PROFESSIONAL BASIC™

A Window-Oriented Programming Language

Morgan Computing Co., Inc.

P.O. Box 112730 • Dallas, Texas 75011

Supplemental Information
Professional BASIC - Version 2.0

This version of Professional BASIC works on the IBM PC, IBM PC XT, IBM PC AT, AT&T 6300, Compaq Deskpro, and other compatible machines with 320k of RAM memory or more. 512k or more RAM is recommended. Programs loaded in Professional BASIC™ expand in memory to take up more room than just the byte size of the file. This is because of the extensive "seeding" operations and development of the PCODE, which add a lot of information upon loading a program, required to operate the sophisticated tracing environment. In addition, the syntax of every line loaded is checked at load time. (This complex process accounts for the reason why programs take longer to load than would be expected.)

Arrays and variables take up the same amount of memory as in PC BASIC™, except that integers are "double precision" and take up 4 bytes each instead of 2 bytes. (In random access disk files, an integer can be stored with the MKI$ command as either 2 bytes or 4 bytes, depending on whether the FIELD statement allocates 2 or 4 character spaces. The conversion is handled by Professional BASIC™ automatically.)

The "semi-compile" process in this release of Professional BASIC™ has been reworked. As a result, the time for the "semi-compile" when a "RUN" or "SRUN" command is given has been cut by about 40%. In addition, when a program is suspended and is not altered, it can be rerun instantly with a "RUN" or "SRUN" command. The semi-compile is not redone in such a case. Any alteration of the program, however, will mean that a semi-compile will be done to restart the program.

New Program Statements

CHAIN
COMMON These statements work as specified by the PC BASIC™ manual. However, you may put one, two, or all three of the optional parameters (line #, "ALL", or "DELETE range") in any order in the statement and the system will accept them.

If you are in a trace window when a CHAIN is executed, a few more things will happen on the screen. The specific commands Professional BASIC™ uses to perform a CHAIN operation will be displayed on the command screen, before the program being chained to begins to execute.

When running a program normally you can see a trace of just the chain operation by entering the command **,showchain**

on the Command Screen sometime before the program is loaded. Or, in general, enter this command anytime before the CHAIN command is encountered in the program. To turn this trace action off enter: **,noshowchain**

Error Trapping

ON ERROR
ERR, ERR2
ERL
ERR$
ERROR
RESUME n These commands work much the same as in PC BASIC. A RESUME 0, however, will take you back to the actual statement in a line which created the error instead of the beginning of a line. What is new in this subsystem is the addition of two special variables: ERR2 and ERR$. Professional BASIC™ has over 100 numbered error messages compared to

about 50 in PC BASIC. Also, it is important to note that many of the PC BASIC codes are not relevant under Professional BASIC. For instance, the first three messages "NEXT without FOR", "Syntax Error", and "RETURN without GOSUB" are not relevant since these problems are caught before a program begins to execute in Professional BASIC™, during the "semi-compile" operation when a "RUN" or "SRUN" command is issued. On the other hand, for the "Illegal Function Call" error (Error Code #5)in PC BASIC, Professional BASIC™ has 57 separate error message codes.

For compatibility purposes, you can use either the PC BASIC error code with the variable "ERR" or you can use the "extended" code in Professional BASIC™ with the special variable "ERR2". See Chapter 14 of the manual for the detailed codes. The "extended" code in ERR2 is equal to the PC BASIC code times 100 plus a number from 1 to 99. For example error number 24 in PC BASIC is "Device not ready". The extended code in PB is 2401. The "Illegal function call" error number 5 becomes an extended message of 501, 502, 503, etc. You can use either code in your programs. If you already have programs which use the PC BASIC codes, you do not have to change the program.

In addition to the extended code number (ERR2) there is an additional variable -- ERR$. This is a string variable which contains the Professional BASIC™ error message text. If a program ends in an error, from the command screen you can type ERR, ERR2, ERL, and ERR$ to get both error codes, the line number of the error, and the message text itself. With the ON ERROR command, you can also use this information in your program to control what happens when an error occurs.

Some common errors are handled automatically by Professional BASIC™ (without the use of the ON ERROR system) and allow a program to continue executing and not have to be restarted. If a printer is not ready or is out of paper or if a disk drive door is open, then a window will open on the screen and allow the user to fix the problem and continue with program execution, at the statement where the error occurred. Thus, much of the need for using ON ERROR has been taken care of by the system, although many people will still want to explicitly maintain control over errors in their programs.

Key Definition/Trapping

KEY(n)
ON KEY(n) The ON KEY system works as indicated in the PC BASIC manual.

KEY n,x$ The assignment of text to a function key works in the program execution mode, but not in immediate mode. If a function key is not being used in an ON KEY statement, then you can use a program statement to assign up to 31 characters to it (PC BASIC accepts up to 15 characters).

KEY ON, KEY LIST, KEY OFF These are not a part of the Professional BASIC™ system. Also, the definitions of the function keys when on the command screen are fixed and cannot be reassigned.

Graphics

The graphics system in Professional BASIC™ works in much the same way as in PC BASIC. Both High and Medium resolution modes are supported as well as 40 and 80 column text modes. The number of colors and the palette choices are all the same. The speed of the graphics operations, per se, are not affected by the use of the 8087/80287 numeric coprocessor, except when the "start" and "end" parameters are used in a circle statement. Where graphics operations depend on calculations to be performed for the paramters in a graphics function, however, there may be a very considerable difference in speed between the same program run under PB.EXE vs. PB8.EXE.

In general, the graphics functions in Professional BASIC™ are much faster than in PC BASIC. In some cases, like drawing horizontal lines, Professional BASIC™ is over 20 times faster.

You can run program that are creating a graphics output screen, but you cannot split the output screen and also see one of the trace windows. When you go to the trace window system it is always in text mode. When moving from a trace window to the output screen, if it is in graphics mode then the screen mode is changed automatically. You can halt full speed program execution and single step while on the output screen by repeatedly pressing <**Alt space bar**>.

The first press puts the system in single step mode and the subsequent presses execute one BASIC statement per press.

The status line indicates whether the user output screen is in "hi", "med", or text mode. Chapter 7 of the Professional BASIC™ manual describes the Status Line. Position "P" (shown in Figure 7-1 and described in the manual) has been replaced with a new indicator. "hi" indicates high resolution graphics mode, "med" indicates medium resolution graphics mode, and three numbers, such as "000" indicate that the screen is in text mode (see Supplemental Manual Note #1 below for details on the meaning of the three numbers, representing the "pages" that can be written to or displayed in 40 or 80 column text mode).

We appreciate the use of the Assembler code given to us by Ray Duncan of Laboratory Microsystems, Inc., Marina del Rey, California, in the implementation of the graphics subsystem in Professional BASIC™.

CIRCLE This works as specified in the PC BASIC manual, except that if you specify an aspect ratio, you cannot leave out the color, start, and end parameters. Put in 0 for the start and end if you want a full circle and a number or variable name for the color. If you want to specify a start and end for the circle, you have to specify the color; but you do not have to identify the aspect ratio if you want the default. In general you can leave out paramenter to the right to get the default setting but you cannot leave out parameters in the middle. That is, a statement such as:

```
CIRCLE (160,100),60,,,,5/18
```

is not valid since the color, start, and end have been left out. However, these two statements are okay:

```
CIRCLE (160,100),60,2
CIRCLE (160,100),60
```

COLOR (graphics)
LINE
PAINT
POINT
PSET & PRESET
SCREEN (statement)
WIDTH 40 or 80 These statements and commands work as indicated in the PC BASIC manual, but only as program statements (i.e., not in immediate mode).

New or Enhanced Immediate Mode Commands

,delete_var_all	Delete all variables currently defined and active, but leave the program in memory.
,delete_var a,b,...	Delete one or more specified variables (a,b,...) and leave the program and all other variables alone.
,just_common	Delete all variables except those specified in the COMMON statement(s) in the current program.
,delete_prog_all	Deletes the current program, but leaves all the variables and their values intact.
,crun n,scrn,speed	This is the "chain run" command, which runs the currently loaded program, but differs from a normal RUN command as follows: 1) leaves all files open, 2) leaves variables intact which are currently defined, and 3) does not clear the screen.
n	is the line number on which execution will begin.
scrn	specifies which screen(s) will be active when running begins (such as "l", or "l>v", etc).
speed	is either "f" for full or "s" for step speed.
,reset_opdef	This command makes the option base and the first letter type definitions (i.e., DEFINT, DEFSNG, etc.) undefined.

Each of these above six commands can be used to manually perform the same operation as a CHAIN statement, or perform a portion of a chain operation. For instance, you could correct an error in a program and rerun it without affecting either the status of the open files or the values of all the variables. You could also delete the current program and load in another one (with MERGE command) and run it, all without disturbing the values of all the variables.

LSORTL
LSORTV Same as SORTV and SORTL but the information is sent to the printer.

GOTO n Immediate mode method of transfering control to a specified line and resuming execution/tracing of the program, beginning with that line. This immediate mode command only works while a program is in suspended mode. If you want to load a program and start execution on a designated line:

- Load the program
- Enter **SRUN** command
- Press <**Break**>
- Enter the **GOTO n** command (line number or label)
- Press <**Enter**> to resume execution mode

You will be place back on the List trace screen, but with the "execution bar" on the line designated, instead of on the first line. The **GOTO n** command is a method of moving the "execution bar" to some line other than the one it is currently on.

RENUM You can specify a range rather than just a starting line number or line label, and lines in the middle of a program can be renumbered without affecting line numbers before or after. Example:

```
renum 1500,1500-1990,5
```

This will take lines 1500-1990 and renumber them beginning at 1500, incremented by a count of 5.

Another new feature is the capability to specify that certain lines will always keep the same line number when the program is renumbered. To do this create a remark line which begins with rem abs and has a line number following somewhere in the line. For example:

```
1000 rem abs This is always line 1000
```

<Backspace> key Removing characters from the program key buffer. There is a buffer of up to 64 characters that is maintained. These characters are accessed and read by either an INPUT or INKEY$ statement in the executing program. Occasionally you may press a key and unintentionally put a keystroke (a character) into this buffer for a running program. The <backspace> key in the trace system lets you back out these characters one at a time, last in first out. If you are on the program output screen, press

<Alt backspace>

Each press removes a character from the buffer, starting with the last one entered. If you are on a trace screen simply press the <backspace> key. Notice that the indicator on the right of the status line which shows the count on the number of characters in the buffer (maximum of 64) is decremented by one on each press. This single character counter goes from 0 to 9 and then a to z. Thus the counter indicates that there are 1 to 35 characters in this buffer or, if it shows a "z", 36 to 64 characters. On a trace screen, press <Alt A> repeatedly. Notice how the indicator is incremented as characters are put into the buffer for the user program. Then press <backspace> and see the counter decrement, as characters are removed from the buffer (or "popped" off the stack).

New Trace System Commands

- # Fast GOSUB. When on the L trace window and the "execution bar" is on a GOSUB statement, pressing # (<Shift-3>) will cause the subroutine to be executed at full native speed on the program output screen every time it is called at any line in the program. Thus, you can save a lot of time tracing through a program since the trace will not go through the code in this subroutine. It is as if you pressed the <**X**> key and a breakpoint were set at the next instruction after the RETURN from the subroutine.

- $ Remove fast GOSUB. When the "execution bar" is on a GOSUB statement that is set to "fast", pressing this key will reset it back to "slow", and tracing will show each line executed in the subroutine.

System States: There are four new system states. A complete list is shown below, with the new ones marked (See manual Section 7-2, F):

i	Initial state of the program - has not been executed
x	Program is executing
s	Program is suspended - was executing - but user presed <**Break**> key or program came to a stop. Execution can be resumed by a <**Enter**>.
t	Was state s, but program has been changed.
u (new)	Was state s, but variables have been deleted.
v (new)	Was state s, but program has been changed <u>and</u> variables deleted.
e	Program ended in error - execution cannot be resumed.

f	Was state e, but program has been changed.
g (new)	Was state e, but variables have been deleted.
h (new)	Was state e, but program has been changed and variables deleted.

Supplemental Manual Notes

1. The status line now has some additional items on it at the right. There is now an indicator for:

APAGE	Active page which the program is addressing
VPAGE	Visual page which the program thinks is displayed
MPAGE	Monitored page which is actually being displayed

The three characters near the right of the status line (e.g., 000) show this, where the first number is the APAGE, the second is the VPAGE, and the third is the MPAGE. This position is where the "P" used to be in previous versions.

When you are on the program output screen (and only then) you can switch the page being viewed from the current page by pressing and holding the <Alt> key and then hitting a number key for the page you want to monitor; **<Alt 0,1,2,or 3>** in 80 column text mode or **<Alt 0,1,2,3,4,5,6,or 7>** in 40 column text mode. The number key to use is at the top of the keyboard. Three pages can be accessed in 80 column mode and 8 pages in 40 column. In this way the program can be writing to one page (APAGE - active page), the program thinks it is displaying another screen (VPAGE - visual page), and you are actually looking at a third screen (MPAGE - monitored page). You may want to observe the APAGE as it is being built by the program when the visual page is a different one.

Unlike PC BASIC, you can have 4 screen pages separately maintained and then switch between them even if you are using an IBM PC Monochrome Display and Adapter Card. RAM memory instead of the memory on the Graphics Adapter card is used by Professional BASIC™ to store these screens.

The cursor location on each page is automatically maintained and restored for you as you or the progam switches from one page to another.

2. To duplicate a line of code in Professional BASIC™, enter an "EDIT n" command or use one of the edit function keys (F6, F8, F9, or F10) to display the line for editing. Then, move the cursor left over the line number and type in a different number. You will then have both lines in your program. Use the DEL command to delete the original line if you do not want it (i.e., you were performing a "move line" edit.

3. You do not have to specify the beginning or ending line number for a **DEL n-m** command. The following commands are valid:

```
DEL  -100
DEL 200-
```

4. After performing a RUN **filespec,** SAVE **filespec, or LOAD filespec** command you can enter a subsequent SAVE or LOAD and the last filespec used is remembered and used again. This default filespec can be seen at the bottom of the Memory or "Y" window.

5. **RANDOMIZE n** -- The parameter n is required in this function. A handy parameter to use for this "reseeding" operation is "timer". The command would be: **RANDOMIZE TIMER** The system clock will provide a number. If the user is to provide a seed, just precede the function with an "**INPUT var**" statement and pass it to a "**RANDOMIZE var**" instructon.

6. If you name a variable and an array of the same type with the same name, the **SORTV** command will only show one or the

other, as will the **FIND** command. The **SEARCH** command can be used instead of **FIND**.

7. **LOF(filenum)**	Returns the actual length of the file in bytes. Note: On a "new" file, this length is zero until the file is closed. On an "old" file opened for output, this length is the previous file length until the file is closed.
EOF(filenum)	This function also works with random files to tell you when you have accessed the last record in the file with a **GET #n,m** statement.
LOC(filenum)	With sequential files, returns the actual number of bytes into the file.

You can read or write to a random access file as a sequential file if you wish.

8. Numeric expressions. (1) Two operators cannot be placed next to one another. If you want to raise 3 to the power of -2, then the -2 must be enclosed in parentheses. (2) If you want to enter 10 to the 30th power, enter: **10.e30** not **10e30** The latter is an error since it is an attempt to calculate an integer number higher than the upper limit of integers. The decimal point in the first number marks it as a floating point number which can have an exponent up to 308 (255 for the BCD version).

Supplemental Information
Professional BASIC - Version 1.08

Professional BASIC™ now works on both the IBM PC AT (with or without the 80287 coprocessor) and on the AT&T 6300 Personal Computer.

New Program Statements - These four options for the OPEN statement are now implemented:

OPEN SCRN:
OPEN LPT1:
OPEN LPT2:
OPEN LPT3:

New Immediate Mode Commands:

,sn This command is used to speed up screen display operations when a color graphics adapter card is installed. It is an on/off toggle. To avoid the problem of screen interference (or "snow") with the IBM Color/Graphics Adapter Card, the screen writing functions in Professional BASIC™ have been slowed down, but only when this card is being used. The monochrome system works fast. If you have a non-IBM color/graphics card or do not mind the "snow" effect, enter **,sn** on the command screen. Program tracing and other operations involving the screen display will be speeded up considerably. Try it if you are not using the monochrome adapter card to see the effect.

Supplemental Manual Notes:

1. A CAUTION: Professional BASIC™ allows you to toggle into a mode whereby the **<Ctrl>**, **<Alt>**, and **<Shift>** keys can be left "on" without physically keeping them pressed down. This can lead to great confusion and unpredictable results if you accidentally get into this mode and aren't aware of it.

This "handicapped keyboard" feature is turned "on" by pressing the <5> key on the numeric keypad (when the Num Lock mode is "off"). In this "sticky" mode pressing one of the three keys shifts it "on". Pressing it again shifts it "off". In the case of the **<Shift>** key this will "shift-lock" every key on the keyboard. The Status Line at the top of a trace screen will let you know if you are in this mode and if one of the three keys is "on". A "K" in inverse video will appear on the Status Line if you are in this mode. A "c", "s", and/or "a" will appear if one of the three keys is "on".

2. Any time a command on the Command Screen causes something to scroll on the screen (i.e., LIST, FIND, SEARCH, LOAD, SORTV, SORTL, or FILES) you can control the scrolling action with three keys:

<Space Bar>	Suspends the scroll or scrolls the next line in single step fashion.
<Enter>	Continues full speed scrolling if currently suspended in single step mode.
<Esc>	Aborts the scrolling operation and returns to the Command Screen prompt.

3. If you try to load a file that has a line with no line number and the line does not begin with at least one space, Professional BASIC™ will try to execute the command as an immediate mode command.

December 1984

8087/80287

ENHANCEMENT PACKAGE

This enhancement package allows Professional BASIC™ to take advantage of your 8087 or 80287 numeric coprocessor for additional speed and accuracy. You also recieve a BCD (Binary Coded Decimal) option for true 16-digit decimal precision. The 8087/80287 option requires prior purchase of Professional BASIC™ and is available at $50.00. See Appendix C for more details on the differences in speed and accuracy.

PROFESSIONAL BASIC™ SLIPCASE

For the protection of your 3-ring binder a Professional BASIC™ slipcase is available for an additional $6.00. Limited quantities.

Add $3.00 Shipping and Handling to all orders

[_]Check Enclosed

[_]VISA [_]MC #____________________ Exp. Date___/___/___

Name: ____

Address: ______________________________________

City: ________________________ State:____ Zip:_________

BASIC Serial Number: ______________________________

PROFESSIONALBASIC™

A Window-Oriented Programming System

by Dr. Neil Bennett

Manual by Chris H. Morgan

September 1, 1984

Morgan Computing Co., Inc.
P.O. Box 112730   Dallas, Texas 75011

FOREWORD

Some of the earliest ideas behind Professional BASIC™ started in 1968. I had been used to computers with a typewriter where just one operation was needed to read a character from the keyboard and then print the character. However, I began to work on a PDP-8, where separate operations were needed to read a character and to print the character. This gave the opportunity to sample each character sent from the keyboard in real time and to determine if that incoming character was one that was considered acceptable. If it was not acceptable, the character could be rejected with a bell character rather than printed. The program I designed on the PDP-8 system implemented a very simple calculator language. In Professional BASIC™, however, the Dynamic Syntax Checker™ is used to validate <u>every</u> keystroke entered, whether an arithmetic command is being entered or a line of program code is being created.

Another fundamental idea behind Professional BASIC is the one of a white line following control through a listing of the program. This evolved from the notion of a small spot following the words on a screen on some of the old sing-along shows. Once the means was found to separately (and efficiently) maintain screens for the command screen, the print screen and the list trace screen, the other screens followed. One observer commented that "you've built a fully instrumented system". I agree.

An earlier (1980) version of this system was implemented on an APPLE II computer with 48kb of memory (<u>Hands on BASIC</u>, now published by and available from EduWare, Inc.). It was regarded as an interesting teaching system but of little practical interest because the language was a minimal subset of BASIC.

This system implements a much richer BASIC. Also, a semi-compiled approach was used to gain execution speed. When the user types RUN, arithmetic expressions are translated to pseudo code strings of very elementary operations like load, add, etc. During execution, this string can be followed easily, whereas an interpreter has to analyze and evaluate arithmetic expressions each time they are encountered. The Dynamic Syntax Checker™ guarantees there are no syntax errors, so the translations can be done more easily. As a result, however, these operations consume large amounts of memory.

A system must be built from scratch to implement this style of BASIC. It was an obvious decision to implement the dialect of BASIC already being used. The aim is a standard BASIC in an enhanced environment. Another advantage is that programs developed under this system can be compiled to run much faster using the BASIC compilers already available.

It was also obvious to address all the memory which could be added to the PC, although implementing that decision on an architecture of 64k byte pages has not been trivial.

Finally, the ideas behind this system are not limited to the IBM PC or the BASIC language. As 16 bit machines and large memories become available, we expect that users will prefer this style of language system.

Neil Bennett
March, 1984

ACKNOWLEDGEMENTS

It is a pleasure to recognize the contributions of those who have helped to bring Professional BASIC™ to its current state of development.

I wish to express my particular indebtedness to Rudi Hoess of Sydney, Australia and to Chris Morgan here in Dallas, Texas for their support, encouragement and guidance. Many other people have given valuable encouragement and comments.

Roger Keating; Sydney, Australia
Juris Reinfelds; Woolongong, Australia
Richard Miller; Woolongong, Australia
Bill Baker; San Francisco, California
Ted Perry; Sacramento, California
Portia Isaacson; Dallas, Texas
Egil Juliussen; Dallas, Texas
Harold Kinne; Dallas, Texas
Dan Rollins; Glenview, California
Art Babick; Dallas, Texas
Dick Barr; Dallas, Texas

and to the members of the APPLE club and the IBM PC club here in Dallas, my thanks.

Special thanks also to the people at Morgan Computing Company, who have given me much encouragement, support, suggestions and comments. They are:

Brad Crumpecker,
Herb Dorf,
Jim Graham,
Ron Hall,
Jim Hoisington,
Bill Hood,
Charles Kroboth,
John Marxer,
Richard Rudd,
Deborah Pinnick
and Scott Wierschem.

This system is dedicated to JOY BENNETT, who was the first programmer that I ever knew. In the 50's my mother wrote instructions for knitting books telling how to make woolen sweaters in sizes 34, 36, and 38. This system is more complex, but the idea is the same.

I am grateful to my wife Judy for her patience and understanding.

Neil Bennett,
March, 1984

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

1. INTRODUCTION

A new world of programming productivity and fun awaits you! Professional BASIC™ is the first of its kind -- a familiar programming language with windowing and other facilities for the pro or the beginner. With Professional BASIC™ programming can become a productive and even playful activity. All the tools you need for understanding how a program really works are available at a keystroke. We think you will find the experience of programming in Professional BASIC™ a very satisfying one.

While Professional BASIC™ is patterned after IBM PC BASIC, there are many enhancements. These include:

- 8087 support
- Ability to access all available memory in the PC
- Windowing -- dynamic tracing of an executing program
- Semicompiler language
- Dynamic Syntax Checker™
- Save programs and merge code without line numbers
- Labeled line referencing
- Large integer numbers ±2,147,438,647
- Large real numbers $10^{\pm 308}$ (double precision) or $10^{\pm 38}$ (single precision)

- EXITFOR and EXITWHILE statements
- and many other innovative features

In a short time you will be able to go through most of this manual and get a good feel for how Professional BASIC™ can help you program better and faster. Professional BASIC™ provides you with a wealth of useful programming tools, so set aside some time to go through the manual and the sample programs.

IMPORTANT: Please take a moment now to fill out and return the enclosed registration card. This will guarantee you automatic access to information on updates and new releases of this new system. Take advantage of this offer!

1-1 REQUIREMENTS

Professional BASIC™ requires:

- IBM PC, XT, or Portable PC
- at least 256k of RAM memory (384k+ preferred)
- One disk drive
- PC DOS 2.x operating system
- Optional - 8087 numeric coprocessor.

Professional BASIC™ will also run on many IBM PC compatible machines including COMPAQ and Columbia as long as the above requirements are met.

1-2 SYSTEM BACKUP

Before beginning to use Professional BASIC™ you should make a backup copy of both Professional BASIC™ system disks. To do this, format a new disk with the command:

```
A>format b:
```

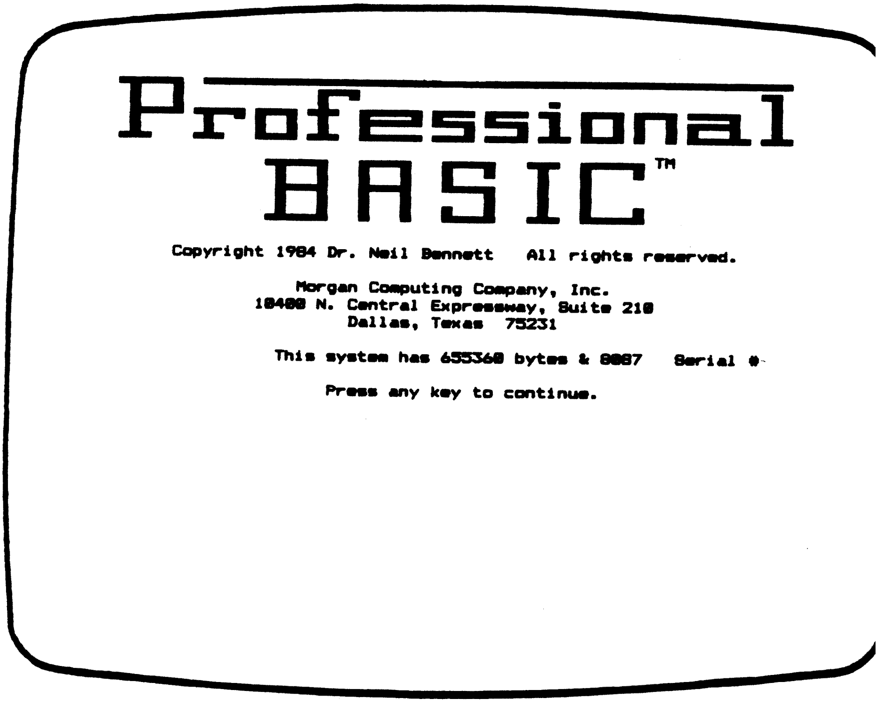
with a DOS 2.0 or 2.1 disk in drive A and a blank disk in drive B. (If you have a one disk system you will be prompted to switch disks in your drive when appropriate.) Then replace the DOS disk with the Professional BASIC™ system disk and enter the following command at the DOS > prompt:

```
A>copy *.* b:
```

This will copy all the Professional BASIC™ files to the newly formatted disk in drive B. (If you have a one disk system you will be prompted to switch between the two disks when appropriate.) Remove the original system disk and store it in a safe place.

1-3 STARTING Professional BASIC™ DEMO

There are two versions of Professional BASIC™ on your system disks. System disk A contains the program file PB.EXE. This version requires the 8087 numeric coprocessor. System disk B contains PB8.EXE. It does not require an 8087. On both disks are a series of demo programs which will be useful in showing the various features and windows provided by Professional BASIC™. If you do not have an 8087, use the disk with PB8. Please refer to Appendix C for more information on the differences between the two versions.



```
Professional
BASIC™

Copyright 1984 Dr. Neil Bennett   All rights reserved.

          Morgan Computing Company, Inc.
      10400 N. Central Expressway, Suite 210
              Dallas, Texas  75231

   This system has 655360 bytes & 8087   Serial #

         Press any key to continue.
```

FIGURE 1-1 Copyright notice screen

Professional BASIC™ has a built-in demonstration program that displays the various program-trace windowing facilities. To get an idea of the unique program tracing capabilities in the Professional BASIC™ system try the following demonstration.

NOTE TO THE EAGER USER: If you are anxious to try the program, skip to Chapters 4 and 5 after looking over the material in Chapters 2 and 3 to understand the differences between Professional BASIC™ and IBM PC BASIC.

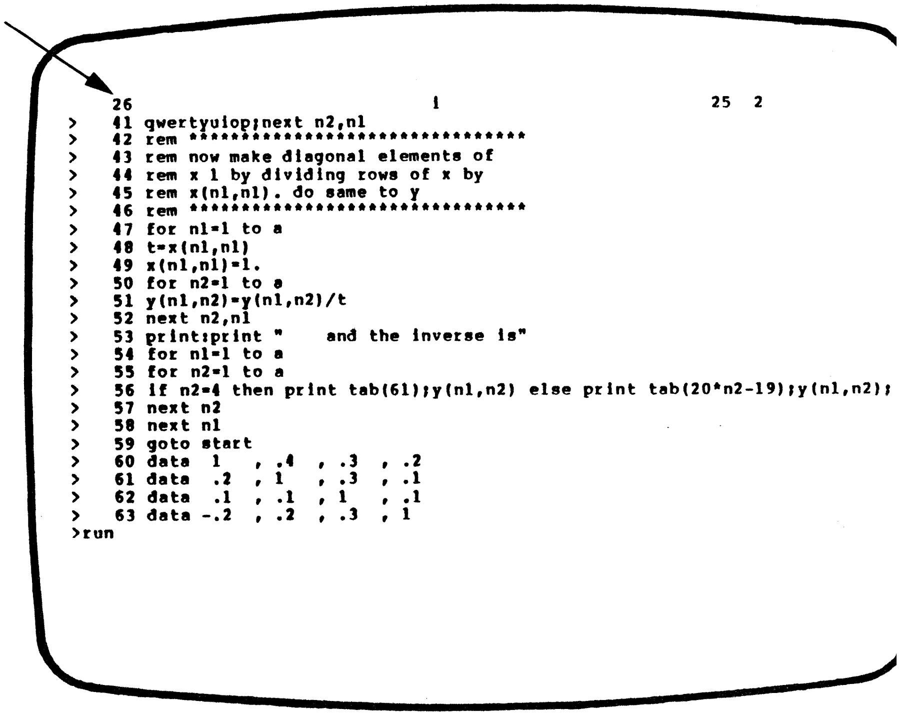
If you have an 8087, place a copy of the Professional BASIC™ System Disk A you just made in drive A, and at the DOS A> prompt enter:

A>**pb** (System disk A)

If you do not have an 8087, then place System Disk B in drive A and enter:

A>**pb8** (System disk B)

Professional BASIC™ will be loaded into your machine and the copyright notice screen will be displayed. If you do not touch the keyboard, a self-running demonstration will start in about 15 seconds and will run for approximately 5 minutes. The demonstration is not just a simulation of the system; it is an actual program being loaded and run. To exit from the demo after the program begins executing, press the <**break**> key twice (the key at the far upper right corner of the keyboard). (If you press the <**space bar**> before the demo begins, the screen will clear and you will be on the command screen. From the command screen you may perform the usual **LOAD**, **SAVE**, **RUN**, and **EDIT** operations. See Chapters 4 and 5 for more on loading, saving, running, and editing programs.)



```
    26                            1                           25   2
>   41 qwertyuiop;next n2,nl
>   42 rem ********************************
>   43 rem now make diagonal elements of
>   44 rem x 1 by dividing rows of x by
>   45 rem x(nl,nl). do same to y
>   46 rem ********************************
>   47 for nl=1 to a
>   48 t=x(nl,nl)
>   49 x(nl,nl)=1.
>   50 for n2=1 to a
>   51 y(nl,n2)=y(nl,n2)/t
>   52 next n2,nl
>   53 print:print "    and the inverse is"
>   54 for nl=1 to a
>   55 for n2=1 to a
>   56 if n2=4 then print tab(61);y(nl,n2) else print tab(20*n2-19);y(nl,n2);
>   57 next n2
>   58 next nl
>   59 goto start
>   60 data  1   , .4  , .3  , .2
>   61 data  .2  , 1   , .3  , .1
>   62 data  .1  , .1  , 1   , .1
>   63 data -.2  , .2  , .3  , 1
>run
```

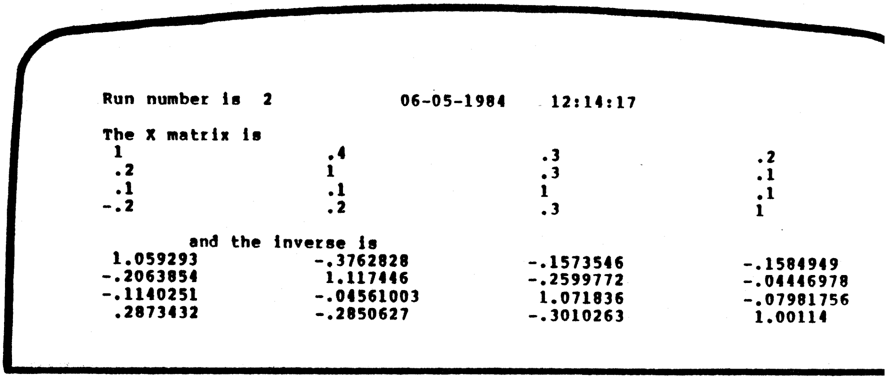
FIGURE 1-2   Screen during semi-compile

The first step of the demonstration is the loading of a program. As part of the loading process the program is listed on the screen. During loading, the program is checked for syntax errors and numerous other set-up operations are performed. Then the "run" instruction is given to initiate execution. At this point, Professional BASIC™ performs a "semi-compilation". If there were any errors in the program involving the flow of execution (e.g., improper GOTO's, GOSUB's, FOR/NEXT loops, etc.) they would be brought to your attention. After about 2 seconds, the semi-compile is completed and the program begins executing. Thus, both the "load" and "run" operations perform a lot of error checking as well as other initialization processes. When running a program in Professional BASIC™ this semi-compile is only done once, unless you edit the program. Therefore, future RUN statements will execute instantly for that program.

What is "Semi-Compiling"

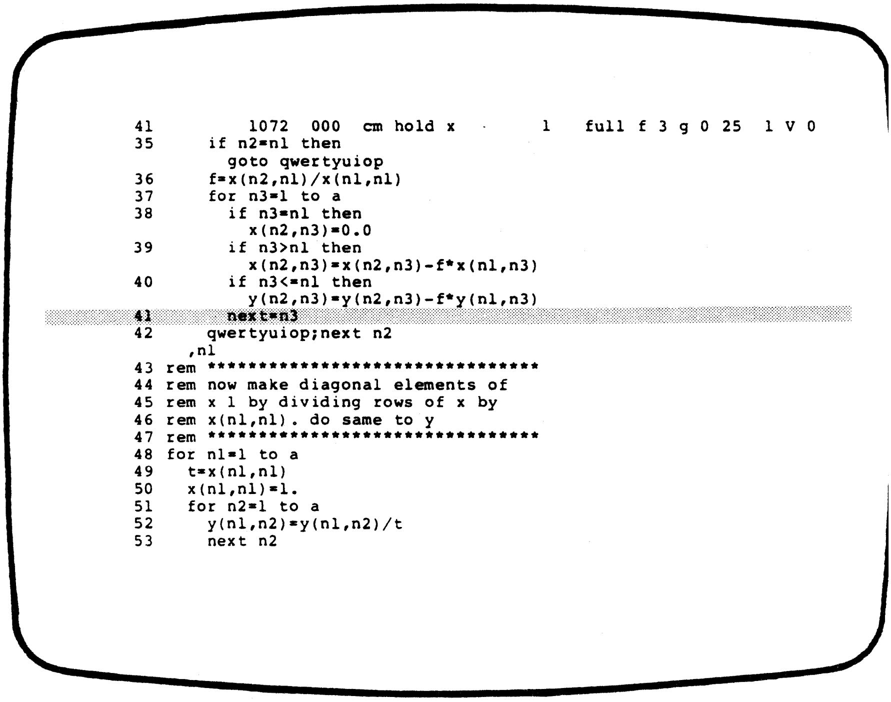
A normal interpreter BASIC begins execution immediately after a "RUN" command is given, performing a "translation" of each instruction as it is executed. In Professional BASIC™ much of this work is done "up front". As a result, several advantages are gained:

- the program executes faster overall
- arithmetic statements are transformed from BASIC source code into a form closer to machine language - a "pseudo code" form
- symbol tables are created to allow rapid access to variables and arrays



```
Run number is  2            06-05-1984    12:14:17

The X matrix is
 1                    .4                   .3                   .2
 .2                   1                    .3                   .1
 .1                   .1                   1                    .1
-.2                   .2                   .3                   1

        and the inverse is
 1.059293            -.3762828            -.1573546            -.1584949
-.2063854             1.117446            -.2599772            -.04446978
-.1140251            -.04561003            1.071836            -.07981756
 .2873432            -.2850627            -.3010263             1.00114
```

FIGURE 1-3 Typical output screen of self-running dem



```
41        1072  000  cm hold x        1   full f 3 g 0 25  1 V 0
35     if n2=nl then
         goto qwertyuiop
36     f=x(n2,nl)/x(nl,nl)
37     for n3=1 to a
38       if n3=nl then
           x(n2,n3)=0.0
39       if n3>nl then
           x(n2,n3)=x(n2,n3)-f*x(nl,n3)
40       if n3<=nl then
           y(n2,n3)=y(n2,n3)-f*y(nl,n3)
41       next=n3
42     qwertyuiop;next n2
     ,nl
43 rem ********************************
44 rem now make diagonal elements of
45 rem x 1 by dividing rows of x by
46 rem x(nl,nl). do same to y
47 rem ********************************
48 for nl=1 to a
49   t=x(nl,nl)
50   x(nl,nl)=1.
51   for n2=1 to a
52     y(nl,n2)=y(nl,n2)/t
53     next n2
```

FIGURE 1-4 Initial trace screen of the self-running dem

- all syntax and control errors have been noted. Hence, these types of errors (or "bugs") will never halt execution once it starts
- actual memory locations are found and used for the GOTO and GOSUB instructions to speed up these transfer operations.

The Demonstration Program

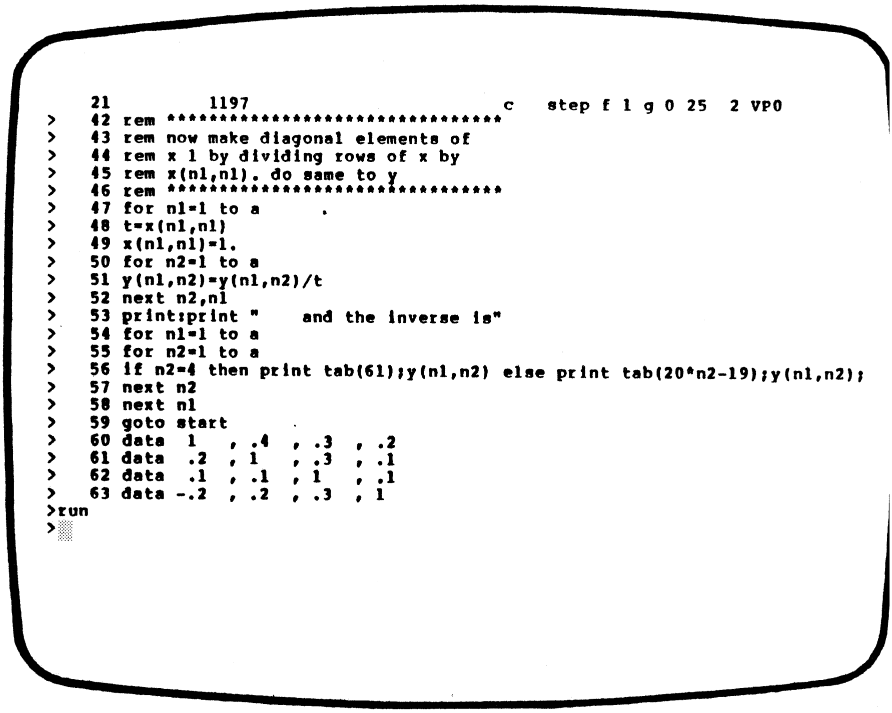
Back to the demo . . . The first thing you will see on the screen is the normal output of the program. The program is inverting a matrix over and over again. This may not mean a lot to you. But don't worry - just look at the various screen windows that will appear.

