# **User's Guide**



# Borland<sup>®</sup> Turbo Debugger<sup>®</sup>GX for 05/2<sup>®</sup>

# **User's Guide**

# Borland® **Turbo Debugger® GX for OS/2**® Version 1.5

Borland may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1987, 1994 by Borland International. All rights reserved. All Borland product names are trademarks or registered trademarks of Borland International, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

### Borland International, Inc.

100 Borland Way, Scotts Valley, CA 95066-3249

PRINTED IN THE UNITED STATES OF AMERICA

1E0R0294 9495969798-987654321 H1

# Contents

Introduction	1
Hardware and software requirements	. 1
Differences between Turbo Debugger GX and	
Turbo Debugger	. 2
Files distributed with Turbo Debugger GX	. 3
Program files	. 3
Online text files	. 3
The README.TD file	. 3
The MANUAL.TD file	. 4
The UTILS.TD file	. 4
Sample programs	.4
TDDEMO	.4
TDDEMOPM	. 4
Typefaces, icons, and conventions	. 5
Using this manual	. 6
Chanter 1 Getting started	7
Installing Turbo Debugger CX	7
Entering and exiting Turbo Debugger GX	. /
Using command-line ontions	.0
Using the Help system	10
Accessing the Control Panel Help menu	11
Using a Help window	11
Displaying the Contents panel	12
Displaying the Help index	12
Getting context-sensitive Help	12
Printing Help information	13
Chapter 2 The Turbo Debugger GX environment	15
What Turbo Debugger GX can do for you	15
What Turbo Debugger GX won't do	16
How Turbo Debugger GX does it	17
The Turbo Debugger GX environment	17
Using the Control Panel	17
The menu bar	19
The SpeedBar	19
The Threads pane	20
The status line	21
Using dialog boxes	21

Working with views Local menus List views and Detail views The views The Breakpoint view The Datapoint view The Datapoint view The Datapoint view The C++ exceptionpoint view The C++ exceptionpoint view The Messagepoint view The Messagepoint view The Source view The Disassembly view The Disassembly view The Disassembly view The Modules view The Modules view The Inspector view The Inspector view The Variable view The Variable view The Watch view The Heap view	$\begin{array}{c} 22\\ 22\\ 24\\ 25\\ 26\\ 27\\ 28\\ 30\\ 31\\ 32\\ 33\\ 34\\ 35\\ 36\\ 36\\ 36\\ 36\\ 36\\ 36\\ 36\\ 36\\ 36\\ 36$
The Numeric Processor view The Register view	38 38
The C++ exception stack view	39
The Log view	39 40
Chapter 3 A guick example	41
The demo program Using TDDEMO Setting breakpoints Using watches	41 43 44 45
Examining simple C data objects Examining compound data objects Changing data values Conclusion	46 47 47 48
Appendix A Turbo Debugger GX for experienced	
Turbo Debugger users	51
Index	55

# **Tables**

1.1	Turbo Debugger GX command-line options	. 9
1.2	The Help menu	11
2.1	Menu bar choices	19

2.2 SpeedBar buttons20A.1 Turbo Debugger GX task list51

# Figures

1.1	Control Panel view
2.1	Control Panel view
2.2	Properties dialog box 21
2.3	Source view local menu
2.4	Breakpoint detail view
2.5	Breakpoint List view
2.6	The Source view
2.7	The five panes of the Disassembly view 32
2.8	The Memory view

3.1	Debugger views after loading TDDEMO 42
3.2	Program stops on return from
	function showargs 44
3.3	A breakpoint set at line 45 45
3.4	A variable in the Watch view 46
3.5	An Inspector window 46
3.6	Inspecting a structure
3.7	The Change Value dialog box
3.8	The Evaluator view

iv

# Introduction

Turbo Debugger GX is a state-of-the-art, source-level debugger with a graphical user interface (GUI). It's designed for programmers using Borland C/C++ and Turbo Assembler to produce programs that run under OS/2. Multiple views with pop-up menus, a SpeedBar displaying buttons for common actions, and integrated window management provide a fast, interactive environment. An online context-sensitive Help system provides you with help during all phases of operation.

Here are some of Turbo Debugger GX's features:

- Full Borland C, C++, and TASM expression evaluation
- Extensive set of views to support all levels of debugging
- Window management facility
- Comprehensive Online Help
- Control Panel with both menu and SpeedBar access to debugging commands
- High-level and low-level code access
- Logging facility
- Powerful control point facility that supports breakpoints, datapoints, messagepoints, and exceptionpoints
- Support for debugging multithreaded applications
- Special tools for debugging of Presentation Manager programs
- Support for hardware debugging registers

### Hardware and software requirements

Turbo Debugger GX runs on any IBM PC-compatible computer that has OS/2 version 2.0 or higher installed. A mouse is recommended. To see the amount of hard-disk space required for Turbo Debugger GX, run the Borland C++ installation program.



Turbo Debugger GX doesn't require a numeric processor chip.

Turbo Debugger GX works with Borland C++ for OS/2 and Turbo Assembler for OS/2. If you want to do source debugging, your application file must be either an executable (.EXE file) or a dynamic-link library (DLL) compiled with full debugging information turned on.

- When you run Turbo Debugger GX, you'll need your application's .EXE file and original source files. Turbo Debugger GX searches for source files in the following places in this order:
  - 1. In the directories specified in the File | Properties dialog box
  - 2. In the directory containing the .EXE file
  - 3. In the current directory

You can override the File | Properties setting by starting Turbo Debugger GX with the **-s** option, which specifies the source directories. (See page 8 for more information on command-line options.)

## Differences between Turbo Debugger GX and Turbo Debugger

Turbo Debugger GX works similarly to Turbo Debugger for DOS and Turbo Debugger for Windows. You'll find that many of the views, commands, and keystrokes you're accustomed to with the DOS or Windows debugger work with the OS/2 debugger. You'll also find that local menus are accessible from the views in much the same way (by rightclicking or pressing *Ctrl+F10*).

There are differences in functionality, some of which are due to the OS/2 environment. They include the following differences:

- A graphical user interface that includes a *SpeedBar*, a series of buttons you can select to perform common functions, like running, stopping, reloading, or stepping. Because the debugger is always in graphics mode, you can see your application running in another window instead of having to switch between the full-screen debugger and the application.
- Dialog boxes that aren't modal. A nonmodal dialog box is like any other window: it stays around until you close or minimize it, and you can switch to another window while the dialog box is displayed. You press *Enter* in a nonmodal dialog box to get text entries to take effect. Radiobutton and check-box selections take effect immediately.
- Windows that can move anywhere on the screen and resize to the full screen size. Each view has its own window, and there's a separate window called the Control Panel for the main menu and the SpeedBar.
- The ability to duplicate any view by choosing New View from the view's local menu. For example, you can open multiple Source views and look at more than one module or DLL at the same time, as long as the module or DLL is used by the currently loaded process.

See Appendix A for detailed information comparing various tasks you can perform with Turbo Debugger and Turbo Debugger GX. Dual form views that show a list of items or the details on one item. Using the detail form of the view, you can set all the options for an item from the list. The views that can switch between list form and detail form are the Breakpoint view, the Datapoint view, the Messagepoint view, the Exceptionpoint view, the Variable view, and the Watch view.

There are also many similarities in functionality. You'll find that many of the shortcut keys are the same, and most of the views between Turbo Debugger and Turbo Debugger GX will be familiar. The table in Appendix A lists some typical tasks and shows how to do them with both products.

### Files distributed with Turbo Debugger GX

The Turbo Debugger GX part of the Borland C++ package includes this manual and a set of files on disk. The files include

- The files needed to run the program (TD.EXE, associated DLLs, and a Help file)
- Online text files
- Utility program
- Sample program files

The installation program (described on page 7) copies these files into various default directories on your hard drive. (You can specify different directories during installation.) For a complete list of files associated with Turbo Debugger GX, see the README.TD online text file.

For a list of the files on your distribution disks, see the FILELIST.DOC file on the Installation disk.

#### **Program files**

The installation program copies the program files into the BIN subdirectory of your main Borland C++ directory. The following files are included:

- TD.EXE
- TDDEBUG.DLL
- TD-LANG.STR
- TDHELP.HLP

#### **Online text files**

By default, the installation program copies the Turbo Debugger GX online text files into the DOC subdirectory of the main Borland C++ directory on your hard drive. These files include README.TD, MANUAL.TD, and UTILS.TD. In addition, there's an overall README file for the entire Borland C++ package that resides in the main Borland C++ directory.

The README.TD file	It's important that you take the time to look at the README.TD file before you do anything else with Turbo Debugger GX. This file contains last- minute information that might not be in the manual or the Online Help.				
The MANUAL.TD file	Be sure to read the MANUAL.TD file for late-breaking changes and additions to the manual. If there are no changes to report, this file won' on the disk.				
The UTILS.TD file	Turbo Debugger GX comes with the TDUMP utility. By default, it's in the BIN subdirectory of the main Borland C++ directory along with the Turbo Debugger GX program files.				
	To get a list of the command-line options available for TDUMP, type the program name on the OS/2 command line and press <i>Enter</i> .				
	TDUMP.EXE displays the contents of object modules and .EXE files in a readable format.				
Sample programs	A number of sample programs are distributed in the Borland C++ package. The two programs associated with Turbo Debugger GX are TDDEMO.EXE and TDDEMOPM.EXE.				
TDDEMO	This program is a simple OS/2 character-mode application that displays text to, and reads text from, a single window. It's the sample program used in Chapter 3, "A quick example." TDDEMO takes lines of text as input. When the user presses <i>Enter</i> on an empty line, the program calculates the number of letters, words, and lines, and how many times each letter occurred, and categorizes words according to length. It then displays all this information on the screen.				
TDDEMOPM	This program does the same work as TDDEMO, except that it accepts input in one window and displays the output in two other windows after each line is entered. It uses some standard Presentation Manager (PM) window types to do its work.				
	The window on top, the one the user enters text in, is a multiline edit window. It uses a standard PM multiline entry field control (WC_MLE) to display text and process the entries the user makes.				
	The two windows below this one are used by TDDEMOPM to display program output. They are standard PM list boxes that use the list box control WC_LISTBOX.				

