1

Introduction

1.1 What Is VossPlot?

VossPlot (**VP**) began at IBM Research in the late 1970's as a software tool for scientific and technical graphics, the sort of two-dimensional drawings that represent most figures in a technical journal. It now includes many text-processing capabilities, color control, programming, analysis, and pixel image display features that make it ideal for generating technical presentation graphics. It provides immediate, interactive views of data and analysis as well as facilitates the convenient composition of publication-quality figures from the data.

1.2 How Is VP Different from Other Programs?

- **VP** does less better.
  It was designed to solve some of the common jobs of scientists and engineers: the routine display and analysis of data and the preparation of publication and presentation 2D graphics from this data. By focussing on the most common tasks, **VP** is able to provide highly efficient solutions without the inherent baggage of a program that attempts to "be everything to everyone."
- **VP** has a long history of testing and evolution.
  VP was developed by a physicist at the IBM Research Division for his own technical presentations in the late 1970s. It has had thousands of users worldwide within the IBM Corporation and has received several internal IBM awards. **VP** has a consistent internal logic (albeit somewhat idiosyncratic) that cannot be achieved in committee-designed programs, as well as a long history of testing and user feedback.
- **VP** is highly programmable and customizable. Every aspect of the creation of a graph may be controlled by the user, but all have

carefully chosen defaults to produce excellent results with minimal tweaking.

- **VP** has an application programming interface for direct communication with other high-level programming languages.
- **VP** runs on a wide variety of computers, including the smallest PCs with 300K of memory.

1.3 How Does VP Work?

Figure 1.1 provides an overview of the functional hierarchy of creating graphics with **VP**. This figure, like all others in this manual, was produced with **VP**.

As can be seen in the figure, **VP** converts user input from the keyboard, mouse, text (`.exg`) files, and other programs to graphics primitives (objects like lines, triangles, etc.) contained in the output `.ugh(bin)` file for display or hard copy.

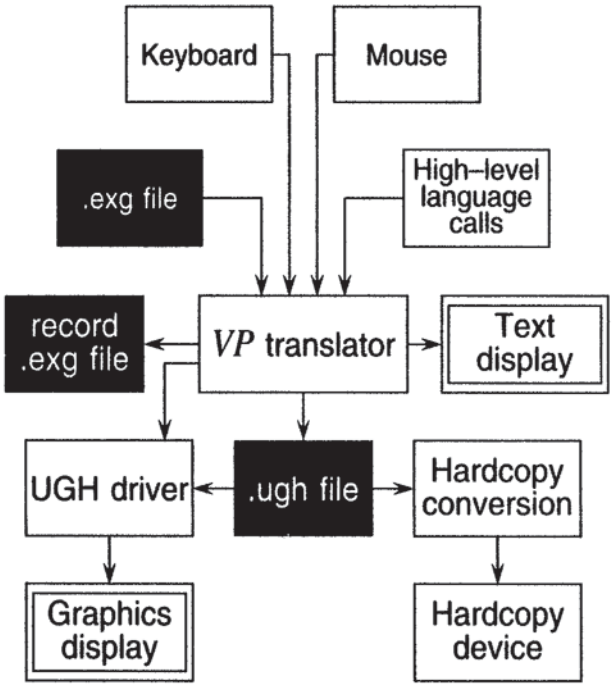

Figure 1.1: VP graphics creation hierarchy.

1.3.1 VP input

Users can instruct **VP** in a number of ways on what graphics to produce:

- Read commands and data from a previously generated text file, the `.exg` file.

  Note: `.exg` is the default name used here for text input files to **VP**. On most systems these are plain ASCII text files with extension `.exg` or `.exgraph`. In fact, **VP** can read text files with any extension, but choices other than `.exg` must be specified explicitly.
- Enter commands and data interactively from the keyboard.
- Use a mouse or other graphics input device to position elements on the graphics figure.
- Pass data and commands from another high-level language, such as C, PASCAL, FORTRAN, or APL. It is even possible to start the usual **VP** keyboard input mode from the calling language. Here, **VP** can serve as a simple graphics display utility for simulations or detailed experimental data analysis. And, of course, **VP** makes it easy to produce the final publication-quality figures once the original data have been reduced.

1.3.2 Recommended usage

The *recommended* way of using **VP** is through the plain text `.exg` file, which you can create and later change with an editor. Once created, it is processed by **VP** with the READ command and its graphics output is displayed immediately with the VIEW command. This interactive process of editing the `.exg` file and displaying the output can be quickly iterated within **VP** until the desired figure has been composed. The DOS version contains a built-in editor interface for this process. At this point the `.ugh(bin)` output file may be used to produce hardcopy output using the utilities described in Chapter 7. The use of an `.exg` file (rather than immediate keyboard input) allows simple changes to the figure at a later date (for example to change the symbols used for plotting or to add the latest experimental data points).