After the program has run for a few seconds the system "switches the channel." A listing of the program with a "white line" flashing about the screen is now displayed, showing program control moving from statement to statement. This trace of the program listing can be stopped and started using the following keys:

- Press the <**space bar**> to stop the program.
- Press the <**Enter**> key to resume trace execution.
- After you press the <**space bar**> to halt program execution, each additional press of the <**space bar**> will cause one instruction to be executed.

Using the <**space bar**>, it is possible to walk through the execution of the program and to see each instruction executed in turn in a slow and controlled fashion.

Continuing with the demonstration (if in single step mode, press the <**Enter**> key), several other windows will be shown.



```
    21          1197                          c    step f 1 g 0 25  2 VP0
>   42 rem *********************************
>   43 rem now make diagonal elements of
>   44 rem x 1 by dividing rows of x by
>   45 rem x(n1,n1). do same to y
>   46 rem *********************************
>   47 for n1=1 to a     .
>   48 t=x(n1,n1)
>   49 x(n1,n1)=1.
>   50 for n2=1 to a
>   51 y(n1,n2)=y(n1,n2)/t
>   52 next n2,n1
>   53 print:print "    and the inverse is"
>   54 for n1=1 to a
>   55 for n2=1 to a
>   56 if n2=4 then print tab(61);y(n1,n2) else print tab(20*n2-19);y(n1,n2);
>   57 next n2
>   58 next n1
>   59 goto start
>   60 data  1   , .4  , .3  , .2
>   61 data  .2  , 1   , .3  , .1
>   62 data  .1  , .1  , 1   , .1
>   63 data -.2  , .2  , .3  , 1
>run
>▒
```

FIGURE 1-5 **Command screen, after <Break> has been pr twice to exit demo**

For now just watch the demonstration, halting it with the <**space bar**> and continuing it with the <**Enter**> key. Notice that the program provides various ways to view execution. The demonstration allows you to become familiar with the various windows which you will learn more about later in this manual.

- Press <**Break**> twice to stop the demonstration (after the loaded program has begun to execute).

- After you stop the demonstration, if you want to exit the Professional BASIC™ system, press the **F7** function key (or type in the command "system") and then press the <**Enter**> key.

1-4 WHY A NEW APPROACH

Normally, you write and run a program and see the results only on the screen or printer. You verify that it has operated properly by examining and interpreting the output it generates. For the most part, the program itself is working away unseen. There is no way to tell if it is performing as expected except by viewing and interpreting the output. If the program does not work properly it can be quite difficult to find out what is going wrong. Errors in the code may prevent the normal output from appearing or, sometimes worse, output may be produced that is erroneous in subtle ways. Sometimes programming and debugging can be like groping in the dark, making adjustments which will, hopefully, put the program onto the desired track.

Debugging programs can be part art and part science. There are systematic procedures a programmer can often employ to analyze how a program is operating and why it is not functioning as it should. However, many times an error can be so difficult to find that only by playing "hunches" can the programmer straighten out the erroneous sections of code.

Professional BASIC™ provides an entirely new environment in which a programmer can work. At last, a program creation, analysis, and debugging system was designed and written from the ground up to aid the beginner as well as the advanced programmer to the maximum extent possible. Many years of careful thought have gone into the concepts behind Professional BASIC™ by a designer who has rethought the whole process of computer-programmer interface. Because of the power of the 16-bit microcomputer, with its large memory, it was feasible to create a new and radical departure from traditional wisdom in the design of the environment for a programming language.

As stated before, a normal programming language has one means of communication -- the output designed by the programmer to be displayed or printed by the program itself. Now, with Professional BASIC™, it is possible to view the execution of a program in many different ways. It is possible to view the actual set of program instructions in such a way that you can see how each instruction step is executed, the sequence of execution, and the effect upon every variable. And all of this is easily accessed in a natural way by the user.

With this system, we expect the world of programming to open up to a much wider audience than ever before. The newcomer to the computer (and programming) can now have a tool with which to learn the fundamentals of programming. For the advanced programmer, Professional BASIC™ is a workshop which can help to perfect the most complex program in a much shorter time than has ever been the case before. We feel that the whole approach to computer programming will be permanently altered and enhanced with the advent of Professional BASIC™.

1-5 ROLE OF THE MANUAL

This manual is not intended to teach programming and it cannot be used in place of your IBM PC BASIC manual. Instead, it is a supplement. The basic operation of each command, statement, and function within Professional BASIC™ is the same as IBM PC BASIC unless specified otherwise in Chapter 3. Use both manuals as your resource. In general, if you have a question about how the BASIC language works, consult the IBM manual or other reference guide on BASIC (there are many good ones to choose from). If you have a question about the special Professional BASIC™ windows, program creation, editing, running, testing, storing, and loading, then refer to the Professional BASIC™ manual.

1-6 ORGANIZATION OF THE MANUAL

Section I introduces you to Professional BASIC™ and describes the differences between it and IBM PC BASIC. It also explains how to load and run a BASIC program and use the built-in editor to create and change programs.

Section II explains the tracing display windows and how to use them for learning and debugging.

Section III discusses breakpoint setting for Professional BASIC™ and includes a section on error messages.

1-7 HOW TO USE THE MANUAL

The next section of this manual describes the scope of the language and how to load, edit, run, and save a program.

Chapter 4 tells you how to load, run, and save programs in Professional BASIC™. A short tutorial will lead you through how it is done in Professional BASIC™ and point out some differences from other BASICs.

Chapter 5 explains how the Professional BASIC™ editor works by presenting a series of examples. Since the editor functions differently than other BASICs it is important to go through these exercises. Also, you will discover some new tools for editing that are not available in other systems.

When you have finished with Section I, you should be comfortable executing a BASIC program in a similar manner to using the IBM BASIC Interpreter.

Then proceed to Section II. The introduction to Section II and Chapter 7 present important information about keyboard control and the windowing environment. Note that the screen on which commands are entered is separate from the screen on which program output is displayed. Plus, all the trace windows are also separately maintained. Thus if you were to run a program without screen I/O (PRINT statements), your display would be blank. In that case <**Break**> would return you to the command screen or <**Alt-S**>, for instance, would put you into a trace window.

Finally, Section III presents a few advanced topics and a list of Professional BASIC™ error messages.

1-8 COMPATIBILITY WITH IBM PC BASIC

Since the language of Professional BASIC™ is patterned after BASIC on the IBM PC, you will be able run programs that work on that BASIC in Professional BASIC™ after a few modifications (some programs may not need to be altered at all).

Professional BASIC™ has built-in features to aid you with your conversion. You will not have to learn much that is new in terms of the language (although there are some extensions you should become familiar with and begin to use).

Once you understand the areas of compatibility between the two languages you should be able to run programs under both BASICs with little or no modification. See Chapter 6 for more information.

1-9 COMPILING PROGRAMS

Use the IBM BASIC compiler to compile programs developed with Professional BASIC™. Of course, be sure to keep in mind the differences between the two systems to maintain compatibility.

SECTION I

PROGRAMMING WITH Professional BASIC™

INTRODUCTION TO SECTION I

In this section of the manual you will learn about the many enhancements offered by Professional BASIC™ and about the differences between Professional BASIC™ and IBM PC BASIC.

Chapter 2 describes some of the differences between Professional BASIC™ and IBM PC BASIC, in areas such as program line length, ranges of numbers, array dimensioning, and file handling.

Chapter 3 explains the differences in commands and statements that exist between Professional BASIC™ and IBM PC BASIC.

Chapter 4 discusses loading, running and saving programs.

Chapter 5 explains the Professional BASIC™ editor and how to effectively use it to create and edit your own programs. Chapter 5 also introduces the Dynamic Syntax Checker™ and gives a brief synopsis of some of the new commands and enhancements to the programing environment including line labels, long variable names, and other new commands.

Chapter 6 explains how to convert and run your programs from IBM PC BASIC to Professional BASIC™.

By the time you finish reading Section I you should know how to:

- Load a program from diskette into memory. Many programs can exist on the diskette, but only one may be in memory for editing or execution at any given moment.
- Run a program.
- Use the built-in editor to create and edit lines of BASIC program code.
- Save a created program to a diskette once it is created (with or without line numbers).
- Convert your existing BASIC programs to run under Professional BASIC™.

It is not the intention of this manual to actually teach BASIC. It is presumed that you are already familiar with how to program and, moreover, have some experience with doing so in IBM PC BASIC. If you are not familiar with BASIC, find a text on how to program in BASIC and use it in conjunction with this manual to create, edit, run, and debug programs.

The specific commands discussed in this Section are:

- LOAD - Read a program from disk into memory.
- RUN & SRUN - Cause the program in memory to begin execution.
- SAVE - Write to disk the program currently in memory.
- SAVEU - Write the program currently in memory to disk without line numbers.
- EDIT - Display a line of code for editing.
- LIST & LLIST - Display or print the current program or a section of it.

- DELETE - Delete a line or set of lines from the program in memory.
- FIND & FINDL - Display every occurrence of a specified variable or label.
- SORTV & SORTL - Display a sorted list of variables or line labels.
- SEARCH & SEARCHL - Display every occurrence of a specified string (e.g., key word like "PRINT" or "NEXT" or a label).
- INCR - sets the increment value for automatically generating line numbers, for the entry of new lines of program code.
- CLS - clear the command screen.
- NEW - clear the current program from memory and clear the screen.
- SYSTEM - exit to the operating system (PC DOS).

Some of the special features of Professional BASIC to be discussed in this section of the manual include:

- Operation of the built-in line editor. (Chapter 5)
- The Dynamic Syntax Checker™ - Each keystroke is monitored by the system for syntactic correctness. (Chapter 5)
- Use of the IBM PC function keys to speed editing. (Section 5-12)
- Use of line labels instead of line numbers in GOTO and GOSUB statements. (Section 5-1)
- Use of variable names and how to effectively use and enter long variable names. (Sections 5-1)
- How to load, merge and save programs and sections of code without line numbers in Professional BASIC™.

- How Professional BASIC™ runs programs created by another system. (Chapter 6)

NOTES: 1. All the commands and operations discussed in this Section are entered on what is called the "Command Screen". There is a difference in Professional BASIC™ between the screen to which program output is sent by the PRINT statement (the "Print Screen") and the screen on which commands are entered and programs created, edited, and listed (the "Command Screen"). Each is maintained separately and you can switch between them before, during, and after the execution of a program. Action on one screen does not disturb what is displayed on the other.

<Alt P> switches from the Command Screen to the Print Screen.
<Break> halts (suspends) program execution and switches to the Command Screen.
<Enter> switches from the Command Screen to the Print Screen (if you had just been there and had pressed **<Break>**) or to the last trace window viewed and resumes execution.

2. The **<Esc>** key is used to abort and erase the entry of any command or line of code after you have begun to type it in on the Command Screen.

In Section II of the manual you will learn more about the power of Professional BASIC™ - the tracing windows. These windows add another dimension to the process of running programs beyond what is described in this Section. It is essential, however, to understand the concepts presented here before you can effectively work with the system.

COMPATIBILITY WITH IBM PC BASIC - RUNTIME SYSTEM AND COMPILING

Many of your existing programs may run under Professional BASIC™ with only minor modifications. Some programs may require more work. However, after changes are made, the program should run under both systems. Hence, programs developed with Professional BASIC™ can be distributed and run with the IBM PC BASIC system. There is no need for others who need to run programs you develop to actually have Professional BASIC™ (unless your program needs to access a large amount of memory).

To compile a Professional BASIC™ program, use the compiler available from IBM. Again this will work as long as you do not need to access a large amount of memory <u>and</u> you have adhered to the requirements of the compiler.

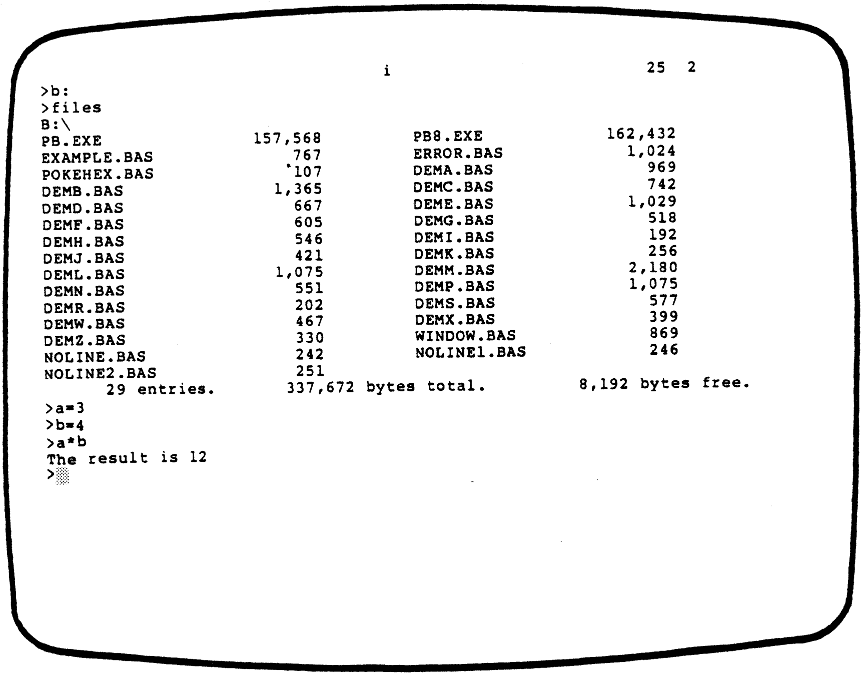
2. Professional BASIC DIFFERENCES

Professional BASIC™ is intended to provide essentially the same BASIC language as is now provided with the IBM PC BASIC. However, there are differences that must be explained to make the transition to Professional BASIC™. In this chapter some fundamental differences in the system and its capabilities are discussed. Differences in specific commands and statements are discussed in Chapter 3.

2-1 IMMEDIATE MODE

In Professional BASIC™ the immediate mode (meaning that the Command Screen is displayed and there is a blinking cursor after the > prompt) provides the functions for program debugging but does not allow immediate execution of all BASIC statements. The principal uses of immediate mode are to:

- Enter arithmetic expressions for immediate evaluation
- Display and set the value of a variable
- Display a list of disk files (directory) and rename or delete files



```
                                  i                        25   2
>b:
>files
B:\
PB.EXE                   157,568         PB8.EXE               162,432
EXAMPLE.BAS                  767         ERROR.BAS               1,024
POKEHEX.BAS                  107         DEMA.BAS                  969
DEMB.BAS                   1,365         DEMC.BAS                  742
DEMD.BAS                     667         DEME.BAS                1,029
DEMF.BAS                     605         DEMG.BAS                  518
DEMH.BAS                     546         DEMI.BAS                  192
DEMJ.BAS                     421         DEMK.BAS                  256
DEML.BAS                   1,075         DEMM.BAS                2,180
DEMN.BAS                     551         DEMP.BAS                1,075
DEMR.BAS                     202         DEMS.BAS                  577
DEMW.BAS                     467         DEMX.BAS                  399
DEMZ.BAS                     330         WINDOW.BAS                869
NOLINE.BAS                   242         NOLINE1.BAS               246
NOLINE2.BAS                  251
      29 entries.        337,672 bytes total.          8,192 bytes free.
>a=3
>b=4
>a*b
The result is 12
>▒
```

FIGURE 2-1 Screen displaying immediate mode execution of default drive change, the FILES command and some arithmetic operations

- Make and remove subdirectories or change from one subdirectory to another
- Change the default drive
- Load, run, and edit programs
- Enter new programs

Pressing <**Break**> from anywhere in the system will place you on the Command Screen. (If a Control Master™ file is running, such as the self-running demo, press <**Break**> twice.)

An example of an immediate mode command is finding the current value of a variable. In regular BASIC this would be done by typing "PRINT X" and <**Enter**>, then the current value of the variable "X" is displayed on the next line. In Professional BASIC™, however, simply enter "X" after the prompt and the current value will be displayed. To change a value, enter, for example, "X=21" and the value of X will become 21.

Appendix B contains a summary of the commands which can be entered while in the Immediate Mode.

2-2 BASIC PROGRAM LINES

<u>Line length</u> can be up to 311 characters, or about 4 screen lines.

<u>Line numbers</u> are used within the Professional BASIC™ system, and can be integers from 1 to 99999. However, you may write programs without line numbers with an external text editor by merely making sure that each line begins with a blank space. Professional BASIC™ will assign line numbers when the

program is brought into the system via the **LOAD** or **MERGE** commands. You may save programs without line numbers via the **SAVEU** command (see section 3-2).

Line number zero is not allowed.

Line labels are allowed. A line in a program can be given a name, which can be used instead of the line number in GOTO and GOSUB statements. The line label appears after the line number, can be any length, and must be followed by a semicolon (;).

Example: **100 start.of.main.routine; Print "Enter Data: "**

2-3 RANGES OF NUMERIC VALUES

Integer values may range from 2,147,483,647 to -2,147,483,648. Integers are 32-bit integers and thus are four bytes each. PC BASIC integers are 16-bit (2 bytes) with a range of 32,767 to -32,768.

Professional BASIC™ permits real numbers between the values of 1.67×10^{308} to 4.19×10^{-307} (double precision) and 3.37×10^{38} to 8.43×10^{-37} (single precision).

2-4 ARRAYS

Professional BASIC™ requires that all arrays must be defined in a program with a dimension (DIM) statement.

If an array xyz(a,b) is to be of size 100x100, then the following statement must appear in the program:

```
20 dim xyz(100,100)
```

This will establish the maximum size of the array in each dimension. (Variable names cannot be used to specify the size of a dimension. Use integer numbers only.) An alternative available in Professional BASIC™ is to use a form like the following to accomplish exactly the same thing:

```
20 dim xyz(1 to 100,1 to 100)
```

The value of this latter form is apparent in a case where you would like the first subscript to be a value other than 1. Perhaps you would like the first subscript to begin with 1900. The statement could be rewritten as follows:

```
20 dim xyz(1900 to 1999, 1 to 100)
```

This permits referencing an element in the array, such as xyz(1950,4), where a program may contain a set of statements like:

```
100 input "Year to look up:  ";year
110 input "Section number :  ";section
120 print "Number of students in section = 
";xyz(year,section)
```

The year may be used in the array reference directly without resorting to a translation routine, such as subtracting 1899 from the year put in line 100.

Size of Arrays

Because (1) Professional BASIC™ can access all available memory in the IBM PC and (2) array subscripts are 32-bit integers and can be as large as two billion (instead of 32,767), it is possible to have very large arrays.

You could specify a one dimensional array with 60,000 elements. Since single precision numbers and integers in Professional BASIC™ occupy 4 bytes each, such an array of numbers would occupy at least 240,000 bytes of memory in the computer, provided you had sufficient memory to also accommodate the program, the BASIC system, and DOS.

The maximum number of dimensions for an array is limited only by the length of a BASIC program line (311 characters).

2-5 FILE HANDLING

Professional BASIC supports both random and sequential files. You may have up to 8 files open concurrently. If you plan on using data files created by IBM PC BASIC programs please pay special attention to Section 6-3 because Professional BASIC™ uses the IEEE floating point format instead of the Microsoft storage format.

2-6 DISK FILES

File numbers

Currently files may only be numbered from 1 to 8.

Number of records in a file

Professional BASIC™ stores the number of records in an unsigned double word integer. Therefore the number of records in a file can be in excess of 4 billion.

Number of bytes in a record

The number of bytes per record is stored in an unsigned single word integer. This puts the upper limit of bytes per record at 65,535.

Number of bytes per file

The total number of bytes in a file is the number of records in the file times the number of bytes per record and cannot exceed 4,294,967,296 bytes. This exceeds the storage capability of most storage devices on the PC.

File buffers

The initial implementation of Professional BASIC™ restricts the disk file system as follows:

- Only files numbered 1 through 8 may be opened
- Files numbered 1 through 4 may have a maximum of 512 bytes per record. Files numbered 5 through 8 may have a maximum of 256 bytes per record. This cannot currently be redefined.

2-7 FILE ACCESS STATEMENTS

In general, the sequential and random file access statements will function in the same manner as they do under IBM PC BASIC. The possible exception to this is the LOC statement.

For random files, the LOC statement functions the same as in IBM PC BASIC. It returns the record number of the last record read from or written to the file. However, in sequential files, LOC returns the number of the last access to the file. That is, it returns a count of the number of reads or writes to the file, instead of the number of 128 byte blocks (records) of data read or written.

2-8 "STANDARD" INPUT/OUTPUT FILES

There are five "standard" files which IBM DOS Version 2.XX always maintains as open files. They are:

- standard input,
- standard output,
- standard error,
- standard auxiliary device and
- standard printer.

A more complete description of them can be found in the DOS 2.XX manual.

The default number of open files in Professional BASIC™ is five (5). By creating a file called CONFIG.SYS (see the IBM DOS Version 2.XX manual chapter titled "Configuring Your System") that contains the one text line **FILES=11** up to 8 files may be opened simultaneously. Otherwise 5 files can be opened at one time, but no more. The files numbered one through four (1 through 4) can have a record length up to 512. The files numbered five through eight (5 through 8) can have a maximum record length of 256.

2-9 COMPILING Professional BASIC™ PROGRAMS

At this time there is no compiler specifically designed for Professional BASIC™. However, because of the high degree of compatibility with IBM PC BASIC, the IBM BASIC Compiler can be used if the program development under Professional BASIC™ takes into account the language features and requirements of the Compiler.

If you wish to compile a program using the IBM BASIC Compiler, then in addition to the memory limitation of the compiler, do not use line labels, the optional format of the DIM statement, DIM(a to b), or the EXITFOR and EXITWHILE statements.

2-10 RUNTIME SYSTEM - INTERPRETER

Due to the language similarities, you can distribute programs to others who do not have Professional BASIC™, as long as these programs conform to the requirements of the IBM PC BASIC interpreter. If these requirements are met, programs can be developed under Professional BASIC™ and then distributed to run under the IBM PC BASIC interpreter.

2-11 CODING IN UPPER OR LOWER CASE

The editor does not care if you type code in upper or lower case. Whatever is typed in will be maintained in that format.

The system does not transform lower case letters into upper case letters. Upper and lower case are not considered unique (i.e., NaMe and nAmE are the same).

2-12 THE CURSOR

In Professional BASIC™ the cursor cannot be turned on or off nor can its size be changed by the LOCATE statement. The cursor is not normally displayed on the output screen but is kept off until an INPUT, LINE INPUT, or INKEY$ statement is encountered in the program. Then it is turned on until the program moves on to another statement. The third, fourth, and fifth parameters in a LOCATE statement will be accepted syntactically by the system but will not have any effect.

2-13 THE TAB KEY

The tab key on the keyboard will insert a tab character, chr$(9), in the program file if pressed while creating or editing a line of code. If a tab character is in a PRINT statement, it will cause the cursor location to be moved to the next tab location on the screen or printer, where the tab locations are at columns 1, 9, 17, 25, etc. (every 8th column).

2-14 COMMAND LINE PARAMETERS

When loading Professional BASIC™ the system does not recognize any parameters passed to the system such as a filename or size of file buffers.

2-15 MEMORY DISK, PRINT SPOOLER, AND OTHER UTILITIES - COMPATIBILITY

Professional BASIC™ may not work properly with some programs which manage the system memory and designate a portion of RAM memory for a memory disk drive or a print buffer. Also, other programs that are designed to run concurrently and attempt to alter the keyboard or are activiatated by special key combinations (like **<Ctrl-Alt>**) will not work with the current version of Professional BASIC™.

For those programs which do operate with Professional BASIC™, the total amount of memory available may be reduced and will be reflected in the number shown on the initial screen when Professional BASIC™ is loaded.

2-16 COLOR GRAPHICS VS. MONOCHROME SCREEN

The performance of Professional BASIC™ is far superior on the IBM monochrome screen system than on the color graphics system. Scrolling on and writing to a display driven by the color graphics adapter card are slower in order to avoid the problem of "snow" on the screen. The system is not slowed down on the COMPAQ computer, however.

3. COMMANDS & STATEMENTS - DIFFERENCES

In this chapter you will learn about some of the specific differences in commands and statements between Professional BASIC™ and IBM PC BASIC as well as some of the new features offered by Professional BASIC™. In most cases you will find that the commands and statements function under Professional BASIC™ in exactly the same way they do under IBM PC BASIC. If you have a specific question regarding a command or statement, first reference your IBM PC BASIC manual and then the Professional BASIC™ manual to check for differences. Since Professional BASIC™ is being implemented in stages, section 3-4 contains a list of those commands and statements not currently implemented.

3-1 COMMANDS and STATEMENTS - REPLACED

The following commands are not implemented in Professional BASIC™. Their functions, however, are included in the Professional BASIC™ programming environment.

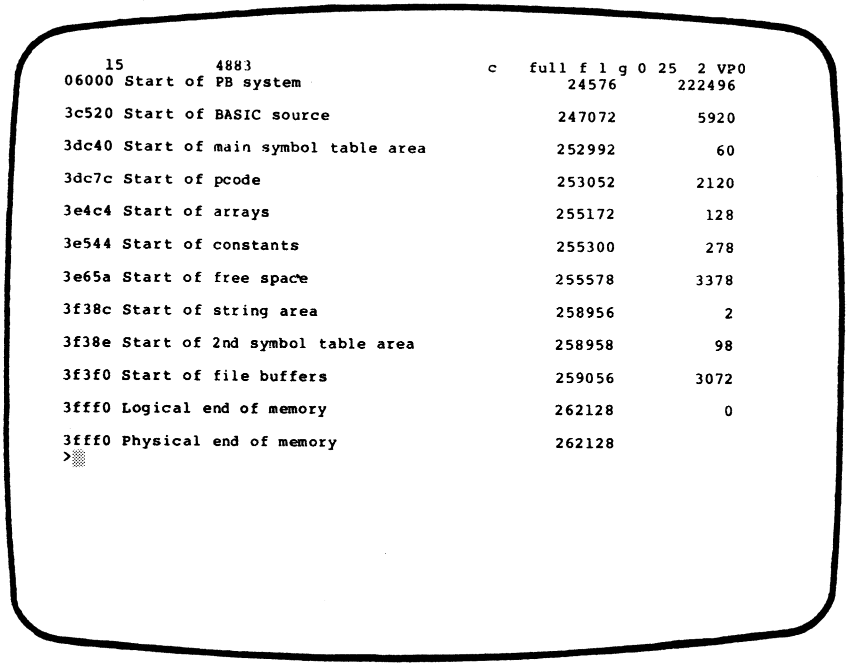
AUTO (see Section 5.3)
TROFF
TRON

3-2 COMMANDS and STATEMENTS - NEW

The following commands are new:

a:, b:, ...		Change the logged drive.
BEEPHI		Like the BEEP command, but with a higher pitch.
BEEPLO		Like the BEEP command, but with a lower pitch.
BREAK	n	Set a breakpoint on a designated line (line number or line label).
FIND	v	Find a specified variable, label, or line number and display on the command screen the lines in which it occurs. This command is for immediate mode only.
FINDL	v	Find a specified variable, label, or line number and highlight it everywhere it occurs in the listing. This command is for immediate mode only.
FINETRACE		Turn on the immediate mode option to see the evaluation of an arithmetic expression one step at a time.
INCR		Set the auto line numbering increment to a specified value. This command is for immediate mode only.

NOBREAK n	Turn off a designated breakpoint (n=line number or label).
NOBREAKALL	Turn off all breakpoints.
NOEX	Reset "executed code" flags (L window, option 2).
NOFINETRACE	Turn off the immediate mode arithmetic feature.
SAVEU	This command allows you to save programs without line numbers. It produces an ASCII text file with the extension .BAS and one leading blank at the beginning of each line. In all other respects it behaves like the SAVE command.
SEARCH a	Search for a specified group of characters and display on the command screen each line that contains the group of characters. This command is for immediate mode only.
SEARCHL a	Search for a specified group of characters and highlight the group everywhere it occurs in the listing. This command is for immediate mode only.
SETTOP n	Set the top of memory accessible by the system to a designated value. The parameter _n_ is a hexadecimal number. Thus to set the top of available memory to 320k, for



```
    15          4883                             c    full f l g 0 25  2 VP0
06000 Start of PB system                                24576      222496

3c520 Start of BASIC source                            247072        5920

3dc40 Start of main symbol table area                  252992          60

3dc7c Start of pcode                                   253052        2120

3e4c4 Start of arrays                                  255172         128

3e544 Start of constants                               255300         278

3e65a Start of free space                              255578        3378

3f38c Start of string area                             258956           2

3f38e Start of 2nd symbol table area                   258958          98

3f3f0 Start of file buffers                            259056        3072

3fff0 Logical end of memory                            262128           0

3fff0 Physical end of memory                           262128
>
```

FIGURE 3-1 **This is an example of the output produced by the ,SIZE command. The self-running demo is currently loaded.**

	instance, the command would be: **SETTOP 50000.** To set the top to 512k: **SETTOP 80000.**
SORTV	Display an alphabetical listing of all variables and array names on the screen.
SORTL	Display an alphabetical listing of all line labels on the screen.
SRUN	Prepare to run (just as if the RUN command had been given), but wait at the first line of code on the List Trace window for you to press <**Enter**> or <**space bar**> to begin executing instructions.
,BUFFERS	Shows list of file buffers.
,SIZE	Shows information on the use of system memory.

The following program statements are new:

EXITFOR	Exit the current FOR loop and continue execution with the first instruction after the NEXT statement.
EXITWHILE	Exit the current WHILE loop and continue execution with the first instruction after the WEND statement.

3-3 COMMANDS and STATEMENTS - WITH DIFFERENCES

The following commands and statements are implemented with the differences described here.

DIM	All array variables must be dimensioned. There is no default value for the size of arrays and variable dimensions are not allowed. You must use integer values only in a DIM statement, not variable names.
EDIT	There is no "EDIT ." option. Press the **F8** key to display for edit the last line edited.
FOR/NEXT	The FOR/NEXT statements must be physically and logically paired. That is, for each and every FOR there must be a matching NEXT. Also, FOR/NEXT loops cannot overlap each other. When nesting loops, the inner loops must be completely contained within the outer loops. Multiple FOR statements can be attached to a single NEXT statement only if the NEXT lists all the corresponding indexes (i.e. next i,j,k). Multiple NEXT statements cannot be attached to a single FOR. That is, NEXT statements cannot be hidden in conditional statements. Instead of using a NEXT in the conditional statement you should use a GOTO statement to direct control to the appropriate NEXT.

LOAD	The quotation marks surrounding the filespec are not needed, but may be used if desired. A filespec extension of .BAS is assumed if no extension is supplied. If a LOAD command is entered with no filename specified, the last file loaded or saved will be loaded again. There is no ",**R**" option to "Load and Run" a program. LOAD is an immediate mode command and is not available as a program statement. The LOAD operation does a number of things: the file is read from disk, each line of code is checked for proper syntax and marked if in error, and the system is "seeded" to set up the multi-window capabilities. Thus the load takes longer than it does in BASICA.
LOCATE	The third parameter in the LOCATE statement which is used to turn the cursor on/off is accepted by the syntax checker in the system, but does not actually turn the cursor on or off when the statement is executed. Cursor size (4th and 5th parameters) cannot be controlled either.
FIELD	Array elements cannot be used in the FIELD statement. Only unsubscripted variables can be used. That is, "20 as ABC$" is permitted but "20 as ABC$(3)" is not.

FILES	Quotes are not needed. You may specify a drive letter followed by a colon to get a list of all files. File sizes in bytes, total number of files, bytes in the displayed files, and bytes remaining on the disk are also presented.
MERGE	You can merge lines of code without line numbers and place that code anywhere you choose in the current program. See Section 5-14.
RUN	This command does not have the "line number" or ",**R**" options.
SAVE	Same as for LOAD above. The filespec extension of .BAS will be added if none is supplied. If no filename is specified in the command, the name of the last program loaded or saved will be used. The default filename is shown on the <**Y**> trace screen. SAVE is an immediate mode command and is not available as a program statement. The file saved is an ASCII file and thus is <u>not</u> in tokenized form.
WHILE/WEND	Comments above on FOR/NEXT also apply for pairing of WHILE/WEND statements.

3-4 COMMANDS and STATEMENTS - NOT YET AVAILABLE

The Professional BASIC™ system is being implemented in stages. At this time, the following commands and statements are not available for use.

BLOAD	ERASE	ON TIMER	STICK
BSAVE	GET(graphics)	OPEN "COM	STRIG
CALL	INPUT$	PEN	STRIG(n)
CLEAR	ON COM(n)	PLAY	USR
COM(n)	ON PEN	PLAY(n)	VARPTR
DEF USR	ON PLAY	PMAP	VARPTR$
DRAW	ON STRIG(n)	PUT (graphics)	VIEW
			WINDOW

The following is a list of commands, functions, and statements supported in Version 2.0x of **Professional BASIC**™:

ABS	DELETE	INPUT#	MERGE	PRINT #	SRI
ASC	DIM	INSTR	MID$	PRINT # USING	STC
ATN	EDIT	INT	MKDIR	PSET	STF
AUTO*	END	KEY	MKKI$,MKS$,MKD$	PUT (files)	STF
BEEP (BEEPHI**	& BEEPLO**)	KEY(n)	MOTOR	RANDOMIZE	SW
CDBL	EOF	KILL	NAME	READ	SYS
CHAIN	ERASE		NEW	REM	TAE
CHDIR	ERROR		NEXT	RENUM	TAN
CHR$	ERR,ERL	LEFT$	OCT$	RESET	TIM
CIRCLE	EXITFOR**	LEN	ON ERROR	RESUME	
CINT	EXITWHILE**	LET	ON..GOSUB	RESTORE	TIM
CLOSE	EXP	LINE	ON..GOTO	RETURN	TRC
CLS	FIELD	LINE INPUT	ON KEY(n)	RIGHT$	TRC
COLOR	FILES	LINE INPUT#	OPEN COM	RMDIR	VAI
COMMON	FIX	LIST	OPEN	RND	
CONT	FOR	LLIST	OPTION BASE	RSET	WA
COS	FRE	LOAD	OUT	RUN	WH
CSNG	GET (files)	LOC	PAINT	SAVE	WE
CSRLIN	GOSUB	LOCATE	PEEK	SCREEN	
CVI,CVS,CVD	GOTO	LOF	POINT	SGN	WII
DATA	HEX$	LOG	POKE	SIN	WR
DATE$	IF	LPOS	POS	SOUND	WR
DEF FN	INKEY	LPRINT	PRESET	SPACE$	
DEF SEG	INP	LPRINT USING	PRINT	SPC	
DEF type	INPUT	LSET	PRINT USING	SQR	

* This command is handled by **Professional BASIC**™ in the environment.
** This command is in **Professional BASIC**, but not in IBM PC BASIC.

4. LOADING, RUNNING & SAVING PROGRAMS

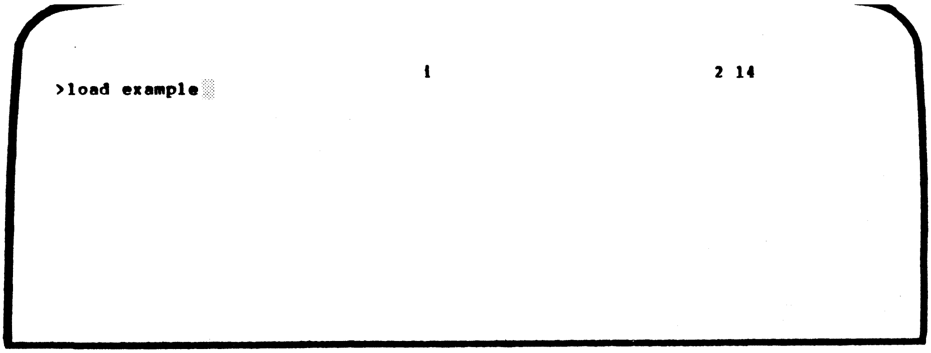
In order to become comfortable with running a program, we are going to load a sample program that is provided on your Professional BASIC™ disk. The program filename is "EXAMPLE.BAS". (Note: If you have questions about the operation of the editor, see the next chapter.)

First, start Professional BASIC™ by typing PB (if you have an 8087) or PB8 (if you don't) at the DOS prompt and pressing **<Enter>**.

```
A>pb
```

Press any key after the initial screen appears. (If you did not touch the keyboard a self-running demo would begin after about 15 seconds. Press the **<Break>** key twice after the demo program is loaded and running to exit it.)

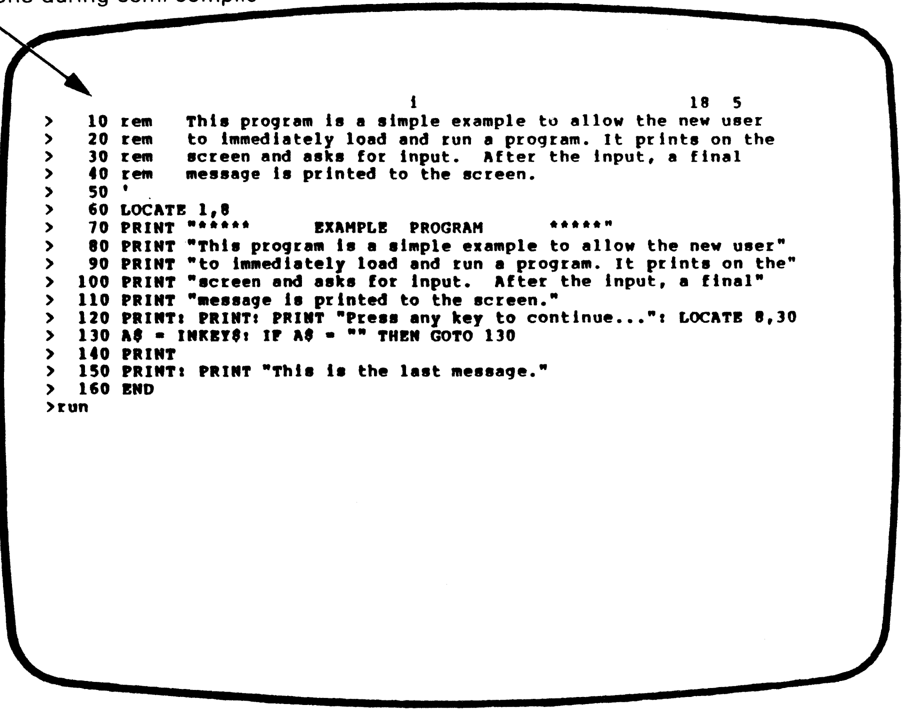
You will now be on the Professional BASIC™ Command Screen. This is where all commands to the Professional BASIC™ system are typed. The line at the top of the screen is called the Status Line. (Explained in Chapter 7) The system prompt consists of the greater than character (>) and a flashing rectangular cursor which indicates that the system is waiting for your input.



```
                                        1                       2 14
>load example
```

FIGURE 4-1 Screen showing LOAD command (before <ENTER> is pressed). EXAMPLE.BAS is one of the demos supplied on your system disk

ne number of
structions during semi-compile



```
                                        1                     18  5
>   10 rem   This program is a simple example to allow the new user
>   20 rem   to immediately load and run a program. It prints on the
>   30 rem   screen and asks for input.  After the input, a final
>   40 rem   message is printed to the screen.
>   50 '
>   60 LOCATE 1,8
>   70 PRINT "*****      EXAMPLE  PROGRAM      *****"
>   80 PRINT "This program is a simple example to allow the new user"
>   90 PRINT "to immediately load and run a program. It prints on the"
>  100 PRINT "screen and asks for input.  After the input, a final"
>  110 PRINT "message is printed to the screen."
>  120 PRINT: PRINT: PRINT "Press any key to continue...": LOCATE 8,30
>  130 A$ = INKEY$: IF A$ = "" THEN GOTO 130
>  140 PRINT
>  150 PRINT: PRINT "This is the last message."
>  160 END
>run
```

FIGURE 4-2 Command screen showing the EXAMPLE.BAS program and the run command just before the <ENTER> key is pressed

Type the LOAD command as follows to load the demo program EXAMPLE.BAS from your copy of the Professional BASIC™ system disk:

```
>load example
```

You do not have to precede (or enclose) the filename with quotes as is required in IBM PC BASIC. The program will list on the screen as it loads. Every line of code is checked for proper syntax as the program loads. If you wish to examine the program in detail as it loads, you may suspend loading and enter single step mode by pressing the **<space bar>**. Each succeeding press of the **<space bar>** will cause one more line to be read and loaded. To return to full speed loading, press the **<Enter>** key. When the program is loaded, the cursor will reappear.