# Typefaces, icons, and conventions

This section explains the meaning of the special typefaces and icons used in this manual.

Monospaced type This typeface represents text as it appears onscreen or in a program. It is also used for anything you must type literally (such as TD to start up Turbo Debugger GX).

- ALL CAPS All capital letters are used for the names of files and C++ constants.
  - [] Square brackets [] in text, syntax statements, or OS/2 command lines enclose optional items. *Text of this sort should not be typed verbatim*.
  - **Boldface** Boldface type indicates
    - C++ predefined types, functions, preprocessor directives, reserved words and keywords
    - Command-line switches (such as **-s**)
    - *Italics* Italic type indicates C++ variable names, data members, user-defined types, and classes. This typeface is also used to emphasize certain words, such as new terms.
  - *Keycaps* This typeface indicates a key on your keyboard. For example, "Press *Esc* to exit a menu."
  - *Key1+Key2* Key combinations produced by holding down one or more keys simultaneously are represented as *Key1+Key2*. For example, you can reset the program by holding down the *Ctrl* key and pressing *F2*. This key combination is represented as *Ctrl+F2*.

Choice1 This command sequence represents a choice from the menu bar followed by a choice from the drop-down menu. For example, instead of saying "Choose File, then choose Load Process from the File menu," we say "Choose File | Load Process."



This icon indicates material you should take special notice of.



This icon indicates a reference to the Help system, where you can find complete, up-to-date information on Turbo Debugger GX.

### Using this manual



This manual covers the basics of using Turbo Debugger GX. It does not cover all features or discuss debugging tasks in detail—you can find that information in the Online Help system. This manual discusses general aspects of the user interface, tells how to use Online Help, how to install, start, and exit the debugger, and shows how to use some of the debugger's features on a sample program. Once you have the debugger running, you can use the extensive Online Help facility to get complete explanations of features or debugging tasks.

If you're an experienced Turbo Debugger for DOS or Turbo Debugger for Windows user, see Appendix A for a list of debugging tasks and how to perform them with Turbo Debugger GX.

The manual contains the following chapters and appendixes:

**Chapter 1: Getting started** discusses how to install Turbo Debugger GX, how to enter and exit the debugger, and how to use the Online Help system.

**Chapter 2: The Turbo Debugger GX environment** discusses some aspects of debugging and provides an overview of the Turbo Debugger GX environment.

**Chapter 3: A quick example** shows how to use Turbo Debugger GX to perform some debugging tasks on a sample program.

**Appendix A: Turbo Debugger GX for experienced Turbo Debugger users** lists some typical debugging tasks and shows how to do them with both DOS or Windows Turbo Debugger and Turbo Debugger GX.

А

Н

Р

Т

### E

R

# Getting started

Your Borland C++ package contains a set of distribution disks and manuals, including the *Turbo Debugger GX for OS/2 User's Guide* (this book). The distribution disks contain all the programs, files, and utilities needed to debug programs written using Borland C++ for OS/2 and Turbo Assembler for OS/2. The online text files README, MANUAL.TD, and UTILS.TD contain documentation on subjects not covered in this manual.

If you aren't familiar with Borland's no-nonsense license statement, now is the time to read the agreement. Mail your filled-in product registration card, so you'll be notified about updates and new products as they become available.

# Installing Turbo Debugger GX

When you installed Borland C++ on your system, INSTALL.EXE (the installation program on your distribution disks) copied files from the distribution disks to your hard disk. If you left the defaults on, the installation program also created a Borland C++ folder on the desktop and put the icons for Borland C++, Resource Workshop, and Turbo Debugger GX into it.

If you chose not to install Turbo Debugger GX when you installed Borland C++, you can install it now:

- 1. Insert the Installation disk in one of your floppy drives (for example, drive A).
- 2. In an OS/2 window, type A: INSTALL. EXE and press Enter.
- 3. In the Installation dialog box, click the Installation Options button.
- 4. Specify Turbo Debugger GX as the only program to install, then click OK.
- 5. In the Installation dialog box, click Install to start installation.

See page 3 for more information on the Turbo Debugger GX files. By default, the installation program copies the Turbo Debugger GX program files and utilities to the BIN subdirectory of the main Borland C++ directory, online text files to the DOC subdirectory, and examples to the EXAMPLES subdirectory.

Before installing the files, you can change default directories by clicking the Directory Options button and entering new directories in the Borland C++ Directory Options dialog box.

### Entering and exiting Turbo Debugger GX



When you've installed Turbo Debugger GX and it appears as an icon in an OS/2 folder, double-click the icon to start Turbo Debugger GX and display the Debugger Control Panel.

	Figui	re 1.1
Control	Panel	view

<u>F</u> ile	⊻iew	<u>B</u> un	<u>S</u> et	<u>W</u> indow	<u>H</u> elp	
E		9	인단			
PID No F	TID Process	Functio	n		Schedule	Priority
lostu		ten Ovi	-10		ſN	> Process1

See page 17 for a description of the Control Panel view.

From the Control Panel, you can choose File | Load Process to load an application program so you can debug it.

When you're finished debugging that application, you can choose File Unload Process to unload the current process, and then load in another application. You can also exit the program by choosing File | Exit, pressing *Alt+X*, pressing *Alt+4* in the Control Panel, choosing Close from the System Menu, or double-clicking on the system menu icon (at the top left corner of the Control Panel's title bar).

Using commandline options There are a number of command-line options you can use when starting Turbo Debugger GX. You can enter these command-line options two different ways:

- Start Turbo Debugger GX from the OS/2 command line (for example, td -m).
- Right-click on the Turbo Debugger GX icon to display the icon's pop-up menu, then
  - 1. Choose the arrow to the right of Open to display the Settings notebook.
  - 2. On the Program page, enter any Turbo Debugger GX parameters in the Parameters field.
  - 3. Close the notebook when you're done.

The command-line format is as follows:

TD [option[optionarg] ... option[optionarg]] [progname [progargs]]

Brackets indicate that an argument is optional. All Turbo Debugger GX command-line arguments are optional.

- *option* is one of the command-line options listed in Table 1.1. Options must be preceded by either a dash (-) or a slash (/).
- *optionarg* is the argument to a switch, such as the path name that follows the **--s** option.
- *progname* is the file name or full path to the file name of the application program you intend to debug.
- *progargs* are arguments to the application program.

The following table lists the Turbo Debugger GX command-line options. For more information on these options, access the Turbo Debugger GX Help system and use the Search button to find *command-line option*.

- · · · · ·				
Table 1.1 Turbo Debugger GX command-line options	Option	Description		
	-cfilename	Indicates the path and filename of the configuration file to be used when Turbo Debugger GX starts up.		
		By default, Turbo Debugger GX uses TD.INI as its configuration file. If you specify a –c command-line option, Turbo Debugger GX reads and writes to the configuration file specified.		
	<b>-h</b> or <b>-?</b>	Opens a window displaying a panel of Help text describing these command- line options.		
	-m	Enables monochrome screen colors for plasma screens and other monochrome video adapters.		

9

ŝ,

-r <expression></expression>	Run to <i>expression</i> on start up. You must also specify an application to be debugged (a <i>progname</i> command-line argument). This switch causes Turbo Debugger GX to run the startup code of the application and position the program counter at <i>expression</i> after it loads the application. Note that <b>main</b> is the default.		
<b>-s</b> dirlist	Indicates where to find the source files for your application. You can enter one search path or multiple search paths separated by semicolons.		

#### Table 1.1: Turbo Debugger GX command-line options (continued)

### Using the Help system

This section provides a detailed overview of the Turbo Debugger GX Online Help system. The Help system is the principal source of information about Turbo Debugger GX. You can go there to get detailed procedural and descriptive help on debugging tasks or to get context-sensitive Help on elements of the user interface (like views, menu choices, dialog boxes, list boxes, and entry fields).

Online Help is available from any view, menu, or dialog box, and provides three kinds of information:

- Context-sensitive Help for all individual elements of the Turbo Debugger GX environment: menu choices, views, entry fields, check boxes, and radio buttons in views and dialog boxes. You can select any menu choice or any element of a dialog box or view (entry field, check box, or radio button), then press *F1* to get Help for that item.
- Task-oriented information on debugging tasks, such as essential information to get you started, how to set breakpoints, how to go on a bug hunt, and debugging tips and techniques.
- Information on the Turbo Debugger GX environment, such as views, menus, and keyboard shortcuts. (Most of this information is available as context-sensitive Help, but you can also access it from within the Help system.)

To access the Help system, do any of the following actions:

- Choose Help from the Control Panel's menu bar.
- Click the Help button on the Control Panel's SpeedBar.
- Press *F1* anywhere in Turbo Debugger GX.
- Press *Shift+F1* (to get the Help index).
- Press *Ctrl+H* in a view or dialog box, or choose Help from the local menu.

How you access the Help system depends on what kind of information you want.

- If you need general help, you can access the Help menu or the Help Contents panel and choose the topic you need.
- If you know what you're looking for but not where it is in the Help system, you can display the Help index and search for the item.
- If you want help with an element of the user interface, you can set the focus to that control and press *F1* for context-sensitive Help.

If you choose Help from the Control Panel menu bar, you see the Help

If you want task-oriented information, such as how to set breakpoints, you can display the Contents panel and select Essentials or Tasks, or you can use the Search button in the Help window to find the topic.

