as XDATA COLUMN 1
ydata  { c2/c3 }                   ; ratio of column 2 to 3
ydata  { r2/r3 } loop &ncol.       ; ratio of row 2 to 3
xydata { i SQRT(i) } loop &nmbr.   ; Y=SQRT(X) for X=1-nmbr
```

The implied calculation loop can also be used with the CURVE command for arbitrary plots with no restrictions on the use of X or Y. When used with CURVE (or the DRAW synonym), two calculated values define the actual X and Y data to be plotted. This data does *not* replace the current XYDATA:

```
curve  { x SQRT(y) }                  ; plot X vs SQRT(Y)
curve  { y (SIN(pi*x))**2.5 }         ; X and Y switched
draw line { i 3*SQRT(i) } loop 100    ; purely calculated
```

Similarly, calculations can be used to supply the four values required by the LINE command (as the endpoints `xmin ymin xmax ymax`) with an implied loop,

```
line box { X-0.5 0 x+0.5 Y } loop &nmbr.  ; line spectra
```

or calculations can be used to supply the six values required by the XYERRDATA command (as `x y xmin xmax ymin ymax`) with an implied loop as in `demoerrb.exg`:

```
dot red              ; set color and symbol type with synonyms
xyerrdat {c1 c2 c1 c1 c2-(0.12*c1) c2+(0.19*c1)} loop &nmbr.
```

When 2 calculated values are supplied for XDATA, YDATA, ZDATA, or when 3 values are supplied for XYDATA or CURVE, or when 7 values are supplied for XYERRDATA, the first value is taken as a logical compression operator (TRUE if INT(value) ≠ 0 and FALSE if INT(value) = 0) for inclusion of the remaining item(s). Thus:

```
xdata  {c1>0 c2} loop &nrow. ; c2 when c1 > 0
xydata {ABS(c1)>0.01 C2 C3}  ; column 2 3 when ABS(c1)>0.01
draw dot {c1==2 C2 C3}       ; plot column 2 3 when (c1)=2
```

This latter format is a particularly powerful method of selecting and plotting subsets of tabular data. Samples of such sophisticated calculations are found in the sample files `democalc.exg` and `demomac.exg`.

Named arrays from the ARRAYDAT command may be used in calculations like TABDATA. Several special attributes may be accessed directly:

```
{ name.min }   min value in array name
{ name.max }   max value in array name
{ name.nmbr }  number of points in array name
{ name.ncol }  number of columns in array name
{ name.nrow }  number of rows in array name (nmbr/ncol)
{ name.Cn }    column n from array name
{ name.Rn }    row n from array name
{ name }       the complete array name as a 1D array
```

TABDATA, XDATA, YDATA, ZDATA may be used for name with these .attributes.

**VP** performs all numeric calculations in HSILOP notation on an internal stack. The stack size is 20. 40 numeric constants are allowed, and the calculation operator stack size is 256. Up to 32K values may be indexed for TABDATA row and column. Normal notation is converted to HSILOP before evaluation. Character synonym substitution of names delimited by & . (&Sname.) is performed before any attempt to evaluate the expression. HSILOP {...} calculations are always performed from left to right. An operator modifies the top stack contents. Constants or variables add a value to the stack. Parentheses "(" and ")" should be used in normal calculations {...} to indicate operator priority. Without parentheses all operators have equal priority and are evaluated from left to right.