Type in the RUN command as follows:

```
>run
```

When the "RUN" command is entered there is a brief pause before program execution begins. This pause occurs while "semi-compilation" is performed. After the pause the screen switches to the user output screen (See the "NOTES" in the Introduction to Section I; Pg. I-iv). The line number being compiled is displayed at the top of the screen. Since there are multiple passes, the sequence of line numbers is displayed more than once. For a long program, a few seconds may be needed to perform the analysis. Once completed, the program will execute significantly faster than an interpreted program, but not as quickly as a fully compiled program. This semi-compilation will only occur once if you do not edit or change your program. The next time you enter RUN, it will begin to execute instantly.

The sample program itself is very simple. It prints to the screen and waits for a key input to continue (see Figure 4-2). Press any key. Next, the program prints one more line and ends. At the end of the program you are returned to the command screen. To see the screen as it was at the end of program execution, press **<Alt P>**. To return to the Command Screen press the **<Break>** key (the key at the far upper right corner of the keyboard that has the words "Scroll Lock" on the top and "Break" on the front).

To exit from a running program and go to the Command Screen, press the **<Break>** key. You may also press **<Ctrl Break>**, as you would normally do in IBM PC BASIC to halt (suspend) an executing program. Both methods produce the same result.

The advantage to having separate screens for entering commands and showing program output is that it is possible to perform various operations like listing lines of code, examining various variable values, setting variable values, etc. while not disturbing the output screen. Then when program execution is continued from the point of suspension, the output screen is left just as it was. The more complex the programs you write and debug, the more you will appreciate this feature.

To save the program, use the SAVE command as follows:

```
>save example
```

You could just enter the command SAVE by itself. The current file would be saved using the last filespec referenced in a SAVE or LOAD. A NEW command nullifies this option until another SAVE or LOAD with a specified filename is executed.

You do not have the option of saving a file in "protected" or "tokenized" format as in IBM PC BASIC. Professional BASIC™ saves and loads programs only in ASCII (text) format.

Except for the separation of the program output screen (Print Screen) from the Command Screen, the above process is identical to IBM PC BASIC.

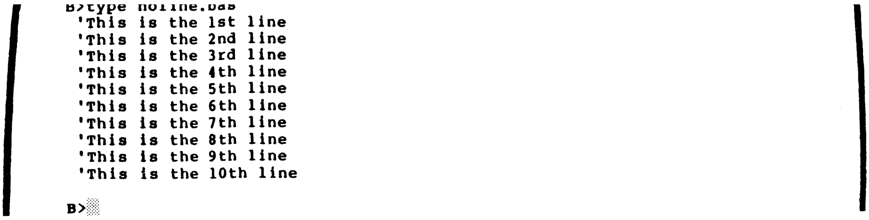
NOTE -- Loading and running programs from other systems: Chapter 6 explains the process of loading and editing programs you may already have on disk. Assuming that the program you wish to run conforms to the language definition in Chapter 3 of this manual and is in ASCII format on disk, Chapter 6 will explain how to proceed. However, we recommend reading about the Professional BASIC™ editor (Chapter 5) before you begin to use your own programs.

4-1 PROGRAMS WITHOUT LINE NUMBERS

Professional BASIC™ allows you to load programs which have no line numbers or have only a few line numbers created by an external text editor or saved from Professional BASIC™ using the SAVEU command. The key to making this work, however, is to remember that:

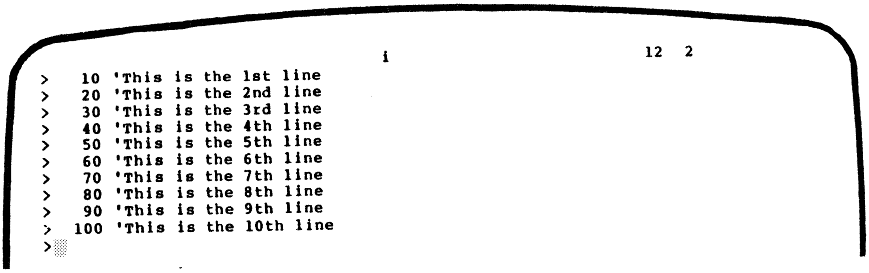
> Each line of code in the program **must** start with <u>either</u> a line number <u>or</u> at least one space.

Professional BASIC™ will assign line numbers for lines that start with a space. The default increment value that will be used between lines is "10". If a line is loaded that already has a line number, followed by unnumbered lines (each beginning with a space), then that line number will be maintained and each subsequent line will be assigned line numbers above it.



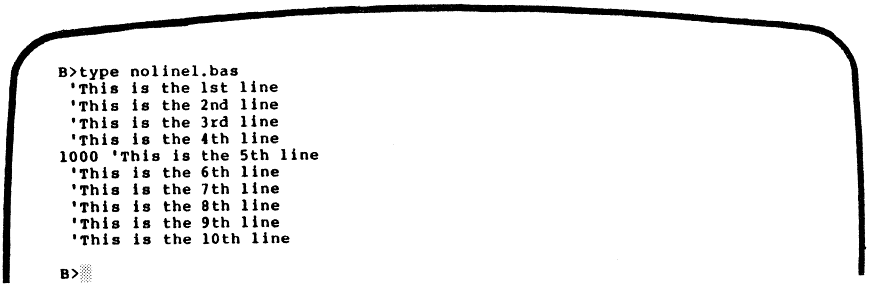
```
B>type noline.bas
 'This is the 1st line
 'This is the 2nd line
 'This is the 3rd line
 'This is the 4th line
 'This is the 5th line
 'This is the 6th line
 'This is the 7th line
 'This is the 8th line
 'This is the 9th line
 'This is the 10th line

B>
```

FIGURE 4-3 **The supplied demonstration program NOLINE.BAS as it appears on your disk**



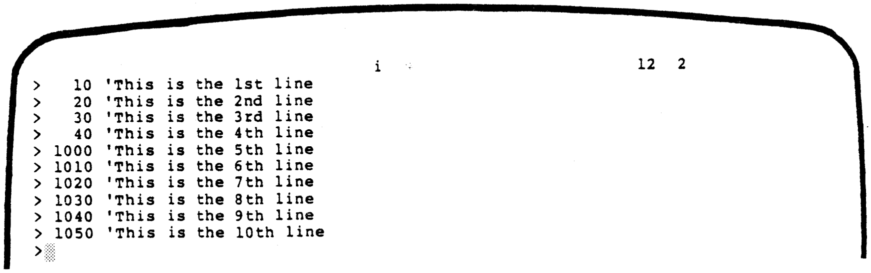
```
                                       i                        12   2
>   10 'This is the 1st line
>   20 'This is the 2nd line
>   30 'This is the 3rd line
>   40 'This is the 4th line
>   50 'This is the 5th line
>   60 'This is the 6th line
>   70 'This is the 7th line
>   80 'This is the 8th line
>   90 'This is the 9th line
>  100 'This is the 10th line
>
```

FIGURE 4-4 **The supplied demonstration program NOLINE.BAS after it is loaded by Professional BASIC™**



```
B>type nolinel.bas
 'This is the 1st line
 'This is the 2nd line
 'This is the 3rd line
 'This is the 4th line
1000 'This is the 5th line
 'This is the 6th line
 'This is the 7th line
 'This is the 8th line
 'This is the 9th line
 'This is the 10th line

B>
```

FIGURE 4-5 **The supplied demonstration program NOLINE1.BAS as it appears on your disk**



```
                                       i                        12   2
>   10 'This is the 1st line
>   20 'This is the 2nd line
>   30 'This is the 3rd line
>   40 'This is the 4th line
> 1000 'This is the 5th line
> 1010 'This is the 6th line
> 1020 'This is the 7th line
> 1030 'This is the 8th line
> 1040 'This is the 9th line
> 1050 'This is the 10th line
>
```

FIGURE 4-6 **NOLINE1.BAS after it is loaded by Professional BASIC™**

Each of the subsequent lines is incremented by 10. A few examples should help illustrate how this works.

Load the program NOLINE.BAS which is on the Professional BASIC™ system disk.

```
>load noline
```

Figure 4-3 shows what the program looks like if you were to use the DOS **type** command (In DOS at the A> prompt enter: **type noline.bas**) Figure 4-4 shows the program after it is loaded in Professional BASIC™

Figure 4-5 shows the same program but with one line having a line number. Figure 4-6 shows the program loaded. This is the program NOLINE1.BAS on your disk.

Figure 4-7 shows the program NOLINE2.BAS which has two lines with line numbers. Figure 4-8 is the same program after loading it into Professional BASIC™.

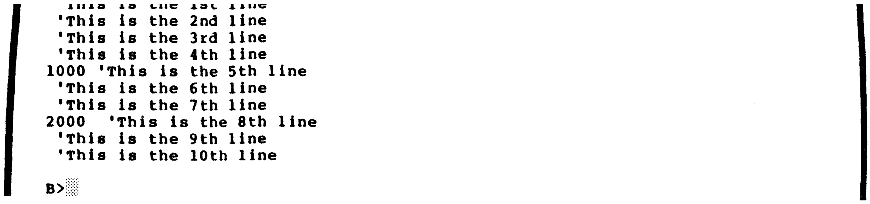
If you wish to load a program from disk using a line number increment other than the default of ten, you must do so using the MERGE command. To illustrate this, enter:

```
>new
```

to clear Professional BASIC™ of any programs that are already in the system, followed by the command:

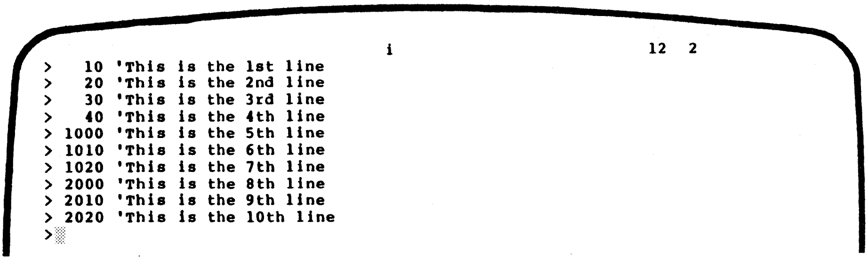
```
>incr 100
```

to set the line number increment to 100. At the Professional BASIC™ prompt enter:



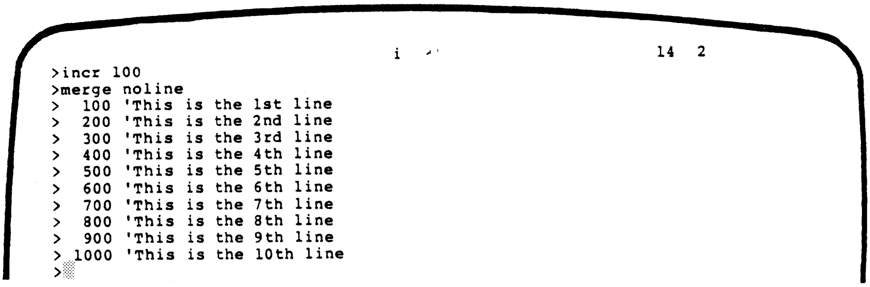
```
 'This is the 1st line
 'This is the 2nd line
 'This is the 3rd line
 'This is the 4th line
1000 'This is the 5th line
 'This is the 6th line
 'This is the 7th line
2000  'This is the 8th line
 'This is the 9th line
 'This is the 10th line

B>
```

FIGURE 4-7 **The supplied demonstration program NOLINE2.BAS as it appears on your disk**



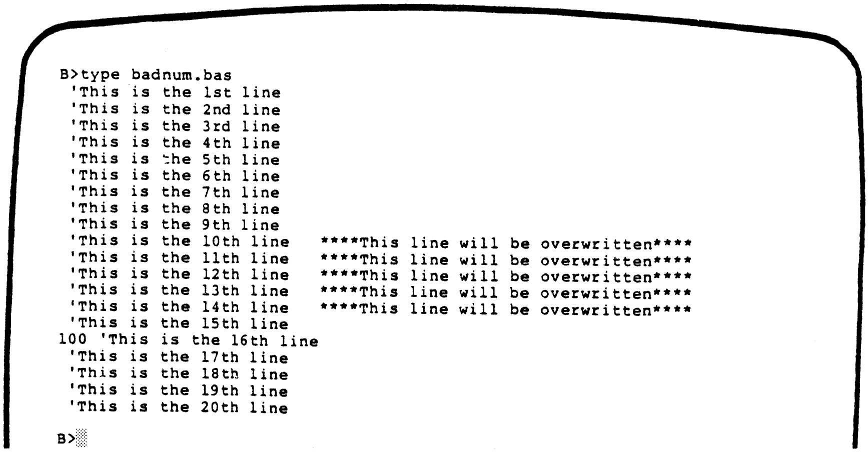
```
                                     i                    12   2
>    10 'This is the 1st line
>    20 'This is the 2nd line
>    30 'This is the 3rd line
>    40 'This is the 4th line
>  1000 'This is the 5th line
>  1010 'This is the 6th line
>  1020 'This is the 7th line
>  2000 'This is the 8th line
>  2010 'This is the 9th line
>  2020 'This is the 10th line
>
```

FIGURE 4-8 **NOLINE2.BAS after it is loaded by Professional BASIC™**



```
                                     i                    14   2
>incr 100
>merge noline
>   100 'This is the 1st line
>   200 'This is the 2nd line
>   300 'This is the 3rd line
>   400 'This is the 4th line
>   500 'This is the 5th line
>   600 'This is the 6th line
>   700 'This is the 7th line
>   800 'This is the 8th line
>   900 'This is the 9th line
>  1000 'This is the 10th line
>
```

FIGURE 4-9 **NOLINE.BAS after it is loaded into Professional BASIC™ via the MERGE command with an increment of 100**



```
B>type badnum.bas
 'This is the 1st line
 'This is the 2nd line
 'This is the 3rd line
 'This is the 4th line
 'This is the 5th line
 'This is the 6th line
 'This is the 7th line
 'This is the 8th line
 'This is the 9th line
 'This is the 10th line    ****This line will be overwritten****
 'This is the 11th line    ****This line will be overwritten****
 'This is the 12th line    ****This line will be overwritten****
 'This is the 13th line    ****This line will be overwritten****
 'This is the 14th line    ****This line will be overwritten****
 'This is the 15th line
100 'This is the 16th line
 'This is the 17th line
 'This is the 18th line
 'This is the 19th line
 'This is the 20th line

B>
```

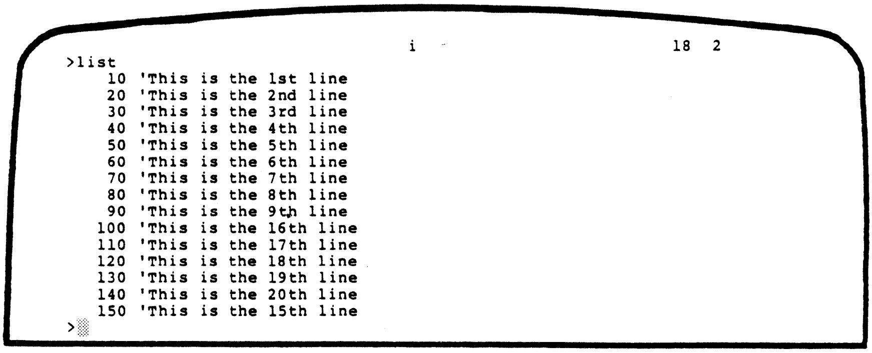
```
        >merge noline
```

Figure 4-9 is an example of the result.

Because Professional BASIC™ can use line labels instead of line numbers in GOTO and GOSUB statements this system can work without the problem of a GOTO or GOSUB having to address a particular line which may get a different line number during subsequent loads, where program lines were deleted or inserted ahead of the target line. Of course, you can leave some line numbers in, if you really want to keep the number option for such lines.

For more on this option of having code without line numbers see the discussion on the MERGE command in Section 5-14 of the next chapter. You can create sections of code without line numbers which can be easily merged into any place in a program. This facility allows you to have libraries of routines which can be used in a flexible way in constructing programs.

Note that you can produce "logical errors" in a program file which has a few line numbers. A designated line number can conflict with a line number that is generated at load time. For instance, if you have a program such as the one in Figure 4-10. The line with the line number 100 will replace the tenth line in the program. The lines after that will also replace the 11th through 14th lines. Figure 4-11 is a listing of the program after loading. Notice that some of the lines from the source file are missing because they were overwritten.



```
                                          i                     18  2
>list
    10 'This is the 1st line
    20 'This is the 2nd line
    30 'This is the 3rd line
    40 'This is the 4th line
    50 'This is the 5th line
    60 'This is the 6th line
    70 'This is the 7th line
    80 'This is the 8th line
    90 'This is the 9th line
   100 'This is the 16th line
   110 'This is the 17th line
   120 'This is the 18th line
   130 'This is the 19th line
   140 'This is the 20th line
   150 'This is the 15th line
>
```

FIGURE 4-11 **The program BADNUM.BAS after it is loaded into Professional BASIC™ and listed with the LIST command**

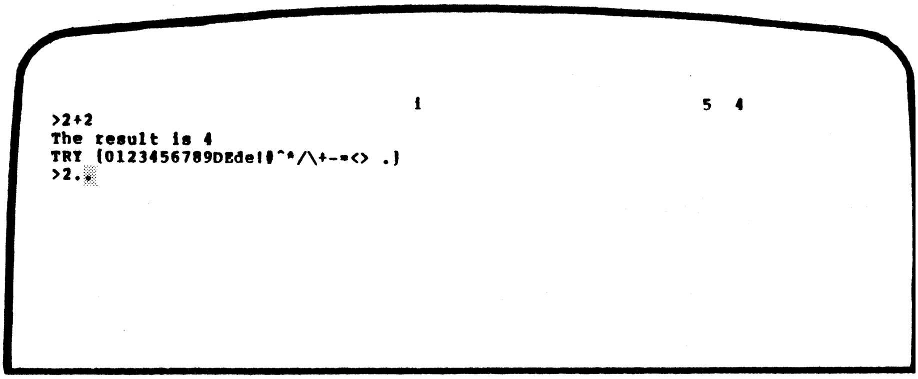
5. CREATING & EDITING PROGRAMS

This chapter explains the Dynamic Syntax Checker™ and the Professional BASIC™ editor -- how to enter commands and lines of code. You will also learn about entering long variable names and several other powerful and useful features of Professional BASIC.

5-1 THE DYNAMIC SYNTAX CHECKER™

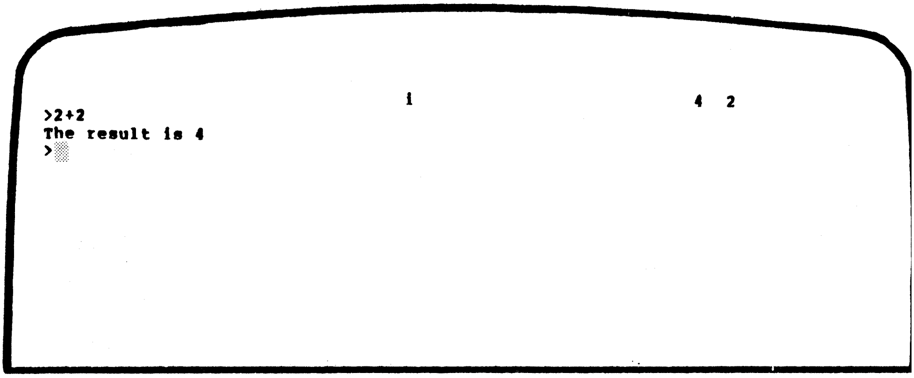
One of the unique features of Professional BASIC™ is its system of syntax checking. All input to the command screen (at a > prompt) from the keyboard is monitored and checked, keystroke by keystroke. An error in syntax is rejected by the system with a beep and the cursor remains in place. If a second error is made at the same point, the system makes an internal evaluation of every possible keystroke and presents a list of acceptable keystrokes to the user on a "TRY" line.

To illustrate, go to the command screen in Professional BASIC™ (press <**Break**> if you are not already there). We will perform an arithmetic operation in the "immediate mode" (command directly entered and executed on the command screen):



```
                                    1                           5  4
>2+2
The result is 4
TRY [0123456789DEde!#^*/\+-=<> .]
>2..
```

FIGURE 5-1 Screen showing "Try" line after syntax erroı made twice at the same place



```
                                    1                           4  2
>2+2
The result is 4
>
```

FIGURE 5-2 Same screen as Figure 5-1 after <Esc> is pressı

- Enter: **2+2** and press the **Enter** key. The answer is shown as: **The result is 4.** Now let's try it again, but make a mistake on purpose.

- Enter: **2..** The second decimal point is rejected. Press the period again. Since you have made two errors at that point, the system presents a list of acceptable keystrokes. The display will look similar to Figure 5-1.

When you type one of the acceptable characters, the "TRY" line will disappear. If you want to get out of the operation, the **<Esc>** will clear both lines.

The syntax checking system is quite powerful. For instance, it will prevent you from entering too many right parentheses for the matching number of left ones. "Immediate mode" commands as well as program instructions are all checked by the system. If you violate a rule of syntax for one of the BASIC commands, functions, or statements, the system will announce with a beep that an error has been made and wait for a correct keystroke before continuing. <u>You are literally prevented from entering a line of code with a syntax error lurking in it</u>.

While a program is running, user input requested by an **INPUT** statement will also be checked by the system. If you try to enter alphabetical characters where a number is expected, the system will reject the characters. If you try to enter too many or too few data elements required by an **INPUT** statement, the system will also respond and indicate your error.

5-2 CREATING, RUNNING, AND EDITING PROGRAMS - SOME BACKGROUND

Creating and editing programs in Professional BASIC™ is easy, with many facilities available to aid you in both the creation of new programs and the editing of existing ones. Most of the procedures followed in IBM PC BASIC are also followed in Professional BASIC™. In addition to the Dynamic Syntax Checker™, there are some other important differences you need to learn. The new features to become familiar with are:

- Line numbers can be automatically generated by pressing the **<space bar>**.
- Line length can be up to 311 characters.
- Auto label generator -- for long variable names and line labels there is a feature which allows you to press a key (the "@" character) to generate the name or label after typing just a few characters (enough to uniquely identify it).
- Line numbers (required) range from 1 to 99999 (0 is not allowed).
- **FIND** command - locates and displays every use of a specified variable, label, or line number.
- **SEARCH** command - locates and displays the use of any designated text in the program. For instance, locate every "PRINT" or "FOR" in the program.

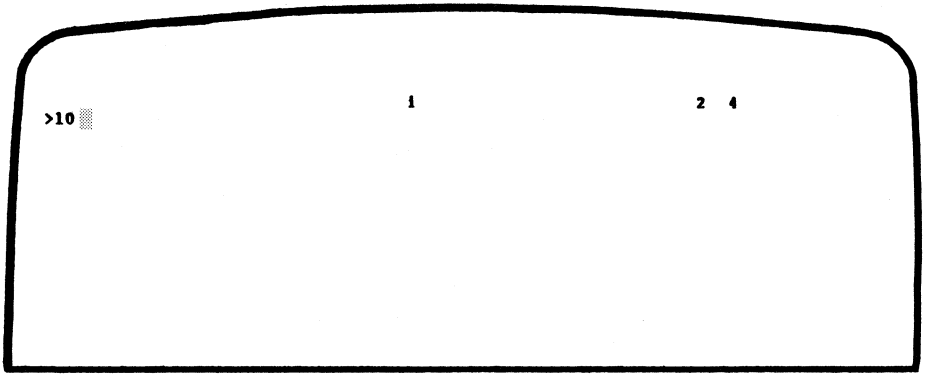
- **SORTV** & **SORTL** commands - These two commands produce a sorted list of all the variables (either simple variable or array) and a list of all line labels in a program. With the SORTV and FIND commands the location and use of any variable can be identified. With the SORTL and FIND commands the location and use of any line label can be identified.

- Before the program is executed (after a "run" or "srun" is entered) the entire program is analyzed for errors. You do not have to wait until each portion of the program is executed to find out, for example, that you have a **GOTO 150** statement but no line 150 in the program.

These facilities, together with the screen windows described in Section II, will help you create and debug programs quickly and easily.

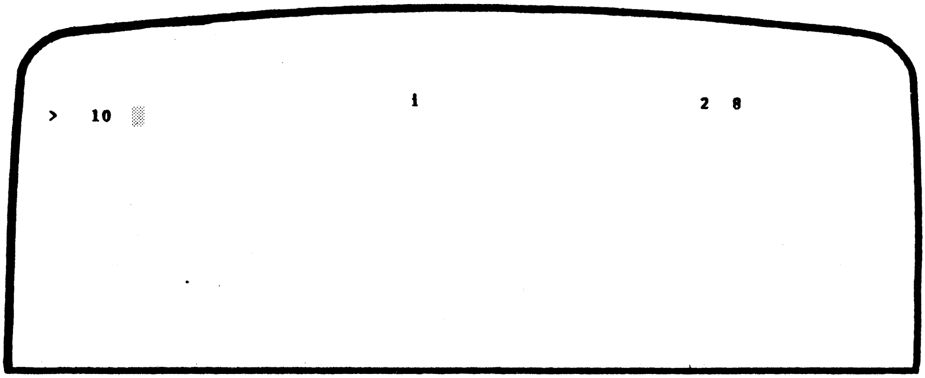
In addition to the above extensions, there are a few differences to note with Professional BASIC™. These include:

- Line oriented editor -- Because of the extensive Dynamic Syntax Checking capability the editor works differently than in IBM PC BASIC. The up and down cursor keys will not move the cursor around the screen for editing purposes. Another facility is used to accomplish much the same thing (**<F9>** and **<F10>** keys). The **Up** and **Down** cursor keys will move the cursor only within a line of program code (a program line can be up to 4 screen lines long).



```
                                   1                  2  4
>10
```

FIGURE 5-3a Command Screen after line number is typed b before <Space bar> is pressed



```
                                   1                  2  8
>   10
```

FIGURE 5-3b Command Screen after <space bar> is pressed

- When using the SAVE and LOAD commands it is not necessary to enclose the filename with quotes. The filename is also optional, with the filename used in the previous LOAD or SAVE, if any, as the default. (The default filename is shown on the <Y> screen.) Thus you can assure using the same filename in the SAVE command as was used in the previous LOAD command. Using the NEW command nullifies this option until another SAVE or LOAD with a specified filename is executed.

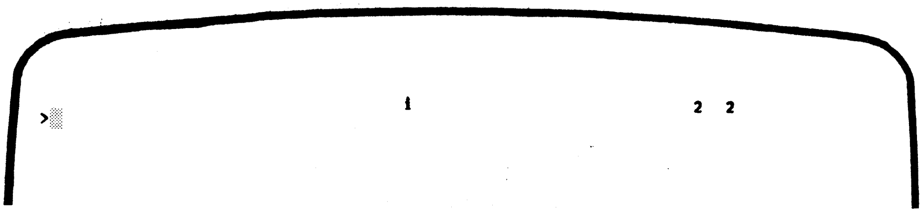
- You may type code in upper and lower case. The system does not change lower case letters into upper case letters. Upper and lower case are not considered unique (i.e., <u>NaMe</u> and <u>nAmE</u> are the same)

With these features in mind let's create a program using them.

5-3 ENTERING AND EDITING PROGRAM LINES

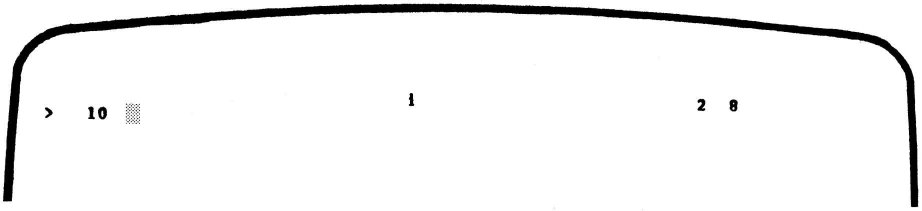
Load in Professional BASIC™ and press a key to clear the initial screen. If it is already loaded in the machine, type **new** on the Command Screen and press the <**Enter**> key. The cursor (the rectangular blinking box) will be sitting to the right of the > symbol, which is the prompt for entering a command in Professional BASIC™. In IBM PC BASIC this condition is indicated by an "**OK**" on a line followed by a blinking underline cursor on the next line, under the "**O**" in "**OK**".

To begin typing in a line of BASIC code, you must first type the line number. Type in a **10** followed by a space. The system



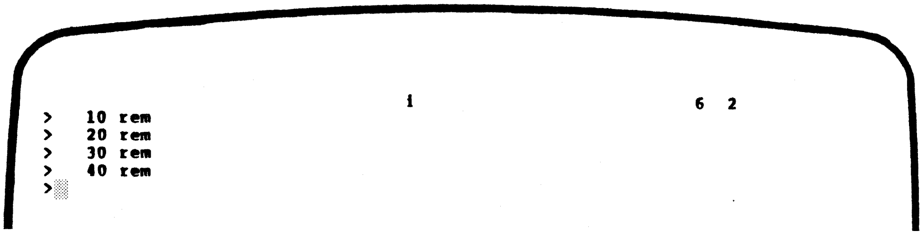
```
>                      1                     2  2
```

FIGURE 5-4a Command Screen before <space bar> is pressec



```
>  10                  1                     2  8
```

FIGURE 5-4b Command Screen after <space bar> is pressed



```
                       1                     6  2
>  10 rem
>  20 rem
>  30 rem
>  40 rem
>
```

FIGURE 5-5 Command Screen after a few program lines ha
been entered

recognizes this number "10" as a line number when the space key is pressed. If you meant to enter in a number 10 which was the first part of a calculation in "immediate mode" (see Section 5-10), you should not put a space between the 10 and the operator (+, -, *, /) or other part of an expression. If you press the backspace key, the display will change to the previous state, before the space was typed.

An alternate way to get a line number is with the default system that generates line numbers.

1. Press the <**Esc**> key to clear (or abort) the current line.

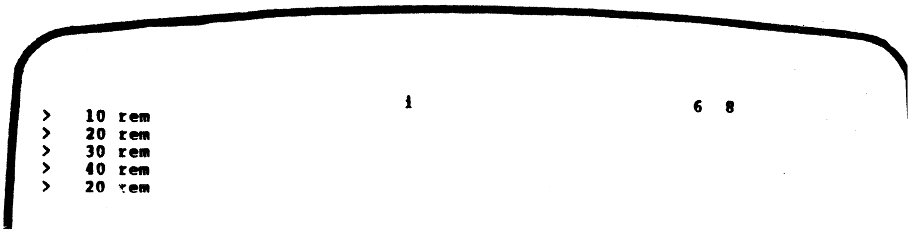
2. Then press the <**space bar**> once. The next logical line number (10) is generated for you and you are positioned to enter in a line of code. (BASIC requires a space following the line number so the cursor is positioned to the second space or column following the line number.) Before proceeding to enter a program try the following to get used to the number generator.

3. Type in **rem** and press the <**Enter**> key.

4. Then press the <**space bar**> to generate the next line number, 20.

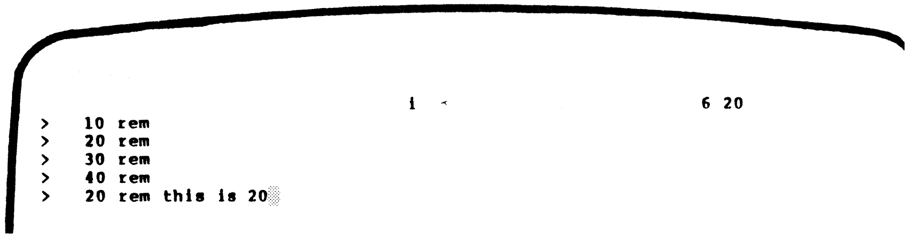
5. Repeat steps 3 and 4 a few times to see how it works. Your screen should look something like Figure 5-5.

You have the option of resetting the increment value of 10 to some other number. To automatically generate line numbers in



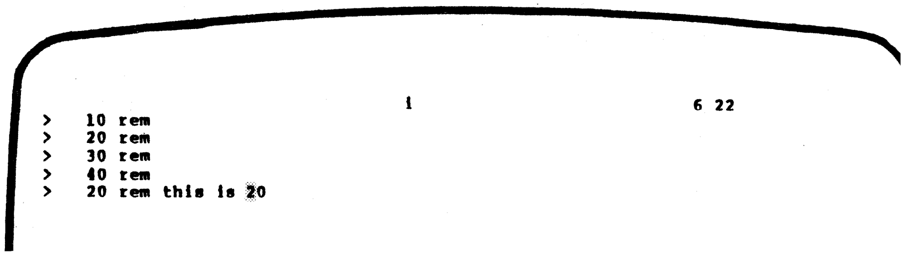
```
                                    i                      6  8
>    10 rem
>    20 rem
>    30 rem
>    40 rem
>    20 rem
```

FIGURE 5-6 Editing line 20 with cursor at the front of the l



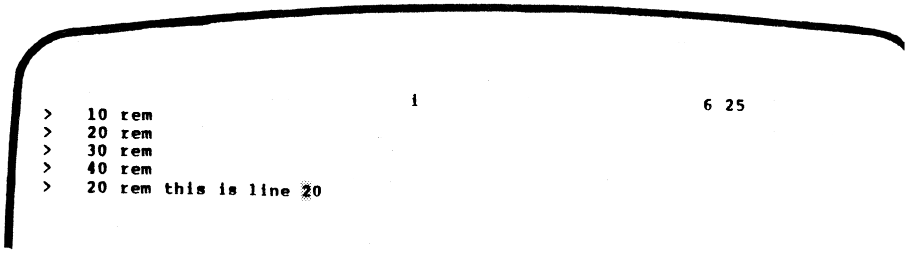
```
                                    i                      6 20
>    10 rem
>    20 rem
>    30 rem
>    40 rem
>    20 rem this is 20
```

FIGURE 5-7 Line 20 after text has been typed



```
                                    i                      6 22
>    10 rem
>    20 rem
>    30 rem
>    40 rem
>    20 rem this is 20
```

FIGURE 5-8 Line 20 with cursor over the "2"; <left cur key pressed twice



```
                                    i                      6 25
>    10 rem
>    20 rem
>    30 rem
>    40 rem
>    20 rem this is line 20
```

FIGURE 5-9 Line 20 after typing in "line"

increments of 5, for instance, enter the following command:

incr 5

Now press the <**space bar**> once and the next logical line number (45) is generated.

NOTE: The Dynamic Syntax Checker™ is monitoring all your keystroke entries. If you make a mistake during this exercise the system will reject the incorrect keystroke with a beep. After two consecutive errors, the set of keystrokes which are valid at that point will be presented to you. Only by typing one of those presented or by pressing <**Esc**> to abort the line can you continue (Refer back to 5-1 for more).

Now let's edit a line. Enter the following:

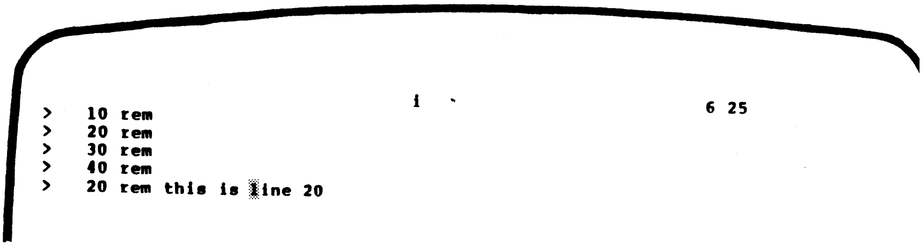
edit 20

Line 20 is displayed and the cursor is positioned at the start of the line.

1. Press <**End**> to move the cursor to the end of the line.

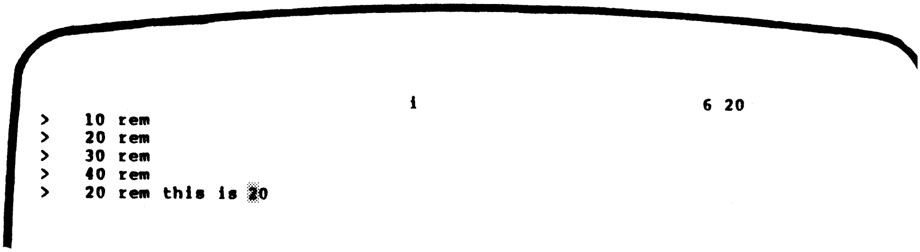
2. Press the <**space bar**> once and type in: **this is 20.**

3. Press the left cursor key (on the numeric keypad) until the cursor box is over the **2** after the word **is.** We want to insert the word **line** here plus a space.



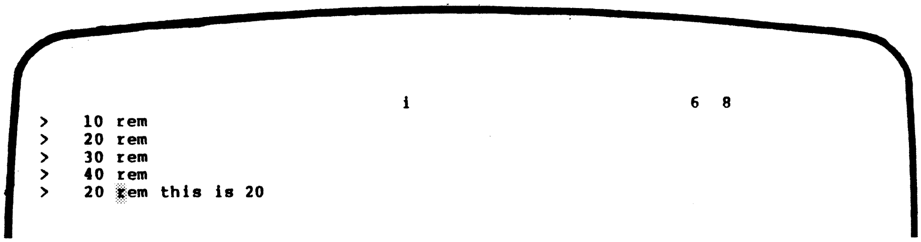
```
                                              i                         6 25
>    10 rem
>    20 rem
>    30 rem
>    40 rem
>    20 rem this is line 20
```

FIGURE 5-10 Line 20 before deleting the word "line"



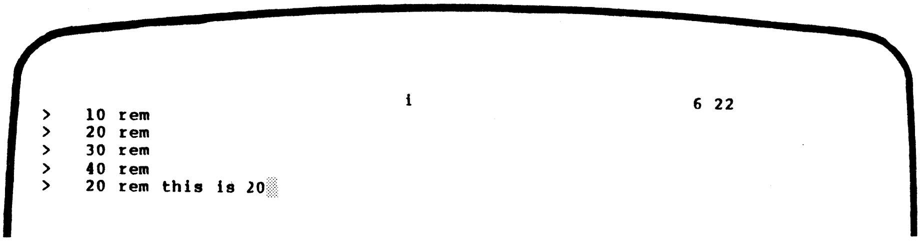
```
                                              i                         6 20
>    10 rem
>    20 rem
>    30 rem
>    40 rem
>    20 rem this is 20
```

FIGURE 5-11 Line 20 after deleting the word "line"



```
                                              i                         6  8
>    10 rem
>    20 rem
>    30 rem
>    40 rem
>    20 rem this is 20
```

FIGURE 5-12 Line 20 after <Home> was pressed



```
                                              i                         6 22
>    10 rem
>    20 rem
>    30 rem
>    40 rem
>    20 rem this is 20
```

FIGURE 5-13 Line 20 after <End> was pressed

4. Press **<Ins>**. This will put the editor in insert mode, indicated by a faster blinking of the cursor. Type in the word **line** and a space. The statement should now read: **20 rem this is line 20.** Any cursor control key or **<Ins>** will exit the insert mode.

5. To delete the word "line", press the left cursor key until the cursor box is over the first letter "l". Then press **<Del>** 5 times, until the appropriate characters are deleted.

Practice the use of the left and right cursor keys and the **<Ins>** and **<Del>** keys until you feel comfortable with how the editing operations work.

Use the **<Home>** and **<End>** keys to move the cursor to the beginning (first character after the line number) or end of the line being edited, respectively.