Menu choice	Description
Contents	Table of contents for the Help system. Each topic preceded by a + can be expanded into subtopics. A - means the subtopics for that topic are already expanded. Double-clicking on a topic brings up a Help panel for it.
Index	Alphabetical list of topics for the Help system. Double-clicking on a topic or subtopic brings up a Help panel for it.
Essentials	Information to help you get started with the debugger. Also available from Contents.
Tasks	A list of debugging tasks, such as compiling your program for debugging, executing your program under the debugger, and setting and using control points. Also available from Contents.
Menus	A list of all the global and local menu choices available in the debugger. Also available from Contents.
Views	A list of all the views, including the Control Panel. Also available from Contents.
Glossary	A glossary of debugging terms. Also available from Contents.
Using Help	Help on how to use the Help system.
Product Information	A panel showing the Turbo Debugger GX name and version.

Accessing the Control Panel Help menu

Table 1.2 The Help menu menu.

#### Using a Help window

Picking any Help menu choice except Product Information displays a Help window.

To get information on how to use Help, choose Help from the window's menu bar. At the bottom of the window are some pushbuttons, which work as follows:

	Contents	Display the Contents panel.			
	Index	Display the Help index.			
	Print	Print selected Help panels.			
	Search	Search for a topic in the Help system.			
	Previous	Go to the previously viewed Help panel. If this is the first panel you displayed, clicking this button exits you from Help.			
	Forward	Display the next Help panel.			
Displaying the Contents panel	To display the from the Cont	e Contents panel from the debugger, choose Help   Contents trol Panel menu bar or click the Help button on the SpeedBar.			
	If you're already in a Help window, you can go to the Contents panel by clicking the Contents button at the bottom of the Help window, pressing <i>Ctrl+C</i> , or choosing Options   Contents with your mouse or by pressing <i>Alt+O+T</i> .				
	The Contents panel shows the same topics as those shown in Figure 1.1.				
Displaying the Help index	To display the Help index, choose Help   Index from the Control Panel, or press <i>Shift+F1</i> from anywhere within the debugger.				
	If you're in a Help window, you can click the Index button at the bottom of the window, press <i>Ctrl+I</i> , choose Options   Index with your mouse, or press <i>Alt+O</i> .				
	The Help index is an alphabetic list of topics in the debugger's Online Help You might want to display it if you know what you're looking for, but you aren't sure where it is. You can scan down the list or search for a topic. When you find the topic you want, double-click it or press <i>Enter</i> to display the associated Help panel.				
Getting context- sensitive Help	To get Help o dialog box en showing info	n a part of the user interface, such as a menu choice or a try field, select it, then press <i>F1</i> . A Help panel comes up rmation on the area you clicked.			
	For example, if you press <i>F1</i> while File   Load Process is selected, you get information on that menu choice. If you press <i>F1</i> when the Source view is active, you get information on that view.				

•

For Help on a menu in its entirety (rather than a single menu choice), go into Help and find the menu name in the Contents or the Index about that menu, then double-click the menu name to display a Help panel.

Printing Help information

You can print Help information from a Help window as follows:

- 1. Select the panels you want to print by doing one of the following:
  - If you want to print one panel, display a single Help panel.
  - If you want to print more than one panel, display the Contents panel and select the topics you want to print. Each topic represents a single Help panel. (The Print facility calls each Help topic a *section* and calls this process *marking sections*.) To select topics, press *Ctrl* and click each topic you want to print. (To deselect topics, repeat this process.)
- 2. Select the Print button, press *Ctrl-P*, or Services | Print (with your mouse) to display the Print dialog box.
- 3. Select the radio button for what you want to print, then click Print.

Choosing All Sections isn't recommended because it will print all the panels in the Help system.

Turbo Debugger GX for OS/2 User's Guide

Н

С

Α

Р

Т

Е

R

(

# The Turbo Debugger GX environment

Debugging is the process of finding and correcting errors (*bugs*) in your programs. It's not unusual to spend more time finding and fixing bugs in your program than writing the program in the first place. Debugging is not an exact science; the best debugging tool you have is your own feel for where a program has gone wrong. Nonetheless, you can always profit from a systematic method of debugging.

The debugging process can be broadly divided into four steps:

- 1. Realizing you have a bug
- 2. Finding where the bug is
- 3. Finding the cause of the bug
- 4. Fixing the bug

## What Turbo Debugger GX can do for you

Turbo Debugger GX helps with the two hardest parts of the debugging process: finding where the bug is and finding the cause of the bug. It does this by controlling program execution so you can examine the state of the program at any given spot. You can even test new values in variables to see how they affect your program. With Turbo Debugger GX, you can perform *stepping, viewing, inspecting, changing,* and *watching*.

Stepping into	You can execute your program one line or one instruction at a time, stepping into each function call.
Stepping over	You can execute your program one line or one instruction at a time, but step over any function calls. If you're sure your procedures and functions are error- free, stepping over them speeds up debugging.

Viewing	You can have Turbo Debugger GX open a special window to show you the state of your program from various perspectives: variables and their values, breakpoints, datapoints, messagepoints, exceptionpoints, the contents of the stack, an event log, a data file, a source file, disassembled code, memory, the heap, registers, numeric processor information, or program output.
Inspecting	You can look at the contents of variables and expressions, including complex data structures like arrays and structures.
Changing	You can replace the current value of a global or local variable with a value you specify.
Watching	You can isolate program variables and keep track of their changing values as the program runs.

You can use these tools to dissect your program into discrete sections, confirming that one section works before moving to the next. In this way, you can work through any program, no matter how large or complicated, until you find where a bug is hiding. You might find there's a function that inadvertently reassigns a value to a variable, or gets stuck in an endless loop. Whatever the problem, Turbo Debugger GX helps you find where it is and what's at fault.

Turbo Debugger GX enables you to debug object-oriented C++ programs. It's smart about classes, and it correctly handles late binding of member functions so that it executes and displays the correct code.

Turbo Debugger GX also enables you to debug both Presentation Manager and OS/2 line-mode programs.

#### What Turbo Debugger GX won't do

With all these features, you might be thinking that Turbo Debugger GX has it all. However, there are at least three things Turbo Debugger GX *won't* do for you:

- Turbo Debugger GX doesn't have a built-in editor to change your source code. You can use the Borland C++ editor or your favorite text editor for this purpose.
- Turbo Debugger GX can't recompile your program for you. You need the original program compiler to do that.
- Turbo Debugger GX can't come up with strategies for finding bugs. It's a powerful tool, but is only that—a tool.

How Turbo Debugger GX does it Here's the good news: Turbo Debugger GX gives you all this power and sophistication, and at the same time it's easy to use.

Turbo Debugger GX accomplishes this blend of power and ease by offering an environment featuring a graphical user interface. The next section describes the advantages of the Turbo Debugger GX GUI environment.

## The Turbo Debugger GX environment

Turbo Debugger GX has been designed for intuitive use. To this end, Turbo Debugger GX provides you with the following features:

- A Control Panel view, from which you can control all aspects of a debugging session.
- Global and local menus that make it easier to access menu commands.
- Online Help, available from any view, menu, or dialog box, that provides context-sensitive and task-oriented information. (See page 10 for a description of the Help system.)
- Dialog boxes you can use to change preferences, look at variables, set control points, and load processes.
- views that show you different aspects of your code and data, and tell you what's going on in memory, with the processor, and with the operating system.

Using the Control Panel

The first thing you see when you start Turbo Debugger GX is the Control Panel view.

Menu bar 	SpeedBar 
C Debuggerd File' View Bu	ntoll.Paret
PID TID Fun No Process	tion Schedule Priority
Use hard mode	[No Process]
Help Ctrl+H	
Local menu	Threads pane Status line

You use the Control Panel to oversee, manage, and control the debugging process. From the Control Panel, you can perform the following tasks:

- Load and unload applications
- Open views
- Run and step through applications
- Set control points (breakpoints, messagepoints, datapoints, and exceptionpoints)
- Manage all the views
- Monitor the status of threads

As you can see in Figure 2.1, the Control Panel view contains the following elements:

- 🛚 Menu bar
- SpeedBar
- Threads pane
- 🛚 Status line
- Local menu

See the Online Help for complete information on the Control Panel and on these tasks.

Figure 2.1 Control Panel view

#### The menu bar

Table 2.1 Menu bar choices There is one menu bar in the debugger, the one at the top of the Control Panel. The menu bar has the following choices:

Choice	Description
File	Use the File menu to load or unload a process, set debugger properties, or exit the debugger.
View	Use the View menu to select any of the debugger views. With these menu choices, you get access to views that show things like source code, disassembled code, control points, program data, memory, and what's going on with the CPU. The views are described later, starting on page 22.
Run	Use the Run menu to run your application in different ways, such as stepping through your program one source line at a time, running to a certain point, or simply running the program. You can also stop or reset the program from this menu.
	There are SpeedBar buttons that correspond to the following choices on this menu: Run, Stop, Reset, Statement Into, Statement Over, Instruction Into, and Instruction Over.
Set	You can use this menu to set four kinds of control points (breakpoints, datapoints, C++ exceptionpoints, and messagepoints) and to set a watch to monitor changes in an expression.
Window	Use this menu to control your debugger views. You can switch to the window of the application (the <i>user</i> window), move from view to view, hide or show all the views (except the Control Panel), save or restore the positions you've put the views in, and choose from a list of open views.
Help	Use this menu to access Online Help. (Note that you can also use the SpeedBar button to access the Help.) See page 10 for more information on using Online Help.
	Choice File View Run Set Window Help



See the Menus topic in Online Help for a complete description of these menu choices.

#### The SpeedBar

New Column and the state of the state						
	? 🕒		ò		010	010
		234,12,621,681,63,43,43,572,57		063661608686	مرزة كالشرق تركزنية	



Use context-sensitive Help for a complete description of each of these buttons.