Figure 1.1 shows that the **VP** program uses two *windows* for user interaction: the *text display* for text message output and keyboard echoing and the *graphics display*.

In a multiwindow environment (such as X-Windows, Windows, or OS/2) **VP** also has a RECORD command that allows keyboard and mouse input (or other graphic positioning) to be saved in an `.exg` file as it is generated. Such input can then be made part of the edit-display cycle previously described. The RECORD command can also save the data passed by another high-level program in an `.exg` file for later detailed graphic editing.

1.3.3 VP output

VP translates your input into UGH graphics primitives that are saved in the `.ugh(bin)` file.

Note: `.ugh(bin)` is the default name used here for the **VP** graphics output files. The `.ugh(bin)` files consist of compressed binary data

that specify the graphics primitives (lines, circles, etc.) in a device-independent integer coordinate system representing a fictitious page. On PCs these files have extension `.ugh`, while systems that support extensions of more than three characters may use `.ughbin` as a reminder of the binary nature of the data.

UGH stands for Universal Graphics Handler. The "universal" is, of course, rather silly. Nothing in graphics is really universal. UGH, however, represents an attempt at providing the most useful primitives for scientific graphics in a device independent manner. The "handlers" are the individual programs that convert these primitives into hardware commands that drive specific display and hardcopy devices. Handlers are available for most of the common graphics devices found in research and engineering labs. Section 7.1 details the format of the binary `.ugh(bin)` files. The utility programs described in Chapter 7 can convert the `.ugh(bin)` files to other common formats for inclusion in word processing documents or for printing.

VP graphics generated on one system should produce the same output (within limitations of available colors and graphics resolution) when the corresponding `.exg` file is transferred to another system.

1.4 Getting Started: Installing VP

Since **VP** can run on a wide variety of computing environments, it is impossible for this manual to provide installation details for all present configurations. Nonetheless, installing **VP** involves a few general steps:

- Create a subdirectory or folder and copy the **VP** programs and files into it. This is typically named **vpugh**.
- Customize your system by setting environmental variables or selecting an UGH graphics driver for PC DOS.
- Run **VP**.

The **README.VP** file that is included with the software gives the details of the installation process for each specific system. In most cases, the installation process will be automated. The DOS version includes the program `install.exe` for this purpose.

2

Learning the Basics of VP

A systematic way to learn **VP** is with a careful examination of some simple examples. We'll look at each line in the three examples presented here and comment on what it does. This chapter is not a reference for the **VP** commands. However, it contains a number of hints for efficient usage of **VP** and should provide a useful review, even for experts.

2.1 A First Example

Figure 2.1 shows the first example we will study line by line. In a normal installation this file can be found as `vpsamp1.exg` in the `vpugh` subdirectory or folder. To try it yourself, start the **VP** program. At the initial prompt (`vp:`) type the command

```
read vpsamp1
```

and press the `[ENTER]` key. You should see the same graphics as shown in the figure.

The first two lines

```
01)  * Simple graph for tutorial purposes
02)  * VERSION 1
```

are comment lines. Any line in an `.exg` file that begins with an asetrisk (`*`) is treated as a comment. **VP** ignores the rest of the line but prints it on your text screen as a reminder of what is happening. Years of experience have shown that such comments are useful in reminding you (or others) of what you were originally doing when you created the file. These comments also make your `.exg` file more readable and easier to edit. Inline comments, and comments that do not print, are also possible, as we'll see in the next example.

```
03)  restart default
```

This command is recommended as the first non-comment in most `.exg` files. Its function is to reset the plotting environment to that estab-

```
01)  * Simple graph for tutorial purposes
02)  * VERSION 1
03)  restart default
04)  save vpsamp1
05)  set top 1 ysize 4 bot 1 xsize 5 right 1 pbcolor 0
06)  xydata
07)  1     -34.47
08)  2     -37.1
09)  3     -27.9
10)  4     -16
11)  5     -.977
12)  6     -2.76
13)  7      5.55
14)  8  3.9
15)  9     -2.7
16)  10    -12.1
17)  11    -17.83
18)  12    -27.3
19)  * The following sets the x limits and number of
20)  * intervals, but y limits will be taken from the data:
21)  axis xmin 1 xmax 12 xint 11
22)  curve isym 1 iline 3 color 4
23)  label
24)  Month
25)  "0" Celsius
26)  Average daytime temperature in Tuktuyaktuk
27)
28)  view erase fill
```