To delete from the cursor to the end of the current line, press and hold **<Ctrl>** while striking **<End>**.

To delete from the left of the cursor to the beginning of the line (excluding the line number itself), hold down the **<Ctrl>** key while you strike the **<Home>** key.

When the line on the screen is edited the way you want it, press **<Enter>** and it will be changed in memory. If you do not wish to save the change, then press **<Esc>**.

To delete a line or set of lines use the DELETE command (see Section 5-9).

The <**F6**> key may be used to generate the "**edit** " string on the command line so that you only have to press that key followed by a line number and the <**Enter**> key. This is just the same as typing "**edit** " on the keyboard.

The <**F8**> key will redisplay the last line edited so you may edit it further.

If you press <**F10**>, the next line after the one previously edited will be displayed for editing. Using this key it is possible to scroll slowly through the program, pausing to edit lines as you go. The <**F9**> key will move back one line with each press (in case you go past the line you intended to edit).

Using the <**F6**>, <**F9**>, and <**F10**> keys it is possible to move about in a program and perform editing functions. Each time an **edit n** command is used, the reference point for the <**F9**> and <**F10**> keys will be moved (previous and next will be relative to the line **n**).

The up and down cursor keys will work only within a multiple line BASIC code line. One line of code in Professional BASIC™ may be 311 characters long, or 4 screen lines. The technique to move to the next or previous line of BASIC code is with the use of the <**F9**> and <**F10**> keys.

A summary of function key action is presented in Section 5-13.

Important note for future reference: If you edit a line in a program which had been running and is now suspended (<**Break**> was pressed to halt execution), the program cannot resume execution unless you first enter the "**run**" command again. If you have not edited or changed your program in any way the program should execute immediately.

5-4 LIST Command

To look at a program in memory while on the Command Screen:

1. Type: **list** and press <**Enter**>. (Or, press <**F1**>). The program listing will scroll quickly on the screen.

2. Pressing <**space bar**> during the scroll will halt the listing and allow viewing a section of code.

3. Each additional press of the <**space bar**> scrolls one more line onto the screen.

4. Press <**Enter**> to go back to full speed listing and scrolling.

5. If you have seen what you want and wish to return to the Command Screen before the listing finishes, press <**Esc**>. The listing operation will be aborted at that point.

To list specific lines of code on the screen, you can enter the line numbers after the list command. To look at lines 100 to 200, for instance, enter the following:

list 100-200

If there is no line 100 or 200, the system will start from the next available line in the program after 100 and list up to the last available line before 200.

If you want to list from line 100 to the end of the program, enter:

```
list 100-
```

To list all lines up to line 200 the command is:

```
list -200
```

You may specify just a single line to list. You may also specify a line label instead of its line number. For instance, to list the line that is named **Start.of.Routine.A**, the command would be:

```
list  start.of.routine.a
```

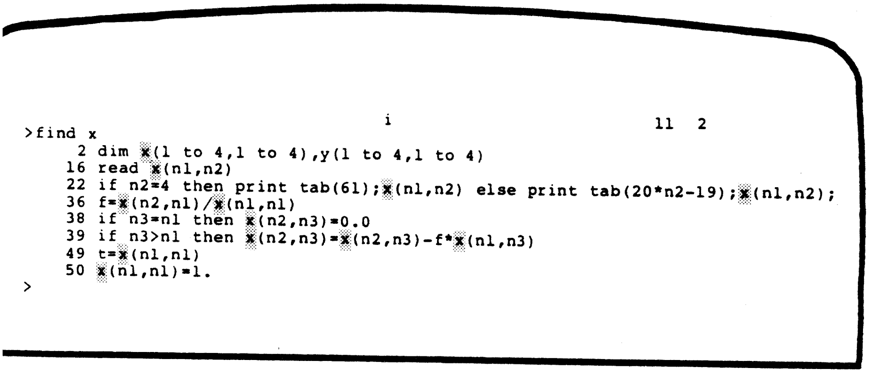
(Notice that upper/lower case is not important)

You may also use labels to specify a range, as in:

```
list  start.of.routine.a - start.of.routine.b
```

To send a listing to the printer instead of the screen, the command is **llist** instead of **list.** You may specify the lines to be printed with the **llist** command the same way you do with **list.**

In Section II, Chapter 8, you will learn about another way to print a listing of your programs (<J> key option).



```
                                          i                          11   2
>find x
    2 dim x(1 to 4,1 to 4),y(1 to 4,1 to 4)
   16 read x(n1,n2)
   22 if n2=4 then print tab(61);x(n1,n2) else print tab(20*n2-19);x(n1,n2);
   36 f=x(n2,n1)/x(n1,n1)
   38 if n3=n1 then x(n2,n3)=0.0
   39 if n3>n1 then x(n2,n3)=x(n2,n3)-f*x(n1,n3)
   49 t=x(n1,n1)
   50 x(n1,n1)=1.
>
```

ꝛE 5-14 **Screen showing use of the FIND command to locate the "x" variable in the DEMM.BAS program**

5-5 LOCATING VARIABLES, LABELS & LINE NUMBERS IN A PROGRAM -- THE "FIND" COMMAND

There is an easy-to-use facility in Professional BASIC™ that allows you to locate every reference to a variable, label, or line number. On the command screen, load the sample program which inverts a matrix by entering: **load demm.** After the program is loaded simply enter, for example:

```
>find x
```

The system will scroll onto the screen every line in the program containing the variable, label, or line number "x". The actual reference is highlighted for you in inverse video.

Rules to follow in using this command are:

1. If there are more than 24 lines to be displayed on the screen, then you can press <**space bar**> to halt the scrolling action.

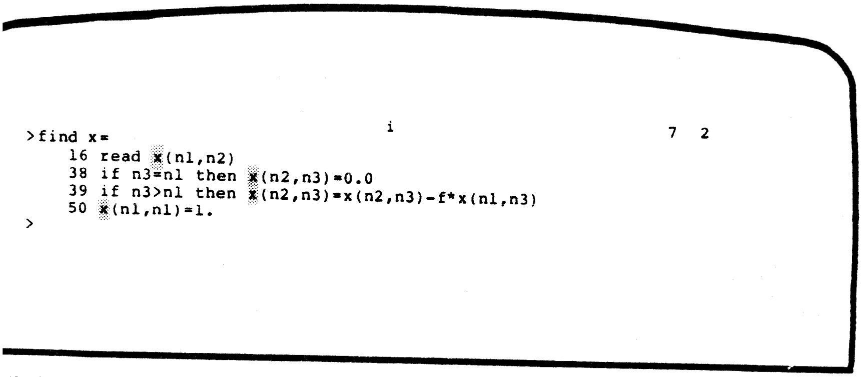
2. Subsequent presses of <**space bar**> will scroll the next line(s), one line with each press.

3. Pressing <**Enter**> will continue the full speed scrolling action.

4. If you press <**Esc**>, the listing will be aborted at that point and control will return to the command screen.

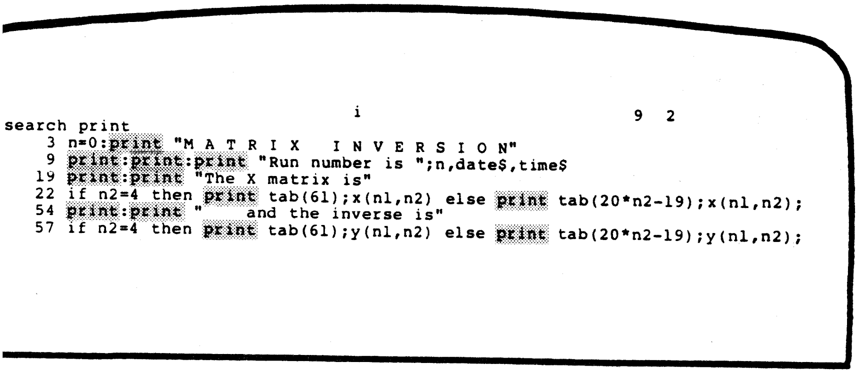
To locate each place in a program a variable is assigned a value. The command is, for example:

```
>find x=
```



```
                                   i                          7  2
>find x=
    16 read x(n1,n2)
    38 if n3=n1 then x(n2,n3)=0.0
    39 if n3>n1 then x(n2,n3)=x(n2,n3)-f*x(n1,n3)
    50 x(n1,n1)=1.
>
```

E 5-15 Screen showing use of the FIND= command to locate where variable "x" is being assigned a new value



```
                                   i                          9  2
search print
   3 n=0:print "M A T R I X   I N V E R S I O N"
   9 print:print:print "Run number is ";n,date$,time$
  19 print:print "The X matrix is"
  22 if n2=4 then print tab(61);x(n1,n2) else print tab(20*n2-19);x(n1,n2);
  54 print:print "    and the inverse is"
  57 if n2=4 then print tab(61);y(n1,n2) else print tab(20*n2-19);y(n1,n2);
```

E 5-16 Screen showing use of the SEARCH PRINT command in the DEMM.BAS program

This variation will cause only those lines in which **x** is assigned a value to be listed on the screen. (Remember, if you were using a long variable name or label, the special facility for completing the typing in of the string described in Section 5-8 is available for your use).

Another form of this command is to do the same search but display <u>all</u> program lines, not just the ones where there is a "hit". The format of this command is:

```
>findl x     or     findl x=
```

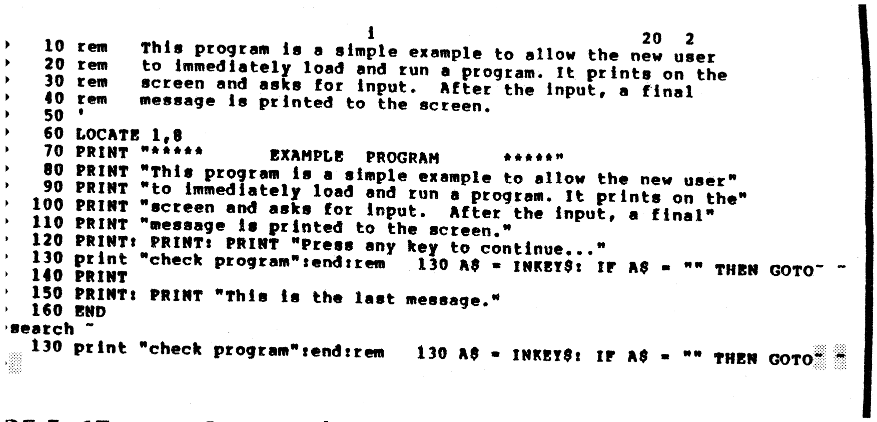
It is similar to the LIST command but with the target variables shown in inverse video.

5-6 LOCATING ANY TEXT STRING IN THE PROGRAM -- THE "SEARCH" COMMAND

The FIND command is limited to variables, labels, and line number references. The SEARCH command is used to locate <u>any</u> sequence of characters in the program. The SEARCH command format is:

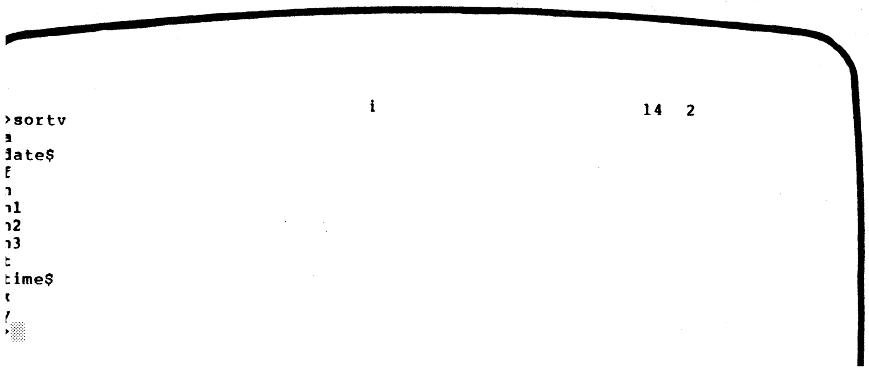
```
>search  character-sequence
```

The boundaries of the sequence are defined by starting at the second character after the "h" in "search" and stopping at the last character entered before pressing **<Enter>**. Note that blanks are valid search characters. In fact, you could search for a group of blanks. This command may be particularly useful in searching for every occurrence of the use of particular instructions in a program. For example, if you wanted to locate every instance of a PRINT statement in a program, enter:



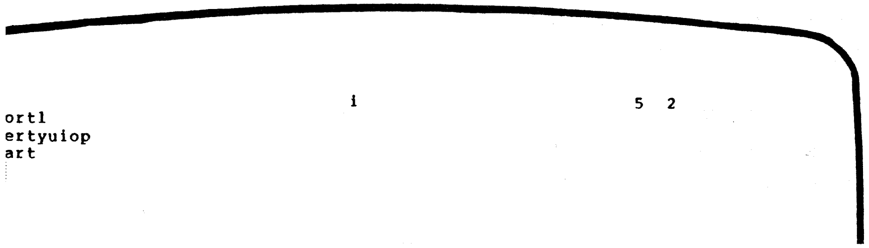
```
                                    i                             20  2
'   10 rem   This program is a simple example to allow the new user
'   20 rem   to immediately load and run a program. It prints on the
'   30 rem   screen and asks for input.  After the input, a final
'   40 rem   message is printed to the screen.
'   50 '
'   60 LOCATE 1,8
'   70 PRINT "*****        EXAMPLE  PROGRAM      *****"
'   80 PRINT "This program is a simple example to allow the new user"
'   90 PRINT "to immediately load and run a program. It prints on the"
'  100 PRINT "screen and asks for input.  After the input, a final"
'  110 PRINT "message is printed to the screen."
'  120 PRINT: PRINT: PRINT "Press any key to continue..."
'  130 print "check program":end:rem   130 A$ = INKEY$: IF A$ = "" THEN GOTO~ ~
'  140 PRINT
'  150 PRINT: PRINT "This is the last message."
'  160 END
'search ~
   130 print "check program":end:rem   130 A$ = INKEY$: IF A$ = "" THEN GOTO~ ~
'
```

RE 5-17 Screen showing use of the SEARCH ~ command in the ERROR.BAS program to find lines with syntax errors



```
                                    i                             14  2
>sortv
a
date$
E
n
n1
n2
n3
t
time$
x
y
'
```

RE 5-18 Screen showing use of SORTV command in the DEMM.BAS program



```
                                    i                             5  2
ortl
ertyuiop
art
```

RE 5-19 Screen showing use of SORTL command in the DEMM.BAS program

```
        >search print
```

Another important use of the SEARCH command is to find error lines when a program is read in from a file not created by Professional BASIC™ (see Chapter 6). Load the example program ERROR.BAS. Enter the search command:

```
        >search ~
```

Use the "tilde" character because it occurs in all lines with errors. Although the example program has only one error and is very short, you can see that in a large program with several errors it is very convenient to be certain that no error lines are skipped during editing.

The SEARCHL command is similar to SEARCH except that it produces a listing like one produced by a LIST command but with the target characters shown in inverse video.

5-7 SORTV and SORTL COMMANDS

By entering **SORTV** on the command screen, an alphabetical listing of all the variables (including arrays) in the program will be scrolled onto the screen. Then, to locate the specific occurrences of any variable in the program, use the FIND or FINDL command.

The **SORTL** command is used to help identify all the line labels used in a program. By entering the command on the command screen, a list of all the line labels will be scrolled onto the screen. Then use the FIND or FINDL command to locate the line with the label and the other lines in which there is a GOTO or GOSUB and the same label.

5-8 ENTRY OF LONG VARIABLE NAMES AND LABELS

With the ability to create long variable names and line labels, programs can be created which are more easily understood than if short, cryptic names are used. The disadvantages of using long names are that they are frustrating to type again and again and that spelling and/or typographical errors are easily made.

There is an elegant solution to this dilemma in Professional BASIC™. All you need to do is:

1. Type in enough of the variable or label to uniquely identify it.

2. Then enter an "@" character. The rest of the name or label will be retrieved from memory and displayed on the screen.

For example, if you have a variable name such as,

first.parameter,

and you type in **fir**, or enough to uniquely identify that variable name, then press @, (hold the shift key and press the "2" key at the top of the keyboard) and then the rest of the variable name will appear. It is as if you had completed the typing of the name yourself from the keyboard. If there are two or more variables starting with "**fir**" (e.g., another variable is **firm.costs**) then the "@" will be rejected with a beep and you must type in more of the name or label. **firs** or **firm** in this latter case would distinguish each label from the other.

This system of automatically completing long strings used as variable names or as labels, is a tremendous help in coding in

Professional BASIC™. There are some disadvantages to using long names, such as using more program space in memory and on disk, and even some potential difficulty in reading programs. Also, the split screen displays (that you will learn about in Section II) truncate long instructions. Therefore, the ability to recognize an instruction which begins with a long variable name or label may be difficult. It should be realized that there is a practical limit to how long your variable and label names should be in order for them not to become a distraction. With some practice, you can develop an optimal strategy to fit your style and needs.

5-9 DELETE Command

To delete a line or set of lines in a program in memory use the **DELETE** command. To use the command simply type in: **delete** or **del** followed by the line or set of lines. Unlike the list command, you must specify the exact lines you wish to delete. You cannot leave out the beginning or ending line number.

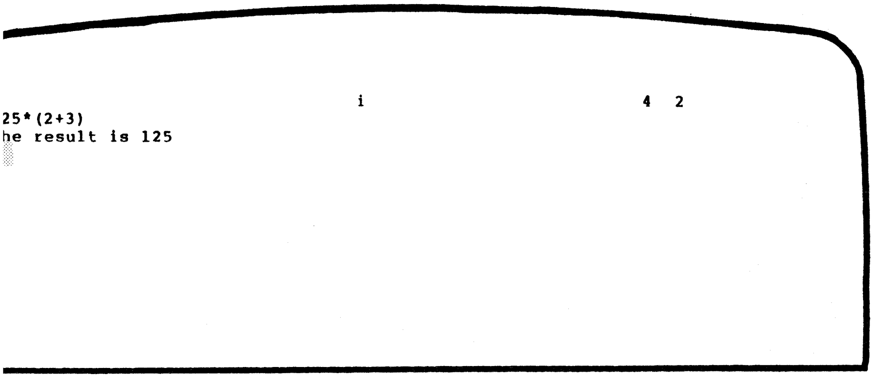
To delete line 100 in a program enter:

>delete 100 or **>del 100**

To delete lines 50-135 in a program enter:

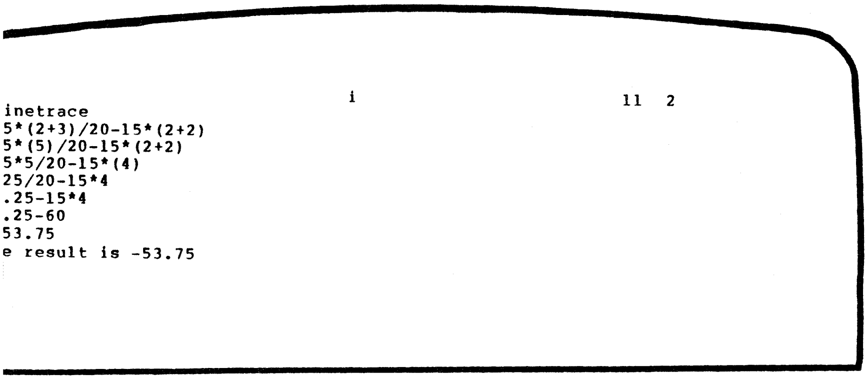
>delete 50-135 or **>del 50-135**

Lines 50 and 135 must be valid lines which are in the program currently in memory at the time the command is given. After deleting the lines you may wish to renumber your program line numbers. Refer to your IBM BASIC reference manual to see how the **RENUM** command operates.



```
                                    i                          4   2
25*(2+3)
he result is 125
```

RE 5-20 Command Screen showing arithmetic in Immediate Mode



```
                                    i                          11   2
inetrace
5*(2+3)/20-15*(2+2)
5*(5)/20-15*(2+2)
5*5/20-15*(4)
25/20-15*4
.25-15*4
.25-60
53.75
e result is -53.75
```

:E 5-21 Command Screen showing arithmetic calculations in Finetrace Mode

5-10 ONSCREEN ARITHMETIC

With the Command Screen active (cursor blinking after the > prompt showing) there are a number of actions which may be taken. Most actions that can be taken in this condition have to do with loading, entering, editing, and running programs. However, in addition to these actions it is also possible to perform arithmetic operations.

For instance, to add two numbers together simply type in the expression and press <**Enter**>. Figure 5-20 is an example of how the screen will look. Notice that if you make a mistake while entering the expression, the system will beep and reject the entry (character will appear but cursor does not move). If you make a mistake twice at the same point, the system will present you with a list of acceptable keys that can be pressed at this point. Only an acceptable character will allow you to continue. Of course, pressing <**Esc**> will erase the line and allow you to restart.

5-11 THE FINETRACE MODE

In this mode it is possible to see in minute detail exactly how the computer evaluates an expression. At the > prompt, type: **finetrace** and press <**Enter**>. Now try this example:

```
25*(2+3)/20-15*(2+2)
```

Press <**Enter**> when you are through typing this in. Next use the <**left arrow**> key to advance to the next operation being performed in the calculation of this expression. It will become quite clear just how the computer resolves this expression to a single number.

A more complex example to try would be to use variables instead of numbers. First, use a series of statements to assign values to each variable. Then, type in the expression using the variable names and step through the finetrace. In addition to seeing the order of calculation as before, you will see the order of substituting numbers for the variables too.

To get out of the "finetrace" mode enter: **nofinetrace**

5-12 PRINTING DATE AND TIME

Professional BASIC™ has a convenience feature to print the current system time and date on the connected printer. This is helpful if you want to put the time and date at the head of a program listing or other printout.

Press <**Shift Alt PrtSc**> at any time to print the time and date. It will look like the following:

```
1984 May 17 15:54:41
```

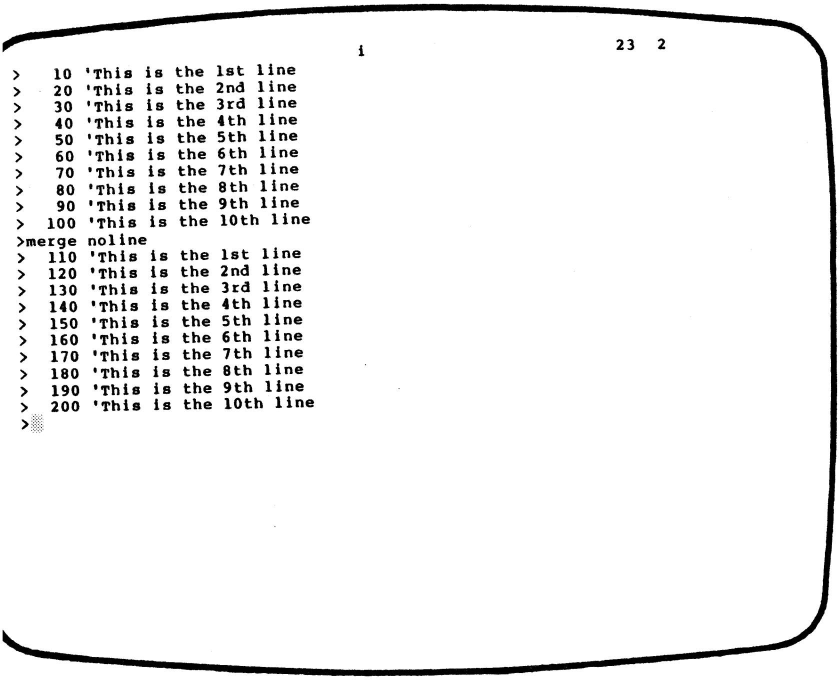
5-13 FUNCTION KEYS

The function keys for the editor have fixed values. That is, you cannot change them. The editor function key values are described below:

F1	Types "**list** " on the screen
F2	Performs the **RUN** command
F3	Types "**load** " on the screen and waits for a filename or <**Enter**>

F4	Types "**save** " and waits for a filename or <**Enter**>
F5	Redisplays the last command entered on the Command screen.
F6	Types "**edit** " and waits for you to enter a line number or label.
F7	Types "**system**" and waits for you to press <**Enter**>. This will exit from Professional BASIC™ and return you to the DOS operating system.
F8	Performs the **Edit** command, displaying the most recently edited line.
F9	Performs the **Edit** command, displaying for edit the line just prior to the last line edited or displayed for edit (next lower line number). By pressing <**F9**> repeatedly you can scroll backwards through the program to get to a line you wish to edit.
F10	Performs the **Edit** command, displaying for edit the next line after the last line edited or displayed for edit (next higher line number). By pressing <**F10**> repeatedly you can scroll forwards through the program to get to a line you wish to edit.

IMPORTANT: If you have the Key-Lock "on" (turned on by pressing the "5" key on the numeric keypad) make sure the Ctrl, Shift, and Alt keys are "off" before trying to use the function keys.



```
                                              i                          23   2
>   10 'This is the 1st line
>   20 'This is the 2nd line
>   30 'This is the 3rd line
>   40 'This is the 4th line
>   50 'This is the 5th line
>   60 'This is the 6th line
>   70 'This is the 7th line
>   80 'This is the 8th line
>   90 'This is the 9th line
>  100 'This is the 10th line
>merge noline
>  110 'This is the 1st line
>  120 'This is the 2nd line
>  130 'This is the 3rd line
>  140 'This is the 4th line
>  150 'This is the 5th line
>  160 'This is the 6th line
>  170 'This is the 7th line
>  180 'This is the 8th line
>  190 'This is the 9th line
>  200 'This is the 10th line
>▒
```

ɜURE 5-22 **Screen showing NOLINE.BAS loaded with the LOAD command and then loaded again with the MERGE command**

5-14 MERGE COMMAND

An unusual and powerful feature of Professional BASIC is the capability of creating programs with only a few or no line numbers. The rule that must be followed with code you wish to merge into a program is:

> Each line of code in the file being merged into your program must start with a line number <u>or</u> at least one space.

In most cases you will probably want to merge program lines where none of the lines have a line number. That is, every line would start with at least one space.

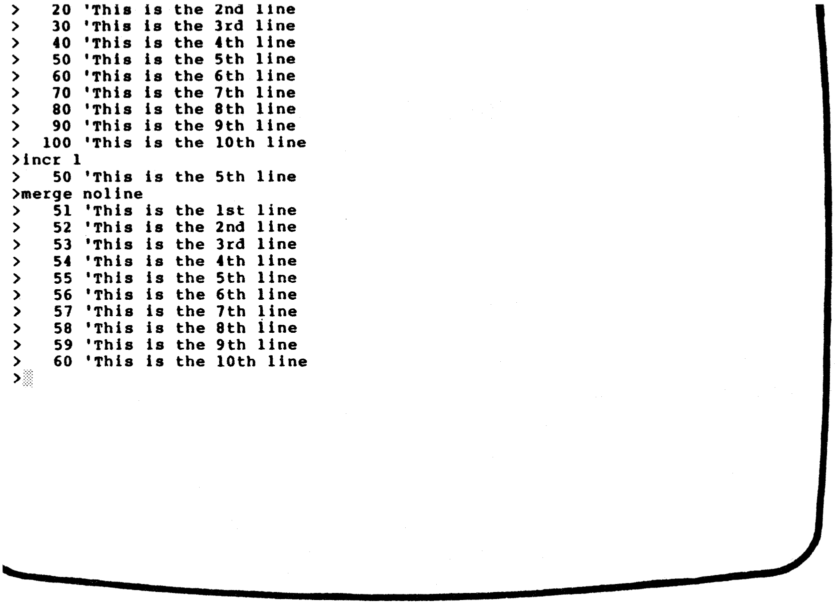
Professional BASIC™ will assign a line number to each line being merged in. The line number assigned to the first line of code in the merge file will be one increment higher than the line which would be displayed if you were to press the **<F8>** function key. There is a "pointer" in the system which is associated with the Professional BASIC™ editor. It is used to scroll through a program to edit lines of code. The "edit current", "edit previous", and "edit next" functions keys (**<F8>**, **<F9>**, and **<F10>** respectively) use this pointer. The MERGE command also uses this pointer to decide where unnumbered lines of code should be placed.

Load the program NOLINE.BAS. Then enter:

```
>merge noline
```

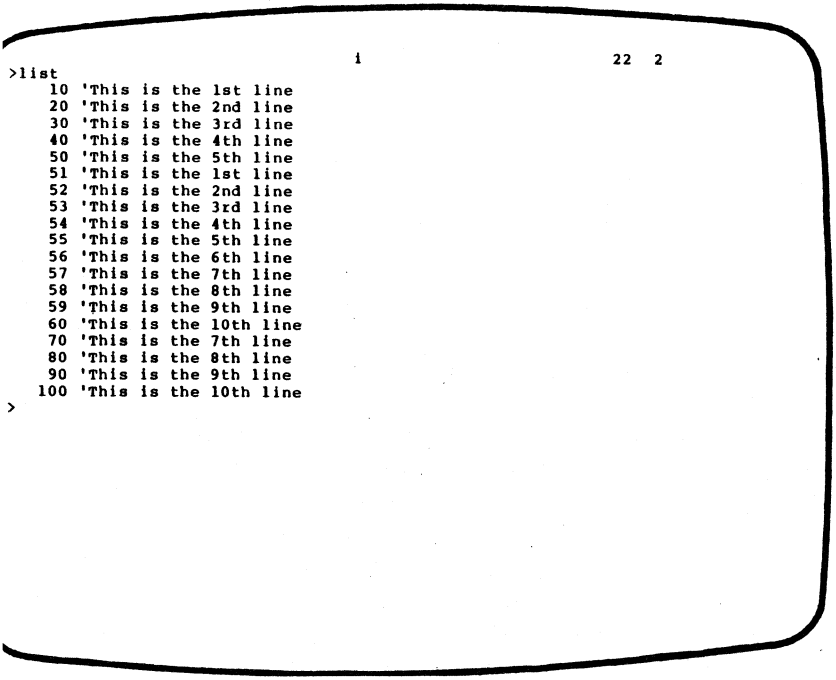
The code is merged at the end of the file (see Figure 5-22). Now enter the edit command:

```
>edit 50
```



```
>    20 'This is the 2nd line
>    30 'This is the 3rd line
>    40 'This is the 4th line
>    50 'This is the 5th line
>    60 'This is the 6th line
>    70 'This is the 7th line
>    80 'This is the 8th line
>    90 'This is the 9th line
>   100 'This is the 10th line
>incr 1
>    50 'This is the 5th line
>merge noline
>    51 'This is the 1st line
>    52 'This is the 2nd line
>    53 'This is the 3rd line
>    54 'This is the 4th line
>    55 'This is the 5th line
>    56 'This is the 6th line
>    57 'This is the 7th line
>    58 'This is the 8th line
>    59 'This is the 9th line
>    60 'This is the 10th line
>
```

URE 5-23a **NOLINE.BAS inserted with the MERGE command after line 50 with the increment set to 1**



```
                                       i                            22  2
>list
     10 'This is the 1st line
     20 'This is the 2nd line
     30 'This is the 3rd line
     40 'This is the 4th line
     50 'This is the 5th line
     51 'This is the 1st line
     52 'This is the 2nd line
     53 'This is the 3rd line
     54 'This is the 4th line
     55 'This is the 5th line
     56 'This is the 6th line
     57 'This is the 7th line
     58 'This is the 8th line
     59 'This is the 9th line
     60 'This is the 10th line
     70 'This is the 7th line
     80 'This is the 8th line
     90 'This is the 9th line
    100 'This is the 10th line
>
```

URE 5-23b **Listing of the resulting program (after a**

Press <**return**> without editing this line. Now enter:

```
>merge noline
```

Notice that the file overwrites part of the existing program beginning at line 60.

Enter the **NEW** command to start over. Enter **merge noline** to load in the program (works just like the LOAD command in this case, where there are currently no program lines in memory). Now enter the auto line number generation command to alter the auto increment as follows:

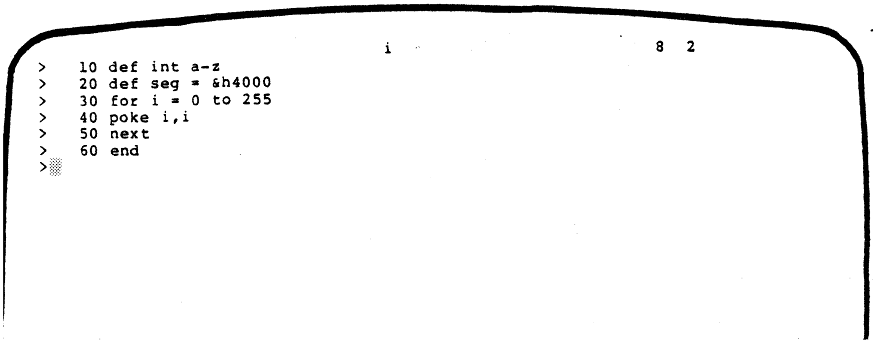
```
>incr 1
```

Then enter the following two commands

```
>edit 50
>merge noline
```

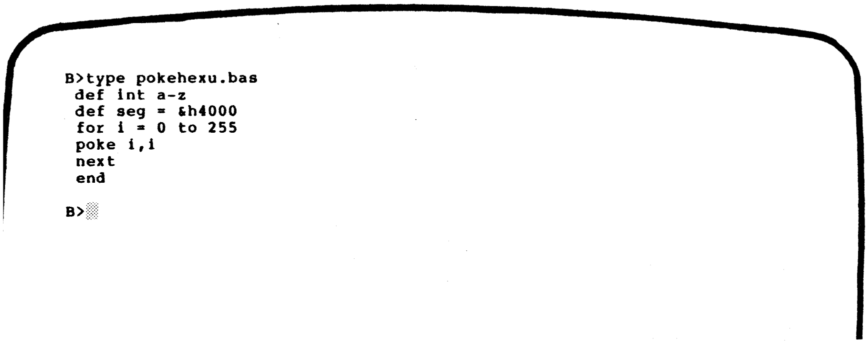
Figure 5-23 shows the result. The lines are merged in with a starting line number of 51 and each subsequent line number is incremented by one. (Note that line 60 has been replaced.)

Thus, with this facility you can see how it is possible to set up libraries of modules which can be merged together into a program. Since Professional BASIC™ has the option to use line labels you do not have to worry about the values of the line numbers generated. If each module merged in is a subroutine, the first line can have a label. Then the program can access that routine through a GOSUB no matter where it may physically be placed in the program. If there need to be GOTO's or GOSUB's internally in that module, then those routines can use labels too. A warning: When merging code be careful that there is room in the target program for the lines of code to fit in. The code being merged in may overwrite and replace lines in the target program, as demonstrated in the example above.



```
                                   i                        8  2
>   10 def int a-z
>   20 def seg = &h4000
>   30 for i = 0 to 255
>   40 poke i,i
>   50 next
>   60 end
>
```

IGURE 5-24 Command screen with POKEHEX.BAS loaded.



```
B>type pokehexu.bas
 def int a-z
 def seg = &h4000
 for i = 0 to 255
 poke i,i
 next
 end

B>
```

IGURE 5-25 POKEHEXU.BAS as it is stored to the disk with the SAVEU command

5-15 SAVING PROGRAMS WITHOUT LINE NUMBERS

Professional BASIC™ allows you the option of saving your program without line numbers. The **SAVEU** command operates in exactly the same fashion as the SAVE command except that it strips off all leading line numbers and replaces them with a single blank. The single leading blank allows Professional BASIC™ to assign line numbers to the code the next time you load or merge it into the system.

This feature can be very useful when creating libraries of commonly used subroutines and small programs. As an example load the program POKEHEX from your disk. Your screen should look like Figure 5-24. Now enter:

```
>saveu pokehexu
```

to save the program without line numbers with a filename of pokehexu. Now press <F7> and <**ENTER**> to return to DOS. To look at the file use the DOS type command as follows:

```
A>type pokehexu.bas
```

The screen should look like Figure 5-25. This is an ASCII text file that can be edited with an outside text editor. Note that each line has a leading blank to allow Professional BASIC™ to assign line numbers the next time you load the program. Remember that if you edit the program with a text editor each line of code must be preceeded with at least one space or a line number.

Now that you have had a grounding in the functioning of the Professional BASIC™ environment and editor, the next section will explain how to use your IBM PC BASIC programs you may already have with the Professional BASIC™ system.

6. CONVERTING IBM PC BASIC PROGRAMS

In most respects the commands and statements supported by the Professional BASIC™ system are identical to those supported by IBM PC BASIC. In some cases you may simply be able to read your ASCII source file into Professional BASIC™ using the LOAD command and it will run perfectly. More often than not, however, there will be minor editing changes that need to be made before Professional BASIC™ will run your IBM PC BASIC programs. You should have both your IBM PC BASIC manual and your Professional BASIC™ manual at hand for syntactical reference when running programs developed for one system and running on the other. Please refer to Chapter 2 for general system differences, and Chapter 3 for specific differences in the way commands and statements operate.
See Appendix A - Professional BASIC™ Differences.

6-1 COMMANDS AND STATEMENTS WITH DIFFERENCES

Below you will find a brief list of the commands that differ between IBM PC BASIC and Professional BASIC™. For more information please refer to section 3-3.

Commands and Statements that differ are:

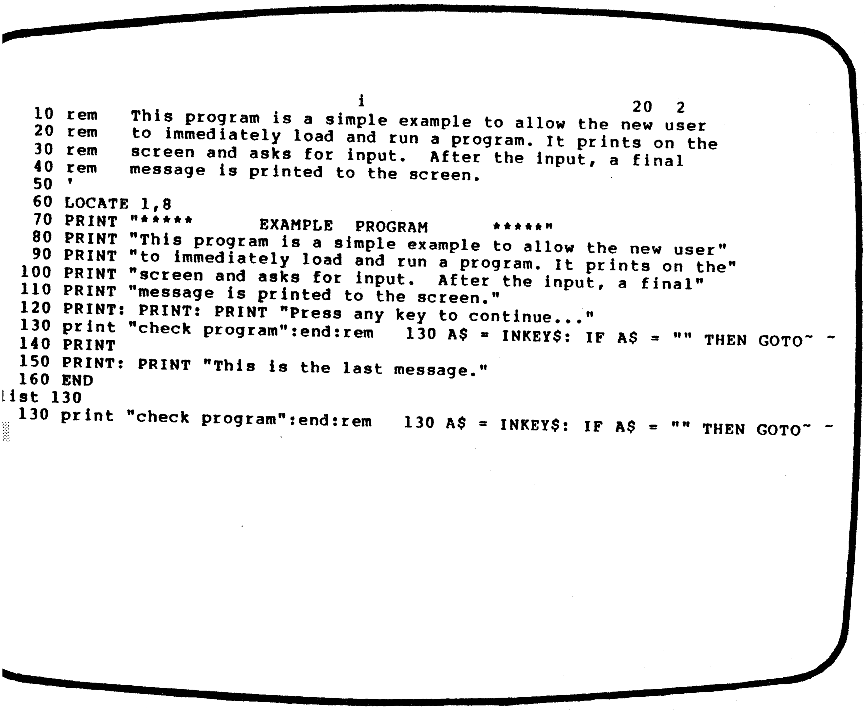
DIM	<u>All</u> arrays must be dimensioned under Professional BASIC™.

FOR/NEXT	All FOR/NEXT statements must be physically and logically paired. A single NEXT without parameters cannot work for multiple FOR's.
LOCATE	The third, fourth and fifth parameters are accepted but have no effect on the cursor.
FIELD	You may not use subscripted variables in a FIELD statement.
FILES (in a program)	Quotes are required if you are specifying a filespec. Specifying a drive letter will give you a list of files on the specified drive. Quotes are not required in immediate mode.
WHILE/WEND	WHILE/WEND statements must be physically and logically paired.

Professional BASIC™ is being implemented in stages, and not all the IBM PC BASIC commands are currently available. For a complete list of commands, both implemented and not yet available, please refer to Appendix A.