The SpeedBar gives you quick access to typical debugging tasks. If you run the mouse across the SpeedBar, the function of each button appears on the status line at the bottom of the Control Panel. The following table describes what these buttons do:

#### Table 2.2: SpeedBar buttons

Button	Description
Start Program Running	Runs the currently loaded process. Same as Run Run.
Stop Program	Stops the currently loaded process if possible and returns control to the debugger. If the Stop button is disabled, it isn't possible to stop the process at this time. Same as Run Stop.
Reset Program	Reloads the current process so you can run it again from the beginning. Any watches and breakpoints you set in an earlier run remain set. Same as Run Reset.
Show Help	Displays the Contents screen of Online Help. Same as Help Contents.
Raise Program Window	Switches to the application's active window. Same as Window/User Window.
Hide Debugger Windows	Hides all open views. You typically do this before minimizing the Control Panel. Same as Window Hide Views.
Show Debugger Windows	Shows all the views hidden by the Hide Views command. Same as Window Show Views.
Statement Step Into	Steps through the application one source statement at a time, and into any functions that are called. Same as Run Statement Into.
Statement Step Over	Steps through the application one source statement at a time, but steps over any function calls (doesn't step into the function, but rather runs the function until it returns). Same as Run Statement Over.
Instruction Step Into	Steps through the application one assembly instruction at a time, and steps into any routines that are called. Same as Run Instruction Into.
Instruction Step Over	Steps through the application one assembly instruction at a time, but steps over any routine calls (doesn't step into the routine, but rather runs it until it returns). Same as Run Instruction Over.

#### The Threads pane

15 1 main() Runnable 512 🗖

The Threads pane shows information about the threads that make up your application. It's most useful with multithreaded applications. Using this pane, you can

- Reset all views to show information about a particular thread you want to debug
- Get information about the current thread and process, such as the process ID (PID), the thread ID (TID), which function is currently active, and the schedule and priority of the current thread

A typical use of this pane is to select a thread so you can get information about it (such as register settings or variable values), and to change the Source view to show where that thread stopped. To select a thread, click the scroll buttons on the right side of the pane till you see the thread you want. Then, double-click the thread. (All the views will then take on the context of that thread.)

The status line

The status line at the bottom of the Control Panel displays error messages, shows the status of the current process (suspended, running, no process), and describes what each SpeedBar button does.

Because the status line displays error messages, keep this part of the Control Panel visible at all times, even when working in other views.

#### Using dialog boxes

A dialog box is a window you can enter information in for some task you want to perform. A dialog box, unlike a view, just lets you enter or change information. (A view shows updated information about your application and might let you enter information as well.)

For example, if you choose File | Properties, you see the Properties dialog box.

Check boxes	Entry fields List box		
🗄 Prop Ites			
Pop Up on Exception:	Source Directories:		
Bource View     Disassembly View     Local Variable View	Add directory:		
Use C Evaluator Use C++ Evaluator Use TASM Evaluator	Run to max on startup		
Smart Messages     Just beep     Just pop up a message     Beep and pop up a message	Case sensive searches  Wrap around searches Always create new views  Save properties  Save views for startup		

Radio buttons Combo box

See the Online Help for more information on this dialog box.

This dialog box contains radio buttons, check boxes, a list box, and three entry fields. Because the dialog box is just another window, there is no OK or Cancel button. Depending on the kind of change you make, it either takes effect immediately or takes effect when you press *Enter*.

#### Figure 2.2 Properties dialog box

For example, if you type a source directory name in the Add Directory entry field, you must press *Enter* for it to take effect. However, if you click any of the radio buttons or check boxes, your selections take effect immediately.

This behavior is common to all dialog boxes and views.

Working with views A view is a window that shows information about your application. Some views also let you enter information. These views follow the same rules as dialog boxes for entering information (see the previous section). There are eighteen different views, all available from the Control Panel's View menu.

#### Local menus

All views have local menus that you can pop up by right-clicking the mouse or by pressing *Ctrl+F10*. For example, right-clicking in the Source view displays the following local menu:

Figure 2.3 Source view local menu

🖧 Source uddens o	BREAK CO	• 🗖
• D	nlines nwords totalch showarg while (:	= 0 = 0; aracters = 0; s(argc, argv); readaline() != 0) {
Set breakpoint Enable break point <u>R</u> un to here Jump to here	F2 Ctrl+E F4 Ctrl+J	<pre>wordcount = makeintowords(buffer); nwords += wordcount; totalcharacters += analyzewords(buffer nlines++; totalcharacters == unade_totalcharacter</pre>
Inspect value Add <u>w</u> atch Add <u>d</u> atapoint	Ctrl+I Ctrl+W Ctrl+D	<pre>/nsress (Enter&gt; when done."); buf); );</pre>
<u>G</u> oto the pc <u>M</u> ove to address Move to line number Search Search <u>ag</u> ain	Ctrl+G Ctrl+M Ctrl+L Ctrl+S Ctrl+S	er into a list of null-terminated word wo nulls squish out white space. >
Display <u>o</u> ptions Views New view Help	Ctrl+0 → Ctrl+V → Ctrl+N Ctrl+H	

The local menu choices have shortcut keys, indicated by an underlined letter in the choice. For example, to search for a string in the Source view, you can press *Ctrl+S* without having to display the local menu first.

#### List views and Detail views

Some of the views have two forms, a List view that shows all items that have been set and a Detail view that shows information about each item. The views that have these two forms all manage lists of items, such as variables or control points. The following views have both forms:

- Breakpoint view
- Datapoint view
- Exceptionpoint view
- Messagepoint view
- Variable view
- Watch view

The view initially displays in a default form (if there are items to display, the list form; if there are no items, the detail form). To switch to the other form, either press *Ctrl+S* or right-click in the view to display the local menu, then choose the first menu choice. The wording of this menu choice changes depending on which form of the view is displayed.

For example, when you first load your application, there are no breakpoints set. Choosing View | Breakpoint in the Control Panel displays the Breakpoint Detail view, which you can use to set a breakpoint.

Figure 2.4 Breakpoint detail view	🏵 Breakpoint 🔹 🗆			
	Breakpoint: KAdd a Breakpoin>			
	Location:			
	Filters: Expression True Pass Count	to go		
	Actions: 🗹 Break Evaluate Expression			

Enter the name of a function (for example, main) in the Location entry field and press *Enter*. (The Location entry field takes a program location, such as function name that evaluates to a program location.) Next, right-click to display the local menu, then choose Show Breakpoint List to display the List view. You see the breakpoint you just set.

E

Figure 2.5 Breakpoint List view

Enabled	Breakpoint	
	<add a="" breakpoir<="" th=""><th>1D</th></add>	1D
Z	tddemo.c#33	int main(int a
		Σ

If you close the Breakpoint view, then choose View | Breakpoint again, notice that the Breakpoint view comes up in list form. That's because there's a breakpoint to display. If you right-click in the Breakpoint view, you see that the first local menu choice has changed to *Show Breakpoint Details*.

If you close the Breakpoint view again, then choose Set | Breakpoint from the Control Panel, notice that the Breakpoint view comes up in detail form, letting you set a breakpoint immediately.

#### The views

Breakpoint	
Exceptionnoint	
C++ Exceptionnaint	
Messagepoint	
Source	
Disassembly	
Modules	
Evaluator	Ctrl+F4
Inspector	
Variable	
Watch	
Call Stack	
Heap	
Memory	
Numeric Processor	
Register	
C++ Exception Stack	
File	
Log	

The View menu is displayed when you choose View from the Control Panel. The View menu is divided into five sections that reflect the functionality of the views.

- The first section groups choices for all the *control point* views. (A control point is a name referring to breakpoints, datapoints, exceptionpoints, and messagepoints, all of which can be used to control program execution.) Use these views to set various kinds of control points that, when encountered, can log information about the control point or return control to the debugger.
- The second section groups choices for views that show you information about your application's code. For example, the Source view shows your application's source code and where the program has executed to, and the Modules view lists the source files contained in your executable program.
- The third section groups choices for views that show you information about your program's data. For example, the Variable view displays all local or global variables.
- The fourth section groups choices for views that show you hardwarerelated information, such as the contents of memory or the CPU registers.

The fifth section contains choices for the Log view and the File view, both used for auxiliary functions (logging information and looking at files that do not contain debug information).

#### The Breakpoint view

Choosing View | Breakpoint from the Control Panel displays the Breakpoint view. You can also choose Set | Breakpoint to display this view in Detail form.

Breakpoints stop the processing of your program and give control of it to you. Use the breakpoint view to set, remove, modify, enable, and disable breakpoints, and to see a list of the breakpoints that have been set in your program.

When you set or change a breakpoint, you can also set filter conditions and actions, which customize the conditions under which a breakpoint is activated and specify the actions that take place when the breakpoint is activated.

#### The local menu

Right-click or press Ctrl+F10 to display the local menu.

With the Breakpoint view local menu, you can perform actions with breakpoints, such as setting or removing them. You can also use the shortcut keys you see on the menu directly from the Breakpoint view, without displaying the menu.

#### The List and Detail views

Press *Ctrl+S* to change view forms.

This view has two forms, a Detail view form and a List view form. The Detail view shows details about a particular breakpoint (if any exist); you use this form to set a new breakpoint or change settings for an existing breakpoint. The List view shows all breakpoints that have been set.

See page 23 for more information on displaying forms.



You can set a simple breakpoint without using the Breakpoint view. There are two ways to do this:

- Select a line of code in the Source view or the Disassembly view, then press *F2*.
- Double-click the mouse either inside a line of disassembled code or near the diamond in the left margin of a line of source code (if you've displayed the diamonds when using the Display Options | Show Attributes submenu).

You can get the following additional information from the Help system:



- For other methods of setting breakpoints, see the Online Help topic "Setting Breakpoints".
- For a complete discussion of breakpoints, see "Breakpoints" and its subtopics in the Online Help under the "Setting and Using Control Points" task.
- For information on the Breakpoint view itself, click in the view, then press *F1* or choose Help from the view's local menu.
- For information on a local menu choice, right-click to display the menu, then select the menu choice and press *F1*.