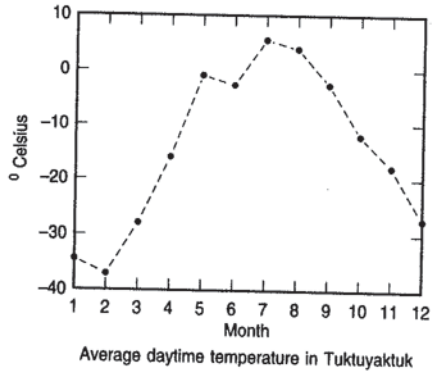

Figure 2.1: Sample `.exg` file and the resulting **VP** output. The data are purely fictitious. The line numbers are not normally part of an `.exg` file. They have been added here for reference in the text.

lished when **VP** was started. All of the internal **VP** parameters that control the graphics are *global* parameters. Upon entry to **VP** they are given default values (you can customize these default values with your `userprof.exg` file). These global parameters retain their values until explicitly changed. Thus, if you process multiple files in a single session, changes from one file will propagate to the others unless you use the RESTART DEFAULT command. Including RESTART DEFAULT helps avoid later "mysterious" results due to leftover (and forgotten) parameter changes. RESTART DEFAULT also sets up the default destination for your graphic output (into the file `vossplot.ugh`).

In this manual **VP** commands and their options are often emphasized with UPPERCASE characters. Commands always indicate some action that **VP** is to perform. If present, they are the first word in a line. An alphabetical list of the available commands is given in Chapter 5.

```
04)  save vpsamp1
```

This indicates that instead of the default we want our graphic output to be saved in the file `vpsamp1.ugh`. This command erases any current contents and directs all following graphics to be saved in `vpsamp1.ugh`. Without this command, the graphic output would be placed in the default file, `vossplot.ugh`. Remember that the `.ugh(bin)` file contains all the primitives (line, circle, text, etc.) that correspond to the graphics you're creating with **VP** commands. Consequently, the SAVE command belongs at the start of your `.exg` file, before you have generated any primitives. The `.ugh(bin)` file is used later to produce a printed copy of your graphics. See Chapter 7 for the various hardcopy options. Changing the output destination to the same name as the `.exg` file that produced it is good practice. It allows you to generate multiple `.ugh(bin)` files in a single session and make hard copies later. It also serves as a reminder of what produced the `.ugh(bin)` files.

```
05)  set top 1 ysize 4 bot 1 xsize 5 right 1 pbcolor 0
```

First, a SET command cannot be found in the reference section. There is no such command. In fact, the character string `"set"` has been defined as a **synonym** in the `profile.exg` file, which is automatically read every time you start **VP**. A **synonym** is a character string defined to stand for another string in order to save typing and improve readability. In this case, since `"set"` is replaced by blanks, it doesn't save typing and the line would have done the same thing without it, but does clarify what is happening on the line. See Section 3.3 for more information about synonyms and how to define your own.

You can probably guess that line (05) sets some parameters that control later processing. You are correct. **Top**, **ysize**, **bot**, **xsize**, **right**, and **pbcolor** are only a few of the many global parameters that are used throughout **VP** to modify the action of the **VP** commands. They retain their values throughout a **VP** session unless explicitly reset or returned

to their default values with the RESTART DEFAULT command. See Chapter 6 for a complete list of all global parameters and their significance. It is useful to remember that **VP** parameters always have a numeric value and are *global* to the **VP** session. Within **VP** you can use the PRINT command to see their current numeric values.

The parameters **top**, **ysize**, **bot** (short for **bottom**), **xsize**, and **right** are used by the AXIS command in line (21) to define the size and placement of the axis box on an imaginary page as shown in Figure 3.5 on page 42. You will understand more about these parameters as we continue with these examples. See Section 3.7 for a detailed discussion of coordinate mappings. The last parameter **pbcolor** stands for the page boundary color. If **pbcolor** > 0, the page boundaries will be visible with a VIEW command. You won't see it in this case since we set it to color zero (the same as the background). The page boundary is never included in the `.ugh(bin)` output file and will not appear in any hardcopy output. Although we have now defined where the axis will be placed, it will not be created until line (21).

```
06)  xydata
```

The XYDATA command indicates that the following lines contain X,Y coordinate pairs that are to be read into internal numeric arrays containing the X and Y data. Numeric values will be added until a nonnumeric field is reached (here line 19). This is just one of the many ways to enter the data you want to see graphically.

```
07) - 018)
```