When using the editor, keep in mind that the Professional BASIC™ editor is not a full screen editor and that there are many new and powerful commands available to you. Refer to Chapter 5 for more information. The editing commands that differ are:

EDIT	There is no "EDIT ." option, **F8** displays last line edited.
LOAD	The quote marks are not needed before or after the filename, if any. An extension of .BAS is assumed. There is no **,R** option.
SAVE	Responds in a fashion similar to LOAD. SAVE is not available as a program statement.



```
                                  i                        20  2
  10 rem   This program is a simple example to allow the new user
  20 rem   to immediately load and run a program. It prints on the
  30 rem   screen and asks for input.  After the input, a final
  40 rem   message is printed to the screen.
  50 '
  60 LOCATE 1,8
  70 PRINT "*****       EXAMPLE  PROGRAM      *****"
  80 PRINT "This program is a simple example to allow the new user"
  90 PRINT "to immediately load and run a program. It prints on the"
 100 PRINT "screen and asks for input.  After the input, a final"
 110 PRINT "message is printed to the screen."
 120 PRINT: PRINT: PRINT "Press any key to continue..."
 130 print "check program":end:rem   130 A$ = INKEY$: IF A$ = "" THEN GOTO~ ~
 140 PRINT
 150 PRINT: PRINT "This is the last message."
 160 END
list 130
 130 print "check program":end:rem   130 A$ = INKEY$: IF A$ = "" THEN GOTO~ ~
```

RE 6-1 Screen showing line with error message in the ERROR.BAS program

6-2 LOADING PROGRAMS FROM OTHER BASIC SYSTEMS

To load a program created on another system, first make sure it is in ASCII format. If it is not, use the other BASIC system to resave the program in ASCII format. For example, to save the program named **test** in ASCII format with IBM BASIC, use the following command: **save "test.bas",a.** The equivalent command in Professional BASIC™ is:

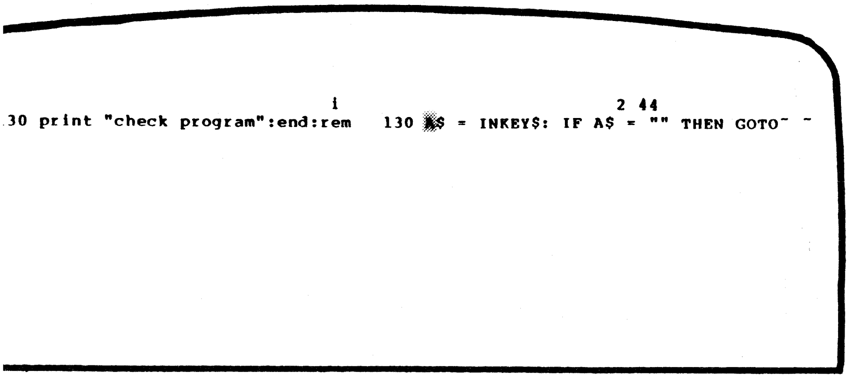
```
>save test
```

Because of the differences explained in Chapters 2 and 3 or if there is a syntax error in the program, Professional BASIC™ will flag "errors" as the program is loaded. The Dynamic Syntax Checker™ is active during the loading of a program. When an error is found, a beep will sound and the line in error will be changed to include an error message. The loading does not stop. If there are errors, you will need to edit the program before running it.

To practice with this, load in the program: ERROR.BAS. It is the same as the program EXAMPLE.BAS except that there is an intentional error. The beep indicates that an error has been encountered; the program will continue to load. The line with the error in it will be changed to include code that would stop the program if you were to run it without fixing the error. The line of code with the error in it has been altered to look like:

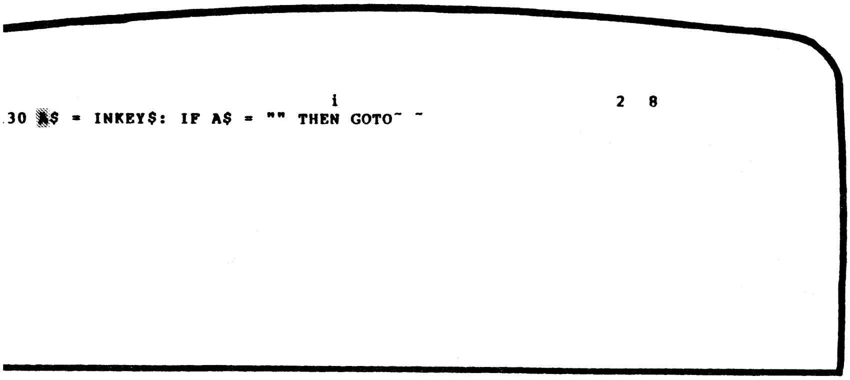
```
130 print "check program":end:rem 130 A$ = INKEY$: IF
A$ = "" THEN GOTO~~
```

Note that the original code is retained as a comment. The error is that there is no line number or label after "GOTO". A "~" (tilde) is included in the line where the error is detected. Place the cursor over the "A" after the "130" in the middle of



```
                                i                                 2 44
30 print "check program":end:rem   130 A$ = INKEY$: IF A$ = "" THEN GOTO~ ~
```

GURE 6-2 Line 130 before <Ctrl Home> is pressed



```
                                i                                 2   8
30 A$ = INKEY$: IF A$ = "" THEN GOTO~ ~
```

IGURE 6-3 line 130 after <Ctrl Home> is pressed the line. Press <**Ctrl Home**> to delete the information which was inserted by the system during loading. Type in "130" after the "GOTO". Now the program can be run.

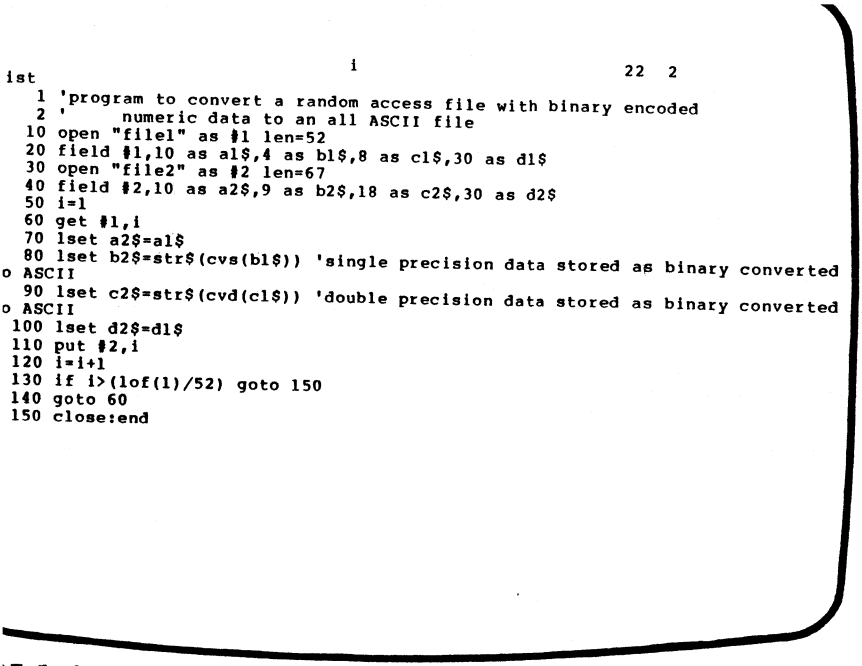
For larger programs, an easy way to find all the error lines is to use the SEARCH command described in Section 5-6. Since the tilde character is unique and always present in error lines, the command "SEARCH ~" will list all of the lines that need to be changed.

6-3 RANDOM ACCESS FILE DATA DIFFERENCES

All real numbers (sometimes called floating point numbers) in Professional BASIC™ use the IEEE format instead of the Microsoft format used by IBM PC BASIC. Real numbers encoded in binary format use the same amount of space on the disk under both BASICs. Single precision real numbers use 4 bytes and double precision real numbers use 8 bytes.

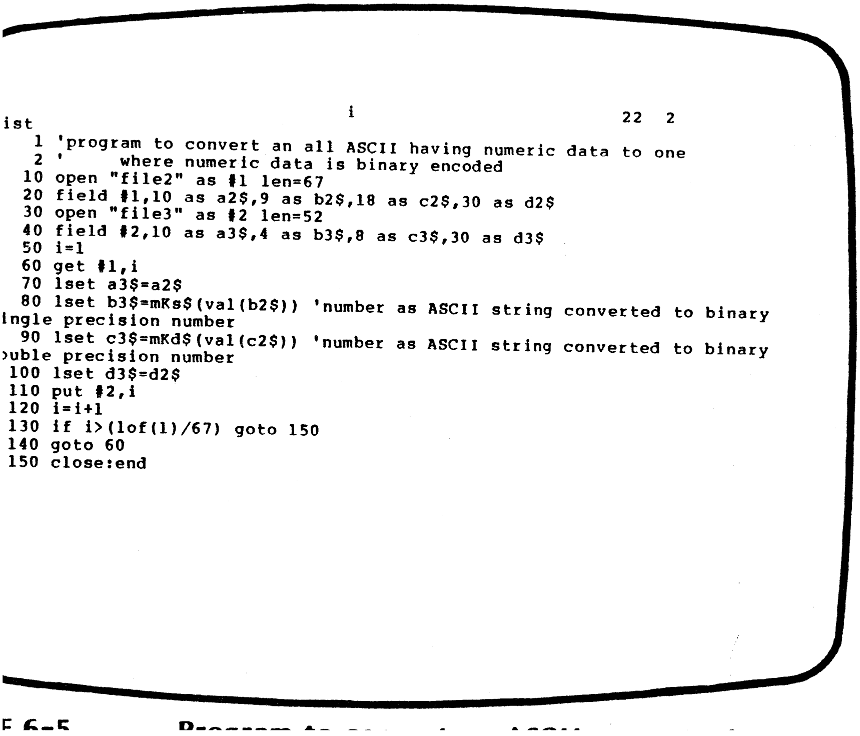
This means that the binary encoding and decoding of single and double precision numbers using the MKS$ and MKD$ commands and the corresponding CVS and CVD commands will be different in Professional BASIC™. (In all cases Professional BASIC™ gives more accurate results than IBM PC BASIC.)

A possible method for converting files created by IBM PC BASIC that have binary encoded numeric data fields is to use IBM PC BASIC to read the file one record at a time and write each record out into another random file, storing the numbers in ASCII (text) format. Then in Professional BASIC, the strictly ASCII random file can be read and then the data written back out into a third random file which reencodes the data with the MKS$ and MKD$ commands.



```
                                   i                           22  2
ist
   1 'program to convert a random access file with binary encoded
   2 '     numeric data to an all ASCII file
  10 open "file1" as #1 len=52
  20 field #1,10 as a1$,4 as b1$,8 as c1$,30 as d1$
  30 open "file2" as #2 len=67
  40 field #2,10 as a2$,9 as b2$,18 as c2$,30 as d2$
  50 i=1
  60 get #1,i
  70 lset a2$=a1$
  80 lset b2$=str$(cvs(b1$)) 'single precision data stored as binary converted
o ASCII
  90 lset c2$=str$(cvd(c1$)) 'double precision data stored as binary converted
o ASCII
 100 lset d2$=d1$
 110 put #2,i
 120 i=i+1
 130 if i>(lof(1)/52) goto 150
 140 goto 60
 150 close:end
```

E 6-4 **Program to convert a binary encoded numeric data file to ASCII (Run in IBM PC BASIC)**



```
                                   i                           22  2
ist
   1 'program to convert an all ASCII having numeric data to one
   2 '     where numeric data is binary encoded
  10 open "file2" as #1 len=67
  20 field #1,10 as a2$,9 as b2$,18 as c2$,30 as d2$
  30 open "file3" as #2 len=52
  40 field #2,10 as a3$,4 as b3$,8 as c3$,30 as d3$
  50 i=1
  60 get #1,i
  70 lset a3$=a2$
  80 lset b3$=mKs$(val(b2$)) 'number as ASCII string converted to binary
ingle precision number
  90 lset c3$=mKd$(val(c2$)) 'number as ASCII string converted to binary
ouble precision number
 100 lset d3$=d2$
 110 put #2,i
 120 i=i+1
 130 if i>(lof(1)/67) goto 150
 140 goto 60
 150 close:end
```

E 6-5 [illegible]

The two programs shown in Figures 6-4 and 6-5 do these conversions. Of the four fields in the file, the second and third are the ones which are converted. Notice that the single precision numbers are represented by an ASCII string that is 9 bytes long. The double precision numbers require a string field 18 bytes long. Program #1 should be run under IBM PC BASIC. Program #2 is run under Professional BASIC™. File1 and File3 are identical except that the two middle fields of the four have numeric data encoded in binary using two different encoding conventions.

Integers

Integers in Professional BASIC™ use four bytes (two 16 bit words) instead of two bytes (one word) as are used in IBM PC BASIC.

However, you may define integer fields which are binary encoded as either 2 or 4 bytes. Thus files established under IBM PC BASIC in which integer values have been stored in 2 byte fields can be read without conversion by Professional BASIC™. The CVI operator will correctly convert either a 2-byte or a 4-byte encoded string to an integer. The MKI$ instruction will encode an integer into a 4-byte string. However, if that string is LSET into a 2-byte field, only the 2 left bytes (which would be all that is required to store a number up to 32,767) would be placed in the file buffer field.

Thus, as long as you used integers of no more than 32,767, it does not matter whether or not you set aside 2 or 4 bytes in the FIELD statement for a binary encoded integer. If you use 4 bytes, however, you may store integers that are as large as 2,147,483,647.

As you begin to convert your programs, you will quickly spot the areas that are likely to cause you frequent trouble. Once you have accomplished one or two complete conversions the process should become simple and relatively easy.

SECTION II

THE TRACE WINDOWS

INTRODUCTION TO SECTION II

In this section we explain the trace windows and special features of the Professional BASIC™ environment. In the introduction you had a chance to sit back and watch a demonstration of some of the features in Professional BASIC™ (to repeat this demo with Professional BASIC™ already loaded, enter **##** on the Command Screen). This section is designed to take you step by step through the process of using the full tracing capabilities of Professional BASIC™ to see your programs run.

To aid you in learning about the many additions to the BASIC environment, there are several programs on the Professional BASIC™ disk. To load these programs enter **load dem** followed by one of the following characters. The example programs are listed below:

a	=	GOSUB with line numbers
b	=	GOSUB with line labels
c	=	GOSUB with multiple statement lines
d	=	DATA/READ statements
e	=	sort a list of names using SWAP statement
f	=	FOR/NEXT with arrays
g	=	LSET/RSET statements and file buffer window
h	=	LSET/RSET statements (no "field" statement)
i	=	indentation example - L window
j	=	SOUND/PEEK statements
k	=	prime numbers calculation

l	=	PRINT/LOCATE/COLOR statement
m	=	matrix inversion
n	=	WRITE statement - sequential file I/O
p	=	INPUT statement and syntax checking
r	=	RND random statement
s	=	"sieve benchmark"
w	=	string variable screen
x	=	WHILE/WEND
z	=	divide by zero - program

To make using the tutorials even easier three of these demos can be loaded from within Professional BASIC™ by simply pressing <#> followed by the letter of the program. Those three demos are:

m	=	matrix inversion
l	=	PRINT/LOCATE/COLOR statement
z	=	divide by zero - program

For example, to load the matrix inversion demonstration, at the > prompt, type either:

```
>#m          or >load demm
```

Then enter **srun** to semicompile the program and be placed on the List Trace Window. Press <**Enter**> to start the trace. To pause the trace and single step press the <**space bar**>. Pressing the <**Enter**> key resumes full speed tracing. The <**Break**> key (upper right corner key on keyboard) will suspend the program and return you to the Command Screen. Remember, the screen on which commands are entered, the screen on which program output is sent (via the PRINT statement) and the trace screens are all maintained separately. Hence if you were to run a program that has no screen output you would be looking at a blank screen, but the program would still be running.

The following chapters will introduce you to how the system works in more detail. There is a lot going on but if you follow the examples you should soon find that after some practice it is a simple system to use.

In this section you should become familiar with the features of program execution and the concept of visible program execution presented in Professional BASIC™. Because you will be working with a number of different screens and keys, keep the Professional BASIC™ Quick Reference Card handy.

7. UNDERSTANDING THE WINDOWS

In this chapter you will begin to explore the true magic of Professional BASIC™. This chapter provides some of the groundwork necessary to understand what is happening while you trace a program.

Load Professional BASIC™ by inserting the program disk in drive A and entering: **PB** (**PB8** if your computer doesn't have an 8087 numeric coprocessor) at the DOS **A>** prompt. When the initial screen appears, press any key to clear it and go to the Command Screen. If the self-running demonstration is operating, press <**Break**> twice to get to the Command Screen.

The standard prompt in Professional BASIC™ is a ">" symbol. Whenever the Command Screen is displayed with this symbol along the left and a blinking cursor (box), you may enter a command.

7-1 KEYBOARD CONTROL

There are multiple screens being independently maintained in Professional BASIC™ - the program output screen, the Command Screen, and all the tracing windows. It is important to understand how the functioning of the keyboard changes when going from one of the three basic types to another.

Pressing the <**Break**> key (the key at the far upper right of the keyboard with "Scroll Lock" written on top of the key and "Break" on the front) will return you to the Command Screen from <u>any</u> screen, while suspending the execution of the program. <**Ctrl Break**> will do the exact same thing.

While the Print Screen (normal program output) is Being Viewed

In Professional BASIC™ the keyboard has a dual role while a program is running and the normal program output screen (Print Screen) is displayed:

- First, in normal mode the keyboard interacts with the program in the typical way. That is, if there are statements within the program that expect keystrokes, then the keyboard is used to input those keystrokes. Operations like selecting menu options, responding to queries from the program, and providing input data to the program are done with the keyboard.

- Second, with the <**Alt**> key held down, the keyboard is used to switch from the normal output screen to one of the Professional BASIC™ windows <u>or</u> to stop and restart program execution while viewing the output. To change to a trace window, hold down the <**Alt**> key while pressing the appropriate key for the window. The Alt key can be used with the <**Esc**> key, <**Enter**> key, <**space bar**>, and all the normal alphabetical keyboard (white colored) keys.

  <**Alt space bar**> will stop program execution, if it is currently running, and put the system in "single

step" execution mode. If the program is currently stopped, each press of **<Alt space bar>** will cause the next instruction to be executed (single step execution of the program). However, the normal program output will continue to be displayed on the screen.

<Alt Enter> will return the program to full speed mode if it is in single step mode. The normal program output will continue to be displayed on the screen.

While a Trace Window is Being Viewed

When a program is running (either at single step or full speed execution mode) and one of the tracing windows is being displayed, the keyboard is controlling the trace window display. Make sure the Caps Lock and shift keys are "off". The Status Line should not show a "c, s, or a" in position G (Section 7-2). The keyboard is "talking" to the trace environment. By pressing the letter corresponding to a window, that window will be displayed. The cursor and other keys used for special control actions on a tracing window will also be in effect. See the quick reference card for a summary of all the key commands which affect the tracing windows and environment.

If you need to input information into the program you are tracing you may do so from any of the trace windows (except the **P** screen) by holding down the **<Alt>** key while entering the information. For instance, as you view an INKEY$ statement waiting for input, hold the **<Alt>** key and press any other key. This will "send" a keystroke(s) to the program.

If the <**P**> or <**X**> key is pressed while in the trace system, control will be switched to the normal output screen of the program (to the "Print Screen"). Once that screen is displayed, the keyboard is then "talking" to the program.

If you are in single step mode in a trace window and then press the <P> key, the program will remain suspended. If you had pressed <X> instead of <P> the system would switch to the output screen and go to full speed execution.

While on the Command Screen

The rule is the same as for the Program Output or "Print Screen". The keyboard does not communicate to the tracing environment unless the <Alt> key is held down.

If a program was running and was suspended, ended, or encountered an error that caused it to end, then it is still possible to reenter the tracing system and examine any window. Thus if a program error halted execution you can switch to any tracing window to see the status of the program and examine the circumstances which lead to the error.

A special note: It is possible to view the Command Screen as a window. If in the window system you press the **C** key, the current status of the Command Screen can be viewed, but not written to. Thus, you can either view the "Command Screen Window" while in the tracing system or be on the Command Screen entering commands and editing programs.

7-2 THE STATUS LINE

The first line on the screen, in the tracing system, contains information about the system. Some of the items are displayed now and some are "dormant", hence not activated and displayed

```
------------------------------------------------------------------------------

Reference    A    B    C     D   E    F  G    H  I    J    K     L   M   N      OPQ  R
Status line  10   1    000   cm  holdx   csa  CNK  I>v  step  f0  g0  24 80  VPf  end

------------------------------------------------------------------------------
```

FIGURE 7-1 Example of status line

at this time. However, we will explain all the potential information that can be displayed on the status line now.

A. The first number on the left of the status line is the line number of the statement about to be executed.

B. The second number is the cumulative number of instructions executed (including the one about to be done). Or, we could refer to this number as the "serial number" of the instruction about to be executed.

C. The third number is a kind of speedometer, showing the number of instructions per second being executed.

D. If the Control Master™ is being used (such as when the self-running demo is in effect), a **cm** will show.

E. If the **<space bar>** is pressed while Control Master™ is being used, the action will stop and Control Master™ will be in "hold" mode. The word "hold" appears on the Status Line between **cm** and the system mode indicator. If the **<space bar>** is pressed again, a single step execution of the next instruction will be executed. To return to continuous operation, the **<Enter>** key must be pressed.

```
------------------------------------------------------------------------

Reference     A   B    C    D   E    F G   H  I   J    K    L   M   N      OPQ  R
Status line   10  1   000   cm  hold x csa CN K  I>v  step  f0  g0  24 80  VPf  end

------------------------------------------------------------------------
```

FIGURE 7-1 **Example of status line**

F. The next position indicates the system state:

i -	initialized state. No program has been run.
x -	the program is in execute state. A press of the **<space bar>** or **<Enter>** will begin execution.
s -	the program is in suspension. This will happen when **<Break>** has been pressed and the user is placed on the Command Screen with the > prompt and blinking cursor displayed. Program execution can be resumed at the point it was halted by pressing **<Enter>**. If the program is edited while in this mode the status changes to a "t".
t -	the system is neither in execution nor suspended state. The program has been altered and execution cannot be simply resumed. To run the program again, enter "run" or "srun" from the Command Screen.
e -	the running program halted in an error.
f -	the program which ended in error has been edited and execution cannot be resumed (similar to the "t" state).

G. The next three positions show the status of the **Ctrl, shift, Alt** keys. When the **Ctrl, shift,** and **Alt** keys are pressed a "c, s, or a" appears on the screen. If all three keys are pressed at once the display would show **csa.**

```
----------------------------------------------------------------------------

Reference     A    B     C     D    E    F  G    H   I    J     K    L    M    N       OPQ   R
Status line   10   1    000    cm   holdx   csa  CNK  I>v  step   f0   g0   24 80   VPf   end

----------------------------------------------------------------------------
```

FIGURE 7-1 **Example of status line**

H. The next two positions show the status of the **Caps Lock** and **Num Lock** keys. If the **Caps Lock** key is toggled on, a "C" is displayed. If the **Num Lock** is toggled on an "N" is displayed. If both are switched on at the same time the display shows **CN**.

I. There is a "keyboard lock" mode toggled by the **5** key when the keypad is functioning in the cursor control (not numeric) mode. Pressing the **5** key causes a "K" to be displayed in inverse video next to the position where the **N** would be displayed. When this mode is toggled on, the **Ctrl**, **shift**, and **Alt** keys can be toggled to be continuously on or off. With this feature it is possible to control all the operations of Professional BASIC™ with one finger at a time. It is even possible to perform a system reset, by pressing the <**5**> key, the <**Ctrl**> key, the <**Alt**> key, and then the <**Del**> key one at a time. This is particularly helpful for the handi-capped.

J. The active window(s) being displayed is indicated. If two windows are shown simultaneously on the screen (see Section 8-5), then a display like **l>v** would be shown. In this example the list trace (l) window would be on the left side of the screen and the variable (v) window would be on the right. The > symbol shows that keyboard control is addressing the right (v) window. To address the left side of the screen, press the <**left cursor**> key. The display would then be **l<v.** The <**right cursor**> key switches control back to the right half of the display. If only one window is displayed then a single character is shown.

```
----------------------------------------------------------------------

Reference    A    B    C     D    E    F  G    H  I   J      K     L    M    N         OPQ  R
Status line  10   1    000   cm   holdx   csa  CNK  I>v    step  f0   g0   24 80     VPf  end

----------------------------------------------------------------------
```

FIGURE 7-1 Example of status line

K. "step" or "full" mode is shown next, to indicate if the program trace is currently set to run at full trace speed or to execute only one instruction per press of the **<space bar>**. If no program is active this space is blank.

L. The "f" followed by a number shows the current number of active FOR/NEXT loops and WHILE/WEND combinations (nestings).

M. The "g" and the number display the number of GOSUB statements that are "active".

N. The next two numbers refer to the row and column of the screen. When you are using the Command Screen, these numbers are the row and column position of the cursor. For any other type screen, they are the row and column of the Print Screen output. Thus you can see where a print statement is placing its output on the screen without going to the Print Screen itself.

O. The next character to be displayed is a "V", displayed with whatever attribute was last used to display program output. If characters are being output to the Print Screen by the program in normal mode (i.e., color 7,0) then the "V" will appear normally. If the current character being output to the coordinates shown is being displayed in inverse video (i.e., color 0,7), then the "V" will be shown in inverse video.

P. The next indicator is a "P". It is related to the program output to the "print" screen. "P" is displayed if one of the Professional BASIC™

```
-----------------------------------------------------------------

Reference    A    B    C     D   E     F  G   H I   J     K  L   M   N      OPQ  R
Status line  10   1  .000   cm  holdx   csa CNK  I>v  step f0  g0  24 80  VPf  end

-----------------------------------------------------------------
```

FIGURE 7-1 Example of status line

screens is being shown instead of the normal program output screen (Print Screen), and that output screen has been changed in some way since the last time it was looked at. With this indicator it is possible to work with the tracing and other screens and be informed when a statement in the program is executed which affects the normal program output screen (Print Screen). You can switch to the Print Screen to check what happened, return to one of the Professional BASIC™ windows, and the "P" indicator will be turned off until another change is made to the Print Screen.

Q. To the right of the position where the "P" is displayed is a number that may range from 0 to f, where a=10, b=11, c=12, d=13, e=14, and f=15. This indicates the number of characters that are in the keyboard buffer for the current BASIC program at any point in time.

R. The last thing which may be displayed on the screen relates to creating or editing lines of BASIC code. When you are approaching the maximum size of a line of code (311 characters), the display will indicate how far you are from the last position. A message like "end-4" will indicate that the cursor is four characters from the maximum line length. At the last position the display will be in inverse video and read "at end".

Figure 7-1 shows every possible element of the status line with something in each position. This example line may not be possible. It is used here for illustrative purposes only to show where and how each of the elements can be displayed relative to each other.

8. ENTERING THE WINDOW ENVIRONMENT

This chapter is designed to introduce you to the windows into program execution that Professional BASIC™ offers. Here we encourage you to examine several example programs as they execute utilizing certain keys to watch and inspect various aspects of execution. You will learn about the following screens and commands:

<L>	List Trace Window
	Profiling Options:
	<1> List Trace
	<2> Unexecuted Statements
	<3> Statement Execution Count
	<4> Execution Histogram
<T>	Time Trace Window
<S>	Split Screen
<W>	Wide (full screen)
<E>	Exchange screens
<J>	Print the listing with the current List Trace option.
<B>	The "back up" command

8-1 OPENING THE DOOR TO WINDOWING

Several sample programs are available. Each helps to demonstrate something about the Professional BASIC™ system. (These programs are included for demonstration purposes only and do not necessarily have any practical utility.) The sample programs can be accessed by typing in **load dem*** where * is one of the call characters (see the first page of the Introduction to Section II).

To start with a simple demonstration enter:

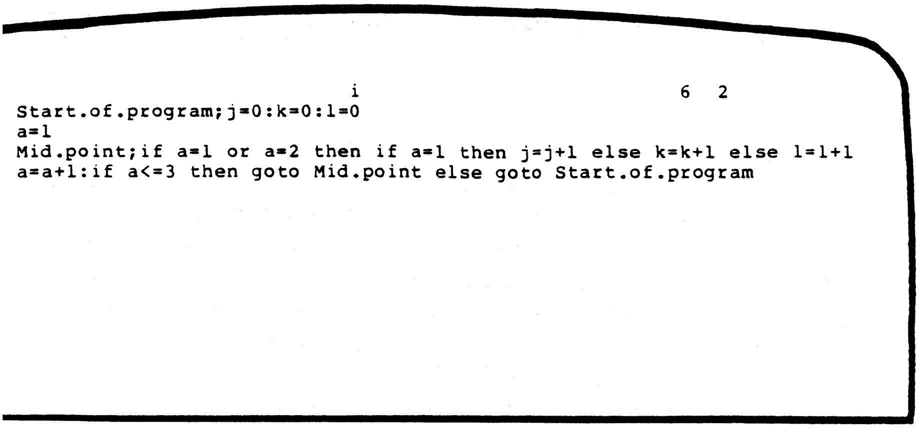
load demi

at the > prompt. The program will be loaded and your screen should look like Figure 8-1. To begin running the program you have two options. First, you could enter **run** as you would normally do in BASIC (or press the <F2> key). However, there is no program output so the screen will just be blank. The second option - the **srun** command - is the one to use for the demonstration.

Enter: **srun** on the Command Screen, or press <S> and then <F2>. This "stop run" command prepares to execute the first statement in the program, switches to the List Trace Window, and waits. At this point, the program is listed and the white line is on the first program line to be executed.

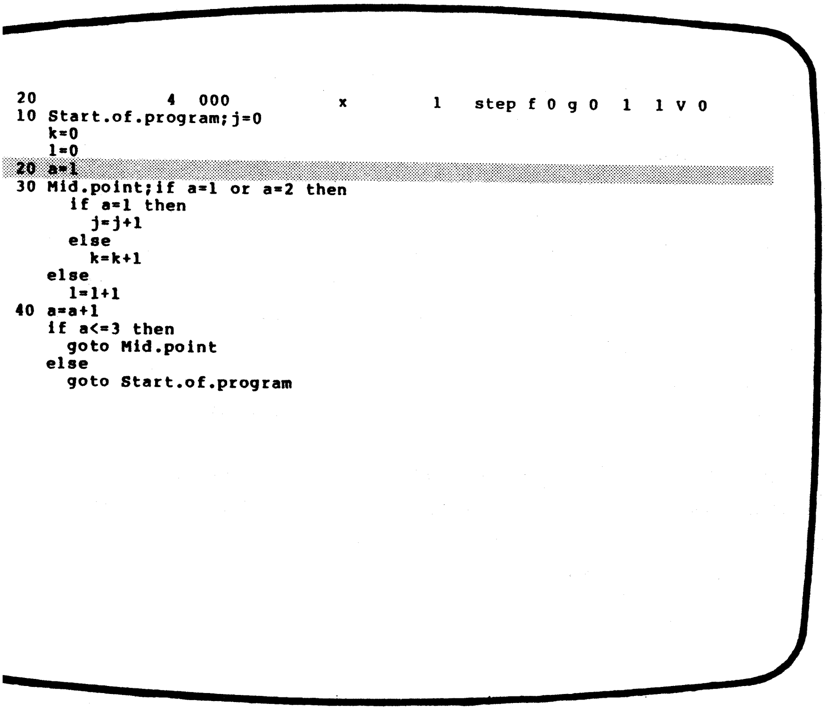
8-2 THE LIST TRACE WINDOW -- L Key

With the program in memory (loaded with the **load demi** command) and with the system at the List Trace Window (put there by the **SRUN** command) we will proceed with the demonstration. You will see the program code, but in an expanded



```
                           i                     6  2
Start.of.program;j=0:k=0:l=0
a=1
Mid.point;if a=1 or a=2 then if a=1 then j=j+1 else k=k+1 else l=l+1
a=a+1:if a<=3 then goto Mid.point else goto Start.of.program
```

8-1 Command Screen after DEMI.BAS program is loaded



```
20             4  000         x       1   step f 0 g 0  1  1 V 0
10 Start.of.program;j=0
   k=0
   l=0
20 a=1
30 Mid.point;if a=1 or a=2 then
     if a=1 then
       j=j+1
     else
       k=k+1
   else
     l=l+1
40 a=a+1
   if a<=3 then
     goto Mid.point
   else
     goto Start.of.program
```

8-2 List Trace window with line 20 highlighted - DEMI.BAS program and indented format (see Figure 8-2). Notice how the program has been rearranged so that the logical order of the statements is more clearly displayed than on the Command Screen. You can go back and forth between the Command Screen Window and the List Trace Window by pressing <C> when the List Trace Window is displayed or the <L> key when the Command Screen Window is displayed. Compare the formats of the program lines. (NOTE: When on the Command Screen Window notice that the cursor is not displayed and blinking. This is not the "active" Command Screen.)

Now press <**space bar**> once. See how the top line has been updated to show that the second instruction is about to be executed (**2**), but that the line number indicator has gone blank. The white line is over the second of three instructions that together constitute line 10.

The system works such that each trace line shows only an individual instruction. Thus, only one instruction (the first of the three) is identified with the line number label 10. As program control (indicated by the inverse video bar) moves to instructions other that the first one on a line of code, the position on the status line which indicates the next instruction to be executed is blank. The other two instructions on line 10 are also shown on separate screen lines in the List Trace window. (This one-to-one correspondence of one instruction per screen line and use of the line number to identify just the first instruction on the line also holds for other parts of the trace system, which we will examine later in the manual.)

Press <**space bar**> two more times and line 20 is highlighted as the next one to be executed. The status row shows line 20 as the next to be executed and it will be the 4th instruction to be executed.

Notice that the count on instructions is actually per instruction and not based on lines in the BASIC program, which may have multiple instructions separated by colons.

Load another sample program at this point by doing the following:

1. Press <**Break**> to go to the Command Screen.
2. Enter: **load demm** or **#m**
3. After the program is loaded enter "srun" or press <S> and then <F2>.

An extremely useful feature of the List Trace Window is the ability to scroll up and down through the program listing.

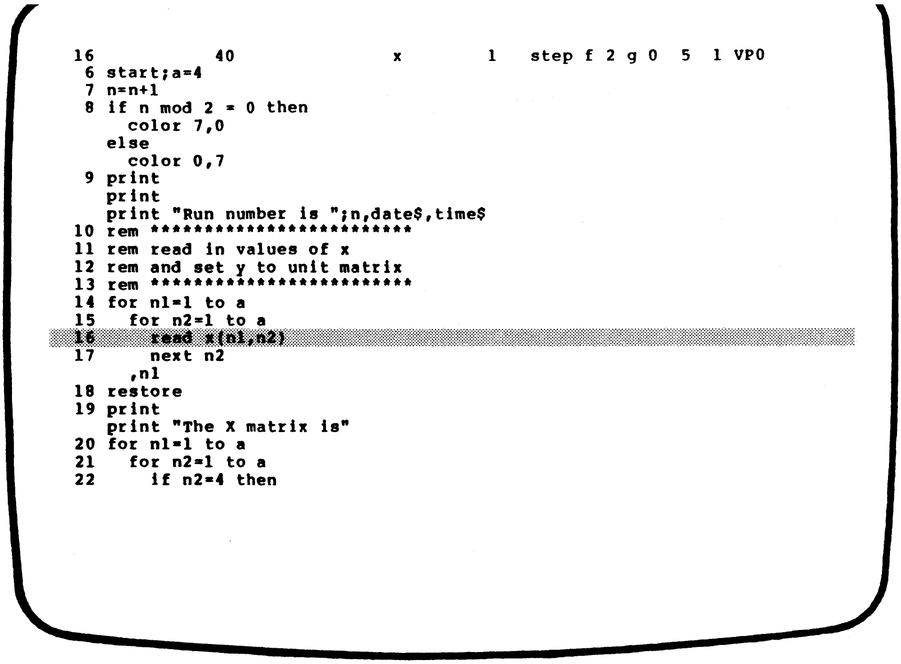
- The up and down cursor keys will scroll one line at a time up or down.

- <**PgUp**> and <**PgDn**> move a whole screen at a time up or down.

- <**Home**> will move to where the first instruction is displayed at the top of the screen.

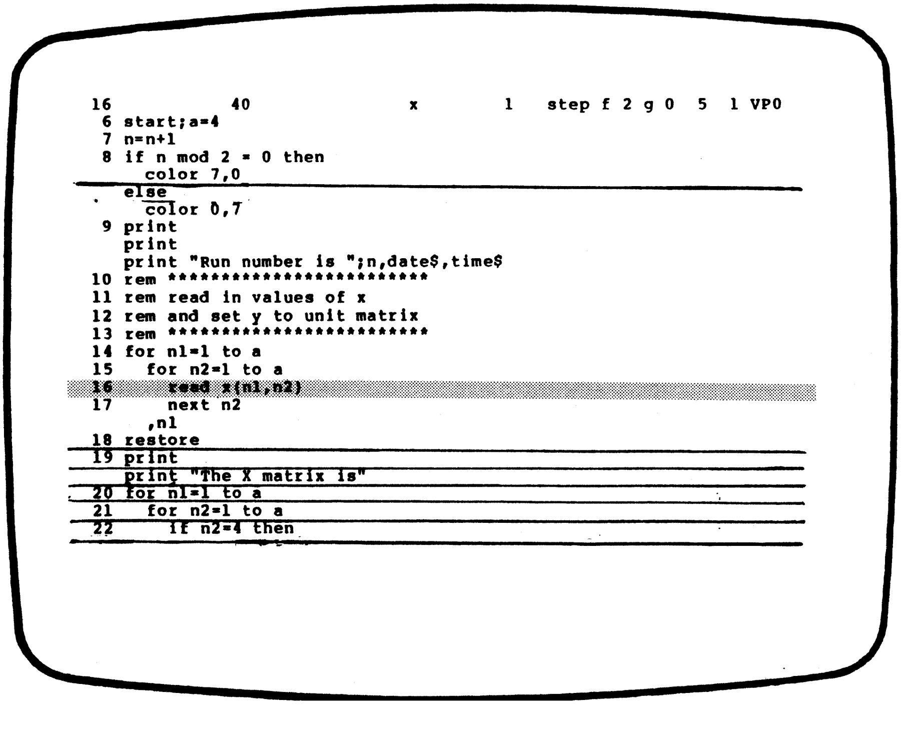
- <**End**> will move to where the last instruction is displayed as the last line on the screen.