#### The Datapoint view

Choosing View | Datapoint from the Control Panel displays the Datapoint view. You can also choose Set | Datapoint to display this view in detail form.

Use this view to set a *datapoint* or see all the datapoints you've set. A datapoint (also known as a watchpoint) is a variable or expression whose memory location the debugger watches during program execution. When the value in that memory location matches a condition, such as being equal to or less than a certain value, the debugger performs the action you've indicated, such as breaking and returning control to the debugger.

A datapoint has characteristics similar to a breakpoint (see the Breakpoint view description starting on page 25). For additional information about datapoints, refer to the topic in the Online Help.

#### The local menu

You use the Datapoint view to perform the following actions on datapoints:

- Set new datapoints
- Adjust the filters and actions associated with those datapoints
- Enable or disable datapoints
- Remove existing datapoints
- Look at the datapoints that have been set

All these choices are available from the Datapoint view local menu (rightclick or press *Ctrl+F10* or *Shift+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Datapoint view, without actually displaying the menu.

#### The List and Detail views

Press *Ctrl+S* to change view forms.

This view has two forms, a Detail view form and a List view form. The Detail view shows details about a particular datapoint (if any exist); you

use this view to set a new datapoint or change settings for an existing datapoint. The List view shows all datapoints that have been set.

For more information on the Datapoint view itself, click in the view, then press *F1* to display a Help screen or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*. For more information on working with datapoints, see "Datapoints" in the Online Help under the "Setting and Using Control Points" task.

# The Exceptionpoint view

¢

กั

Choosing View | Exceptionpoint from the Control Panel displays the Exceptionpoint view.

Use this view to change settings for an *exceptionpoint* or see all the exceptionpoints. An exceptionpoint tells the debugger what to do when it intercepts a particular *exception* or signal sent to your application. (An exception is an asynchronous notification from OS/2 that an event has occurred, such as a divide-by-zero exception or a guardpage exception.)

When the exception comes in, the debugger performs the action you've indicated, such as breaking and returning control to the debugger, then passing the exception to the application when you run the application again.

An exceptionpoint has characteristics similar to a breakpoint (see the Breakpoint view description starting on page 25). For further information about exceptionpoints, refer to the topic in the Online Help.

#### The local menu

You can use the Exceptionpoint view to perform the following actions on exceptionpoints:

Indicate whether an exceptionpoint pauses program execution

Adjust the filter conditions and actions associated with exceptionpoints

Look at the list of exception points

All these choices are available from the Exceptionpoint view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Exceptionpoint view, without actually displaying the menu.

#### The List and Detail views

Press *Ctrl+S* to change view forms.

This view has two forms, a List view form and a Detail view form. The List view shows all exceptionpoints. The Detail view shows details about a particular exceptionpoint; you use this view to change settings for an exceptionpoint.



For more information on the Exceptionpoint view itself, click in the view, then press *F1* to display a Help screen or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

#### The C++ exceptionpoint view

Choosing View | C++ exceptionpoint from the Control Panel displays the C++ exceptionpoint view.

Use this view to customize the action that the debugger should take when a C++ exception is thrown. By default, the debugger stops on all C++ exception throws.

When a C++ exception is thrown by your program, the debugger performs the actions you've indicated, such as breaking and returning control to the debugger, then with the throw when you run the application again.

A C++ exceptionpoint has characteristics similar to a breakpoint (see the Breakpoint view description starting on page 25). For further information about C++ exceptionpoints, refer to the topic in the Online Help.

When the debugger stops on a C++ exception, it displays a dialog box that displays the C++ exception's type and value. With this dialog box, you can choose to run to either the catch or stack-unwinding destructors associated with this C++ exception.

#### The local menu

You can use the local menu of the C++ exceptionpoint view to enable, disable, remove, and add C++ exceptionpoints.

#### The List and Detail views

C++ exceptionpoint.

Press *Ctrl+S* to change view forms.

8

For more information on the C++ exceptionpoint view itself, click in the view, then press *F1* to display a Help screen or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

This view has two forms, a List view form and a Detail view form. The List

view shows all C++ exceptionpoints. The Detail view shows details about a particular C++ exceptionpoint; you use this view to change settings for a

# The Messagepoint view

All these choices are available from the Exceptionpoint view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Exceptionpoint view, without actually displaying the menu.

Choosing View | Messagepoint from the Control Panel displays the Messagepoint view. You can also choose Set | Messagepoint to display this view in detail form.

Use this view to track PM messages sent to the window functions in your application. You can also have your application break and return control to the debugger or perform some other action when it encounters a message for one of your window functions.

When you designate a window message to be tracked, you're setting a *messagepoint*. A messagepoint has characteristics similar to a breakpoint (see the previous section). For additional information about messagepoints, see the topic in the Online Help.

#### The local menu

You can use the Messagepoint view to perform the following actions on messagepoints:

- Set new messagepoints
- Adjust the filter conditions and actions associated with those messagepoints
- Enable or disable messagepoints
- Set messagepoints on your own custom messages
- Remove existing messagepoints
- Look at the messagepoints that have been set on window functions

Most of these choices are available from the Messagepoint view local menu (right-click or press *Ctrl+F10* or *Shift+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Messagepoint view, without actually displaying the menu.

#### The List and Detail views

Press *Ctrl+S* to change view forms.

á

This view has two forms, a Detail view form and a List view form. The Detail view shows details about a particular messagepoint (if any exist); you use this view to set a new messagepoint or change settings for an existing messagepoint. The List view shows all messagepoints that have been set.



For more information on the Messagepoint view itself, click in the view, then press *F1* to display a Help screen or choose Help from the view's local menu. For information on a local menu choice, select it and press *F1*. For more information on working with messagepoints, see "Messagepoints" in the Online Help under the "Setting and Using Control Points" task.
The Source view

If you load an application that has debugging information and source code, the debugger displays the source code for the current module in the Source view. You can also display the Source view by choosing View | Source from the Control Panel.

The following figure shows the Source view opened on TDDEMO, one of the sample programs distributed with Turbo Debugger GX. For demonstration purposes, a breakpoint has been set and the program has been run to **main**.

#### Breakpoint glyph

Program counter glyph

Figure 2.6 The Source view

<u>ol</u> de la	dema e
∘B D	TRAIN(JULY STORE BAREAR MARINY)
	unsigned int nlines, nwords, wordcount; unsigned long totalcharacters;
۰	nlines = 0;
٥	nwords = 0;
٥	totalcharacters = 0;
٥	showargs(argc, argv);
0	while (readaline() = 0) {
<u>ہ</u>	<pre>wordcount = makeintowords(buffer);</pre>
0	nwords += wordcount;
•	totalcharacters += analvzewords(buffer
٥	nlines++:
٥	γ · · · · · · · · · · · · · · · · · · ·
0	printstatistics(nlines_nwords_totalcharacter
ଚ	printf("\nPress (Enter) when done.");
0	gets(rethuf)
0	return (0)

Executable line of source code indicator

- The diamond at the left of a line of source code indicates that the line is executable (not a declaration or comment) and has a valid address. It's a location where you can set a breakpoint.
- If you set a breakpoint on a line of source code, a breakpoint glyph (a blue B in a box) appears to the right of the diamond marking that line of source code.
- The program counter glyph indicates the line of code that will execute next when you run your program. The first line of that code is also selected.

You're likely to spend much of your time in the Source view when you're debugging an application.

#### The local menu

From this view, you can do the following things:

- Set, delete, enable, and disable breakpoints
- Run or jump to the current insertion point position
- Add datapoints and watches
- Inspect variables and expressions
- Move around in the source code by searching, moving to an address or line number, or returning to the program counter

All these choices are available from the Source view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Source view, without actually displaying the menu.

For more information on the Source view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

Choosing View | Disassembly from the Control Panel displays the Disassembly view.

This view shows your disassembled source code. You use it to see the assembly language instructions that correspond to your source code. You must use this view if the program you're debugging wasn't compiled with debugging information or doesn't have source code available.

- If you set a breakpoint on a line of disassembled code, a breakpoint glyph (a blue *B* in a box) appears to the left of the line of code.
- The program counter glyph indicates the instruction that will execute next when you run your program. That line of code is also selected.

#### The local menu

In this view you can do things such as the following:

- Set, delete, enable, and disable breakpoints
- Run or jump to the current insertion point position
- Move around in the code by moving to an address or jump target, or returning to the program counter

All these choices are available from the Disassembly view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Disassembly view, without actually displaying the menu.

#### **Displaying panes**

The Disassembly view can represent five views as panes. By default, this view has one pane, the Disassembly pane. Using the Display Options local

#### The Disassembly view

Q

menu choice (*Ctrl+O*), you can add a Memory pane, a Stack pane, a Registers pane, and a Flags pane.

Figure 2.7 The five panes of the Disassembly view

Jisas	ssembly pane	Register pane Flags pane
B .	rassembly a marked in the marked based on the same	
B	tddemo.c#33 int main(int argc bit ordonomic average of the second ox0001001C PUSH EDI ox0001001D PUSH EDI tddemo.c#37 nlines = 0; 37 0x0001001F XOR EDI.EDI tddemo.c#38 nwords = 0; 38 0x00010021 XOR ESI.ESI tddemo.c#39 totalcharecter; 39 0x00010023 XOR FEY ERY	▲         EAX         0×00000000         ↓         C0           EEX         0×00020034         C0         C1         C1           ECX         0×00020BAC         A0         C1         C1 <td< th=""></td<>
	tddemo.c#40 showargs(argc. 40 0x00010025 PUSH DWORD FTR	ESP 0x0003BFA8 00011d41 0x0003BFAC 00000001
EDI	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	<ul> <li>▲ 0x0003FFB4 0000000</li> <li>● 0x0003FFB4 0000000</li> <li>● 0x0003FFB5 0000000</li> <li>● 0x0003FFC0 0000000</li> <li>● 0x0003FC4 0000000</li> <li>● 0x0003FC4 0000000</li> <li>● 0x0003FC6 fffffff</li> <li>▲ 0x003FC6 ffffffff</li> <li>▲ 0x003FC7 00011997</li> </ul>
<u>Edited</u>	<u> </u>	New York, New Yo

#### Memory pane

Discourselate

Stack pane

The local menu that comes up for all these panes is the Disassembly local menu. If you want to perform a pane-specific task that is only available from the corresponding view's local menu (such as clearing register ESP in the Register pane), open the corresponding view and perform the task there.