The next twelve lines contain the X and Y data pairs to plot. The values are written in free format. You do not need to place numbers at the same position in each column, and you can have a different amount of numbers on each line. Even decimal points are not necessary. See Section 3.1 for details about the format of commands and numbers.

```
21)  axis xmin 1 xmax 12 xint 11
```

This line explicitly specifies the numeric limits of the *x*-axis (`xmin 1 xmax 12 xint 11`) and then executes the **VP** AXIS command. Because of this explicit specification, the *x*-axis runs from 1 to 12 in exactly 11 intervals. Note, however, that nothing was specified about the *y*-axis limits. In this case, **VP** looks at the range of the YDATA already entered (the actual limits are found in the parameters **ydmin** and **ydmax**) and chooses "appropriate" Y limits and number of intervals so that all of the YDATA will be visible. If **xmin**, **xmax**, and **xint** have not been set, then **VP** would also have made estimates of the *x*-axis limits based on the previous data.

By default, AXIS will produce a box with tic marks all around, and numeric labels on the left for Y and on the bottom for X. This can, of course, be changed to almost anything you want.

VP usually does a good job of choosing AXIS limits, intervals, and tic marks. You will probably find that most of the time you don't need to specify any limits. If the AXIS command is encountered before any X and/or Y data have been entered, the limits will default to (0.0, 10.0) in both directions.

```
22)  curve isym 1 iline 3 color 4
```

This is the command that actually plots our data. As with the AXIS line above, **VP** first sets any of the *global* parameters (here **isym**, **iline**, and **color**) and then executes the actual CURVE command. In effect, this line translates to

> Draw a curve (plot) with the current XY data inside the current axis box, with dots (**isym 1**) as symbols at each data point, connected by dashed lines (**iline** 3), all in red (**color** 4).

Both the CURVE command and the AXIS command place the graphic primitives they create in the current `.ugh(bin)` file (here `vpsamp1.ugh`). Later, when we get to the VIEW command, we will be able to see the contents of this file.

Figure 3.1 on page 32 shows the available symbols and lines types as found in the `demosylt.exg` sample file. As we will see in the following example, it isn't necessary, however, to remember the numeric values for **isym**, **iline**, and **color**.

```
23)  label
24)  Month
25)  "0" Celsius
26)  Average daytime temperature in Tuktuyaktuk
27)
```

The LABEL command on line (23) is used to add text comments around the current AXIS box. The first line following the LABEL command provides a text label for the *x*-axis; the second line gives the text label for the *y*-axis, and all remaining lines up to a blank line (27) provide a caption that is positioned beneath the X label. Default positioning centers the X and Y labels (with Y rotated) along the corresponding axis. As always, positioning can be customized. The double quotes (") in the Y label mark off (delimit) the 0 as a superscript. There is much more available for text processing than demonstrated here, and we'll see more in the next example.

```
28)  view erase fill
```

Finally, the VIEW command permits us to see the graphics we have created. It begins a replay of stored graphic data from the `vpsamp1.ugh` file as set by line (04). The ERASE option erases any existing graphics on the screen before the VIEW. The FILL option resets display limits to approximately fill the screen with the contents of the `.ugh(bin)` file.

If you understand this example, you already know most of the basics of using **VP**. For more practice, you can edit the vpsamp1.exg file to change parameters and add or delete parts to see how **VP** reacts. Meanwhile, we'll look at the next sample, which adds some useful frills to our first example.

2.2 A Second Example

Figure 2.2 shows our second example, which is found as vpsamp2.exg in the vpugh subdirectory or folder. To try it yourself, start the **VP** program. At the initial prompt (vp:) type the command

```
read vpsamp2
```