The usual operation of the **LIST** command is available on the Command Screen. But the operation of the cursor keys to scroll up and down a program listing in the List Trace Window can be a much easier and quicker way to look at your program.



```
16              40                x        1   step f 2 g 0  5  1 VP0
 6 start;a=4
 7 n=n+1
 8 if n mod 2 = 0 then
     color 7,0
   else
     color 0,7
 9 print
   print
   print "Run number is ";n,date$,time$
10 rem *************************
11 rem read in values of x
12 rem and set y to unit matrix
13 rem *************************
14 for n1=1 to a
15   for n2=1 to a
16     read x(n1,n2)
17     next n2
     ,n1
18 restore
19 print
   print "The X matrix is"
20 for n1=1 to a
21   for n2=1 to a
22     if n2=4 then
```

FIGURE 8-3 List Trace window (1 - normal screen) DEMM.BAS program



```
16              40                x        1   step f 2 g 0  5  1 VP0
 6 start;a=4
 7 n=n+1
 8 if n mod 2 = 0 then
     color 7,0
   else
     color 0,7
 9 print
   print
   print "Run number is ";n,date$,time$
10 rem *************************
11 rem read in values of x
12 rem and set y to unit matrix
13 rem *************************
14 for n1=1 to a
15   for n2=1 to a
16     read x(n1,n2)
17     next n2
     ,n1
18 restore
19 print
   print "The X matrix is"
20 for n1=1 to a
21   for n2=1 to a
22     if n2=4 then
```

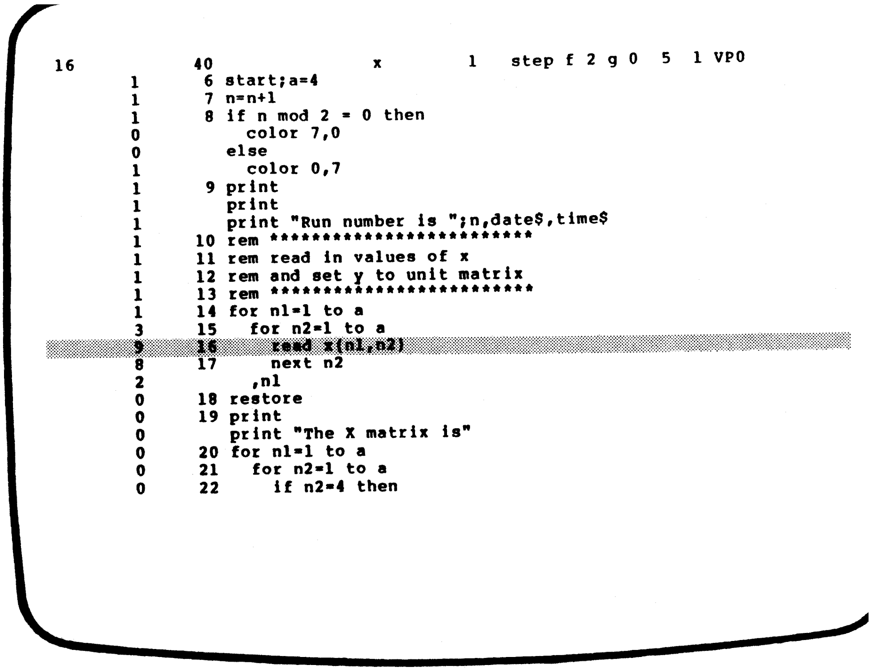
Additional Features of the List Trace Screen

Profiling -- There are four variations of the List Trace window. The normal window with the program lines shown has already been discussed. The other three variations present information about the profile of instructions executed. Options 3 and 4 are not available on a split screen. With keyboard control addressing the List Trace window the options are:

1. Press <1> to display the normal List Trace window.
2. Press <2> to underline on the screen all lines in the program which have not been executed at least once. (On a color monitor the lines will be blue. On the COMPAQ computer, the lines will be highlighted instead of underlined.)
3. Press <3> to show a count of the number of times each instruction in the program has been executed since execution was begun.
4. Press <4> to show a histogram of the relative frequency of instruction execution.

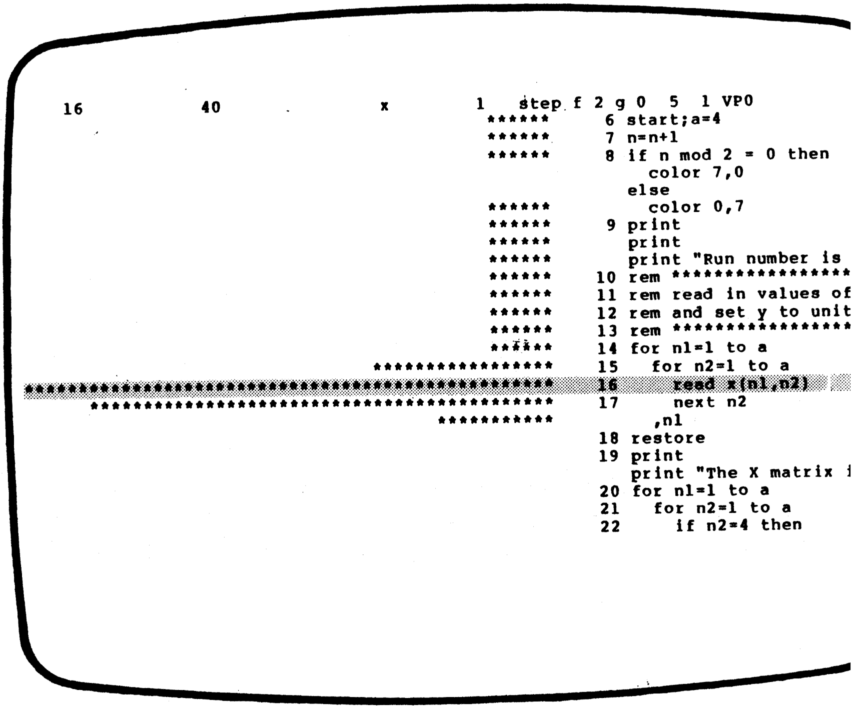
The second option can be very useful in finding out about instructions in the program which have not been executed after a set number of iterations or "execution paths." The third and fourth options may underscore the relative importance, at least in terms of execution frequency, of certain sections of code within a program.

The **noex** command entered on the Command Screen at the > prompt will reset the counters associated with this profiling system to zero. This will allow you to run a program, suspend



```
16          40                x         1   step f 2 g 0  5  1 VPO
        1      6 start;a=4
        1      7 n=n+1
        1      8 if n mod 2 = 0 then
        0          color 7,0
        0        else
        1          color 0,7
        1      9 print
        1        print
        1        print "Run number is ";n,date$,time$
        1     10 rem *************************
        1     11 rem read in values of x
        1     12 rem and set y to unit matrix
        1     13 rem *************************
        1     14 for n1=1 to a
        3     15   for n2=1 to a
        9     16     read x(n1,n2)
        8     17     next n2
        2          ,n1
        0     18 restore
        0     19 print
        0        print "The X matrix is"
        0     20 for n1=1 to a
        0     21   for n2=1 to a
        0     22     if n2=4 then
```

FIGURE 8-5 List Trace window (3 - counts frequency executions) - DEMM.BAS program



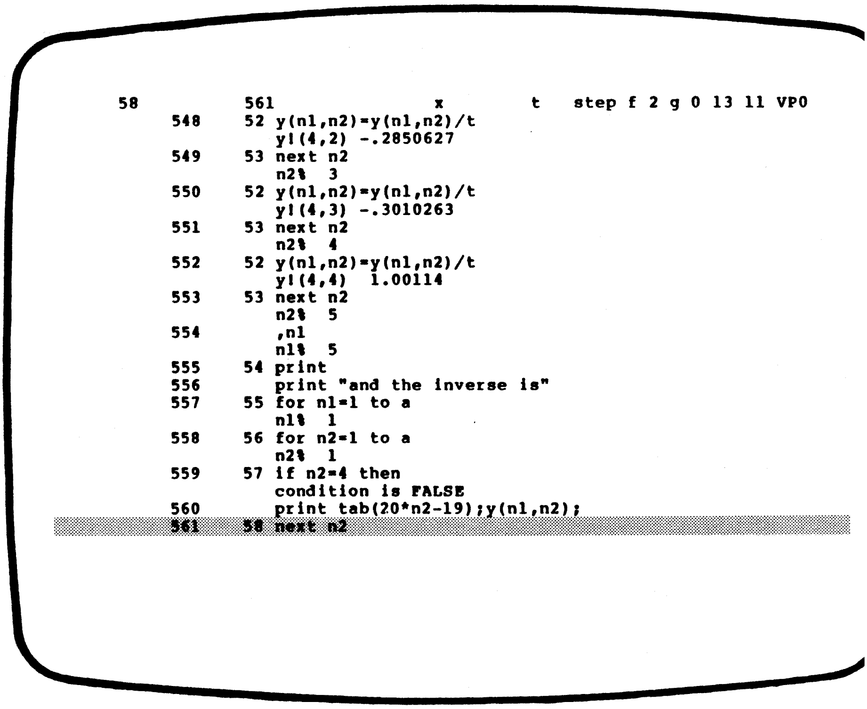
```
16          40              x          1   step f 2 g 0  5  1 VPO
                                   ******      6 start;a=4
                                   ******      7 n=n+1
                                   ******      8 if n mod 2 = 0 then
                                                   color 7,0
                                                 else
                                   ******          color 0,7
                                   ******      9 print
                                   ******        print
                                   ******        print "Run number is
                                   ******     10 rem *****************
                                   ******     11 rem read in values of
                                   ******     12 rem and set y to unit
                                   ******     13 rem *****************
                                   ******     14 for n1=1 to a
                       ******************     15   for n2=1 to a
***************************************************     16     read x(n1,n2)
      *********************************************     17     next n2
                             ************          ,n1
                                              18 restore
                                              19 print
                                                 print "The X matrix i
                                              20 for n1=1 to a
                                              21   for n2=1 to a
                                              22     if n2=4 then
```

execution at some point by pressing <**Break**>, and continue program execution from that point on, seeing the profiler information as it has accumulated information from the suspended point forward. Only this command and the LOAD command reset the counters. Editing and inserting new lines and performing multiple "RUNS" can be done while "accumulating" profile information.

Printing the Program Listing -- Instead of using the LLIST command to print the program, you may want to have a listing that corresponds to the format of the List Screen. The <**J**> key prints the program using the current profile option. The List Screen must be displayed at the time the <**J**> key is pressed. The entire program is printed no matter which part of the program is currently being displayed. If you want to quit the listing before it is entirely printed, use the <**Esc**> key. The status of the Professional BASIC™ system is not changed by this command. When printing is complete, processing can continue from the current status.

8-3 THE "BACK UP" COMMAND -- B Key

Press <**space bar**> and execute at least one instruction. Press and hold <**B**>. This is a "back up" command which will display the screen that was displayed one instruction back. Literally, if before <**B**> is pressed you were to change to one of the other screens, then pressing <**B**> would not only back up one instruction but would also show the screen exactly as it appeared before. To try this press <**T**> to display the Time Trace window. Then press <**B**> and you will see the display flip between the two trace screens.



```
58          561               x        t    step f  2  g  0  13  11  VPO
       548      52 y(n1,n2)=y(n1,n2)/t
                   y!(4,2) -.2850627
       549      53 next n2
                   n2%  3
       550      52 y(n1,n2)=y(n1,n2)/t
                   y!(4,3) -.3010263
       551      53 next n2
                   n2%  4
       552      52 y(n1,n2)=y(n1,n2)/t
                   y!(4,4)  1.00114
       553      53 next n2
                   n2%  5
       554         ,n1
                   n1%  5
       555      54 print
       556         print "and the inverse is"
       557      55 for n1=1 to a
                   n1%  1
       558      56 for n2=1 to a
                   n2%  1
       559      57 if n2=4 then
                   condition is FALSE
       560         print tab(20*n2-19);y(n1,n2);
       561      58 next n2
```

FIGURE 8-7 Example of Time Trace window (T) - DEMM program

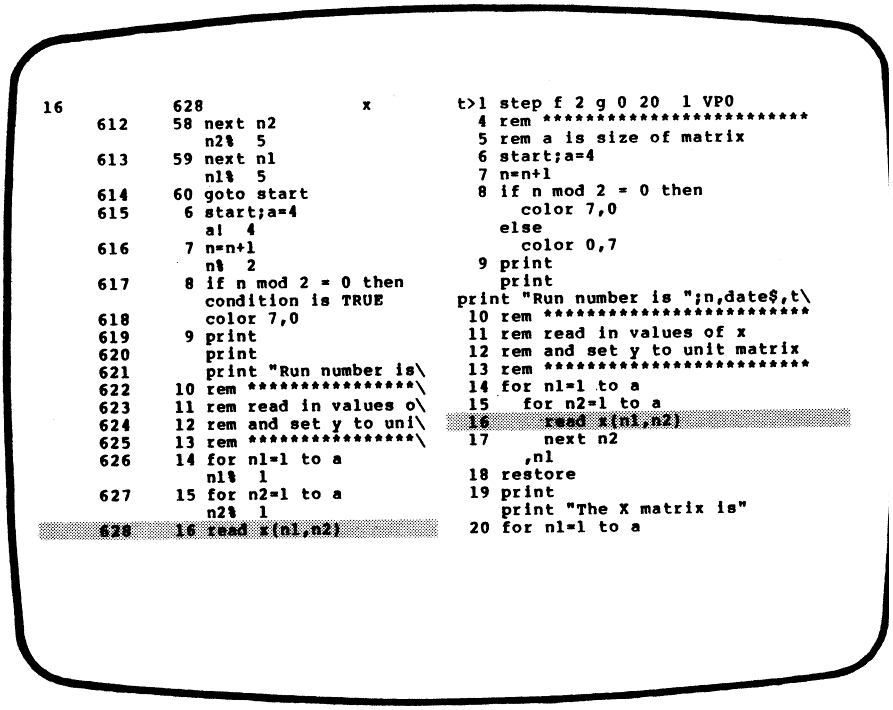
8-4 TIME TRACE WINDOW -- T Key

The Time Trace window displays a list of each instruction executed as it is run. The next instruction to be executed is always shown at the bottom of the screen in inverse video. If you press **<Enter>** the instructions will scroll up off the screen as the program is run. This screen represents a time sequence (or historical) record of program execution, while the List Trace window presents a spatial view of execution.

The historical record rewind -- A very useful feature with this window is to scroll back into the history of instructions that have been executed, by 256 to 512 instructions (depending on how many instructions use two lines, to display the instruction and the assigned value for a variable or array element). Simply press **<up cursor>** to scroll back through the record and **<down cursor>** to move forward again. **<Home>** will move all the way back into the record to the earliest instruction kept. **<End>** will move forward in time to the current instruction about to be executed.

Notice in the Time Trace Window that as each instruction is executed four things are shown:

- First, the instruction itself is shown.
- Second, the sequence (line) number of the instruction, if appropriate, is shown. As with the status line at the top of the screen, if the instruction is not the first one on a program line then no sequence number is shown.
- Third, the "serial number" of the instruction is shown. This represents the cumulative total number of instructions executed by the program



```
16          628              x
      612    58 next n2
                n2%  5
      613    59 next n1
                n1%  5
      614    60 goto start
      615     6 start;a=4
                a!  4
      616     7 n=n+1
                n%  2
      617     8 if n mod 2 = 0 then
                condition is TRUE
      618       color 7,0
      619     9 print
      620       print
      621       print "Run number is\
      622    10 rem ****************\
      623    11 rem read in values o\
      624    12 rem and set y to uni\
      625    13 rem ****************\
      626    14 for n1=1 to a
                n1%  1
      627    15 for n2=1 to a
                n2%  1
      628    16 read x(n1,n2)
```

```
t>1 step f 2 g 0 20  1 VPO
  4 rem **************************
  5 rem a is size of matrix
  6 start;a=4
  7 n=n+1
  8 if n mod 2 = 0 then
      color 7,0
    else
      color 0,7
  9 print
    print
print "Run number is ";n,date$,t\
 10 rem **************************
 11 rem read in values of x
 12 rem and set y to unit matrix
 13 rem **************************
 14 for n1=1 to a
 15   for n2=1 to a
 16     read x(n1,n2)
 17     next n2
      ,n1
 18 restore
 19 print
    print "The X matrix is"
 20 for n1=1 to a
```

FIGURE 8-8 Split (S) Time Trace window (T) and List T
window (L)

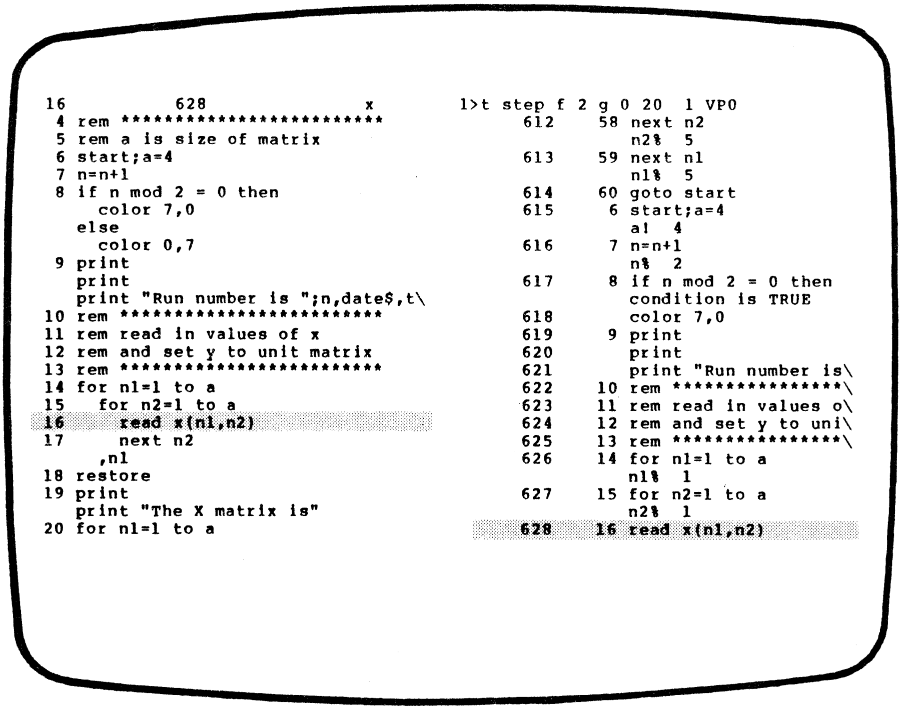
since the **run** or **srun** command was given. This number can easily be a five or even six-digit number, if the program has been running for a long time.

- Fourth, variable values are displayed to provide a convenient way to track what is happening as the program executes. For any variable assigned a value by the instruction, that value will be shown on the line after the instruction. For all "IF" statements, the conditional result is indicated as "TRUE" or "FALSE". For all "READ" statements, the data read is displayed. For POKE statements the full address, the old contents, and the new contents are given. For the MOTOR statement its on or off status is given. For DEF SEG statements the full address is given.

8-5 SPLITTING THE SCREEN - TWO WINDOWS AT ONCE -- S Key

Both the List Trace and Time Trace windows are quite valuable in themselves. But it is possible to put both on the screen at the same time by pressing <S> to "split" the screen into a left and a right half.

With the Time Trace Window displayed, press <S>. The status line will show the screen displays as "**t>l**". This means that the Time Trace Window is on the left side of the screen and the List Trace Window is on the right. Keyboard control is pointing to (or addressing) the right window (List Trace Window). Press the up and down cursor keys to verify this. Now press the left cursor key to switch control to the left window. The display will appear as "**t<l**". Now press the up



```
16           628                x
 4 rem *************************
 5 rem a is size of matrix
 6 start;a=4
 7 n=n+1
 8 if n mod 2 = 0 then
     color 7,0
   else
     color 0,7
 9 print
   print
   print "Run number is ";n,date$,t\
10 rem ************************
11 rem read in values of x
12 rem and set y to unit matrix
13 rem ************************
14 for n1=1 to a
15   for n2=1 to a
16     read x(n1,n2)
17     next n2
     ,n1
18 restore
19 print
   print "The X matrix is"
20 for n1=1 to a
```

```
1>t step f 2 g 0 20  1 VP0
      612     58 next n2
                 n2%  5
      613     59 next n1
                 n1%  5
      614     60 goto start
      615      6 start;a=4
                 a!  4
      616      7 n=n+1
                 n%  2
      617      8 if n mod 2 = 0 then
                 condition is TRUE
      618        color 7,0
      619      9 print
      620        print
      621        print "Run number is\
      622     10 rem ****************\
      623     11 rem read in values o\
      624     12 rem and set y to uni\
      625     13 rem ****************\
      626     14 for n1=1 to a
                 n1%  1
      627     15 for n2=1 to a
                 n2%  1
      628     16 read x(n1,n2)
```

FIGURE 8-9 Exchanged (E) Time Trace window and List Tr window - DEMM.BAS program

and down cursor keys and verify that keyboard control addresses the left window.

At this point the Time Trace Window is on the left and the List Trace Window is on the right. Press **<Enter>** or **<space bar>** to see the coordinated trace action with both screens. Since each side of the screen is limited to 40 columns there may be some information displayed on the wide screen not shown on the split, narrow window. For instance, long instructions will be truncated with a "\" character at the end of the line to indicate there is more. By pressing the **<W>** key, the left screen will return as a wide, single screen.

Coordination of List Trace and Time Trace Windows -- Press <S> to split the screen. With the system in the "**t>l**" configuration, press <E> to "exchange" the two screens. This is a very special configuration. With this setup (shown on the status line as **l>t**), and in single-step mode, it is possible to reverse backward through the most recent instructions executed by pressing **<up arrow>**, and see on both screens the coordinated replay of the program execution. In a sense, you can run the program backward and forward (using the cursor keys) and look at an "instant replay" of the flow of execution of a section of the program. (A note on this: you are not really running the program in reverse and changing the whole state of the system. It is a coordinated method of replaying the Time Trace Window and doing it in coordination with the List Trace Window.) This can be a very useful capability to understand just how a part of a program works. It is also an exceptional educational tool.

As you run back through the historical record, notice that the last line on the Time Trace screen is no longer shown in inverse video. It is shown underlined (on some computers, such as the COMPAQ, it is shown in high intensity). Thus you can know if the current instruction is the next one to be executed by the system or a previous instruction. When you press <**space bar**> or <**Enter**> the system will leap forward to the "current" state and execute the next instruction(s).

In summary the new commands just reviewed are:

<**L**>	List trace window (options: <**1**>, <**2**>, <**3**>, <**4**>, and <**J**>
<**T**>	Time trace window
<**S**>	split the screen into two windows, left and right
<**W**>	change left window to single, wide screen with one window
<**E**>	exchange the left and right windows when screen is split
<**B**>	Back up one screen.

Play with these windows until you feel comfortable with how to control them and the format of the information. These are only the first of the many visual presentations Professional BASIC™ offers you as you begin to trace your programs.

9. TRACING PROGRAM VALUES

This chapter deals with the screens that keep track of program data and values. Professional BASIC™ has a number of screens which display the current values of variables and array elements. In this chapter you will learn about the following windows and commands:

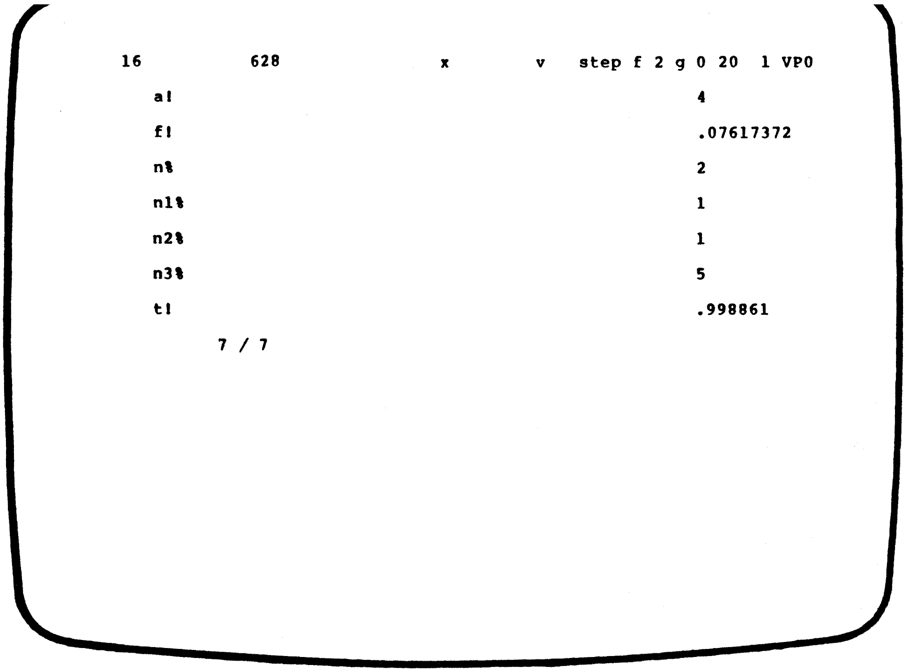
<V>	Variable Window
<A>	Array Window
<R>	Two Dimensional Array Window (single precision)

Because most programs have a number of variables and array elements, you will learn how to scroll through the values on the screen. There are numerous commands to allow you to travel with varying speed through the elements, so have your Quick Reference Card available for reference.

9-1 Variable Values

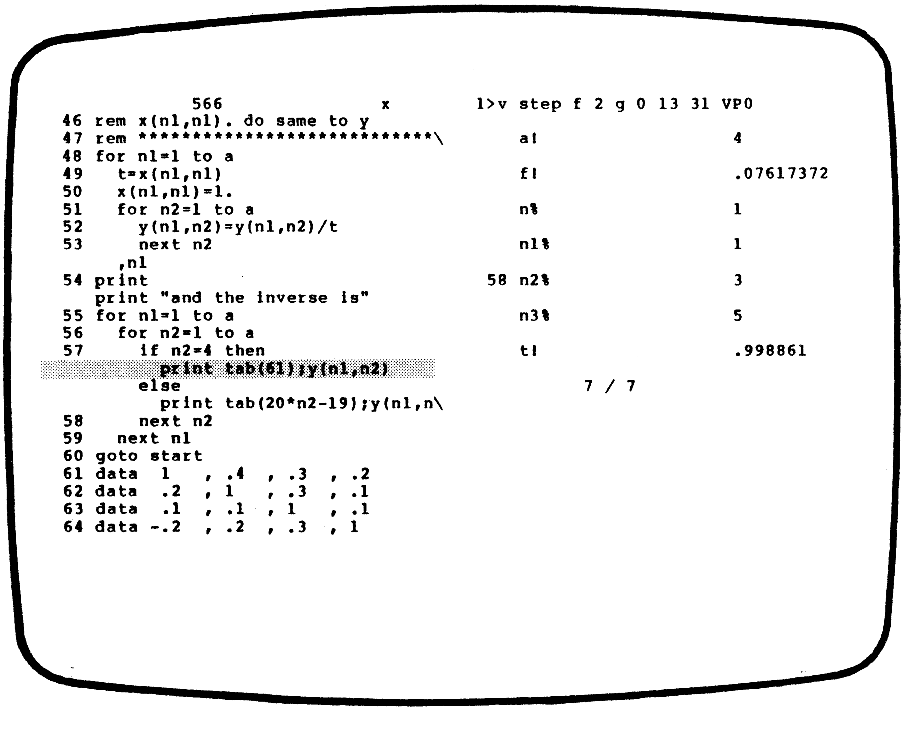
Load the **#m** sample program, press <S> and then <F2> after the program is loaded.

Now press <V>. The variable screen will be displayed (Figure 9-1). Seven variables will be displayed - all the variables in the current program. The screen indicates several things:



```
    16          628              x         v   step f 2 g 0 20  1 VPO

       a!                                              4

       f!                                              .07617372

       n%                                              2

       n1%                                             1

       n2%                                             1

       n3%                                             5

       t!                                              .998861

             7 / 7
```

FIGURE 9-1 Sample Variable (V) window - DEMM.BAS pro gram



```
              566              x          1>v step f 2 g 0 13 31 VPO
46 rem x(n1,n1). do same to y
47 rem ****************************\         a!                  4
48 for n1=1 to a
49   t=x(n1,n1)                              f!                  .07617372
50   x(n1,n1)=1.
51   for n2=1 to a                           n%                  1
52     y(n1,n2)=y(n1,n2)/t
53     next n2                               n1%                 1
   ,n1
54 print                                  58 n2%                 3
   print "and the inverse is"
55 for n1=1 to a                             n3%                 5
56   for n2=1 to a
57     if n2=4 then                          t!                  .998861
         print tab(61);y(n1,n2)
       else                                          7 / 7
         print tab(20*n2-19);y(n1,n\
58     next n2
59   next n1
60 goto start
61 data  1   , .4  , .3  , .2
62 data  .2  , 1   , .3  , .1
63 data  .1  , .1  , 1   , .1
64 data -.2  , .2  , .3  , 1
```

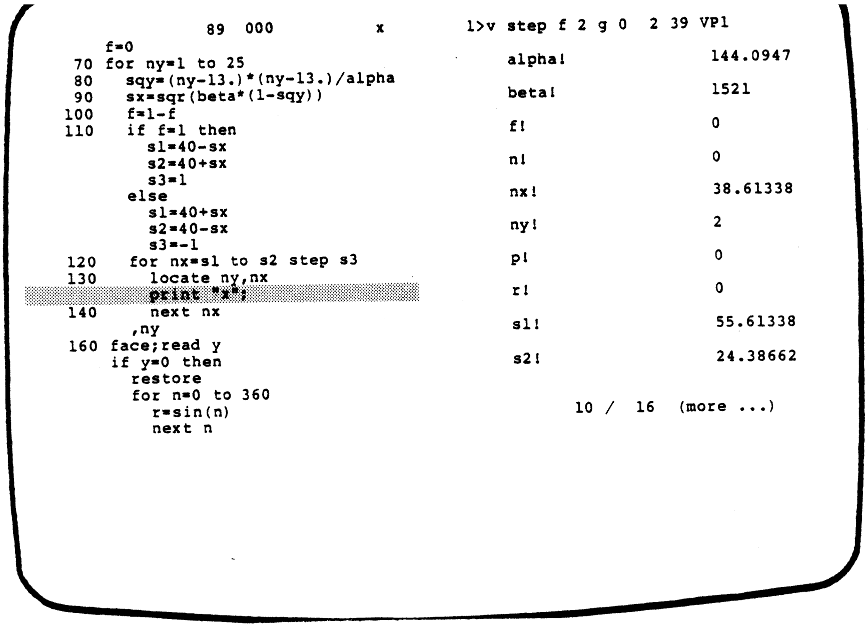
- The total number of variables in the program is shown. The *7/7* displayed below the list of variables says that there are a total of 7 variables in the program and the 7th variable is shown on the screen.

- The current value of each variable is shown to the right. As the program runs you can see the value of each variable change. The line number in which the variable was last assigned a value is shown to the left of each variable name. (The fact that no line number is shown may mean that the instruction was not the first instruction in the line. This is consistent with other displays in which the line number is represented.)

Press <**Enter**> to run the program in full trace mode (if not there already) and watch the variables change value. At any point you can stop by pressing <**space bar**>, and then restart by pressing <**Enter**>.

Display the Command Screen by pressing <C>. Note that the cursor is not present. No entry is allowed. The screen is presented for reference purposes only. (The <**Break**> would have to be used to get the active Command Screen.) Now press <S> to split the display. From the Command Screen this command will default to showing the List Trace Window on the left and the Variable Window on the right. This very useful combination lets you see the program execution proceed while monitoring the values of all the variables.

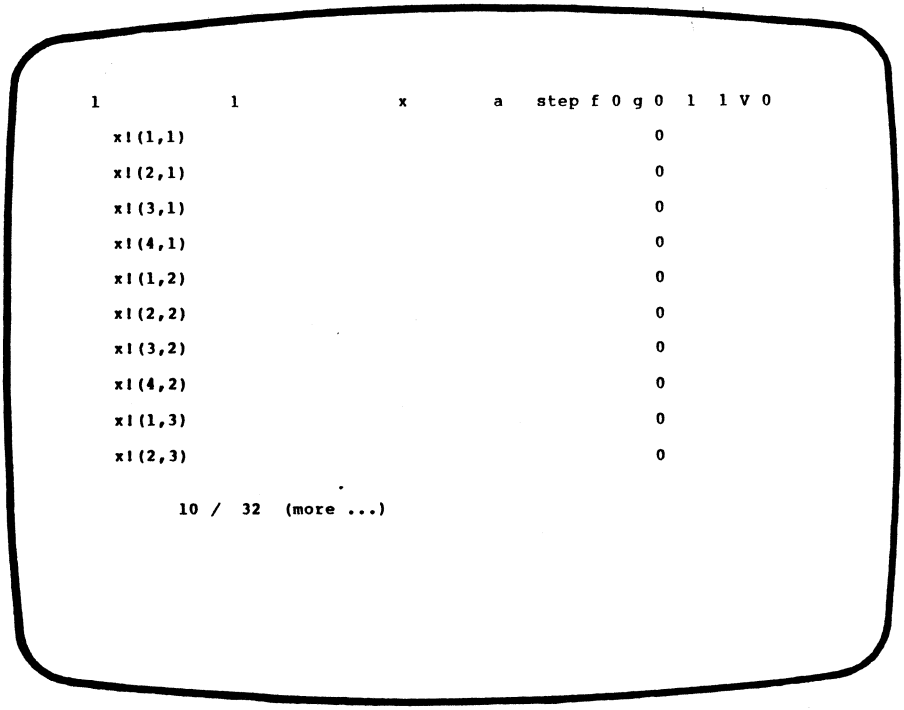
Now press <**Break**> and load in another program; enter **load deml**. Enter **srun** at the > prompt. Then split the screen by pressing the <S> key. The split screen display will show the List Trace Window on the left and 10 out of 16 variables on the



```
              89  000          x

    f=0
 70 for ny=1 to 25
 80   sqy=(ny-13.)*(ny-13.)/alpha
 90   sx=sqr(beta*(1-sqy))
100   f=1-f
110   if f=1 then
        s1=40-sx
        s2=40+sx
        s3=1
      else
        s1=40+sx
        s2=40-sx
        s3=-1
120   for nx=s1 to s2 step s3
130     locate ny,nx
        print "x";
140     next nx
      ,ny
160 face;read y
    if y=0 then
      restore
      for n=0 to 360
        r=sin(n)
        next n
```

```
1>v step f 2 g 0  2 39 VP1

alpha!              144.0947

beta!               1521

f!                  0

n!                  0

nx!                 38.61338

ny!                 2

p!                  0

r!                  0

s1!                 55.61338

s2!                 24.38662

         10 /  16  (more ...)
```

FIGURE 9-3 **Split List Trace window and Variable window - DEML.BAS program**



```
1            1                x          a   step f 0 g 0  1  1 V 0

  x!(1,1)                                        0

  x!(2,1)                                        0

  x!(3,1)                                        0

  x!(4,1)                                        0

  x!(1,2)                                        0

  x!(2,2)                                        0

  x!(3,2)                                        0

  x!(4,2)                                        0

  x!(1,3)                                        0

  x!(2,3)                                        0

        10 /  32  (more ...)
```

right side. In this case there are more variables than can be shown on the screen. To look at the 11th through 16th variables the variable screen can be scrolled with the cursor keys:

- **<up cursor>** and **<down cursor>** to move one variable,
- **<PgUp>** and **<PgDn>** to move 10 variables,
- **<Home>** and **<End>** to move from the first to the last variable.

Try using these keys. (Remember, the left and right cursor keys switch control back and forth from the left side of the screen to the right when the screen is split. The cursor and other control keys will act upon whichever screen side is designated at the time.)

9-2 Array Window -- A Key

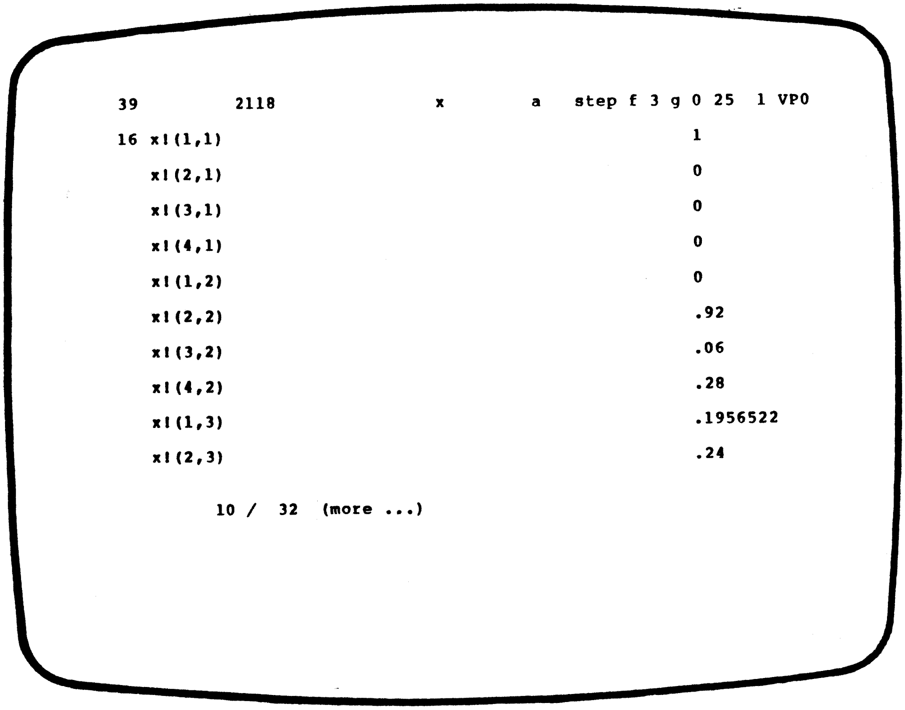
With another sample program, we can examine the Array Window.

Press **<Break>** and load in the matrix inversion program by entering **#m** at the **>** prompt. Then enter **srun** followed by <A>. The Array Window will now be displayed. See Figure 9-4 for a sample Array Window. Arrays are maintained in separate areas of memory by Professional BASIC™ and have their own system of access and display. Since array size can be extremely large, there is a special set of keys that let you navigate through the array space to find the elements you want to see.

Key:	1	2	3	4	5	6	7	8	9	0
	− 10,000	− 1,000	− 100	− 10	− 1	+ 1	+ 10	+ 100	+ 1,000	+ 10,000

Cursor Keys: (V, A Windows)	down cursor (↓)	+ 1	Home key	-first element
	up cursor (↑)	− 1	End key	-last element
	PgDn	+ 10	+ key	-to next array
	PgUp	− 10	− key	-to previous array

FIGURE 9-5 **Number of array elements scrolled in the Arra window (A) - numeric keys along top of keyboard**



```
39        2118              x          a    step f 3 g 0 25   1 VPO

16 x!(1,1)                                       1

   x!(2,1)                                       0

   x!(3,1)                                       0

   x!(4,1)                                       0

   x!(1,2)                                       0

   x!(2,2)                                       .92

   x!(3,2)                                       .06

   x!(4,2)                                       .28

   x!(1,3)                                       .1956522

   x!(2,3)                                       .24

          10 /  32  (more ...)
```

FIGURE 9-6 **Array window (A) displaying first ten elements the "x" array after program execution has beg - DEMM.BAS program**

As with the Variable Window, the cursor keys perform the same type of operations to scroll through the elements of the arrays. In addition to these cursor keys there are two more sets of keys.

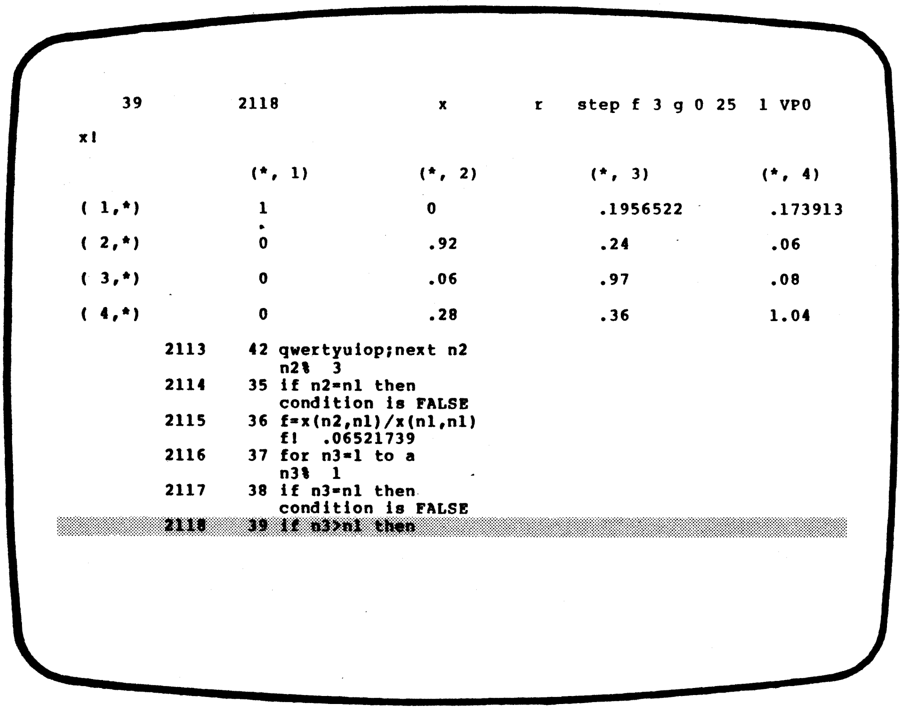
First, <+> and <-> (use the grey + and - keys to the right of the numeric keypad) allow you to move forward in the array space to the first element of the next or previous array. If an array has 10,000 elements and a subsequent array has 10 elements followed by another very large array, it could be a long and tedious search to find the smaller array nestled in between the two large ones. Pressing <+> lets you jump from the beginning of one array to the next with ease. <-> jumps back.