```
print { 1 + 3 * 4 }          ; gives (1+3)*4 = 16
print { SQRT 4*4 }           ; gives SQRT(4) * 4 = 8
```

The allowed operations and their effect on the stack are summarized in the following tables. $S_0$ is the top stack item and $S_{-1}$ is the previous item.



| Symbol | Operation performed and its effect on the stack | |
|---|---|---|
| | **Binary operations use the two top items and reduce stack size by 1** | |
| + | $S_{-1} + S_0 \Longrightarrow S$ | (addition) |
| - | $S_{-1} - S_0 \Longrightarrow S$ | (subtraction) |
| * | $S_{-1} \times S_0 \Longrightarrow S$ | (multiplication) |
| / | $S_{-1}/S_0 \Longrightarrow S$ | (division) |
| ** | $S_{-1}^{S_0} \Longrightarrow S$ | (exponentiation) |
| MAX | $MAX(S_{-1}, S_0) \Longrightarrow S$ | (largest of two values) |
| MIN | $MIN(S_{-1}, S_0) \Longrightarrow S$ | (smallest of two values) |
| | **Logical operations use the top stack item, return 1 if TRUE, 0 if false, and reduce stack size by 1** | |
| ==, =, or EQ | 1 if $S_{-1} = S_0$, else $0 \Longrightarrow S$ | |
| <> or NE | 1 if $S_{-1} \neq S_0$, else $0 \Longrightarrow S$ | |
| >= or GE | 1 if $S_{-1} \geq S_0$, else $0 \Longrightarrow S$ | |
| <= or LE | 1 if $S_{-1} \leq S_0$, else $0 \Longrightarrow S$ | |
| > or GT | 1 if $S_{-1} > S_0$, else $0 \Longrightarrow S$ | |
| < or LT | 1 if $S_{-1} < S_0$, else $0 \Longrightarrow S$ | |
| AND | 1 if $INT(S_{-1}) \neq 0$ *and* $INT(S_0) \neq 0$, else $0 \Longrightarrow S$ | |
| OR | 1 if $INT(S_{-1}) \neq 0$ *or* $INT(S_0) \neq 0$, else $0 \Longrightarrow S$ | |
| | **Unitary operations replace the top item on the stack and leave the stack size unchanged** | |
| SIN | $sin(S_0) \Longrightarrow S$ | |
| COS | $cos(S_0) \Longrightarrow S$ | |
| TAN | $tan(S_0) \Longrightarrow S$ | |
| ATAN | $atan(S_0) \Longrightarrow S$ | |
| SQRT | $sqrt(S_0) \Longrightarrow S$ | |
| EXP | $e^{S_0} \Longrightarrow S$ | |
| LN | $log_e(S_0) \Longrightarrow S$ | |
| LOG or LOG10 | $log_{10}(S_0) \Longrightarrow S$ | |
| ABS or \| | $abs(S_0) \Longrightarrow S$ | (absolute value) |
| INT | $int(S_0) \Longrightarrow S$ | (integer part) |
| SINH | $sinh(S_0) \Longrightarrow S$ | |
| COSH | $cosh(S_0) \Longrightarrow S$ | |
| TANH | $tanh(S_0) \Longrightarrow S$ | |
| ATAN2 | $atanh(S_1, S_0) \Longrightarrow S$ | |
| ASIN | $asin(S_0) \Longrightarrow S$ | |
| ACOS | $acos(S_0) \Longrightarrow S$ | |

Continued on next page.

| Symbol | Operation performed and its effect on the stack |
|---|---|
| | Other names and constants add a value to the stack and increase the stack size by 1 |
| num. const. | numeric value ⟹ *S* |
| I | current INDEX value ⟹ *S* |
| X | XDATA[index] ⟹ *S* |
| Y | YDATA[index] ⟹ *S* |
| Z | ZDATA[index] ⟹ *S* |
| Cn | TABDATA column n row INDEX ⟹ *S* |
| Rn | TABDATA column INDEX row n ⟹ *S* |
| pname | add value of parameter pname ⟹ *S* |
| Aname | ARRAY[index] ⟹ *S* (from ARRAYDATA) |
| Aname.min | minimum value in ARRAY ⟹ *S* |
| Aname.max | maximum value in ARRAY ⟹ *S* |
| Aname.nmbr | number of values in ARRAY ⟹ *S* |
| Aname.ncol | number of columns in ARRAY ⟹ *S* |
| Aname.nrow | number of rows in ARRAY ⟹ *S* |
| Aname.Cn | (column n in ARRAY)[index] ⟹ *S* |
| Aname.Rn | (row n in ARRAY)[index] ⟹ *S* |

Two special operators use the same sequence in both HSILOP and normal notation. The colon (:) pops the top stack element for use as a subscript on the following variable (if allowed). Thus, `{10:X}` specifies XDATA(10). The characters => store the top stack value in the following REAL or INTEGER parameter. Thus, `{XDMIN+1 => XDMIN}` increments the value of XDMIN.

Unrecognized alphanumeric names are given a numerical value based on a left-justified, 8 character, uppercase number algorithm that can be used for logical operations. This feature is often used for testing case-independent equality of text strings:

```
read wait Hit ENTER to continue or Q to quit:   ; prompt
if { &%1.Z = QZ  } read END                     ; evaluate
```

Note that the simpler calculation `{ &%1. = Q }` would not work correctly in the above example. If the user responds with ENTER to the prompt in READ WAIT, the synonym %1 will be blank and the construction `{ &%1. = Q }` reduces to `{ = Q }`, which is incorrect.

## 3.3 Synonyms

User-defined SYNONYMs customize `.exg` file processing and simplify **VP** operation with easy-to-remember mnemonics.

- A synonym is an arbitrary character string that may be referenced by its name.



- Once defined, a synonym is global to the **VP** session. Synonyms defined in an `.exg` file remain valid after the file is complete.
- Synonyms may be defined anywhere on a command input line with the keyword DEFINE followed by the synonym name and its text definition:

```
define name1 ok
define name2 'Scotty'
define name3 "beam me up"
```

After the above definitions, the input line `read name2` is interpreted as `read Scotty`. The single or double quotes are necessary when the synonym text contains normal delimiters such as spaces.

- If the synonym definition field is a hexadecimal number beginning with a $, the synonym will consist of the single corresponding ASCII character. Thus,

```
define hat $5E
```

will define hat as the ^ character. Since the ^ is the logical end-of-line character, this is a convenient way to include it in text. Normal numeric input may also use an underline (_) or 0x for hexadecimal.

- Synonyms can be redefined with a `DEFINE Sname NewText` sequence.
- Synonyms can be removed with a null definition `DEFINE Sname ''`. The `RESTART DEFAULT` command also removes any new synonyms defined since the last `RESTART SAVE`.
- A maximum of 128 synonyms may be defined (96 under DOS), including those in `profile.exg`.
- The command `PRINT SYNONYM` may be used to examine the current synonym names and their definitions.

### 3.3.1 %0 ... %8 synonyms

- A single special set of %n synonyms, %0, %1, ..., %8, is used for passing parameters to `.exg` files. When the READ command is encountered (either explicitly or implicitly), these parameters are defined from the remaining fields on the command line. Thus, either