Note, however, that many pane-specific tasks can be performed directly in the Disassembly view without using the local menu. For example, to change a register value, you can double-click the register and enter the new value in the dialog box that appears. You can perform this type of task in the pane without opening the associated view.



For more information on the Disassembly view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Modules view

Choosing View | Modules from the Control Panel displays the Modules view.

Use this view to display source modules in addition to the one currently displayed in the Source view. This view initially displays the name of your application's .EXE file and any DLLs used by your .EXE.

Each DLL or .EXE has a + preceding it. Click this icon to see all the source modules for the DLL or .EXE. If you double-click one of the modules, the

debugger loads it into the Source view, where you can do things like set control points, set watches on expressions, and so on.

If you want to see more than one module at a time, double-click on the module you want to examine in the Module view, and a new Source view will open with the desired source code.



For more information on the Modules view, click in the view, then press *F1* to display Help, or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

#### The Evaluator view

Choosing View | Evaluator from the Control Panel displays the Evaluator view.

Use this view to change the values of variables and expressions and to evaluate expressions that cause side effects (like function calls). It's especially useful for changing the values of complex variables (like *letterinfo* from TDDEMO).

#### The local menu

In this view you can do the following things:

- Enter a new expression to evaluate
- **n** Inspect the value of an expression (open an Inspector view on the current expression)
- Display the stack concurrently with the expression (a Display Option— *Ctrl+O*)

These choices are available from the Evaluator view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Evaluator view, without actually displaying the menu.



For more information on the Evaluator view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

#### The Inspector view

Choosing View | Inspector from the Control Panel menu bar displays the Inspector view. You can also display this view by choosing Inspect Value from the local menu of the Source view, the Variable view, or the Watch view (or pressing *Ctrl+l* in any of these views). Note that in the Source view, you must have the insertion point on the variable you want to inspect.

Use this view to display or change the current value of a selected variable or expression. Double-clicking on a variable or expression in the Source view automatically displays it in the Inspector view. The Inspector is useful for taking a quick look at a variable or expression or seeing the elements of a complex variable or expression. You can also use the Inspector to change the value of a simple variable (or a single element of a complex variable).

#### The local menu

In this view you can do the following things:

- Enter a new expression
- Change the value of an expression
- Show type information
- Change the form of the data display for an expression

All these choices are available from the Inspector view local menu (rightclick or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Inspector view, without actually displaying the menu.



For more information on the Inspector view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

#### The Variable view

Choosing View | Variable from the Control Panel displays the Variable view.

Use this view to display a list of variables whose values you want to see. (To change which variables display, press *Ctrl+O* or choose Display Options from the local menu.)

#### The local menu

In this view you can do the following things:

- Change between list form and detail form
- Inspect a value
- Add a watch
- Add a datapoint

All these choices are available from the Variable view local menu (rightclick or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Variable view, without actually displaying the menu.

#### The List and Detail views

This view also has a Detail view. To see details on a variable, first select the variable in the List view, then press *Ctrl+S* to display the Detail view.

The Detail view shows the variable's address, its type, and its value. If the variable is a complex type, such as an array or structure, you also see a list showing each element and its value.

While in the Detail view, you can see details on other variables in the list by clicking the drop-down button to the right of the variable name (in the combo box at the top of the Detail view), then choosing a variable from the list that appears.

ິຄ

For more information on the Variable view itself, click in the Variable view, then press *F1* to display Help. For information on viewing variables, see the Help topic "Viewing Program Data in the Variable View" or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

#### The Watch view

Choosing View | Watch from the Control Panel displays the Watch view. You can also display this view by choosing Set | Add Watch from the menu bar or choosing Add Watch from the local menu of the Source view or the Variable view (or pressing *Ctrl+W* in either of these views). Note that in the Source view, you must have the insertion point on a variable.

You use this view to track the values of variables and expressions as they change, or to change their values yourself. Using this view, you can watch more than one expression or variable at a time and get a quick picture of what's going on in your application.

#### The local menu

In this view you can do the following things:

Change between list form and detail form
 Add, remove, or disable a watch
 Change the value of a variable or expression
 Inspect the value of a variable or expression

All these choices are available from the Watch view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the menu directly from the Watch view, without displaying the menu.

#### The List and Detail views

Press *Ctrl+S* to change view forms.

This view has two forms, a List view form and a Detail view form. The List view shows all variables and expressions you are watching. The Detail view shows details about a particular variable or expression and allows you to change its value in memory. Note: you can't change the value of a constant expression.

35



For more information on the Watch view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Call Stack view

Choosing View | Call Stack from the Control Panel displays the Call Stack view.

This view shows the current state of the stack. If you haven't run your application yet, no routines are listed. You can add this view to the Disassembly view as a pane. You can also open a stack pane in several other views, such as: Variable, Watch, Evaluator, Inspector, and Memory.



For more information on the Call Stack view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

#### The Heap view

Choosing View | Heap from the Control Panel displays the Heap view.

Use this view to look at your application's heap. The Heap view represents each memory object in the heap as a line in the Heap view. For each object there is an index (a line number), an address where the object starts, the object's size in bytes, and an indication of whether the object is being used.



For more information on the Heap view, click in the view, then press *F1* to display Help, or choose Help from the view's local menu. For information on looking at memory, see the Help topic "Viewing Memory". For information on a menu choice, select it and press *F1*.

#### The Memory view

Choosing View | Memory from the Control Panel displays the Memory view.

Figure 2.8 The Memory view

$ \begin{array}{c c c c c c c c c c c c c c c c c c c $	Second
0x00010090 eb 01 47 8a - 07 3c 20 74 0x00010098 f9 8a 07 08 - c0 74 01 43 0x000100A0 eb 06 8a 07 - 88 06 47 46 0x000100A8 8a 07 08 c0 - 74 06 8a 07	<pre>&gt;</pre>
Register Addresses Byte representation	

Use this view to look at the contents of memory. When it first opens, you see memory contents displayed as hexadecimal bytes and their ASCII representation at the right side of the window. You can change the form of the display (for example, to **short**) using the local menu Display Options choice (press *Ctrl+O*).

You can also display this view as a pane in the Disassembly view. (See page 31 for a description.)

#### The local menu

In this view you can do the following things:

- Go directly to an address
- Search memory for an expression
- Clear an area of memory
- Move an area of memory to another location (nondestructive copy)
- Change the contents of an area of memory
- Read an area of memory into a file
- Write from a file into an area of memory
- Go to an area of memory indicated by the four bytes at the current text selector location

All these choices are available from the Memory view local menu (rightclick or press *Ctrl+F10* to display it). You can also use the shortcut keys you

37

see on the local menu directly from the Memory view, without actually displaying the menu.



For more information on the Memory view, click in the view, then press *F1* to display Help, or choose Help from the view's local menu. For information on looking at memory, see the Help topic "Viewing Memory." For information on a menu choice, select it and press *F1*.

#### The Numeric Processor view

Choosing View | Numeric Processor from the Control Panel displays the Numeric Processor view.

Use this view to look at or change the state of the numeric processor. You must instruction-step through code that uses the numeric processor in order to see anything meaningful in this view since the numeric stack is usually left clean at the end of each high-level statement. This view indicates

- Contents of the registers
- Control word and control flag settings
- Status word and status flag settings
- NPX Tag word
- Addresses pointed to by the instruction and data pointers
- Current instruction being executed

You can do the following in this view:

- Change control flag values (*Ctrl+G* or double-click)
- Change status flag values (*Ctrl+S* or double-click)
- Change register values (use the entry field)
- Change the value of the control or status word (use the entry field)
- Change the value of the NPX tag word (use the entry field)
- Choose hexadecimal or decimal as the display form (*Ctrl+O*)

For more information on the Numeric Processor view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

#### The Register view

Choosing View | Register from the Control Panel displays the Register view.

Use this view to look at the contents of the CPU registers and flags. You can also display the different panes in this view in the Disassembly view. (See page 31 for a description.)

#### The local menu

In this view you can do the following things:

- Change a register value
- Clear, increment, or decrement a register
- Toggle the value of a flag

All these choices are available from the Register view local menu (rightclick or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Register view, without actually displaying the menu. In addition, you can double-click a register or a flag to change its value.

For more information on the Register view, click in the view, then press *F1* to display Help, or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The C++ exception stack view

ň

The C++ exception stack view displays the state of all pending C++ exceptions. This view displays the most recent C++ exception thrown as the first entry in the list.

Use this view to control how *events* associated with C++ exceptions are handled by the debugger. Events are C++ exception catches or destructor invocations caused by stack unwinding. The check boxes for each entry in this view controls whether or not the debugger stops for destructors or catches for the C++ exception throw associated with the entry. By default, the debugger stops on both of these events.

#### The File view

Choosing View | File from the Control Panel displays the File view.

Use this view to look at files that do not contain Debug information. Typically, you'll use this view on files you can't load into the Source view, such as header files or source files from programs other than the one you're debugging. The default form is hexadecimal bytes with ASCII displayed on the right side. You can change to ASCII form by pressing *Ctrl+O* and choosing Show ASCII.

#### The local menu

In this view you can do the following things:

- Go to a location in the file by entering a C expression (such as a string in quotation marks)
- Change display form (hexadecimal with ASCII on the side is the default)

These choices are available from the File view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the File view, without actually displaying the menu.



For more information on the File view, click in the view, then press *F1* to display Help, or choose Help in the view's local menu. For information on a menu choice, select it and press *F1*.