Second, the number keys along the top of the keyboard, 1 through 0, are used. The keys perform the movement operations illustrated in Figure 9-5. This symetrical and methodical layout can provide you with all the power you need to browse through the array elements set up in the array space in memory.

Figure 9-6 shows a typical Array window. Note that it is nearly the same as the variable screen, except that the subscripts in parentheses indicate the elements in the array.

9-3 Two Dimensional Single Precision Arrays - R Key

This window allows you to view two-dimensional single precision arrays in an easily comprehensible form. This graphic presentation can be a great aid in both teaching and debugging. While you are still in the array screen (the **#m** program should still be executing) press <**Home**>. The screen should be displaying the first 10 elements of the "x" array. Look at them closely and then press <**R**>.



```
     39          2118              x         r    step f 3 g 0 25  1 VPO

x!

                 (*, 1)          (*, 2)          (*, 3)          (*, 4)

( 1,*)            1               0              .1956522        .173913

( 2,*)            0              .92             .24             .06

( 3,*)            0              .06             .97             .08

( 4,*)            0              .28             .36             1.04
           2113     42 qwertyuiop;next n2
                       n2%  3
           2114     35 if n2=nl then
                       condition is FALSE
           2115     36 f=x(n2,nl)/x(nl,nl)
                       f!  .06521739
           2116     37 for n3=1 to a
                       n3%  1
           2117     38 if n3=nl then
                       condition is FALSE
           2118     39 if n3>nl then
```

FIGURE 9-7 Two-Dimensional Array (R) window displaying t "x" array (This window for two-dimension single-precision numeric arrays only) DEMM.BAS program

Due to the physical size limitations of the screen, the Two Dimensional Array screen only supports single precision arrays. If your two dimensional array is double precision you must use the Array Screen to view it.

Figure 9-7 is an example of a typical Two Dimensional Array screen. The screen is split horizontally to allow you to view sixteen elements of an array. The bottom half of the screen is devoted to the Time Trace window to allow you to view the execution of the code that affects the arrays. Press the <**Enter**> key to watch the screen in action.

As with the Array and Variable Windows, there are keys to allow you to scroll and page through the different arrays and their associated elements.

If you wish to view your values in hexadecimal, press the **U** key while the Two Dimensional Array Screen is active.

To change from one array to another you may use the grey <+> and <-> keys on to the right of the numeric keyboard, in the same fashion you have used them on the Array Screen. Whenever you move forward to a new array, or backward to a previous array, the first element of the array will be in the upper left corner of your screen.

To move horizontally through the array, you may use the 4, 5, 6, and 7 keys located along the top of your keyboard. Pressing the 5 or 6 key will move the display one column to the left or right. Pressing the 4 or 7 key will move the display 10 columns to the left or right. Vertical motion is accomplished using the cursor keys, as with the Variable window.

Now you have looked at two fundamental sets of windows. The first set, in Chapter 8, dealt with viewing the program code and how control flowed through the program. This chapter dealt with the windows which allow you to view the current values of all variables defined in the program.

The next chapter will deal with windows which allow you to examine the execution of certain statements in BASIC -- the FOR/NEXT pairs, GOSUBS, and READ/DATA statements.

10.
LOOPS, SUBROUTINES & DATA STATEMENTS

In this chapter we deal with iteration, subroutines, and reading data. You will learn about the following keys:

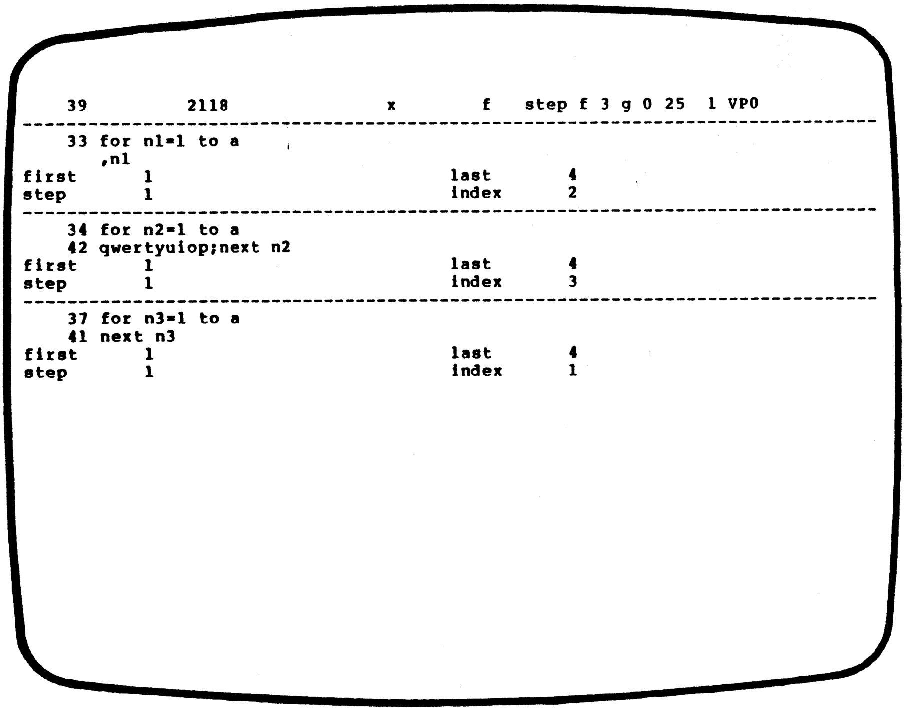
<F>	FOR/NEXT Window
<G>	GOSUB Window
<D>	READ/DATA Window

10-1 FOR/NEXT WINDOW -- F Key

Looping within a program using the combination of the "FOR" and "NEXT" statements can be viewed via the FOR/NEXT Window. To see how this special window works, load in a sample program:

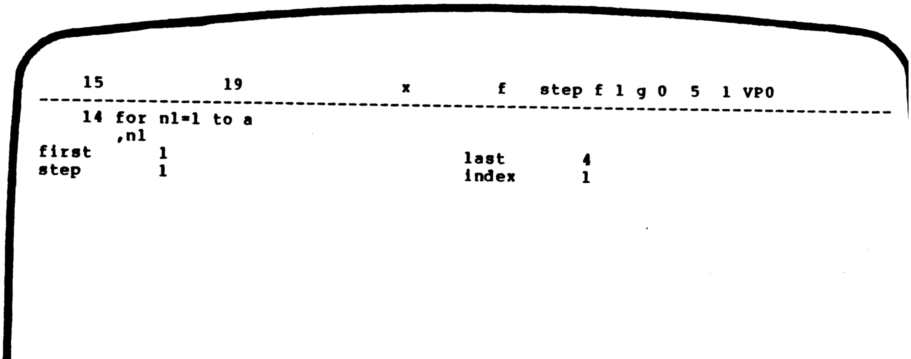
- Enter **load demm** (or **#m**) at the > prompt
- Enter **srun** to initiate execution
- Then press <F> to look at the FOR/NEXT window.

Initially the screen will indicate that there are no active loops. With the <**space bar**>, single step until you get to the 19th instruction (Remember, the "serial" number of the instruction about to be executed is displayed at the top of the screen, to the right of the line number about to be executed). This will



```
   39        2118                 x         f    step f 3 g 0 25   1 VPO
--------------------------------------------------------------------------------
   33 for n1=1 to a
      ,n1
first      1                            last       4
step       1                            index      2
--------------------------------------------------------------------------------
   34 for n2=1 to a
   42 qwertyuiop;next n2
first      1                            last       4
step       1                            index      3
--------------------------------------------------------------------------------
   37 for n3=1 to a
   41 next n3
first      1                            last       4
step       1                            index      1
```

FIGURE 10-1 For/Next window (F) - the DEMM.BAS program



```
   15           19                x         f    step f 1 g 0   5   1 VPO
--------------------------------------------------------------------------------
   14 for n1=1 to a
      ,n1
first      1                            last       4
step       1                            index      1
```

FIGURE 10-2 For/Next window (F) displayed at the 19th struction - DEMM.BAS program (Note num "19" on status line at top of screen)

initialize the first FOR/NEXT loop. There are five parts to the display on this loop:

- The pair of statements containing the "FOR" part of the statement and the "NEXT" part.

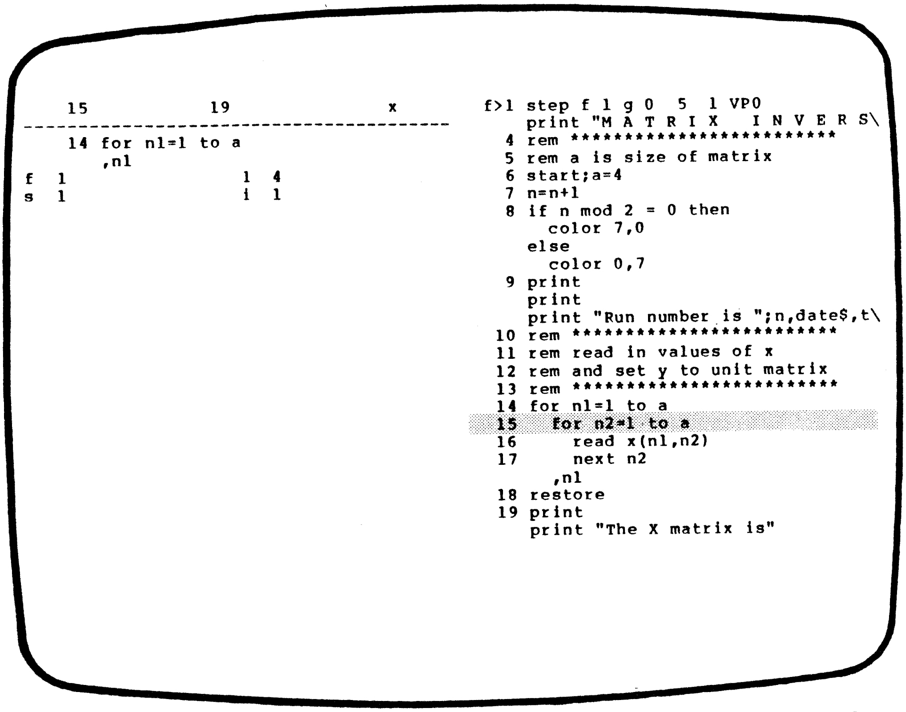
- The four parameters involved with the loop:

  - The beginning index value; **first**
  - The ending index value; **last**
  - The increment value in the loop; **step**
  - The current value of the index; **index**

Now press <**space bar**> once more. The second FOR/NEXT loop will become active. Up to four active loops can be displayed on this screen (the four "most active"). Notice that the number of active loops is displayed to the right on the status line next to the **f**. (The **g** indicates the number of GOSUBS that are active.)

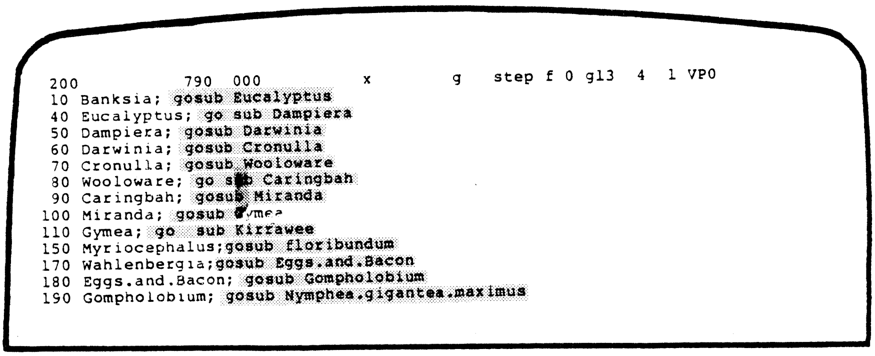
Now press <**Enter**> to run the program at full trace speed. Notice how the screen display changes as the number of concurrently active FOR/NEXT loops changes. Notice how the value of the index changes. As it reaches the last value of the loop, that loop becomes inactive and is dropped from the display.

Now split the screen by pressing <S>. The List Trace Window will be displayed on the right. Because of space limitations the display has become abbreviated. Instead of the words "first, last, step, and index" now the initials "f, l, s, and i" are shown.



```
   15              19                   x       f>1 step f 1 g 0  5  1 VPO
---------------------------------------             print "M A T R I X   I N V E R S\
   14 for n1=1 to a                                4 rem *************************
      ,n1                                          5 rem a is size of matrix
f  1                    1  4                       6 start;a=4
s  1                    i  1                       7 n=n+1
                                                   8 if n mod 2 = 0 then
                                                       color 7,0
                                                     else
                                                       color 0,7
                                                   9 print
                                                     print
                                                     print "Run number is ";n,date$,t\
                                                  10 rem *************************
                                                  11 rem read in values of x
                                                  12 rem and set y to unit matrix
                                                  13 rem *************************
                                                  14 for n1=1 to a
                                                  15   for n2=1 to a
                                                  16     read x(n1,n2)
                                                  17     next n2
                                                       ,n1
                                                  18 restore
                                                  19 print
                                                     print "The X matrix is"
```

FIGURE 10-3 Split For/Next and List Trace window DEMM.BAS program



```
200          790  000          x        g   step f 0 g13  4  1 VPO
 10 Banksia; gosub Eucalyptus
 40 Eucalyptus; go sub Dampiera
 50 Dampiera; gosub Darwinia
 60 Darwinia; gosub Cronulla
 70 Cronulla; gosub Wooloware
 80 Wooloware; go sub Caringbah
 90 Caringbah; gosub Miranda
100 Miranda; gosub Gymea
110 Gymea; go  sub Kirrawee
150 Myriocephalus;gosub floribundum
170 Wahlenbergia;gosub Eggs.and.Bacon
180 Eggs.and.Bacon; gosub Gompholobium
190 Gompholobium; gosub Nymphea.gigantea.maximus
```

FIGURE 10-4a Sample Gosub window (G) - DEMB.BAS progran

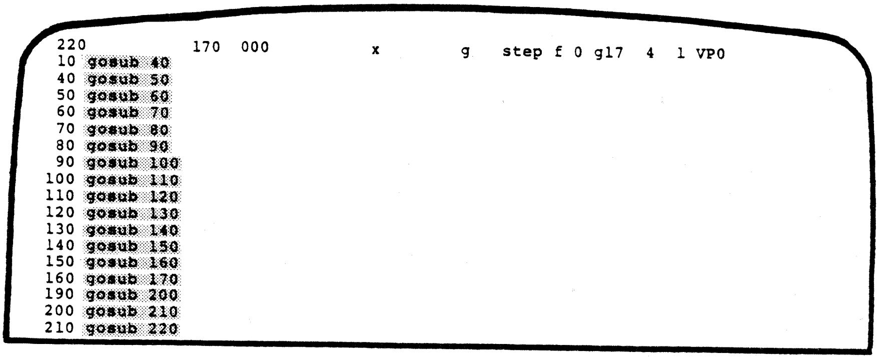
10-2 THE "CLICKER" -- N Key

As a program runs in either normal execution mode or while a trace window is active, it is possible to hear it executing by toggling the "clicker" on by pressing <N> (or, <**Alt N**> if you on the Print Screen). Each instruction executed is a click. As program speed increases or decreases the frequency of the clicks increases or decreases. By toggling the "clicker" on while running a few programs you will be able to get a feel for the speed of execution of various parts of BASIC code just by listening. If you ever want to get a feeling of how fast double precision calculations are versus integer or single precision calculations, you can construct a simple example program which consecutively executes the same calculation each way. Put each calculation in a FOR/NEXT loop which executes it about 200-500 times.

10-3 GOSUB WINDOW -- G Key

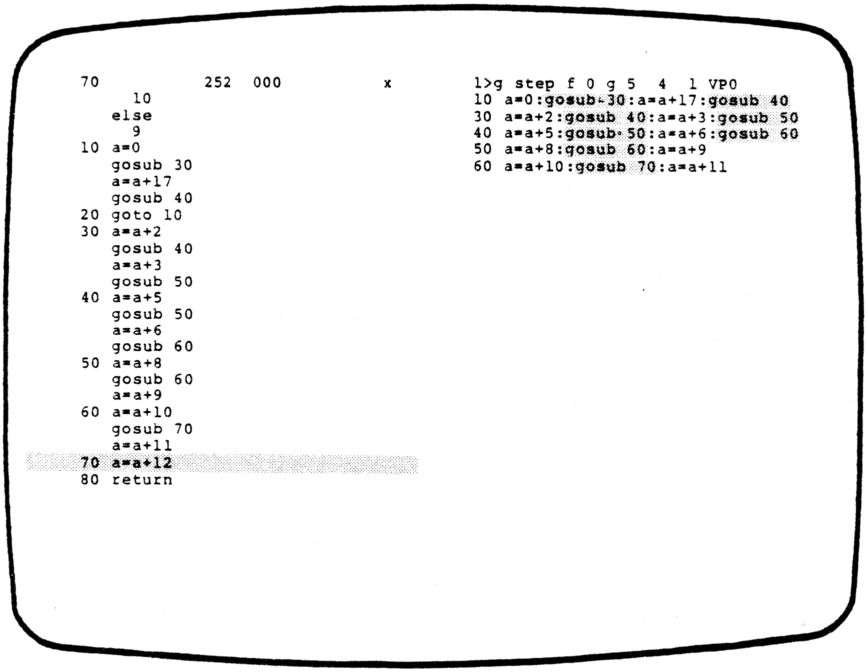
The GOSUB Window shows the list of active subroutines and is displayed by pressing <G>. Only the statement containing the GOSUB statement is shown. There may be several statements on the line of code with the GOSUB. Therefore, the GOSUB statement that is active for the line will be shown in reverse video. Like the FOR/NEXT loops, the number of active subroutines (the nesting level) is shown on the right of the Professional BASIC™ status line, next to the **g**.

There are three demonstration programs which specifically show the operation of the GOSUB display: **dema**, **demb**, and **demc**. The **dema** and **demb** programs are identical except that one uses line numbers and the other takes advantage of one of the unique features of Professional BASIC™ -- labels for lines. Any time a GOSUB or GOTO is used, you can reference a line by a line number or, if the line is labeled, by a name.



```
220             170   000          x          g   step f 0 g17   4   1 VP0
 10 gosub 40
 40 gosub 50
 50 gosub 60
 60 gosub 70
 70 gosub 80
 80 gosub 90
 90 gosub 100
100 gosub 110
110 gosub 120
120 gosub 130
130 gosub 140
140 gosub 150
150 gosub 160
160 gosub 170
190 gosub 200
200 gosub 210
210 gosub 220
```

FIGURE 10-4b **Sample Gosub window (G) - DEMA.BAS progran**



```
70            252   000          x          1>g step f 0 g 5   4   1 VP0
     10                                     10  a=0:gosub 30:a=a+17:gosub 40
   else                                     30  a=a+2:gosub 40:a=a+3:gosub 50
      9                                     40  a=a+5:gosub 50:a=a+6:gosub 60
10 a=0                                      50  a=a+8:gosub 60:a=a+9
   gosub 30                                 60  a=a+10:gosub 70:a=a+11
   a=a+17
   gosub 40
20 goto 10
30 a=a+2
   gosub 40
   a=a+3
   gosub 50
40 a=a+5
   gosub 50
   a=a+6
   gosub 60
50 a=a+8
   gosub 60
   a=a+9
60 a=a+10
   gosub 70
   a=a+11
70 a=a+12
80 return
```

FIGURE 10-5 **Split (S) List Trace window (L) and Gosub wind (G) - DEMC.BAS program**

From the Command Screen enter: **load dema** or **load demb.** Begin running the program by entering **run.** Then switch to the GOSUB window by pressing <**Alt** G>. As the program runs notice how the display always shows the GOSUB statements that are active and the order of their nesting.

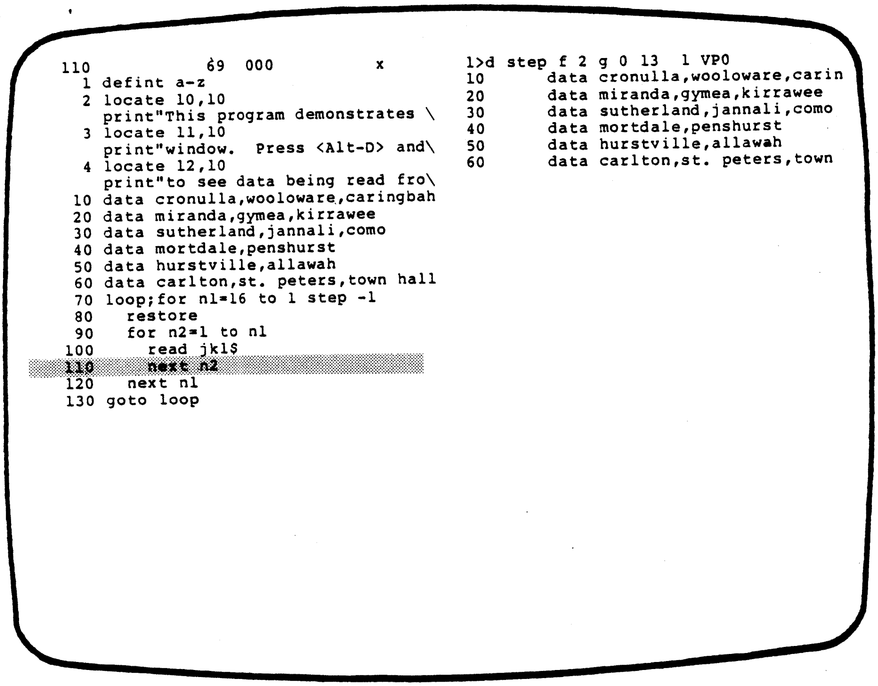
Go back to the Command Screen and invoke the **load demc** demo program. Run this program with the screen split, showing the List Trace window and the GOSUB Window. To do this:

1. Press <**Break**> to go to Command Screen.
2. Enter: **load demc** to load demo program
3. Enter: **run** or press <**F2**>.
4. Screen will be blank
5. Press <**Alt S**> and then <G> to display List Trace Window on left and the Gosub Window on the right.

In this program the GOSUB statements are contained on lines with multiple instructions. See how the active GOSUB statements are highlighted in inverse video. Slow program to single step mode with the <**space bar**> and observe the change in the screen while single stepping.

10-4 DATA WINDOW -- D Key

The next window to be discussed is the DATA Window, which traces the activity of READ and DATA statements. The **demd** program will demonstrate how this window works. From the Command Screen, load in this program and run it. (Enter **load demd** to load program, then press <**F2**>.) Split the screen with the <**Alt S**> keys and then bring up the DATA Window by pressing <D>. (**Remember**: the part of the status line that shows the active windows will show a **l>d** when the List Trace



```
110            69  000         x
  1 defint a-z
  2 locate 10,10
    print"This program demonstrates \
  3 locate 11,10
    print"window.  Press <Alt-D> and\
  4 locate 12,10
    print"to see data being read fro\
 10 data cronulla,wooloware,caringbah
 20 data miranda,gymea,kirrawee
 30 data sutherland,jannali,como
 40 data mortdale,penshurst
 50 data hurstville,allawah
 60 data carlton,st. peters,town hall
 70 loop;for n1=16 to 1 step -1
 80   restore
 90   for n2=1 to n1
100     read jkl$
110     next n2
120   next n1
130 goto loop
```

```
1>d step f 2 g 0 13  1 VP0
10      data cronulla,wooloware,carin
20      data miranda,gymea,kirrawee
30      data sutherland,jannali,como
40      data mortdale,penshurst
50      data hurstville,allawah
60      data carlton,st. peters,town
```

FIGURE 10-6 Split (S) List Trace window (L) and Data windo (D) - DEMD.BAS program

Window is shown on the left, the DATA window is shown on the right, and the keyboard is currently addressing the right side of the display. If you press <**left cursor**> the display changes to **l<d** and the keyboard addresses the left half of the display.)

As the program runs it will encounter the READ statement and data will be picked up from the DATA statements and used by the program. You can easily see which data element is next to be read since it is highlighted in inverse video. Notice that as the RESTORE statement is executed, the pointer to the next data element to be read is reset to the first one. (The **load deml** program and the **load demm** programs also use the data statement. Run them with the DATA and List Trace windows viewed together to see how data is read in and used by the program.)

As with the <**A**> and <**V**> windows, if there are more DATA statements than can be put on 24 lines, use the cursor keys to scroll up and down or the <**Home**> and <**End**> keys for the beginning and end.

11. PROGRAM OUTPUT

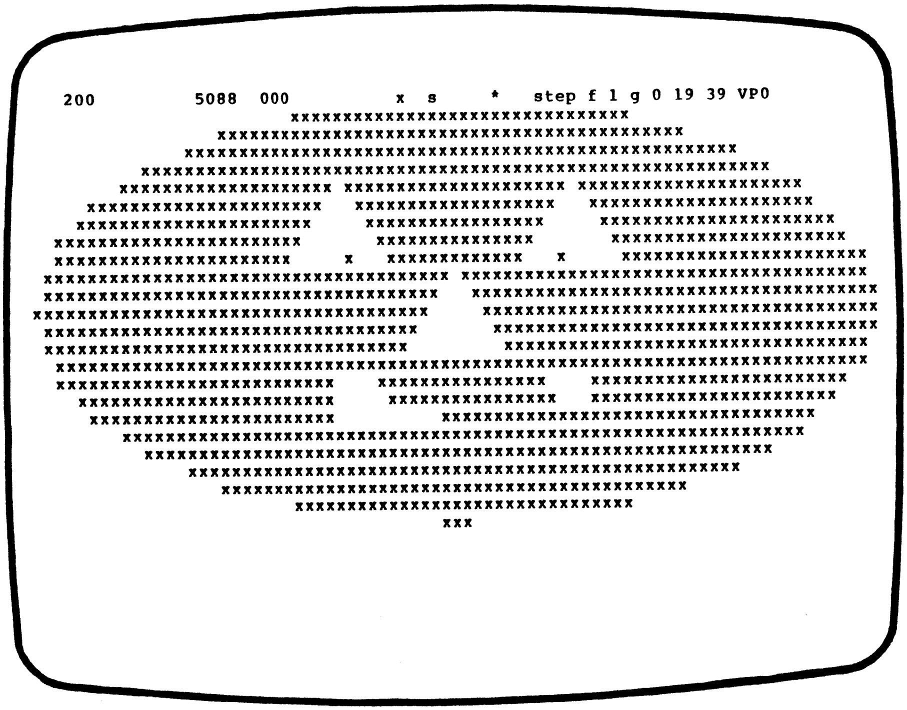
This chapter deals with that all-important aspect of your program, the one that is most visible to the user, the Output Screen. There are several different Print Screen options in Professional BASIC™. In this chapter you will learn about:

<P>	Print Screen
<X>	Full Speed Print Screen
<*>	(**Prtsc**) Print Trace Window
<Q>	Print/List Window

There are four keys which will switch to some form of showing the program output. The **P** and **X** keys will switch to the normal output screen and let you see the output of program PRINT statements. The other two options will stay within the tracing system - **PrtSc** and **Q** keys.

11-1 Print Screen -- P Key

Load a sample program which demonstrates the Print Screen. Enter: **load deml** at the > prompt. Enter **run** (or press <F2>) to begin execution of the program. The pumpkin design will be repeatedly drawn on the screen, alternating between inverse video and normal output. To stop the action press **<Alt space bar>**. You can control the output to this Print window



```
   200         5088  000          x  s     *   step f 1 g 0 19 39 VP0
                        xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
          xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxx
     xxxxxxxxxxxxxxxxxxxxxx   xxxxxxxxxxxxxxxxxxx   xxxxxxxxxxxxxxxxxxxxx
    xxxxxxxxxxxxxxxxxxxxxx     xxxxxxxxxxxxxxxxx     xxxxxxxxxxxxxxxxxxxxxx
  xxxxxxxxxxxxxxxxxxxxxxx       xxxxxxxxxxxxxxx       xxxxxxxxxxxxxxxxxxxxxx
  xxxxxxxxxxxxxxxxxxxxxx     x   xxxxxxxxxxxxx   x     xxxxxxxxxxxxxxxxxxxxxxx
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx     xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx       xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx         xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  xxxxxxxxxxxxxxxxxxxxxxxxxx    xxxxxxxxxxxxxxxx    xxxxxxxxxxxxxxxxxxxxxxxx
    xxxxxxxxxxxxxxxxxxxxxxxx     xxxxxxxxxxxxxxxx   xxxxxxxxxxxxxxxxxxxxxxx
     xxxxxxxxxxxxxxxxxxxxxxx          xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
          xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
              xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                        xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                                      xxx
```

FIGURE 11-1 Sample Print Output screen (* - PrtSc) - DEML.BAS program

with the **<Alt space bar>** and the **<Alt Enter>** keys. Remember, the normal keyboard control is for program input on the Print Screen. The <Alt> key is needed to direct keystrokes to the trace system. You can switch to the List Trace or any of the other windows and then back to the Print Screen at will. What may be interesting is to go to the List Trace Window and execute up to the point that an "x" is about to be printed on the screen by a "print" statement (line 130). With the system in step mode, switch to the Print Screen <P>and press the **<Alt space bar>** once and watch the "x" be printed on the screen.

Very minute tracing of screen building can be viewed by using the various Professional BASIC™ windows and by controlling program execution with the single step control.

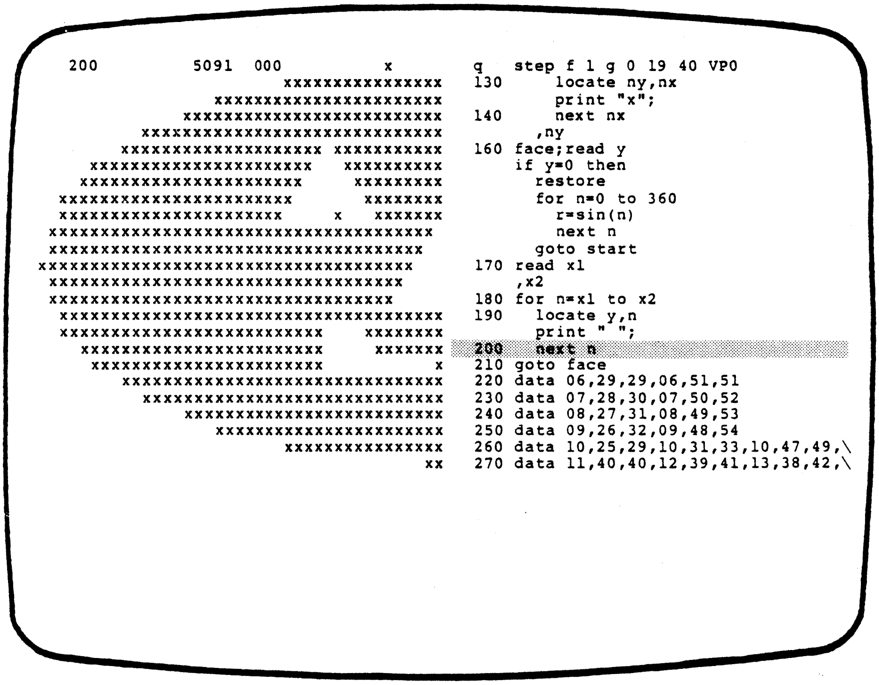
11-2 Full Execute on Print Screen -- X Key

Pressing the **X** key is somewhat like pressing the **P** key. The difference is that the **X** key always places the system in full speed execution mode and turns the "clicker" off if it is activated.

11-3 Print Trace Window -- * (PrtSc) Key

The Print Trace window is a copy of the Print screen (P key) but is in the tracing/viewing system. There are a few differences, however:

- The top line of the screen shows the status line, not the top line which would normally appear on the Print screen.



```
   200         5091  000          x        q   step f 1 g 0 19 40 VP0
                        xxxxxxxxxxxxxxxx   130     locate ny,nx
                 xxxxxxxxxxxxxxxxxxxxxxx           print "x";
              xxxxxxxxxxxxxxxxxxxxxxxxxx   140     next nx
          xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx         ,ny
        xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxx   160 face;read y
     xxxxxxxxxxxxxxxxxxxxxx   xxxxxxxxxx       if y=0 then
    xxxxxxxxxxxxxxxxxxxxxx     xxxxxxxxx         restore
  xxxxxxxxxxxxxxxxxxxxxxx       xxxxxxxx         for n=0 to 360
  xxxxxxxxxxxxxxxxxxxxxx     x   xxxxxxx           r=sin(n)
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx             next n
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx            goto start
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx       170 read x1
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx            ,x2
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx         180 for n=x1 to x2
  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx   190   locate y,n
  xxxxxxxxxxxxxxxxxxxxxxxxxx    xxxxxxxx         print " ";
    xxxxxxxxxxxxxxxxxxxxxxxx     xxxxxxx   200   next n
     xxxxxxxxxxxxxxxxxxxxxxx           x   210 goto face
        xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx   220 data 06,29,29,06,51,51
          xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx   230 data 07,28,30,07,50,52
              xxxxxxxxxxxxxxxxxxxxxxxxxx   240 data 08,27,31,08,49,53
                 xxxxxxxxxxxxxxxxxxxxxxx   250 data 09,26,32,09,48,54
                        xxxxxxxxxxxxxxxx   260 data 10,25,29,10,31,33,10,47,49,\
                                      xx   270 data 11,40,40,12,39,41,13,38,42,\
```

FIGURE 11-2 **Print/List window (Q) - DEML.BAS program**

- Execution of the BASIC program is slowed if you watch execution with this window. It is to be used just for trace/viewing purposes. The real output screen allows the BASIC program to run at full speed.

- The window may be split via the "S" key and then combined with any other window. The * Window will appear on the left when the horizontal cursor location on the Print Screen is in the range 1-40. If the horizontal cursor location is in the range 41-80, the * Window is shown on the right and the other window on the left.

11-4 Print/List Window -- Q Key (adaptive L and * windows)

The **Q** Key is a split screen combination of the List Trace and the Print Trace windows. It allows you to see output as it is sent to the output screen at all times (except for the first screen line which is replaced by the status line). This is the only combination accessible via the Q Key. In all other respects it behaves as any other screen when split with the * Screen as detailed in Section 11-3.

12. ADVANCED TRACING

This chapter is for the programmer who is more familiar with the machine. The features described here are most useful to someone with a grounding in file I/O (sequential and random access files) or with Assembler.

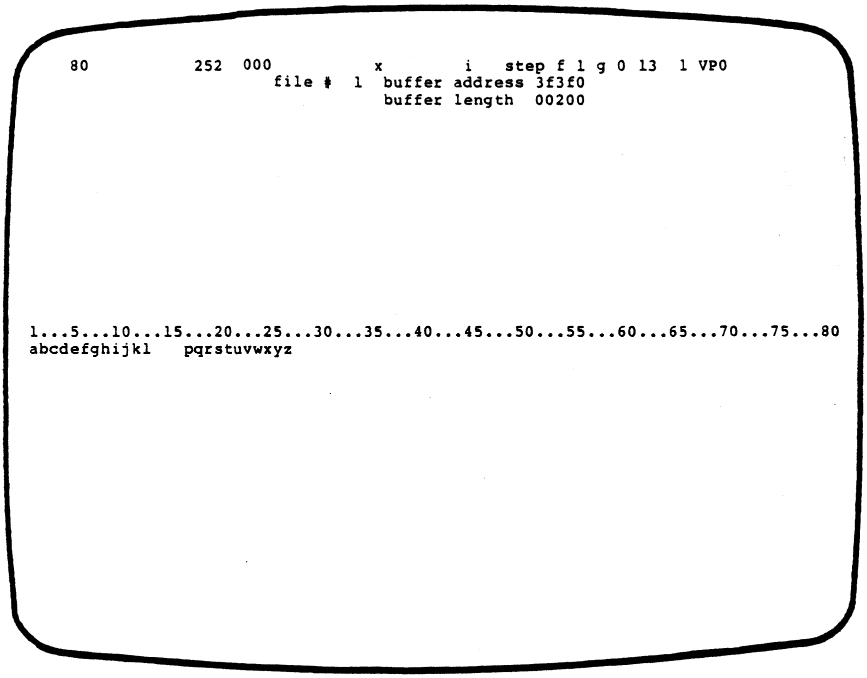
12-1 THE FILE INPUT/OUTPUT WINDOW -- I Key

When you are reading and writing information to sequential or random access files the Input/Output Window may be viewed to see the information currently in the file buffer for each opened file. When the size of the file buffer exceeds the width of the screen you may use the "5" and "6" keys along the top of the keyboard to scroll horizontally through the buffer and view the information in it. The "5" and "6" keys move the buffer left and right one character, respectively.

With a full screen, 80 bytes of the buffer may be displayed. If the screen is split, then 40 bytes are shown.

To move from one file buffer to another press the <+> key repeatedly to see the next file buffer in ascending order and the <-> key to look at the next lower file buffer.

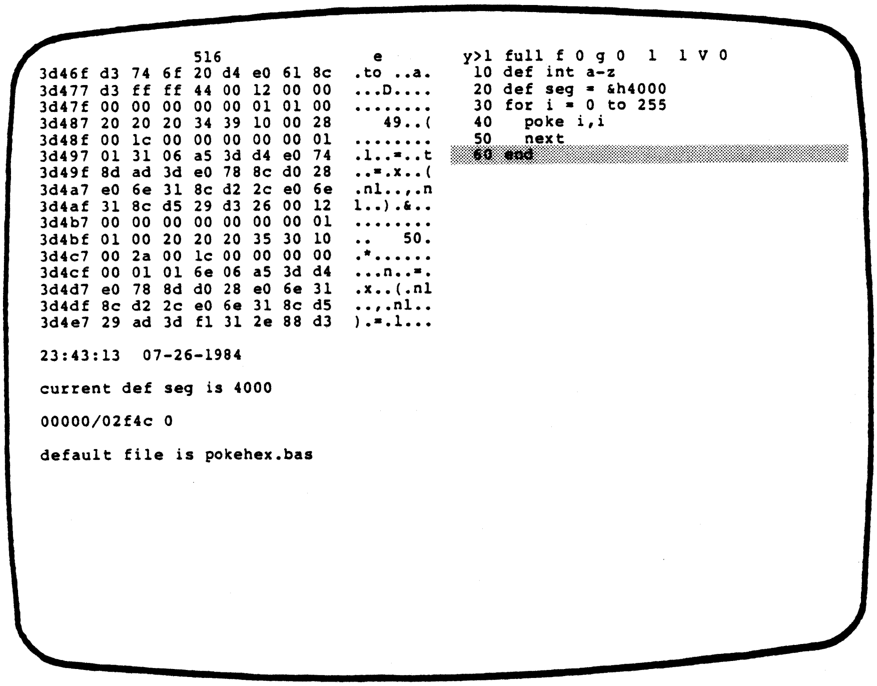
You may toggle between seeing the data in the file buffers in ASCII or Hexadecimal representation with the <U> key. This



```
  80           252  000           x         i   step f l g 0 13  1 VP0
                         file #  1  buffer address 3f3f0
                                    buffer length  00200
```

```
l...5...10...15...20...25...30...35...40...45...50...55...60...65...70...75...80
abcdefghijkl   pqrstuvwxyz
```

FIGURE 12-1 Sample File I/O window (I) - DEMG.BAS prograı



```
                516                  e       y>l full f 0 g 0  l  l V 0
3d46f d3 74 6f 20 d4 e0 61 8c  .to ..a.     10 def int a-z
3d477 d3 ff ff 44 00 12 00 00  ...D....     20 def seg = &h4000
3d47f 00 00 00 00 00 01 01 00  ........     30 for i = 0 to 255
3d487 20 20 20 34 39 10 00 28     49..(     40   poke i,i
3d48f 00 1c 00 00 00 00 00 01  ........     50   next
3d497 01 31 06 a5 3d d4 e0 74  .l..=..t     60 end
3d49f 8d ad 3d e0 78 8c d0 28  ..=.x..(
3d4a7 e0 6e 31 8c d2 2c e0 6e  .nl..,.n
3d4af 31 8c d5 29 d3 26 00 12  l..).&..
3d4b7 00 00 00 00 00 00 00 01  ........
3d4bf 01 00 20 20 20 35 30 10  ..   50.
3d4c7 00 2a 00 1c 00 00 00 00  .*......
3d4cf 00 01 01 6e 06 a5 3d d4  ...n..=.
3d4d7 e0 78 8d d0 28 e0 6e 31  .x..(.nl
3d4df 8c d2 2c e0 6e 31 8c d5  ..,.nl..
3d4e7 29 ad 3d f1 31 2e 88 d3  ).=.l...

23:43:13  07-26-1984

current def seg is 4000

00000/02f4c 0

default file is pokehex.bas
```

FIGURE 12-2 File I/O (I) and List Trace window (L)

can be valuable in cases where control characters, spaces, blanks, and binary encoded data are placed in the buffer and you want to see exactly what is there at all times.