```
read testcs sample one   ; explicit READ or
\vp\testcs sample one    ; implicit READ in DOS
```

will define %1 as `'sample'` and %2 as `'one'`, while %3–%8 remain blank during the processing of `testcs.exg`. %0 is defined as the name of the file being read without any path prefix (here, `testcs.exg`). Any explicit path prefix is saved in the PATHEXG synonym (\vp\ in the 2nd case). If `testcs.exg` contains the command

```
print % pathexg
```

the values of the %n synonyms and PATHEXG will be displayed.

- The %n synonyms may be included in LABELs and COMMENTs with &%n. where both the '&' and '.' are necessary.
- Since there is a single global set of %n synonyms that are redefined with each new READ, it may be necessary to "save" a %n synonym for use outside its .exg file by

```
DEFINE save1 &%1.
```

- The initial values of the %1–%8 synonyms are defined from the first 8 fields (excluding options) of the initial **VP** command. These fields are then available for use in the profile.exg or userprof.exg files for batch processing.

### 3.3.2 PROMPT synonym

The PROMPT synonym contains the **VP** prompt text when keyboard command line input is requested. For example, one can use

```
DEFINE PROMPT 'hit me: '      ; for "hit me:" as prompt
DEFINE PROMPT ' '             ; for blank prompt.
```

to change or completely eliminate the prompt message.

### 3.3.3 MSGFILE synonym

The MSGFILE synonym can be set to place **VP** output messages in a file (or NULL) rather than displaying them at the console. For example:

```
DEFINE MSGFILE batch.msg
```

will place all messages in the file batch.msg. This is useful for examining progress of a batch job. The combination

```
DEFINE MSGFILE null
DEFINE PROMPT ' '
```

will discard most **VP** console messages.

### 3.3.4 Defining new REAL parameters

Although all synonym names correspond to character strings, the DEFINE name text sequence can also be used to create new REAL parameters. When the text field begins with a plus (+) and text can be evaluated as a number, or when text is a [...] or {...} numeric calculation, then name becomes a REAL parameter with the corresponding value rather than a character synonym. For example,

```
define pi 3.14159     ; creates SYNONYM pi as '3.14159'
define pi +&pi.       ; creates REAL pi and removes synonym
define pi +3.14159    ; also creates REAL pi
define pi [3.14159]   ; also creates REAL pi
```



The resulting user-defined REAL parameters, like synonyms, may be removed with a null definition DEFINE pi ''. A given name can never refer to both a synonym and a real parameter. The most recent DEFINE determines the type.

Some commands (such as FIT and AVERAGE) may define additional REAL parameters as the result of processing or graphics input. These are discussed in more detail in Section 6.3.

### 3.3.5 Including synonyms and parameter values in text

Parameter values or synonym definitions may be included in any **VP** text line (such as those processed by the COMMENT or LABELS commands) by delimiting the names between an ampersand (&) and a period (.). This is particularly useful for including the FIT results SLOPE and Y0 in the text of a comment. Thus, for example, to include the current value of parameter XMAX in a comment, use

```
comment cursor              ; position comment with cursor
XMAX has the value &xmax. in file &%0.
```

The current value will be substituted for &xmax. with NDECPL decimal places while the current file name will replace &%0.. A similar technique can be used to equate parameters.