and press the [ENTER] key. Once again, you should see the same graphics as shown in the figure. The contents of vpsamp2.exg are

```
01)  * Simple graph for tutorial purposes
02)  * VERSION 2
03)  restart default
04)  save vpsamp2
05)  set top 1 ysize 4 bot 1 xsize 5 right 1 pbcolor 0
06)  tabdata ; Notice that we replaced XYDATA by TABDATA...
07)  1    -34.47   -29.2
08)  2    -37.1    -36.4
09)  3    -27.9    -30.2
10)  4    -16      -20.8
11)  5    -.977    -4.1
12)  6    -2.76    -3.8
13)  7     5.55     .553
14)  8     3.9    7.73
15)  9     -2.7   -3.3
16)  10    -12.1   -9.1
17)  11    -17.83  -20.11
18)  12    -27.3  -26.02
19)  xdata column 1
20)  ydata column 2
21)  * The following sets the x limits and number of
22)  * intervals, but y limits will be taken from the data:
23)  axis noxtics xmin 1 xmax 12 xint 11
24)  ticmarks x bottom
25)  1 2 3 4 5 6 7 8 9 10 11 12
26)  J F M A M J J A S O  N  D
27)  draw dot dashed red
28)  ydata column 3
29)  draw diamond dashed green
30)  label c2font 20210 c2color 6 ulcolor 7
31)  Month
```



```
32)  "0" Celsius
33)  Average _daytime_ temperature in \Tuktuyaktuk\
34)
35)  comment 7 -25 smult 3
36)  ?1.0.4? Year 1887+
37)  ?2.0.2? Year 1987
38)  comment &xsize. &ysize.+0.1 inch right cfont 15150
39)  &%0. &date. &time.
40)  view erase fill
```

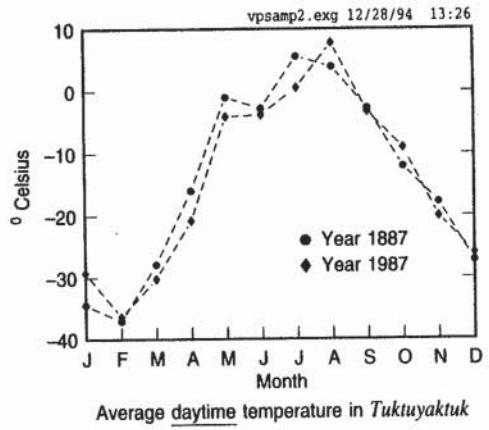

Figure 2.2: The graphics output from vpsamp2.exg.

In this case we will examine only the lines that have changed significantly or have been added.

```
06)  tabdata ; Notice that we replaced XYDATA by TABDATA...
```

First, we see that "inline" comments are initiated with a semicolon. Everything that follows the semicolon will be ignored by **VP** and not even echoed at the terminal. With the (*) comments in the first example, the * must start the line and the entire line is echoed to the text screen. Second, TABDATA has replaced XYDATA to start data entry. With TABDATA we are not limited to pairs (X,Y) of values. The numeric data are read until a nonnumeric field is reached and placed into the TABDATA array or table. Only later will we specify from which columns X and Y values are to be taken.

```
07) - 18) Xi YAi YBi
```

These lines contain the data we wish to plot. Again, it is in free format. However, TABDATA requires the same number of numbers on each line. Here, we have three on each line, corresponding to a table of 3 columns by 12 rows.

```
19)  xdata column 1
20)  ydata column 2
```

Simple enough—this indicates that data for the X array should be taken from the first column in TABDATA while the second column gives the Y data. If you look in the command reference section, you can see that we could also have used `XYDATA COLUMN 1 2`. Of course, we could have given any column numbers, as long as they exist.

```
23)  axis noxtics xmin 1 xmax 12 xint 11
```

We have already encountered most of this line. The change is the NOXTICS option, which eliminates both tics and numeric labels from the *x*-axis. It is used to replace the horizontal numeric labeling by something more appropriate to our specific example.

```
24)  ticmarks x bottom
25)  1 2 3 4 5 6 7 8 9 10 11 12
26)  J F M A M J J A S O N D
```

We now construct our own labeling. `ticmarks x` indicates that the two following lines will refer to the *x*-axis. `bottom` indicates that the labels and tics will appear at the bottom of our graph. Line (25) contains the numeric values for the placement of the tics. Finally, the third line (26) gives the corresponding labels. The AXIS command itself generates a TICMARKS command to add tics and labels to the axis box.

Note. *Options vs Parameters*

Here, `bottom` is used as an OPTION for the **VP** command TICMARKS. If `bottom` had been followed by a number in line (24), then the parameter **bottom** would have been set to that number. Occasionally **VP** uses the same word for either a command OPTION or a numeric **parameter**. It can distinguish between them since a **parameter** is always followed by a numeric value. A **parameter** is *global* to the **VP** session, while an OPTION is *local* to a specific command.

```
27)  draw dot dashed red
```

This is equivalent to line (22) in `vpsamp1.exg`:

```
22) curve isym 1 iline 3 color 4
```


The difference is that we use default synonyms (defined by `profile.exg` as we enter **VP**) to make the line more readable. The synonyms also eliminate the need to remember the specific **isym**, **iline**, and **color** numeric values. Remember the *set* synonym from the first example? The words in line (27) are synonyms that respectively stand for:

draw → `curve isym -1 iline -1`
dot → `isym 1`
dashed → `iline 3`
red → `color 4`

The *draw* synonym is simply the CURVE command, preceded by a reset of the symbol and line types. Here it wouldn't have been necessary because we set them anyway with other synonyms. But the use of *draw* is recommended over CURVE because it always resets **isym** and **iline**. The usage and definition of synonyms are explained in Section 3.3.