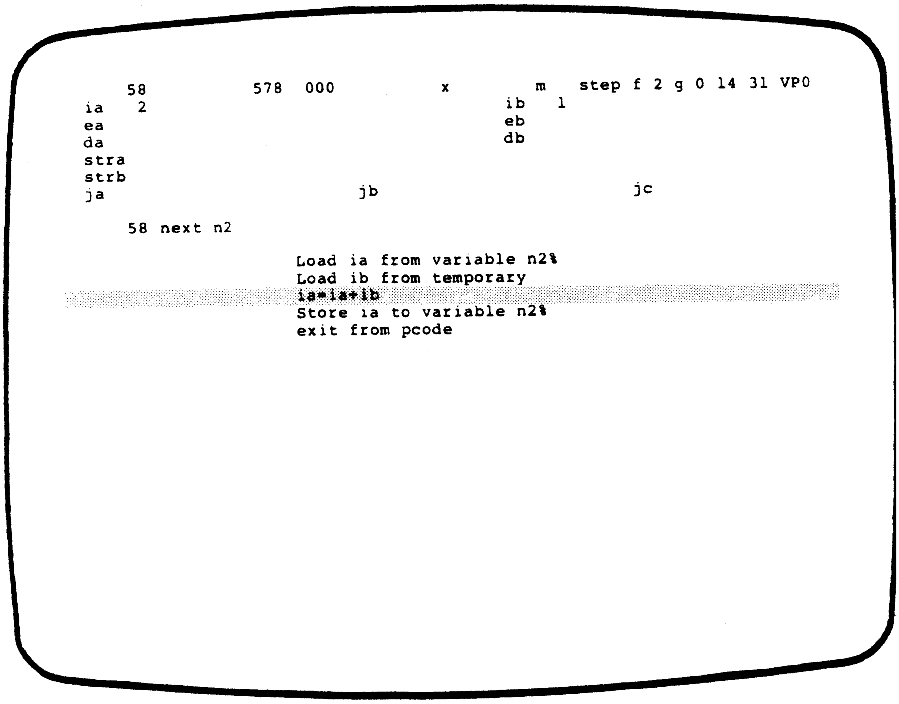
The **demg** demo program may be run to see the action in the <I> Window. Enter: **load demg** on the Command Screen and press <F2>. Then press <I> to see the buffer action as the program is running. Press <S> to split the screen, showing the I/O Window on the left and the List Trace Window on the right.

12-2 THE PSEUDO CODE WINDOW -- M KEY

The M Window shows the small steps involved in evaluating an arithmetic expression with Professional BASIC™. The Pseudo Code Window is invoked by the "M" key. This window can only be full width.

At the top of the screen, you will see 11 fields labeled. These are:

Pseudo-Code Register	Contents of Register	Normal Form
ia	Integer "A"	Decimal
ib	Integer "B"	Decimal
ea	Single Precision "A"	Decimal
eb	Single Precision "B"	Decimal
da	Double Precision "A"	Decimal
db	Double Precision "B"	Decimal
stra	String "A"	Decimal length + ASCII
strb	String "B"	Decimal length + ASCII
ja,jb,jc	Array element address	5 Hexadecimal digits



```
     58           578  000          x         m   step f 2 g 0 14 31 VP0
ia    2                                     ib   1
ea                                          eb
da                                          db
stra
strb
ja                       jb                           jc

     58 next n2

                        Load ia from variable n2%
                        Load ib from temporary
                        ia=ia+ib
                        Store ia to variable n2%
                        exit from pcode
```

FIGURE 12-3 **Pseudo code window (M) - DEMM.BAS prograı**

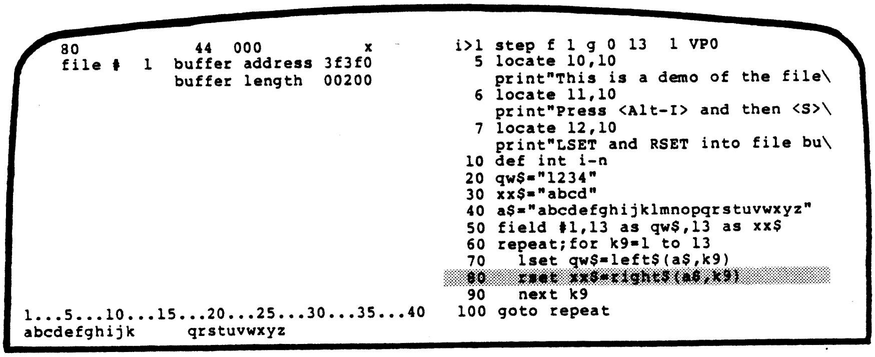
The statement to be executed is shown below the registers, and the rest of the screen is blank. At this point you could decide to switch to another screen. If you go further, you must stay and view the window for the entire statement calculation.

Press the <**space bar**> and the lower half of the screen is filled with the pseudo code associated with evaluating that expression. For each P-code a description of the operation is given.

Each single step will do a P-code operation. Note that many steps are needed to perform a complete BASIC statement. When single stepping through the pseudo code, values are loaded to registers and displayed. As they are stored or used in operations, their values will disappear. This is an attempt to make the operations clearer by only showing relevant registers. The <**K**> key may be used to display the values in all the registers.

Values are normally shown in decimal using the print format. By pressing <**U**> the values in the numeric registers may be shown as two hexadecimal words with the high order number first and the low order number second. The single precision and double precision numbers are shown in the 8087 format as 2 and 4 words with the high order word first and the low order word last. Strings are shown as a length (always in decimal) followed by a 2 digit hexadecimal number for each byte. Pressing <**U**> again toggles back to decimal representation.

The display may be run at full speed with the <**Enter**> key, or a screen switch may be made. The statement is always completed at full speed before the screen switch is done.



```
80              44  000           x          i>1 step f 1 g 0 13  1 VP0
file #  1  buffer address 3f3f0                5 locate 10,10
           buffer length  00200                  print"This is a demo of the file\
                                               6 locate 11,10
                                                 print"Press <Alt-I> and then <S>\
                                               7 locate 12,10
                                                 print"LSET and RSET into file bu\
                                              10 def int i-n
                                              20 qw$="1234"
                                              30 xx$="abcd"
                                              40 a$="abcdefghijklmnopqrstuvwxyz"
                                              50 field #1,13 as qw$,13 as xx$
                                              60 repeat;for k9=1 to 13
                                              70   lset qw$=left$(a$,k9)
                                              80   rset xx$=right$(a$,k9)
                                              90   next k9
1...5...10...15...20...25...30...35...40     100 goto repeat
abcdefghijk     qrstuvwxyz
```

FIGURE 12-4 **Split Memory (Y) window and List Trace wind (L) - POKEHEX.BAS program**

12-3 THE MEMORY DISPLAY WINDOW -- Y KEY

During the execution of a program, you can use the Memory Display Window to observe the contents of a section of memory. The wide screen displays 256 bytes and the split screen displays 128 bytes. You can change the location of the section at any time. The system date and time, default filename, and the current DEF SEG are displayed. On the bottom line there are two hexadecimal numbers separated by a slash followed by a space and then a decimal number. The first hex number is the amount of string space used by the current program. The second hex number is the amount of string space available. The last number is a decimal number representing the percentage of string space used.

To see the Memory Window, enter **load demi** and then enter **srun.** Then press the **Y** key to obtain the Memory Display Window. The first column on the display is the address of the first byte displayed on that same line. Each line displays 16 bytes, thus each address in the first column is 10 hex greater than the address above it. On the right side of each line is the ASCII character display that matches positionally each hex value on that line. If you attempt to view memory beyond the point that DOS recognizes as the end of physical memory all values will be displayed as question marks to avoid causing a PARITY CHECK 2 error.

The window is controlled with the following keys:

Home The **Home** key sets the address to zero.

+ The **Plus** key toggles the display between byte-and-character format and word format.

- The **minus** (or **dash**) key toggles the character display between showing all characters and showing only printable characters, with non-printing characters replaced by a period. The printable character values range from 20 hex to 7E hex.

0-9 The number keys are used to increment and decrement the address. On the full screen, each key is associated with a hex digit position in the full address format. Thus each hex digit position can be "scrolled" to get the desired value. Since the split screen is only half as wide as the full screen, the values are different so they can match the display change action. Keys **1** through **5** decrement the address where the **1** key is maximum decrement and the **5** key decrements by one. Keys **6** through **0** increment the address where the **6** key increments by one and the **0** key is maximum increment. Although this technique is awkward to explain with words, using it is very straightforward and easy. The following table is presented for your information, but you can happily control the display address without reference to the table. The table values are hexadecimal.

Lets do an example to watch the Memory Window work. Enter "**load pokehex**" or type in the following program.

```
10 def int a-z
20 def seg = &H4000
30 for i = 0 to 255
40   poke i,i
50 next
```

Enter "**srun**".
Press <**Y**> to get the Memory Window.
We want the window address to be hex 40000.
Press the <**zero**> key 4 times to get an address of 40000 hex.
Press <**S**> to split the screen.

Press the <**space bar**> repeatedly to step the program until you see the values being "poked" into memory.
Press <**left cursor**> to change screen control from right to left.
Press <**Enter**> to get continuous execution.
As the new characters approach the bottom of the display, use the <**seven**> key to keep the action on the display.
At the end of the program, you will be at the Control Screen.
Press <**Alt Y**> to go to the Memory Window. Use the <**four**> key to position the display to see the beginning of the "poked" characters.
Toggle the <**minus**> (or <**dash**>) key to see the two formats of character display - all characters or just printable characters.
Toggle the <+> key to see the word format.

Practice using the number keys to position the display at a specific byte such as placing the "A" character in the left hand corner.

Note the current DEF SEG value, default filename (for the LOAD and SAVE commands), date and time display, and the 3 numbers showing string space used.

SECTION III

BREAKPOINT SETTING AND ERROR MESSAGES

13. BREAKPOINT SETTING

When running a program you can tag one or more lines for the program to stop execution on. That is, by identifying a given line in the program as a break point, you can let the program run at full speed and if that line is encountered in the running of the program then execution will halt at that precise place. This can be quite valuable in program debugging when you want the program to charge ahead and execute a number of steps and then halt at a particular place so that you can view, in detail, execution of the next instructions and/or examine the status of variables. Without this facility you would have to slowly execute up to this point in single step mode or in a slow trace mode. Such a search could take a long time, plus you might overshoot the mark and miss what you wanted to see.

Breakpoints remain set until you explicitly remove them.

You can set a breakpoint in either of two ways:

1. On the Command Screen you may enter the command: **break n** where **n** is a line number or a label. Multiple breakpoints may be set with a series of **break n** commands. One line is specified with each "**break**" command. To remove a breakpoint use a "**nobreak n**" command. The "**nobreakall**" command removes all breakpoints.

2. When on the List Trace Window you can toggle a breakpoint on or off by pressing the <Z> key while the inverse video bar is on the line. An asterisk is placed between the line number and the first character on the line to indicate a breakpoint. If a "**break n**" command was used to set a breakpoint, the asterisk is also shown.

With the List Trace Window displayed you can scroll up and down the listing and see the lines where breakpoints are set. Note that you set breakpoints on a program line, not an individual instruction on a line (where there are multiple instructions on the line).

Setting one or more breakpoints does not affect the speed of program execution.

When the program encounters a breakpoint and the Print Screen (normal output) is displayed, the system switches to the list trace screen and is halted on the instruction where the break was set.

When you are in the tracing system in "FULL" mode and a breakpoint is encountered, it is as if the <**space bar**> were pressed to put the system into "STEP" mode. The program is not suspended from execution in this case. You may continue execution by pressing the <**space bar**> or <**Enter**>.

The commands in summary are:

break n	set a breakpoint on line **n**
break label	set a breakpoint on the line with the designated label

nobreak n	remove the breakpoin on line **n**
nobreak label	remove a breakpoint on the line with the designated label
nobreakall	remove all breakpoints

14. ERROR MESSAGES

14-1 SYNTAX ERRORS

Syntax error messages are handled by the "Try abc..." prompt instead of numbered, fixed messages. The Dynamic Syntax Checker™ catches syntax errors as they occur. After the second attempt to input an invalid character from the keyboard, the Dynamic Syntax Checker™ presents a list of valid characters to try. Thus, this prompt is called the "try" line. See Chapter 5 for more on the Dynamic Syntax Checker™. Syntax errors in externally created programs are checked as the program is loaded into Professional BASIC™. See Chapters 5 and 6 for more details.

14-2 RUN TIME ERRORS

All other errors are caught during the semi-compile or during the execution of the program. Those errors caused by temporary current equipment conditions are handled by Professional BASIC™ with a windowed error message that allows the user to correct the condition and continue the program. Examples of this type error are: "printer not ready", "drive not ready", and "write protected disk". Most other types of errors are caught during the semi-compile before execution begins.

14-3 LISTING OF ERRORS

There are three listings of error messages that follow. The first list contains those errors that can occur during the semi-compile. The second and third lists are errors that can occur during run time and therefore can be trapped by the ON ERROR statement. Professional BASIC™ provides an extended error number and literal in addition to the ERR and ERL values. ERR corresponds to the error numbers in PC BASIC. ERR2 is a special variable that is the Professional BASIC™ extended error code. For example: when ERR equals 9, ERR2 could be 901 or 902. This extended error code allows clearer definition of errors while maintaining consistency and compatibility with PC BASIC error code numbers. In addition, if you want to display the Professional BASIC™ error message instead of coding your own message, the special variable ERR$ is a string that contains the same error message that would have been displayed by the system if the ON ERROR statement was not being used. Thus on an error condition, the MicroSoft BASIC error number, the extended Professional BASIC™ error number, the error line, and the error message are available for your use.

14-4 SEMI-COMPILE ERROR MESSAGES

Since these error messages are displayed only during a semi-compile and will not be trapped by the ON ERROR statement, they do not have error numbers.

Attempt to set the character,[char] ,to ,[type], it is already set to ,[type].

There is no room to build the address table for resolving goto targets.

The transfer to (line number, or label), is invalid.

The two variables in a swap are not the same type.

There is no program.

A function has more than 8 arguments.

A number is trying to be assigned to a string variable.

A multi-_argument function name is not recognized or has an argument of the wrong type.

A multi-argument function was referenced with no arguments.

A multi-argument user-defined function has been defined more than once.

A multi-argument user-defined function is not found.

A function with no arguments was referenced as one with arguments.

A user-defined function with no arguments was defined more than once.

A user-defined function with no arguments was not found.

A single argument function name is not recognized or has an argument of the wrong type.

A string is trying to be assigned to a numeric variable.

An array reference has a string as a subscript.

An array reference has the wrong number of subscripts.

An operator has mismatched operands, one numeric and the other a string.

Argument duplicated in DEF FN().

Assignment with that special variable is invalid.

Expression should be a string.

FIELD variable is not a string.

More than 8 arguments in DEF FN().

Not found.

Illegal function reference, or array not dimensioned.

LINE INPUT variable is not a string.

LSET or RSET variable should be a string.

That transfer is not valid.

The 1st argument in a MID$()= is not a variable.

The 1st argument in a MID$()= is not suitable.

The expression is too complex, please simplify.

The index must be the same in both FOR and NEXT.

The label,labelname,is doubly defined.

The label,labelname,is not defined.

The last clause in this IF has an incomplete loop.

The name has already been dimensioned.

The substitution of the user defined function with no arguments caused the expression to grow too large.

There is a number where there should be a string.

There is a string where there should be a number.

There is an unknown function on the left-hand side of an assignment.

There is no match for this NEXT.

There is no match for this WEND.

These loops are not completed.

There is no program.

There is no room to build the address table for resolving goto targets.

There is no room to store a descriptor for (variable).

The transfer to (line number,or label), is invalid.

This EXITFOR was not contained in a FOR loop.

This EXITWHILE was not contained in a WHILE loop.

This loop is not completed.

This loop is not paired correctly.

14-5 NUMERICALLY LISTED ERROR CODES

MS Code	PB Code	Message
2	200	Data read is of wrong type.
3	301	RETURN without GOSUB.
4	401	READ has run out of data.
5	501	The index for an ON ... GOTO is negative.
	502	The index for an ON ... GOSUB is negative.
	503	Too deep in GOSUBs.
	504	LOCATE row out of range.
	505	LOCATE column out of range.
	506	COLOR foreground out of range.
	507	COLOR background out of range.
	508	PRINT TAB() value out of range.
	509	Unrecognized calculated error.

MS Code	PB Code	Message
5	510	PRINT SPC() value out of range.
	511	RESTORE, but no DATA statement(s).
	512	FOR - too deep in loops.
	513	Float $ + exp notation is illegal.
	514	Float $ + negative number is illegal.
	515	PRINT USING internal string overflow.
	516	No data for the PRINT USING statement.
	517	No characters in format string for PRINT USING statement.
	518	No format field defined in PRINT USING statement.
	519	No numeric field defined in format for PRINT USING statement.
	520	No string field defined in format for PRINT USING statement.
	521	The field width is out of range.
	522	The field has exceeded the buffer size.
	523	DEF SEG value out of range.
	524	POKE address is out of range.
	525	POKE value is out of range.
	526	In an INPUT # field, a number in a numeric field is too large.
	527	In an INPUT # field, there is a CR inside a quoted string.
	528	A numeric field has more than 24 digits.
	529	Scientific notation double precision number with single precision field - ^^^^.

MS Code	PB Code	Message
5	530	Scientific notation single precision number with double precision field - ^^^^^.
	531	WIDTH for lpt1: is out of range.
	532	OUT address is out of range.
	533	OUT value is out of range.
	534	WAIT port value is out of range.
	535	WAIT AND value is out of range.
	536	WAIT XOR value is out of range.
	537	Invalid transfer via RETURN nn.
	538	The filespec in a CHAIN statement is not valid.
	539	The transfer line no. in a CHAIN statement is not valid.
	540	RESUME nn transfer is invalid.
	541	RESUME next climbs into a loop.
	542	Unknown command.
	543	Bad request structure length.
	544	Seek error.
	545	Unknown media type.
	546	Sector not found.
	547	Calculated error out of range.
	549	OPEN record length is out of range.
	560	SCREEN mode value out of range.
	561	SCREEN apage value out of range.
	562	SCREEN vpage value out of range.
	563	Attempt to switch mono screen to 40 col. or graphics.
	564	WIDTH value is out of range [108].
	565	Attempt to do graphics in non graphics mode.

MS Code	PB Code	Message
5	566	The value of color is out of range.
	567	Not in graphics mode for POINT function.
6	601	Overflow.
	604	WHILE - too deep in loops.
9	901	Subscript is higher than bound.
	902	Subscript is lower than bound.
13	1301	Function has signaled an error.
14	1401	Run out of string space.
20	2001	RESUME encountered without an error.
24	2401	Device not ready.
25	2501	Write fault.
	2502	Read fault.
	2503	General failure.
	2504	Printer problem.
26	2602	Attempt to start a FOR loop with an index already in use.
27	2701	Printer out of paper.

MS Code	PB Code	Message
	5201	WRITE # file number is out of range.
	5202	PRINT # USING file number is out of range.
	5203	PRINT # file number is out of range.
	5204	GET/PUT file number is out of range.
	5205	OPEN file number is out of range.
	5206	CLOSE file number is out of range.
	5207	INPUT # file number is out of range.
	5208	The file number in a FIELD statement is out of bounds.
53	5301	The file cannot be found on this disk(ette) or directory.
	5302	OPEN file specification string is too long.
	5303	OPEN file specification string has an imbedded zero byte.
54	5401	OPEN file mode expression is not one of I, O, R or A.
	5402	The file is a READ ONLY file. It cannot be opened as Output, Append or Random.
	5403	Cannot read from an OUTPUT or APPEND or WRITE ONLY file.
	5404	Access to this file is denied because of it's file mode.
	5405	The file cannot be accessed because it is not open.
55	5501	The file cannot be opened because it is already open.

MS Code	PB Code	Message
58	5801	Cannot delete because a file by this name is open.
	5802	A file by this name already exists.
61	6101	Less data was written to the disk(ette) than was requested. Check for possible disk(ette) full condition.
62	6201	An attempt has been made to read beyond the end of the file.
63	6301	The record number requested is zero or negative.
67	6701	Exceeds maximum open files allowed by PC DOS.
	6702	Cannot create new file. Check for directory full.
	6703	Exceeds maximum open files allowed by Professional Basic.
68	6801	Unknown device.
70	7001	Attempt to write on write protected diskette.
72	7201	Data error (crc).
74	7401	Device and directory must be the same for both names.
75	7501	An attempt has been made to write to an INPUT or READ ONLY file.

MS Code	PB Code	Message
75	7502	The directory cannot be deleted because it is not empty.
	7503	The pathname was not found.
76	7601	A PC DOS error # nn has occured.

14-6 ALPHABETICAL LISTING OF ERROR MESSAGES

MS Code	PB Code	Message
76	7601	A PC DOS error # nn has occured.
58	5802	A file by this name already exists.
5	528	A numeric field has more than 24 digits.
54	5404	Access to this file is denied because of it's file mode.
62	6201	An attempt has been made to read beyond the end of the file.
75	7501	An attempt has been made to write to an INPUT or READ ONLY file.
5	565	Attempt to do graphics in non graphics mode.
26	2602	Attempt to start a FOR loop with an index already in use.
5	563	Attempt to switch mono screen to 40 col. or graphics.
70	7001	Attempt to write on write protected diskette.
5	543	Bad request structure length.
5	547	Calculated error out of range.

MS Code	PB Code	Message
67	6702	Cannot create new file. Check for directory full.
58	5801	Cannot delete because a file by this name is open.
54	5403	Cannot read from an OUTPUT or APPEND or WRITE ONLY file.
52	5206	CLOSE file number is out of range.
5	507	COLOR background out of range.
5	506	COLOR foreground out of range.
72	7201	Data error (crc).
2	200	Data read is of wrong type.
5	523	DEF SEG value out of range.
74	7401	Device and directory must be the same for both names.
24	2401	Device not ready.
67	6703	Exceeds maximum open files allowed by Professional Basic.
67	6701	Exceeds maximum open files allowed by PC DOS.
5	513	Float $ + exp notation is illegal.
5	514	Float $ + negative number is illegal.
5	512	FOR - too deep in loops.
13	1301	Function has signaled an error.
25	2503	General failure.
52	5204	GET/PUT file number is out of range.
5	526	In an INPUT # field, a number in a numeric field is too large.

MS Code	PB Code	Message
5	527	In an INPUT # field, there is a CR inside a quoted string.
52	5207	INPUT # file number is out of range.
5	537	Invalid transfer via RETURN nn.
61	6101	Less data was written to the disk(ette) than was requested. Check for possible disk(ette) full condition.
5	505	LOCATE column out of range.
5	504	LOCATE row out of range.
5	517	No characters in format string for PRINT USING statement.
5	516	No data for the PRINT USING statement.
5	518	No format field defined in PRINT USING statement.
5	519	No numeric field defined in format for PRINT USING statement.
5	520	No string field defined in format for PRINT USING statement.
5	567	Not in graphics mode for POINT function.
54	5401	OPEN file mode expression is not one of I, O, R or A.
52	5205	OPEN file number is out of range.
53	5302	OPEN file specification string is too long.
53	5303	OPEN file specification string has an imbedded zero byte.

MS Code	PB Code	Message
5	549	OPEN record length is out of range.
5	532	OUT address is out of range.
5	533	OUT value is out of range.
6	601	Overflow.
5	524	POKE address is out of range.
5	525	POKE value is out of range.
52	5202	PRINT # USING file number is out of range.
52	5203	PRINT # file number is out of range.
5	510	PRINT SPC() value out of range.
5	508	PRINT TAB() value out of range.
5	515	PRINT USING internal string overflow.
27	2701	Printer out of paper.
25	2504	Printer problem.
25	2502	Read fault.
4	401	READ has run out of data.
5	511	RESTORE, but no DATA statement(s).
20	2001	RESUME encountered without an error.
5	541	RESUME next climbs into a loop.
5	540	RESUME nn transfer is invalid.
3	301	RETURN without GOSUB.
14	1401	Run out of string space.
5	529	Scientific notation double precision number with single precision field - ^^^^.
5	530	Scientific notation single precision number with double precision field - ^^^^^.

MS Code	PB Code	Message
5	561	SCREEN apage value out of range.
5	560	SCREEN mode value out of range.
5	562	SCREEN vpage value out of range.
5	546	Sector not found.
5	544	Seek error.
9	901	Subscript is higher than bound.
9	902	Subscript is lower than bound.
75	7502	The directory cannot be deleted because it is not empty.
5	522	The field has exceeded the buffer size.
5	521	The field width is out of range.
54	5405	The file cannot be accessed because it is not open.
53	5301	The file cannot be found on this disk(ette) or directory.
55	5501	The file cannot be opened because it is already open.
54	5402	The file is a READ ONLY file. It cannot be opened as Output, Append or Random.
52	5208	The file number in a FIELD statement is out of bounds.
5	538	The filespec in a CHAIN statement is not valid.
5	501	The index for an ON ... GOTO is negative.
5	502	The index for an ON ... GOSUB is negative.
75	7503	The pathname was not found.

MS Code	PB Code	Message
63	6301	The record number requested is zero or negative.
5	539	The transfer line no. in a CHAIN statement is not valid.
5	566	The value of color is out of range.
5	503	Too deep in GOSUBs.
5	542	Unknown command.
68	6801	Unknown device.
5	545	Unknown media type.
5	509	Unrecognized calculated error.
5	535	WAIT AND value is out of range.
5	536	WAIT XOR value is out of range.
5	534	WAIT port value is out of range.
6	604	WHILE - too deep in loops.
5	531	WIDTH for lpt1: is out of range.
5	564	WIDTH value is out of range [108].
52	5201	WRITE # file number is out of range.
25	2501	Write fault.

APPENDICES

A.

Professional BASIC DIFFERENCES

1. IF...THEN...ELSE

You cannot have a conditional NEXT statement. The following line is not valid:

```
20 FOR I=1 to A: IF I=5 THEN GOTO 10 ELSE NEXT I
```

2. FOR...NEXT

a. There must be one and only one NEXT for each FOR statement. The following is not valid:

```
10 FOR I = 1 TO N
20 FOR J = 1 TO M
30 A(I,J) = I*J
40 NEXT
```

Line 40 may read, however: 40 NEXT J,I

You may not have a NEXT statement within an IF...THEN...ELSE construct where the NEXT is executed only if a condition is met.

b. You may not have a GOTO or RETURN n which enters into the middle of a FOR...NEXT loop from outside the loop. Returning to a call from a GOSUB is valid however.

3. Arrays and DIM statements

a. All arrays must always be dimensioned, even if there are 10 or fewer elements.

b. Only integer constants (integers can range up to 2,000,000,000+ not just 32,267) may be used in a dimension statement. Variables are not allowed. DIM A(100000) is acceptable but DIM A(n) is not.

c. Arrays can only be dimensioned once and may not be redimensioned during program execution.

d. An alternate form for dimensioning arrays is

```
DIM A(1901 to 2000)
```

This will set up an array A(i) as a 100 element array. Only array element references in the range specified are valid. You may also have negative references if you define a dimension such as:

```
DIM A(-100 to 100)
```

Thus:

```
10 I = -50
20 A(I) = I + I
30 PRINT A(-50)
```

would be a valid set of program lines which would assign -100 to the 51st element of the array and then print -100 on the screen.

e. An array cannot be used as a field name in a FIELD statement. Hence,

```
FIELD #1, 20 as A$(1), 20 as A$(2), 5 as A$(3)
```

is not valid.

f. Remember that the only restriction on array size is the amount of available RAM.

4. Line Labels

A program line may begin with a name or label, such as:

```
10 Start_initial_procedure; rem
```

The label can be any length (up to more than 300 characters), have upper or lower case letters and use an underscore or period to separate words. However, there cannot be any blank spaces in the middle of the label.

You may have GOTO and GOSUB statements which refer to this line by the label instead of the line number. In general, anywhere you would normally use a line number in a program, you can use a label instead. It is also possible to use combinations of labels and line numbers in the same program.

The @ key may be used to finish typing already defined labels, just as with a long variable name.

SORTL will list all labels defined in a program and the FIND command will show where in the program they are used and referenced.

5. Line Numbers

a. Program line number 0 is not allowed.

b. Line numbers may range from 1 to 99999.

c. It is possible to load, save, and merge files without line numbers. To load a file without line numbers be certain that column one in the program file is a blank. To save a program without line numbers, use the SAVEU command.

6. Program storage

Programs are saved and read-in in ASCII format only. A file from BASICA must have been saved with the ,A option before it can be read into Professional BASIC™.

7. LOAD

In immediate mode you do not need to use quotes around a filespec.

```
LOAD DEMA
```

will load the program DEMA.BAS for example. and RUN DEMA will load and run this program. Quotes are required around a filespec only when it is being referenced in a program.

8. MKS$ and MKD$ - Random Access

The MKS$ and MKD$ functions will save single and double precision numbers in 4 or 8 bytes using the standard IEEE format, not the Microsoft format.

Integers are saved the same in both systems, except that you may set aside either 2 or 4 bytes in a file field, depending on whether you want to store single precision integers or double precision integers. Professional BASIC™ always works with integers as double precision, except when the MKI$ function is used to store an integer, if a FIELD statement has set aside 2 bytes, not 4.

9. Cursor

Cursor size cannot be set, nor can the cursor be explicitly turned on or off. Professional BASIC™ turns the cursor off unless an INPUT or INKEY$ statement is soliciting input from the user. The third, fourth, and fifth parameters in a LOCATE statement are ignored.

10. Function Keys

a. The 25th line on the screen never displays function key values.

b. The function key values are always set by Professional BASIC™ unless an application is running. Unlike the function keys under BASICA, function keys under Professional BASIC™ will not retain the last set of values assigned to them.

c. A running program may assign a string of up to 31 characters to any function key instead of only 15 characters.

11. BASIC Keywords

Under Professional BASIC™ keywords may be used as parts of variable and label names. As a result, it is usually necessary to put spaces between key words and other characters. For instance:

GOTO125 is not acceptable but GOTO 125 is.

12. Integer Values

Integers can range from -2,147,483,648 to 2,147,483,467.

13. Single and Double Precision Values

Single and double precision numbers can have exponents from -308 to 308. Hence 15e300 is a valid number. Do not use more than 7 digits in a single precision number. If you want to have more significant digits to work with a value such as 1.23456789 x 10^{200}, then enter 1.23456789d200 (i.e. use "d" instead of "e" before the exponent).

14. Entering Scientific and Exponential Notation

a. The number 10e40 is not valid since it represents an integer "10" raised to a power exceeding the value of integers. Enter 10.0e40 instead.

b. 10^-3 should be entered as 10^(-3). Two operators cannot be together.

15. SCREEN statement

You can access 4 screens in text mode on a monochrome screen similar to the way multiple screens (4 or 8) can be written to and switched among with a graphics adapter card. 16k of RAM memory is used for these screens, instead of using the RAM on the graphics card. Thus this multiple screen option is available regardless which screen adapter card is installed.

16. CIRCLE

If you want to specify the aspect ratio but leave out the "color", "start", and "stop" parameters, do not put in commas such as:

```
CIRCLE (100,100),3,,,,5/4
```

instead enter:

```
CIRCLE (100,100),3,7,0,0,5/4
```

17. CHAIN

The ALL, DELETE n-m, and line number to start on paramters may be entered in any order. If one is omitted, do not put in an extra comma.

18. Arithmetic

a. All single and double precision calculations in Professional BASIC™ are performed either by the 8087 numeric coprocessor or an emulation of the 8087 in 80 bits. Thus the system insures a high degree of accuracy. Integer arithmetic is never done in the 8087, thus the speed of integer calculations is the same for both pb.exe and pb8.exe. All arithmetic in both pb.exe and pb8.exe is done in standard binary floating point format.

b. PBD.EXE, the optional BCD version of Professional BASIC™ requires the 8087 numeric coprocessor. Since the arithmetic here is direct Binary Coded Decimal, you will not find the typical round off error standard with the binary floating point arithmetic versions. If you have purchased the 8087/80287 support package, please refer to Appendix E for more information.

19. Error Handling

In addition to the standard error handling facilities available in Microsoft BASIC, Professional BASIC™ offers you extended features. In addition to the ERR value Professional BASIC™ provides an ERR2 value that includes the Microsoft error code and a two digit extension for more accurate error handling. For instance, when ERR = 5, ERR2 could be any one of 54 values between 501 and 567, two examples could be: when ERR2 = 518 the error message is "No format field defined in PRINT USING statement", or when ERR2 = 565 then the error message would read "Attempt to do graphics in non graphics mode". The Professional BASIC™ error messages are stored in a string named ERR$, if you would prefer to use them instead of writing your own. Please refer to chapter 14 for more information.

B. IMMEDIATE MODE COMMANDS

The following commands are available from the immediate mode:

ABS	HEX$	NEW
ASC	INP	OCT$
ATN	INSTR	OUT
CDBL	INT	PEEK
CHDIR	KILL	RENUM
CHR$	LEFT$	RIGHT$
CINT	LEN	RMDIR
CLOSE	LET	RND
CLS	LIST	RUN
CONT	LLIST	SAVE
COS	LOAD	SGN
CSNG	LOC	SIN
DATE$	LOF	SQR
DELETE	LOG	STR$
EDIT	LPOS	STRING$
EOF	LPRINT	SYSTEM
EXP	MERGE	TAN
FILES	MID$	TIME$
FIX	MKDIR	TIMER
FRE	NAME	VAL

C.
ACCURACY AND SPEED

The arithmetic in Professional BASIC™ is based on the 8087 numeric coprocessor. To make Professional BASIC™ available to users without the 8087, a module has been built which needs only the 8088 processor. To be compatible with the 8087, however, this new module must work internally in a similar fashion; thus it is much slower than an arithmetic module without this constraint. The version of Professional BASIC™ using the 8088 for arithmetic is called PB8.

The 8087 transforms all data types to an 80 bit internal format. Single precision numbers are expanded from their 32 bit format to the 80 bit format; and results are also in this 80 bit format. If a result is to be stored in single precision, the number must first be transformed back to the 32 bit format.

The functions for the 8088-only version software arithmetic have been biased toward accuracy rather than speed. If speed is needed, then the 8087 should be installed and used to give both speed and accuracy.

A severe test of the math functions was published in the September, 1983 issue of "Dr. Dobbs Journal" and reprinted in the March, 1984 issue of "Personal Computer Age". It is:

```
10      'Time and General Accuracy Test
        Program
20      defint i
30      iloop = 2500
40      a=1
50      for i=1 to iloop-1
60      a=tan(atn(exp(log(sqr(a*a)))))+1
70      next i
80      print using "a=####.####";a
90      stop
```

The results for PB and PB8 are:

Mode	Time(secs.)	Result	Time ratio to PB
PB Single Precision	15.93	2500.0000	1
PB Double Precision	15.78	2500.0000	1
BASICA Single Precision	156.98	2179.8470	9.85
BASICA Double Precision	160.28	2179.8464	10.16
PB8 Single Precision	1012.83	2500.0000	63.58
PB8 Double Precision	1028.26	2500.0001	65.16

(Note: Both versions of PB use the 8088 for integer add, subtract, and multiply)

The correct result for this test is 2500. Professional BASIC™ gives this result in all cases.

The times for the other operations are:

	PB Time (ms)	PB8 Time	Ratio
Int divide	.99	23.90	24.05
Sng addition	.29	1.49	5.12
multiplication	.29	2.05	7.07
division	.32	18.35	7.79
Dbl addition	.81	4.09	4.99
multiplication	.82	4.68	5.71
division	.44	19.01	43.14
Rnd function	.55	82.89	151.57

Both PB and PB8 are accurate to the full precision for the add, subtract, multiply and divide operations. PB gives 16 digit accurate results for the functions such as Log. PB8 is designed to give results for the functions accurate to 8 digits. If better speed or accuracy are needed, the 8087 version is available and should be used.

Lastly, the times for the sieve benchmark are given.

Mode	Time (secs.)	Ratio to PB
PB int	109.47	1
sng	124.84	1
dbl	137.80	1
PB8 int	109.47	1
sng	278.20	2.23
dbl	330.17	2.40
BASICA int	191.63	1.75
sng	233	1.87
dbl	(program will not run in the 64kb limit of BASICA)	

Sieve Benchmark Program

```
 10       defint a-z
 20       dim v(0 to 8190)
 30       Start;print
          date$,time;:timmy!=timer
 40       size=8190
 50       count=0
 60       for i=0 to size
 70       v(i)=1
 80       next i
 90       for i=
100       if v(i)=0 then goto 1180
110       prime=i+i+3
130       1130;if k>size then goto 1170
140       v(k)=0
150       k=k+prime
160       goto 1130
170       1170;count=count+1
180       1180;next i
190       print timer-timmy!;"    seconds"
200       goto Start
```

D. ENTERING ASCII VALUES

Non-keyboard characters with any ASCII value from 1 to 255 can be used with Professional BASIC™. Most characters (see list on next page) can be loaded from the disk or entered via the keyboard. Programs with these characters can also be saved to disk.

To view and enter these characters, begin by holding down the <**Alt**> key and pressing the <**Ins**> key. The line above the cursor will be replaced by a menu line with special characters. The character in inverse video is the "currently selected character". You can find the decimal ASCII value of this character in the top right hand corner of your screen.

The character in inverse video may be changed one character to the right or left by using the <**left cursor**> or <**right cursor**> keys. The <**Home**> key selects the lowest character (ASCII value 001) and the <**End**> key selects the highest character (ASCII value 255). You may also use the <**4**>, <**5**>, <**6**>, and <**7**> keys (in the same fashion as on the variable screen) to scroll left 1 or 10 characters (<**5**> or <**4**> key) or to scroll right 1 or 10 characters (<**6**> or <**7**> key). To insert the selected character at the cursor position, continue to hold down the <**Alt**> key while striking the <**Ins**> key again. If you wish to repeat the character, continue to press the <**Ins**> key. To return to normal operation, let up on the <**Alt**> key.

The following characters are ones which you may not use. This is not an exhaustive list, there may be other exceptions in certain applications.

1.	009	**horizontal tab** - This character is always interpeted as a horizontal tab.
2.	013	**carriage return** - It is always interpreted as a carriage return.
3.	026	This character can be used, but it also acts as an End of File mark when it is written out to disk and then read back in.

INDEX

INDEX