```
ndecpl 1              ; portray REALs with 1 decimal place
ymax = &xmax.         ; set ymax to NDECPL version of xmax
ymax   [xmax]         ; numeric equality
```

The latter example (ymax [xmax]) provides a numeric identity, while the former (ymax = &xmax.) first creates the character representation of XMAX to 1 decimal place, then evaluates the characters as the new numeric value for YMAX. The = sign in the example is unnecessary.

Two special synonyms, &date. and &time., will be replaced by the current date and time in any label or comment. Section 6.3 describes other special and default synonyms.

### 3.3.6 Synonyms for graphic symbols and lines

Many of the default synonyms described in Section 6.3 provide easy-to-remember names for the graphic symbols and line types produced by **VP**. These are summarized in Fig. 3.1.

| ISYM | Synonym | Graphic |
|---|---|---|
| −1 | nosym | |
| 1 | dot | • |
| 2 | diamond | ♦ |
| 3 | triangle | ▲ |
| 4 | square | ■ |
| 5 | del | ▼ |
| 6 | arrow | → |
| 7 | plus | + |
| 8 | xmark | × |
| 11 | odot/circle | ○ |
| 12 | odiamond | ◇ |
| 13 | otriangle | △ |
| 14 | osquare | □ |
| 15 | odel | ▽ |
| 16 | oarrow | → |

| ILINE | Synonym | Graphic |
|---|---|---|
| −1 | noline | |
| 1 | connect | •——• |
| 2 | dotted | •·····• |
| 3 | dashed | •– – –• |
| 4 | dotdash | •–·–·• |
| 11 | line | —— |
| 12 | | ····· |
| 13 | | – – – – |
| 14 | | –·–·– |

Line types 1–4 produce lines with space for symbols if ISYM ≥ 0 and continuous lines if ISYM < 0.
Line types 11–14 ALWAYS omit symbols.
The PATTERN command changes the pattern for ILINE 2–4, 12–14.

Synonyms following CURVE reset ISYM and ILINE before plotting.

Since the CURVE command uses the global ISYM and ILINE, it is best to specify BOTH ISYM and ILINE synonyms after CURVE or use the DRAW synonym to reset ISYM and ILINE:

| | | |
|---|---|---|
| CURVE nosym dotted | or | DRAW dotted |
| CURVE square noline | or | DRAW square |
| CURVE dot connect | or | DRAW dot connect |

**Figure 3.1: VP** default synonyms for graphics elements.

## 3.4 Text Processing

All **VP** text is treated as Hershey fonts made up of arbitrarily sized stroke characters for immediate display and hard copy. The **u-ps** utility can be used to convert the Hershey fonts to PostScript fonts as described in Section 7.5. Fonts are specified to **VP** by a 5-digit number, FFSSS, that determines both the font style and size. FF is the 2-digit Hershey font number shown in Fig. 3.2. Any other font number will simply display in the current font. A negative font number gives characters that are rotated 90 degrees. The 3 digits SSS give the vertical size of the characters in thousandths of an inch. The parameter FONTWIDTH gives the width relative to height. A FONTWIDTH of 0.4 to 0.5 is a good choice.



### 3.4.1 Fonts

*VP/UGH* Hershey font samples

| | | | |
|---|---|---|---|
| 12 | Roman Simplex | *22* | *Script Complex* |
| *14* | *Script Simplex* | 24 | **Roman Triplex** |
| 15 | Roman monospace | *25* | ***Italic Triplex*** |
| 18 | **Roman Complex** | 26 | **German Gothic** |
| 20 | ***Italic Complex*** | 27 | **English Gothic** |
| 21 | **Roman Duplex** | 28 | **Italian Gothic** |
| 30 | APL monospace | 31 | APL proportional |

**Figure 3.2:** The Hershey font styles recognized by **VP** drawn with FONTWIDTH 0.5.

### 3.4.2 Text processing features

**VP** provides many special text-processing features that may be used whenever text input is expected. These include Greek characters, subscripts, superscripts, underlined text, secondary font, special characters, formatting, and others. Most of these features are obtained by using *special characters* to delimit the desired text. Some of the special characters may cause problems with DOS keyboard extensions that monitor keyboard input. There should, however, be no problem with `.exg` file input.

Some of the text-processing features are provided by the low-level UGH driver commands described in Section 7.1. The remainder are part of **VP**. This separation, however, should be transparent to the user. Figure 3.3 demonstrates these features, which are described in more detail below the figure. The TEXTBOX command is described in detail in Reference Section 5.31.

The UGH driver provides the following text features:

Greek letters
: Delimited by exclamation points (`!c!`). Each font includes a set of Greek equivalents, along with many special symbols. Figure 3.4 shows many of the character translations.

Subscripts
: Delimited by single quotes (`'c'`).

Superscripts
: Delimited by double quotes (`"c"`).

Arbscripts
: Delimited by back quotes (`‘c‘`).

*VP/UGH* Text-processing features

| | | | |
|---|---|---|---|
| ! ! | delimit Greek | !abc Greek! | → αβχ Γρεεκ |
| " " | delimit superscripts | Text"upper !u!" | → Text$^{\text{upper } \upsilon}$ |
| ' ' | delimit subscripts | Text'lower !!!' | → Text$_{\text{lower } \lambda}$ |
| _ _ | delimit underlining | _under_ | → under |
| \ \ | delimit C2FONT text | Text \C2font\ | → Text *C2font* |
| \| \| | delimit box highlights | \|boxed\| | → boxed |
| ? ? | delimit VP symbols | ?isym.iline.color? | |
| | | a) ?4.1.2? | → a) ■–■ |
| | | b) ?12.–2.4? | → b) ◊···· ◊ |

*TEXTBOX* adjusts the following text to fit just inside the box margins with optional right or left justification

**Figure 3.3:** VP/UGH text-processing features.

*VP/UGH* Greek-character translation table

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| a α | n ν | A Å | N ∇ | 0 ° | / ÷ | : ∩ | |
| b β | o ο | B ‖ | O ∞ | 1 ∮ | * × | ; ∪ | |
| c χ | p π | C ⊥ | P ∏ | 2 ∏ | − ÷ | '' ⊃ | |
| d δ | q ϑ | D Δ | Q φ | 3 ∑ | + × | `` ⊂ | |
| e ε | r ρ | E ~ | R ∝ | 4 □ | > ≧ | " " © | |
| f ϕ | s σ | F Φ | S Σ | 5 ◊ | < ≦ | !! ∈ | |
| g γ | t τ | G Γ | T × | 6 ↑ | = ≠ | ~~ ∄ | |
| h η | u υ | H ħ | U ⊤ | 7 ↓ | ( ⟨ | ## ✲ | |
| i ι | v ψ | I ∫ | V ψ | 8 → | ) ⟩ | \\ ∓ | |
| j θ | w ω | J Θ | W Ω | 9 ← | [ ≤ | ?? * | |
| k κ | x ξ | K ∂ | X Ξ | % o | ] ≥ | ‖ ∃ | |
| l λ | y ψ | L Λ | Y ψ | & ▾ | { ⟨ | @@ ◯ | |
| m μ | z ζ | M √ | Z ± | , • | } ⟩ | __ ≡ | |

**Figure 3.4:** VP/UGH Greek characters and special symbols.



Arbscripts are like sub- and superscripts but with complete control over size and baseline offset. They can be used to provide subscripts within superscripts or just different size characters within a text string. The default arbscripts are the same size as sub- and superscripts (75% of normal) but with no baseline offset. To change the characteristics of all following arbscripts, begin an arbscript with the sequence `~siz+off+dx~` or `~siz-off-dx~`, where `siz` is the arbscript size in 32nds of font size, and `+off` or `-off` is the baseline shift in 32nds of arbscript size. The third numeric value, `+dx` or `-dx`, gives an X shift in 32nds of the font X size for this arbscript only. The `+-dx` may be omitted. For example, to set all following arbscripts to act like superscripts, include `'~24+18~'`; to have arbscripts imitate subscripts, include `'~24-10~'`. To use arbscripts as subscripts within superscripts, try `'~16+10~'`. If a `~` does not immediately follow the `'`, it indicates a `color+font` change as below.

The `~siz+off~` specification can also be used immediately after a `'` or `"` to change the characteristics of all following sub- and/or superscripts.

In the following example

`e"-E'~16+10~i/k'B"T"` produces $e^{-E_i/k_BT}$

note that the first occurrence of any type of quote (`' " '`) always activates sub-, super-, or arbscripts even if the text was already in one of the other scripts.

Color+Font+Size change
: Specified within tildes (`~`).

  The format is `~color+font+siz±off~`. Thus, `~6~` changes to color 6 for the remainder of the text line, while `~6+24~` changes to color 6 and font 24 for the remainder. Size and offset may also be changed (in units of 32nds of the normal font size) as with arbscripts above.

  Remember that the usage is different when a `~` immediately follows a `'`. See **arbscripts** above.

CR/LF
: Provided by `@`. It also sets Greek, subscript, superscript, and secondary font modes off.

Backspace
: Provided by `#`. It allows overwriting of characters and production of special characters.

Underlining
: Delimited by underscore and tilde (_~col+wid-bot~ ..._). Draws a line of color col and width wid a distance bot (in 32nds of the font size) below the text between underscores. See "underlining" below for **VP**'s implementation of a simpler form of underlining.

Boxing
: Delimited by vertical bars and tilde (|~col+wid-bot+top~ ...|). Draws a box of color col and width wid a distance bot below and top above (in 32nds of the font size) the text between vertical bars. The box also extends a distance bot before and after the text. If wid<0, a filled box is drawn before the text is written. See "boxing" below for **VP**'s implementation of a simpler form of boxing.

The following text features are part of **VP**:

Underlining
: Delimited by underscore (_c_). All text between underscores is underlined by a line in the color specified by the parameter ULCOLOR. The actual position will vary with the parameter ULSPACE. **VP** adds ~col+wid-bot~ specifications to produce the UGH underlining format (see "underlining" above).

Boxing
: Delimited by vertical bars (|c|). All text between vertical bars appears enclosed within a box of color ULCOLOR. The actual position will vary with the parameter ULSPACE. **VP** adds ~col+wid-bot+top~ specifications to produce the UGH boxing format (see "boxing" above). If ULCOLOR < 0, the box will be solid in color ABS(ULCOLOR). The use of solid boxes is not recommended for hard copies unless different colors are supported. ULCOLOR=0 (background) can be used to "erase" any existing graphics in the box around the text.

Secondary font
: Delimited by backslashes (\c\). All text between backslashes is written in font C2FONT and color C2COLOR. Secondary font text will not work properly if C2FONT does not refer to an allowed font number in the form FFSSS discussed in Section 3.4.1. **VP** adds ~color+font+siz+off~ specifications to produce the UGH color/font/size format (see "Color+Font+Size change" above).

Multiple lines
: When a COMMENT text line is ended by + or @, the following line is also considered part of the same COMMENT and will be placed a distance LSPACE (times font size) below the first line.