Note. *Decoding input lines*

It is useful to remember the sequence of operations **VP** uses in decoding an input line:

1. Replace all synonyms by their character definitions. Line 27 becomes

   ```
   curve isym -1 iline -1 isym 1 iline 3 color 4
   ```

   after substitution. The *draw* synonym includes default definitions for **isym** and **iline** as invisible. These initial settings eliminate the surprise of having data plotted with **isym** and **iline** values defined (and forgotten) long ago.

2. Reset any *global* parameters that are found on the line, moving from left to right, and remove the corresponding fields from the line. Thus, only the latter settings of **isym** and **iline** will be used.

3. Execute the actual command with current parameter settings.

```
28)  ydata column 3
29)  draw diamond dashed green
```

We now replace the Y data by the contents of the third column of TABDATA and plot this data (using the previous X values) with different symbols, line type, and color, so that it will be easily distinguished.

```
30)  label c2font 20210 c2color 6 ulcolor 7
33)  Average _daytime_ temperature in \Tuktuyaktuk\
```

Some parameter settings have been added to our LABEL command. Remember that parameters are not specific to any given command but exist as **global** numeric values used throughout a **VP** session. They could have been set in any of the previous lines. On a given line they are set

before the command is executed. **c2font** and **c2color** are parameters for the type and color of a *secondary* font, which is delimited in text by backslashes '\..\', as in line (33). **ulcolor** stands for the underlining color, used whenever text is underlined, delimited by the underscore characters '_..._' in line (33). Examine the graphic results of the new caption. See Section 3.4 for more details about text capabilities. Obviously, the different colors all appear as black on the output shown here, but if you use **VP** from a color screen, you will see the specified colors.

```
35)  comment 7 -25 smult 3
36)  ?1.0.4? Year 1887 +
37)  ?2.0.2? Year 1987
```

The COMMENT command is another way to add text to the graph. (Yet another is with the TEXTBOX command.) It allows you to place text at any position. The coordinates (7, −25) are in the axis coordinate system, so it's really easy to know where the text will be placed. The coordinates could also have been entered in inches, centimeters, or even with a cursor or a mouse. See the reference section for more details.

The following line begins the actual text. If a COMMENT text line ends with + (or @), an additional line is read and placed below the first. So we have two lines of text. The strange-looking strings delimited by question marks indicate **VP** symbols. The three numbers represent **isym.iline.color**. They are the same as those used with CURVE. So `?1.0.4?` gives a red dot without any line. Finally, `smult 4` on line (35) indicates that the symbol size should be multiplied by 3 before drawing it, making it 50% larger than the default value of 2 used with the actual data.

```
38)  comment &xsize. &ysize.+0.1 inch right cfont 15150
39)  &%0. &date. &time.
```

This type of comment is useful to place information about when and where the figure came from directly onto the figure itself. Later, when you look at a copy of the figure, you will remember where to find the `.exg` file that generated it. Line (38) may seem difficult to understand at first. A verbose interpretation is

> Add a comment at position given by the **xsize** and **ysize** parameters (plus 0.1 inch for y), aligning the right edge of the text with the specified X position and use font 15150 for the text.

You now see that we can "retrieve" and use the value of a parameter in a line by delimiting its name between an ampersand (`&`) and a period (`.`). In effect, we make parameter values into synonyms that are substituted before the line is evaluated.

The text line (39) also contains some special synonyms delimited by an ampersand and a period:

&%0. The synonym %0 stands for the name of the current `.exg` file.
&date. is replaced by the current system date.
&time. is replaced by the current system time.

2.3 A Third Example

Our second example included the data to be plotted directly in the `.exg` file that produced the figure. This is a common technique for relatively small data sets and unique figures. Moreover, by including the name of the `.exg` file directly on the plot, it is very simple to find and alter the plot later. It becomes inefficient if one is trying to produce many similar plots from different data sets. Consider the problem of producing yearly temperature profiles on data collected from many different places. As we shall see in this example, it may be more convenient to read the data from a separate file.

Figure 2.3 shows our third example, which is found as `vpsamp3.exg` in the `vpugh` subdirectory or folder. As before, you can try it yourself with the command

```
read vpsamp3 tut-temp.dat
```

at the **VP** prompt. Although the graphics shown in Figure 2.3 are nearly identical to those in Figure 2.2, they illustrate **VP**'s powerful programmability and advanced features in producing multiple plots from related data sets.