#### The Log view

Choosing View | Log from the Control Panel displays the Log view.

Use this view to examine the event log. The event log is a dynamic listing of the control points that your program encounters during execution. To indicate that a control point is to be logged, check the Log Expression check box on a control point's Detail view.

For example, to log a breakpoint, choose Set | Breakpoint, enter the breakpoint information, and then check the Log Expression check box at the bottom of the Breakpoint detail view. (This check box is also on the detail views for messagepoints, datapoints, and exceptionpoints.) Whenever your program encounters this breakpoint, the debugger logs its action to the event log.

#### The local menu

In this view you can perform the following actions:

- Erase the contents of the event log
- Open a log file to store the contents of the event log
- Disable and enable event logging

All these choices are available from the Log view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Log view, without actually displaying the menu.



For more information on the Log view, click in the view, then press *F1* to display Help, or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

Н

Т

Ρ

Е

R

## 3

## A quick example

Α

This chapter gives you enough information to debug your first program. Once you've learned the basic concepts described here, the graphical environment and context-sensitive Help system assist you in learning as you go along.

This chapter leads you through the basic features of Turbo Debugger GX. After describing the demo program, it shows you how to do the following procedures:

Run and stop program execution

**Examine the contents of program variables** 

Look at complex data objects, such as arrays and structures

Change the value of variables

### The demo program

This tutorial uses the TDDEMO.C demo program to introduce the two main things you need to know to debug a program: how to stop and start your program, and how to examine your program's variables and data structures. The demo program itself isn't meant to be very useful—some of its code and data structures exist solely to show you the capabilities of Turbo Debugger GX.

The demo program prompts you for lines of text, then counts the number of words and letters you entered. It finishes by displaying some statistics about the nature of the text entered, including the average number of words per line and the number of times each letter occurred.

Make sure your current directory contains the two files needed to debug the demo: TDDEMO.C and TDDEMO.EXE.

**Getting in** To start the program, run the debugger, then:

- 1. Choose File | Load Process from the Control Panel.
- 2. If necessary, change to the directory containing TDDEMO.C and TDDEMO.EXE. (The default directory is \BORLANDC\EXAMPLES\ TD.)
- 3. Enter TDDEMO.EXE as the file to open, then click OK.

Turbo Debugger GX loads the demo program, opens the Source view and the application's Program Window, and positions the text selector in the Source view at the start of the program.

Debugger Control Panel       • □         File View Run Set Window Help         Int main(int argo, char **argv) {         Int set = 0;         Show	0×m	DEMO.EXE				Al Angel Lander	
PID TID Function Schedule Priority PID TID Function Schedule Priority 47 1 0xD0TF.0280 Runneble 512 I Suspended Int main(int argc, char **argv) { unsigned int nlines, nwords, wordcount; unsigned long totalcharacters; nlines = 0; nwords = 0; totalcharacters = 0; showargs(argc, argv); while (readaline() != 0) { wordcount = makeintowords(buffer); nwords += wordcount; totalcharacters += analyzewords(buffer ) printstatistics(nlines, nwords, totalcharacter	Debugger C ile <u>V</u> iew <u>B</u> r	Control Panel un <u>S</u> et <u>W</u> inc	Jow <u>H</u> elp	<b>B</b>			
<pre>PD TID Function Schedule Priority 7 1 0xDU1F.0280 Runnable 512 [Suspended] [Superded] Superded long totalcharacters; nlines = 0; nwords = 0; totalcharacters = 0; showargs(argc, argv); while (readaline() != 0) {     wordcount = makeintowords(buffer);     nwords += wordcount;     totalcharacters += analyzewords(buffer</pre>	⇒0¢	260	6877	010 010		and a second	
[Suspended]	ID TID Fur 7 1 OxE	nction 201F:0280	Schedule Runnable	Priority 512			
<pre>int main(int argc, char **argv) {     unsigned int nlines, nwords, wordcount;     unsigned long totalcharacters;     nlines = 0;     nwords = 0;     totalcharacters = 0;     showargs(argc, argv);     while (readaline() != 0) {         wordcount = makeintowords(buffer);         nwords += wordcount;         totalcharacters += analyzewords(buffer         nlines++;     }     printstatistics(nlines, nwords, totalcharacter</pre>			[St	uspended]			100 (S. 1
<pre>int main(int argc, char **argv) {     unsigned int nlines, nwords, wordcount;     unsigned long totalcharacters;     nlines = 0;     nwords = 0;     totalcharacters = 0;     showargs(argc, argv);     while (readaline() != 0) {         wordcount = makeintowords(buffer);         nwords += wordcount;         totalcharacters += analyzewords(buffer         nlines++;     }     printstatistics(nlines, nwords, totalcharacter</pre>	Charles 16	Concess	Contraction of the				• 🗆
<pre>totalcharacters += analyzewords(buffer nlines++; } printstatistics(nlines, nwords, totalcharacter</pre>		unsign nlines nwords totald showa: while	ned long t s = 0; characters rgs(argc. (readalin wordcou nucrds	otalchar s = 0; argv); we() != ( wht = mak	acters; )) { ceintowor	ds(buffer)	
printstatistics(nlines, nwords, totalcharacter			totalch nlines+	aracters +;	s += anal	yzevords(ł	ouffer
<pre>print(`\nress <enter> when done. ); gets(retbuf); return(0);</enter></pre>		prints print: gets(: return	statistics f("\nPress retbuf); n(0);	(nlines) (Enter)	nwords, when do	totalchan ne.");	racter

Debugger views after loading TDDEMO

Figure 3.1

By default, the debugger doesn't run your program's startup code. You can change this default setting in the Properties dialog box by specifying **main** in Run To \_\_\_ On Startup and checking that button to enable it, or by running the debugger with the **-r** command-line switch (see Table 1.1 on page 9).

The application's user screen appears with TDDEMO because TDDEMO is a character-mode program that requires an OS/2 window, which OS/2 starts automatically before the program is run. If TDDEMO were a

	Presentation Manager program (like TDDEMOPM), the user screen wouldn't appear until you actually ran the program.
Getting out	To exit from the tutorial and Turbo Debugger GX at any time, press <i>Alt+X</i> . If at any point you want to reload the program and start at the beginning, press <i>Ctrl+F2</i> or click the Reset Program button on the SpeedBar.
Getting Help	Press <i>F1</i> whenever you need help with the current view, menu choice or dialog box. You can learn a lot by working your way through the menu system and pressing <i>F1</i> at each menu choice to get a summary of what it does. You can also learn a lot by reading the online <i>User's Guide</i> , which consists of all the subtopics under the Tasks topic in the Help Contents panel. You can read the <i>User's Guide</i> online, taking advantage of its hyperlinks and modular design, or you can print individual topics or sections for reading offline. See Chapter 1 for information about printing Help topics.
the debugger	The Control Panel's menu bar, SpeedBar, and status line, and the various views and their local menus are the keys to using the debugger effectively. For more information, see Chapter 2, "The Turbo Debugger GX environment."

### Using TDDEMO

Using



If you haven't loaded TDDEMO yet, do so now. The text selector in the Source view is on the first executable line of your program, the **main** function. Since you haven't run your program yet, the program counter doesn't show. Press the SpeedBar's Statement Step Into button (or F7) to run the startup code for the program. The program counter now appears to the left of **main**, indicating that the debugger has run the startup code and is ready to start execution with this line.



To position the text selector on a line in the Source view. press Ctrl+L, type the line number, and press Enter.

**F**7

Look at the left margin of the Source view. You see diamonds indicating lines that generated executable code. To see line numbers, click the Source view, press *Ctrl+O*, and choose Show Line Numbers. Now line numbers appear in the left margin.

As you can see from the Run menu, there are a number of ways to control the execution of your program. Let's say you want to run the program until it reaches line 40.



First, position the text selector on line 40, then press F4 to run the program up to (but not including) line 40. Now press F7, which executes one line of source code at a time and enters into any functions called; in this case, it

executes line 40, a call to the function **showargs**. The cursor immediately jumps to line 167, where the definition of **showargs** is found.

Ctrl F8

Continuing to press *F7* would step through the function **showargs** and then return to the line following the call—line 41. Instead, press *Ctrl+F8*, which causes **showargs** to execute and then return, at which point the program stops. This command, too, returns to line 41 and is very useful when you want to run past the end of a function.

If you had pressed F8 (or used the Statement Step Over button on the

executes a function or source line, but skips any function calls.

SpeedBar) instead of F7 on line 40, the program counter would have gone directly to line 41 instead of into the function. F8 is similar to F7 in that it



Figure 3.2 Program stops on return from function showargs

32	*Z
33 ♦ i	nt main(int argc, char <b>**</b> argv) {
34	unsigned int nlines, nwords, wordcount;
35	unsigned long totalcharacters;
36	Real-sector and an end of the sector of the se
37 *	nlines = 0:
38 <b>o</b>	nwords = 0:
39 0	totalcharacters = 0:
40 *	showargs(argc, argv);
410	
42 ×	<pre>wordcount = makeintowords(buffer);</pre>
43 *	nwords += wordcount;
44 *	totalcharacters += analyzewords(bu
45 ×	nlines++;
46 *	
47 *	printstatistics(nlines_nwords_totalchara
48 0	printf("\nPress (Enter) when done "):
49 0	gets(rethuf)

To execute the program until a specific program location is reached, you can directly name the function or line number, without moving the text selector to that line in a source file and then running to that point. Press *Ctrl+F9* (or choose Run | Execute To from the Control Panel) to specify a label to run to. A dialog box appears. Type readaline and press *Enter*. The program runs, then stops at the beginning of function **readaline**.

## Setting breakpoints

Ctrl || F9

Another way to control where your program stops running is to set breakpoints. The simplest way to set a breakpoint is with the *F2* key. Move the text selector to line 45 and press *F2*. Turbo Debugger GX puts a blue *B* in a box to the left of the line, indicating there is a breakpoint set on it.