```
comment 9 9 right
This text comment+
consists of three lines@
right justified
```

**Warning:** The following **VP** text features will not work properly with the rotated text in vertical comments or Y AXIS labels.

Symbols/lines
: Delimited by question marks (?c?) as ?isym.iline.color? where isym is the symbol number, iline is the line type, and color is the color of the included symbols. If iline<0, a longer line will be drawn of type -iline. See Fig. 3.1 for examples. The symbol size will be determined by the parameter SMULT, and any line widths will be controlled by the parameter WIDTH.

  **Warning:** Symbol inclusion will not work properly if XMULT and YMULT are not 1.0.

Tabs
: The TABS command (described in Section 5.30) can be used to set tab positions for horizontal labels. Any character may be used as the tab delimiter.

Box formatting
: Use the TEXTBOX command to format text inside a user-specified box. This command also provides a set of word-processor-like control commands to allow flexible text formatting, as described in detail in Section 5.31.

### 3.4.3 Including delimiter characters in text

Any of the standard text delimiters (' " ` ! ? | \ ~ _) may be included in text by doubling them ('' "" `` !! ?? || \\ ~~ __).

The ASIS option of the LABELS and COMMENT commands can also be used to stop any **VP** special character processing and include the **VP** special characters without doubling.

Similarly, the UGH special character processing can be suppressed by including the string ~-13~ at the start of each line.

### 3.4.4 Creating special characters

Overwriting symbols with the backspace character (#) can be used to create special characters. For example, a copyright symbol can be approximated by:

```
comment x y
C#!@@!
```

Although the copyright symbol is not part of the Hershey fonts, **u-ps** UGH-to-PostScript conversion can transform this sequence back to a standard PostScript character for hard copy. See Section 7.5 for more details.

## 3.5 Macros, Loops, and Conditional Execution

Internal macros may be defined for short sequences of frequently used commands with argument substitution. Such Macros act like internal .exg files. Once defined, they may be used like any other **VP** command. The following example illustrates the use of a MACRO to select several TABDATA columns to plot and fit:

```
macro pfit                    ; defines a new macro called PFIT
  xydata column &%1. &%2.     ; XYDATA from specified columns
  draw dot color &%3.         ; plot in specified color
  fit                         ; least-squares fit
  return                      ; end of macro definition
                              ;
pfit 1 2 6                    ; plot X,Y as columns 1,2, color 6
pfit 3 4 5                    ; plot X,Y as columns 3,4, color 5
```

- There is a limited amount of internal storage for macros and at most 32 macros may be defined at one time.
- The RESTART DEFAULT command does not remove macro definitions. To remove all macro definitions issue the RESTART MACRO command.
- The PRINT **pname** command may be used to see the stored macro definition of **pname**. PRINT MACRO lists all current macro names.

In a similar manner, REPEAT-END loops may be used to repeat a sequence of commands a specified number of times. During successive iterations the integer parameter COUNT is incremented by 1, starting with value 1 on the first iteration. This is used as the default index (subscript) for [...] and {...} calculations with the XDATA, YDATA, TABDATA, and ARRAYDATA commands. Outside of a REPEAT-END loop, COUNT has the value 0.

The REPEAT command is implemented by defining an internal macro named REPMACRO and calling the macro the specified number of times while incrementing COUNT.

The following example illustrates the use of a REPEAT-END loop to plot all TABDATA columns after the first in alternating red and green colors:

```
tabdata
                              ; define the TABDATA array
xdata column 1                ; always use column 1 for xdata
color 4                       ; start with color 4 = red
repeat &ncol.-1               ; repeat count = NCOL-1
  ydata column &count.+1      ; take YDATA from next column
  color &color.+2             ; increase color number by 2
  if { color > 4 } color 2    ; reset from red to green
  draw dot color &count.      ; plot as DOTs in the next color
```



```
  if { color = 2 } then       ; print message for GREEN
    echo column &count.+1 drawn in green
  else                        ; print message for RED
    echo column &count.+1 drawn in red
  endif
end                           ; end of loop
```

The above example also illustrates the two forms of the IF command. With the simple form

```
IF  { calc }  command         ; or
IF  [ calc ]  command
```

command will be executed if the result of the normal {...} or HSILOP [...] calculation is TRUE (i.e., INT(calc)≠0). With the compound form