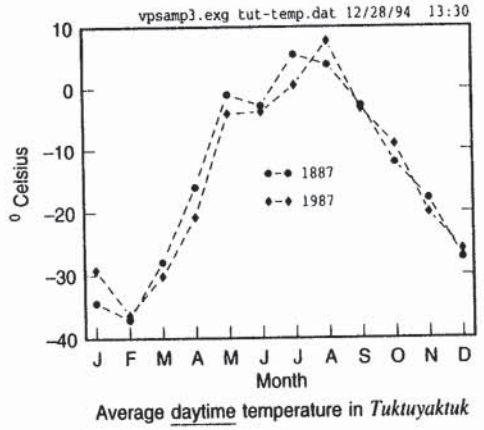

Figure 2.3: The output from the **VP** command `vpsamp3 tut-temp.dat`.

The contents of the temperature variation data file `tut-temp.dat` are

```
01)  ; temperature data for Tuktuyaktuk
02)  -34.47   -29.2
03)  -37.1    -36.4
04)  -27.9    -30.2
05)  -16      -20.8
06)  -.977    -4.1
07)  -2.76    -3.8
08)   5.55     .553
09)   3.9    7.73
10)   -2.7   -3.3
11)   -12.1   -9.1
12)   -17.83  -20.11
13)   -27.3  -26.02
14)  define year1 1887
15)  define year2 1987
16)  define place Tuktuyaktuk
```

Similar data files could, of course, be produced for many different years and places. There are several changes between this and the data that appeared in `vpsamp2.exg`.

1. The initial line is a nonprinting comment (which begins with a semicolon) to remind us of the contents of the data file.
2. Only two columns of data are included. The first column in `vpsamp2.exg` was a simple integer series that need not be included in each data file.
3. At the end of the data, several synonyms are defined with the construction `DEFINE sname substitution-text` that provide information about the data in this particular file. We shall see how these are used by the `.exg` file that reads this data.

The `vpsamp3.exg` file processes this (or similar data files):

```
01)  * Simple graph for tutorial purposes
02)  * VERSION 3 using separate data file
03)  restart default
04)  if { &%1.Q = Q } then
05)    echo usage: vpsamp3 fname.dat  for tabdata data
06)    read end
07)  endif
08)  save vpsamp3
09)  set top 1 ysize 4 bot 1 xsize 5 right 1 pbcolor 0
10)  tabdata  &%1.    ; use variable data filename
11)  if {nrow<12} then
12)    echo data file "&%1." not found or incomplete
13)    read end
```



```
14)  endif
15)  set ymin { tabdata.min } ymax { tabdata.max }
16)  axis noxtics xmin 1 xmax 12 xint 11 xind 0.03
17)  ticmarks x bottom
18)  1 2 3 4 5 6 7 8 9 10 11 12
19)  J F M A M J J A S O  N  D
20)  label c2font 20210 c2color 6 ulcolor 7
21)  Month
22)  "0" Celsius
23)  Average _daytime_ temperature in \&place.\
24)
25)  comment &xsize. &ysize.+0.1 inch right cfont 15150
26)  &%0. &%1. &date. &time.
27)
28)  macro plotyear
29)    xydata { i c&%1. }
30)    draw isym &%1. color &%1. dashed
31)    comm { 6 ypos}
32)    ?&%1..3.&%1.? &year&%1..
33)    ypos { ypos-((ymax-ymin)/10) }
34)  return
35)
36)  define ypos { (ymax+ymin)/2 }
37)  repeat &ncol.
38)    plotyear &count.
39)  end
40)
41)  view erase fill
```

The first change to note is that `vpsamp3.exg` must be called with an argument (the name of the data file to process) as

```
read vpsamp3 tut-temp.dat
```

at the **VP** prompt. This allows the same sequence of commands in `vpsamp3.exg` to be used with multiple data files. Once `vpsamp3.exg` is started, the synonym *%0* is set to the field `"tut-temp.dat"`. Lines (04)–(07) provide a check that an argument was actually entered:

```
04)  if { &%1.Q = Q } then
05)    echo usage: vpsamp3 fname.dat  for tabdata data
06)    read end
07)  endif
```

through the use of the IF command in line (04). The curly brackets `{...}` delimit a numeric calculation. These are discussed in great detail in Section 3.2. In evaluating the `{...}` brackets, **VP** substitutes the first argument for `&%1.` and calculates numeric values for the character

strings `&%1.Q` and `Q`. If these are equal, which can only happen if the first argument is blank, the statements up to line (07) are executed. These statements echo a message on the text screen to remind the user that an argument was expected, and they stop executing the remainder of the file. Try

```
read vpsamp3
```

to see what happens if no argument is given.

If any argument is given, then these statements are skipped and execution continues with line (08). It is useful to include such a construction at the start of `.exg` files that expect arguments.

```
10)  tabdata  &%1.    ; use variable data filename
```

The argument is actually used with the TABDATA command in line (10). In this case, TABDATA attempts to read a numeric table from the data file named in the argument `tut-temp.dat`. Initial comments in this data file are ignored. Once the numeric data has been read into TABDATA, the remaining lines of `tut-temp.dat` are treated as normal **VP** commands. When the end of `tut-temp.dat` is reached, processing of the calling file, `vpsamp3.exg`, continues.

```
11)  if {nrow<12} then
12)    echo data file "&%1." not found or incomplete
13)    read end
14)  endif
```

Lines (11)–(14) are another check that a valid data set has been read. The **VP** integer parameter **nrow** is set to the number of rows in the TABDATA array (**ncol** is set to the number of columns). If this is less than 12, then either the data were incomplete or perhaps the data file could not be found. For either case, a message is again echoed on the text screen and further processing of `vpsamp2.exg` is halted.

```
15)  set ymin { tabdata.min } ymax { tabdata.max }
16)  axis noxtics xmin 1 xmax 12 xint 11 xind 0.03
```

Line (15) uses the numeric calculation `{...}` brackets to define the Y limits for the axis. As discussed in Sections 3.2 and 5.1, the `.min` and `.max` suffixes give the extrema of a numeric array. Thus, no matter what the range of temperatures in the entire TABDATA array, the axis limits will be chosen so they will all be visible. Line (16) uses these **ymin** and **ymax** parameters in drawing the axis box. There is one other addition to the AXIS command line: The parameter **xind** is set to 0.03. This has the effect of indenting the *x*-axis tics by 3% from each side so that extrema do not fall exactly on the axis. Compare the *x*-axis of Figure 2.3 with that of Figure 2.2.

Lines (17)–(26) provide the axis labels and special tic marks similar to the previous sample. The only change is the addition of `&%1.` in line (26) to also display the name of the data file being processed.


```
28)  macro plotyear
29)    xydata { i c&%1. }
30)    draw isym &%1. color &%1. dashed
31)    comm { 6 ypos}
32)    ?&%1..3.&%1.? &year&%1.
33)    ypos { ypos-((ymax-ymin)/10) }
34)  return
```

Lines (28)–(34) introduce the MACRO command or `mini.exg` file that acts just like a separate file. This MACRO can be executed as a **VP** command with one argument to plot and generate a text label for an individual data column. When the MACRO command is encountered in an `.exg` file, all lines up to the RETURN command are saved for later execution when the macro name is called as a command. Thus, the MACRO can be placed at any convenient place in the `.exg` file. Common MACROs may even be included in your `userprof.exg` file.

Suppose the argument `&%1.` has value 1. Then, after substitution, the MACRO commands become

```
29)    xydata { i c1 }
30)    draw isym 1 color 1 dashed
31)    comm { 6 ypos}
32)    ?1.3.1? &year1.
33)    ypos { ypos-((ymax-ymin)/10) }
```

Line (29) uses `{...}` calculations to define the XYDATA. The `"i"` is just an integer index, while `"c1"` gives column 1 of TABDATA. Since column 1 has a length of 12, this defines 12 X, Y data pairs. The use of `"i"` in calculations eliminates the need to include the initial index column in the data files.

Line (30) plots the data using symbol and color numbers corresponding to the column. Each column will be shown in a different symbol and color. Lines (31) and (32) add a text comment that identifies the symbol and color by the `&year1.` synonym that was part of the data file. Line (33) reduces the Y position of the comment by one tenth of the Y range in case another column will be plotted. By making the reduction depend on the Y coordinates used with this particular data, the reduction should be appropriate for any data, independent of the range. As we shall see, **ypos** is a user-defined REAL parameter. The calculations in line (33) correspond to $ypos = ypos - (ymax - ymin)/10$. The **ypos** parameter is actually defined in line (36).

```
36)  define ypos { (ymax+ymin)/2 }
```

This is the same sequence used to define the usual character synonyms. The only difference is that the substitution text is replaced by a `{...}` calculation. This creates a user-defined numeric parameter whose initial value is the result of the calculation, instead of a character string. Such

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:`).