```
IF {...} THEN          ; or   IF [...] THEN
  command T1           ; T1 - Tn are executed if {...} true
  ...
  command Tn
ELSE
  command F1           ; F1 - Fm are executed if {...} false
  ...
  command Fm
ENDIF
```

either the TRUE commands T1 - Tn or the FALSE commands F1 - Fm are executed. In the compound form THEN, ELSE, and ENDIF must be spelled completely. Moreover, only one level of IF commands is supported. Nested IF statements will yield unpredictable results.

## 3.6 Graphical Input

Many **VP** commands that accept positional input such as XYDATA or COMMENT allow the CURSOR option to indicate interactive graphic positioning. A cursor is displayed on the graphics screen, and a mouse or the keyboard (on PCs) can be used to change its position. A mouse button or keyboard character is pressed to indicate the desired position.

With PC DOS, mouse positioning can be used only if a standard mouse driver (`mouse.com` or `mouse.sys`) has been loaded and responds to interrupt 33H. Keyboard movement control is not activated under X-windows.

If the RECORD command has been issued, any cursor coordinates will be saved in graph (as opposed to screen) units in the specified record file. If the INCHES option is also specified, the saved coordinates will be in inches relative to the AXIS box's lower left corner. If PAGE is specified, the saved coordinates will be in inches relative to the lower left page corner.

**Keyboard controls** are available with PC UGH drivers:

- **Movement** is accomplished by the usual cursor arrow keys on the PC keyboard. In normal mode they jump the cursor by a large amount (default 16 screen pixels) with each hit. Shifted (uppercase) cursor keys on the numeric keypad (or the corresponding numeric keys 4 ←, 6 →, 2 ↓, 8 ↑ ) give fine positioning by a single pixel. The **Alt-F** and **Ctrl-F** keys can also be used to change the step amount.
- Pressing a **normal character** ends cursor movement and returns the current position and character hit.
- Pressing **Q** usually terminates the input operation.
- Pressing **V** when using COMMENT CURSOR causes preview drawing of the comment at the current position. The next character erases the preview and returns to cursor positioning of the text.
- **Alt-S** and **Alt-C** change the motion of rubber lines and boxes on the screen. **Alt-S** will toggle the movable corner, and **Alt-C** will toggle between corner movement and displacement of the entire line or box.
- **Alt-F** will halve the size of the large movement step, while **Ctrl-F** will double the step size.
- **Alt-X** will halve the display size of the cursor crosshair, while **Ctrl-X** will double the crosshair size.

**Mouse and trackball** control:

- **Movement** is, of course, tracked from the mouse movements.
- Mouse buttons, which are read upon release, represent the following keyboard conventions (see above):

| | |
|---|---|
| LEFT | Return position as though the ENTER key were hit. |
| MIDDLE | **Q** if present. Quit current operation. |
| RIGHT | **V** For COMMENT CURSOR, toggle temporary view. For rubber lines and boxes, cycle between **Alt-S** and **Alt-C**, (i.e., switch movable ends and the center). |
| LEFT+RIGHT | **Q** (Both buttons hit simultaneously). Same as MIDDLE; used for two-button mice. |

The following table summarizes control of graphical input:



| Action | Keyboard | Mouse |
|---|---|---|
| cursor movement | arrow keys | just move it |
| fine positioning | numeric pad arrow keys | just move it |
| terminate (QUIT) | Q | middle button or left+right |
| COMMENT CURSOR preview | V to view | right button toggles draw or erase |
| return position | any character | left button |
| rubber lines or boxes<br>toggle moving end<br>toggle center vs. end | <br>Alt-S<br>Alt-C | <br>right button (cycles)<br>right button (cycles) |
| halve step size<br>double step size | Alt-F<br>Ctrl-F | <br>not applicable |
| halve X-hair size<br>double X-hair size | Alt-X<br>Ctrl-X | <br>not applicable |

## 3.7 Understanding VP and UGH Coordinate Systems

This section describes the multiple coordinate systems used by **VP** and UGH for locating text and graphics.

**AXIS** These are the real number coordinates in which user data are specified to **VP**. The AXIS or BOXLIMIT command sets up the transformation (which may, for example, include LOGs) to the virtual integer units used by UGH.

**INCHES** These are the real number coordinates in which comments and other text may be specified when the INCHES option is given. INCH coordinates are relative to the lower left corner of the AXIS box. Their transformation to UGH units is set by the AXIS or BOXLIMIT command and the PIXPIN (PIXels Per INch) parameter.

**INCH PAGE**
These are the real number coordinates in which comments and other text may be specified when the INCH PAGE option is given. INCH PAGE coordinates are always relative to the lower left corner of the virtual page. Their transformation to UGH units does not depend on AXIS box placement, but it does depend on the PIXPIN parameter.

**CMS** For those of metric persuasion, CMs (centimeters) may be used instead of INCHES or INCH PAGE.

**Virtual UGH**
This is the integer coordinate system used by UGH for storing and displaying graphics data. It is a 64K by 64K virtual space

(derived from a 16-bit integer). A mapping of this space onto inches is conventional, with the PIXPIN parameter specifying the number of virtual units per inch. The **VP** default for PIXPIN is 800. When combined with the number of pixels/inch on the output device, it allows explicit control of the display scaling in familiar units. In many cases, the default value makes one virtual inch (800 units) correspond to one real inch on output devices such as plotters and printers.

**Display** This coordinate system is device-dependent and represents the pixel mapping of the device, with (0,0) being the lower left corner. For example, the PC VGA's display coordinates run from 0 to 639 in X and from 0 to 479 in Y.

The AXIS or BOXLIMIT command defines all necessary mappings to virtual UGH units. The TOP, YSIZE, BOTTOM, LEFT, RIGHT, and XSIZE parameters define the position and size of the AXIS box in inches (or centimeters) on a virtual page as shown in Fig. 3.5 below. Either the current XYDATA limits (via the parameters XDMIN, XDMAX, YDMIN,

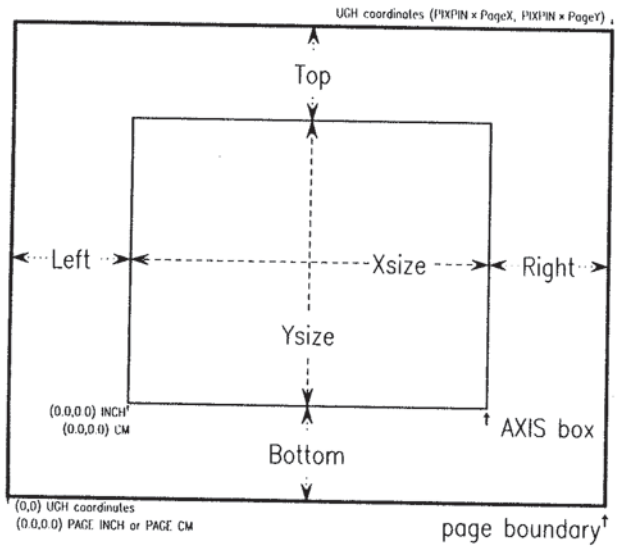


**Figure 3.5: VP** parameters determine AXIS box size and placement on an imaginary graphic page.



and YDMAX) or specifically set XMIN, XMAX, YMIN, and YMAX determine the mapping of user AXIS units onto the AXIS box. A specific example may be useful. Consider the following simple `.exg` file:

```
pixpin 800                          ; UGH units per inch
top 1 ysize 4 bot 6                 ; Y page param. in inches
left 1 xsize 6 right 1              ; X page param. in inches
axis xmin 0 xmax 10 ymin 0 ymax 10 ; USER coordinate mapping
```

This defines a page of X size of 8 inches (1+6+1) by Y size of 11 inches (1+4+6) corresponding to a virtual space of size 6400 (8*800) by 8800 (11*800) in UGH units. In this case, AXIS coordinates 0,0 map to 800,4800 in UGH units while 10,10 maps to 5600,8000 in UGH units. 0,0 INCH also maps to 800,4800 as does 1,6 INCH PAGE.

**Note:** The page does not restrict where graphics may be placed in the virtual space; it is only a convenient concept for eventual hard copy. The **VP** parameter PRUNE controls plotting of data outside the AXIS box, while text may be placed anywhere. Text commands have INCHES and INCH PAGE or CMS and CM PAGE options that allow positioning the text at a particular place on the page rather then using user coordinates based on the axis.

The final mapping is from the virtual UGH coordinates onto a display. This mapping is controlled in two ways. First a subset of the display area (possibly the entire display area) is mapped as a *window*, and then a region of virtual space is mapped onto that window. Both aspects are usually accomplished by the VIEW command.

UGH drivers for display devices have the following default windows (see UGH command 9 in Section 7.1):

```
0  full screen     4  upper left  1/4
1  left half       5  upper right 1/4
2  right half      6  lower left  1/4
3  center 1/4      7  lower right 1/4
```

One of these can be selected with the VIEW command. If no window number is specified, the previous window is not changed. Thus,

```
VIEW fill erase
```

will display the current `.ugh(bin)` file in the current window (default 0, fullscreen) and adjust the final mapping so that all of the graphic elements will "fill" the window, keeping a 1:1 aspect ratio. The VIEW command never alters the contents of the `.ugh(bin)` file; it only determines which part of virtual UGH space can be seen on the graphics screen.

```
VIEW page erase 2
```

will display the current UGH file in window 2 (right half) and adjust the mapping so the current virtual page will "fill" this window.

It is also possible to use the **VP** UGH command as

```
UGH 9 2
```

to set the UGH driver to window 2 for all following VIEW commands (until explicitly reset).

**Note:** Users with detailed knowledge of a particular display device's coordinates can use UGH command 7 to set a particular portion of the graphics screen for display.

## 3.8 Invoking VP: Arguments and Options

**VP** may be started from a system (or shell) command line such as:

**vp** arg1 arg2 ... arg8 ( opt1 opt2 ... or
**vp** arg1 arg2 ... arg8 -opt1 -opt2 ...

In normal operation, the following events occur:

- The UGH display and internal variables are initialized.
- Any initial values of the %1 to %8 synonyms are set to the values of the corresponding argument fields (fields are separated by spaces).
- Any *options* are evaluated. The options are all fields after the first occurrence of "(" (or "\(" in UNIX) in the command line or all fields that begin with a dash "-".
- The initial profile `.exg` file is opened for input, and **VP** begins executing the commands in this profile.
- If, as recommended, the default `profile.exg` is used as the initial profile, then this will be followed by a read of the file `userprof.exg` with the arguments %1,...,%8. This is the recommended location of any user-specific changes.
- When the profile has been read, **VP** displays a prompt and waits for keyboard input.

The options can change this sequence of events. The valid options may be given in upper or lower case, and only enough characters need be given to distinguish between options:

**PROFILE** `filespec`
: Uses `filespec` as the initial profile `.exg` file instead of the default `profile.exg`. If multiple `profile` options are given, only the last is actually read.



**NOGRAPHICS**
: All processing will be completed, but no graphics will be drawn to an UGH display and no graphics or cursor input will be accepted. This is used for batch processing, where immediate views are not required.

**QUIT**
: **VP** will QUIT after the end of the profile `.exg` file. This is used with batch processing as described at the end of this section.

**DISPLAY disp**
: Changes the display destination for UGH graphics. For X-windows implementations, this takes the usual form `-display machine_name:0`. For the PC SVGA implementation this specifies the VESA numeric hex mode of the display. For example, `-display 103` attempts to operate the SVGA adapter at 800 by 600 pixels with 256 colors.

**CM**
: Uses centimeters instead of inches for specifying AXIS box size and placement. This option also converts default values from inches to centimeters.

**FLOAT**
: Uses 4-byte floating point numbers for storage of REAL values. The default uses 8-bytes for each number.

If other system-specific options are available, they will be described in README files that accompany each implementation of **VP**.

The use of command line arguments and options as described above allows **VP** to conveniently operate in a batch-processing environment. The standard `profile.exg` file supplied with **VP** ends with the line

```
if {$ ne ($&%1.)} &%1. &%2. &%3. &%4. &%5. &%6. &%7. &%8.
```

Thus, if **VP** is called with any arguments, the standard `profile.exg` file will end by reading the `.exg` file specified by the first argument and passing the remaining arguments. In other words, the PC command line

```
vp demovp ( quit
```

will start **VP**, read `profile.exg`, read `userprof.exg`, if it exists, read `demovp.exg`, and quit.

```
vp demofit
```

will start **VP**, read `profile.exg`, read `userprof.exg`, if it exists, read `demofit.exg`, display the prompt, and wait for input from the keyboard.