You can also use the mouse to toggle breakpoints by clicking near the diamond to the left of a line of source code.

Figure 3.3 A breakpoint set at line 45

🗄 ana ao alaba	and generated with the set of the set of the first of the second set of the set of the second set of the set of
35 36	unsigned long totalcharacters;
37 ×	
38 ×	nvords = 0;
39 *	totalcharacters = 0;
40 ↔	showargs(argc, argv);
41 >	<pre>while (readaline() != 0) {</pre>
42 🗢	<pre>wordcount = makeintowords(buffer);</pre>
43 <b>o</b>	nwords += wordcount;
<b>44</b> $\diamond$	totalcharacters += analyzewords(buf
45 ◊ B	ALCORES THE
46 《 圓	}
47 <b>*</b>	printstatistics(nlines, nwords, totalcharac
48 ↔	<pre>printf("\nPress <enter> when done.");</enter></pre>
49 <b>*</b>	gets(retbuf);
50 *	return(0);
51 }	
52	
E Blit Windersteinering	>

Notice the small box containing horizontal lines to the left of line 46. This stack glyph indicates the next line that will execute after a return from a procedure call. It appears at the end of this **while** loop because you previously ran the program to **readaline**, which is called by this **while** statement.

Now press *F9* to execute your program without interruption. The focus switches to the program's display. The demo program is now running and waiting for you to enter a line of text. Click the application window, and type abc, a space, def, and then press *Enter*. The display returns to the Source view with the arrow on line 45, where your breakpoint has stopped the program.

Now press *Ctrl+E* to disable the breakpoint. You see the capital *B* change to a lowercase *b*, indicating that the breakpoint is still set (preserving any filters, conditions, and actions) but is disabled.



Ctrl || E

F9

See page 25 for more information on breakpoints. The Online Help also provides a complete description of setting and using breakpoints, under "Tasks."

#### Using watches



The Watch view shows the value of variables you specify. For example, to watch the value of the variable *nwords*, move the text selector to the variable name on line 43, choose Add Watch from the Source view local menu (or press either *Ctrl+F7* or *Ctrl+W*), then press *Enter* to accept that expression.

Figure 3.4 A variable in the Watch view

Watch:	nwords 2	Y
		and a standard a second second

The symbol *nwords* now appears in the Watch view, along with its value. As you execute the program, Turbo Debugger GX updates this value to reflect the variable's current value.

If you pass out of the variable's scope (for example, if you continue statement-stepping-into and step into the **readaline** function), the Watch view shows the variable as undefined. As soon as the variable is back in scope (for example, you statement-step through **readaline**, enter another line of characters, and step back into the **while** loop containing *nwords*), you can see its value again.

### Examining simple C data objects

Once you have stopped your program, there are a number of ways of looking at data using the Inspector view. This facility lets you examine data structures in the same way you visualize them when you write a program.

With the Inspector view (available from all local menus and from the View menu), you can examine any variable you specify. Suppose you want to look at the value of the variable *nlines*. Double-click *nlines* in the Source view: an Inspector view pops up with *nlines* in it.

Figure 3.5 An Inspector window

E	In second s	ЦЦ
33 0	int main(int argc, char **argv) {	12
34	unsigned int nlines, nwords, wordcount;	80
35	unsigned long totalcharacters;	
36		
37 0	nlines = 0;	
38 0	nvords = 0:	-
39 0	totalcharacters = 0;	
40 0	Shovenes (Argo Argo)	3
41 <b>o</b>	while (readaline() != 0) {	
42 >	wordcount = makeintowords(buffer);	
43 0	nvords += wordcount:	
44 0	totalcharacters += analyzewords(buffer):	
45 *	nlines++:	
46 0	19 19년~1977년 642일 · 661년 61일 전 1971년 11월 11월 12월 12월 12월 12월 12월 12월 12월 12월	1
47 0	printstatistics(plines_pwords_totalcharacters);	
48 0	printf("\nPress (Enter) when done ");	1
19 0	rate(rathuf)	
50 0	return(0):	~

The address, type, and name of the variable are listed on the first line and its value on the second. Because *nlines* has been optimized into a register variable, its address is the EDI register.

To examine a data item that isn't conveniently displayed in the Source view, choose View | Inspector. The Inspector view appears, asking you to enter the expression to inspect. Type letterinfo and press *Enter*. The

Inspector view lists the values of the *letterinfo* array elements. The first line of the list shows the address, type, and name of the data you're inspecting. Scroll through the 26 elements that make up the *letterinfo* array. The next section shows you how to examine this compound data object.

#### Examining compound data objects

A compound data object, such as an array or structure, contains multiple components. Double-click the fourth element of the *letterinfo* array (the one indicated by [3]). A new Inspector view appears, showing the contents of that element in the array.

Figure 3.6 Inspecting a structure

0x00020BD0	struct linfo[26] let	terinfo	
[U] [1]	🔆 Inspector		• 🗆
[2] [3] [4] [5]	0x00020BE8 count firstletter	struct linfo 1 1	letterinfo.[3]
(6) (7) (8) (9)	{0,0} {0,0} {0,0}		
(10) (11) (12)	(0,0) (0,0) (0,0)		

When you double-click one of the member names, it appears in yet another Inspector view. If one of these members was in turn a compound data object, you could double-click it and dig down further into the data structure.

Now return to the Source view by clicking on it.

Changing data values

So far, you've learned how to look at data in the program. Now you'll see how to *change* the value of data items.

Use the mouse to go to line 39 in the source file. Double-click the variable *totalcharacters* to inspect its value. With the Inspector window open, right-click to bring up the Inspector's local menu, then choose the Change Value option. (You could also have done this directly by pressing *Ctrl+G*.) A dialog box appears, asking for the new value.

Figure 3.7 The Change Value dialog box

	Change Value	
Expression	n: letterinfo[3].count + 4	

At this point, you can enter any C expression that evaluates to a number. Type totalcharacters + 4 and press *Enter*. The value in the Inspector window now shows the new value, 10.

You can also use the Inspector view to change the value of a structure or array member. For example, if you double-click *letterinfo*, then double-click the fourth element of the array (element [3]), you get an Inspector view for *count* and *firstletter*. If you double-click *count*, you get an Inspector view for that single element. If you then press *Ctrl-G*, you can change the value of that element.

To change a data item that isn't displayed in the Source view, choose View | Evaluator (or press *Ctrl+F4*). A dialog box appears. In the Expression entry field, enter the name of the variable to change. Type nlines, press *Enter*, then press *Tab* to move to the New Value entry field. Type 123 and press *Enter*. The Result field shows 123.

🕃 Evalu	altor 🗖 🗖
Expression:	nlines 👔
New Value:	123
Result:	123

You can use the Evaluator view to change values of complex data types, but you can change only one element at a time. For example, to change the fourth element of *letterinfo* (*letterinfo*[3]), you must bring up the Evaluator and change *letterinfo*[3].*count* first, then change *letterinfo*[3].*firstletter*.

Conclusion

Figure 3.8 The Evaluator view

That's a quick introduction to using the Turbo Debugger GX with a character-mode program written using Borland C++ for OS/2. If you're interested in Presentation Manager debugging, try using the demo program TDDEMOPM and playing around both with the features mentioned in this chapter and with the Messagepoint and Exceptionpoint views.

Another view you might find useful for general debugging is the Disassembly view, which shows disassembled code and can simultaneously show you registers, flags, the stack, and memory contents. You can also use this view on code you have no source or no debugging information for. (Note that you have to select the local menu display options to select these displays.) See page 31 for more information, or see the Online Help topic "Disassembly View."



For more information on debugging tasks, see the Online Help topics "Essentials" and "Tasks." For more information on the Turbo Debugger GX environment, see Chapter 2, "The Turbo Debugger GX environment" or the Online Help topics "Menus" and "Views."

Turbo Debugger GX for OS/2 User's Guide

Ρ

E N

. 1

D

### Х

A

# Turbo Debugger GX for experienced Turbo Debugger users

The following table lists major tasks you can perform when you're debugging, and shows you the commands or keystrokes to accomplish each task in Turbo Debugger for DOS (or Windows) and in Turbo Debugger GX.

#### Table A.1: Turbo Debugger GX task list

Task	TD for DOS or TDW	Turbo Debugger GX
Access the local menu of a view	Right mouse click or <i>Ctrl+F10</i>	Right mouse click or Ctrl+F10.
Add a breakpoint and change characteristics	Breakpoints At ( <i>Alt+F2</i> ) View Breakpoints Add	Set Breakpoint. View Breakpoint.
Animate	Run Animate	Run Animate.
Back Trace	Run Back Trace (Alt+F4)	No equivalent.
Change directories	File Change Dir	File Load Process.
Check the value that a function is about to return	Data Function Return	No equivalent.
Close a file	File Open	File Unload Process.
Copy from current window to Log window	Edit Copy to Log Edit Dump Pane to Log	No equivalent.
Delete all breakpoints	Breakpoints Delete All	Breakpoint view local menu  All Breakpoints Remove all Breakpoints.
Display Breakpoints view	View Breakpoints	View Breakpoint.
Display CPU view	View CPU	View Disassembly. View Call Stack. View Memory. View Register.
Display Dump view	View Dump	View Memory.

Appendix A, Turbo Debugger GX for experienced Turbo Debugger users

51

Table A.1: Turbo Debugger GX task list (continued)

Display Execution History view	View Execution History	No equivalent.
Display File view	View File	View File.
Display Log view	View Log	View Log.
Display Module (source) view	View Module (F3)	View Source.
Display Numeric Processor view	View Numeric Processor	View Numeric Processor.
Display Registers view	View Registers	View Register.
Display Stack view	View Stack	View Call Stack.
Display Variables view	View Variables	View Variable.
Display Watches view	View Watches	View Watch.
Display Windows Messages view	View Windows Messages	View Messagepoint.
Enable/Disable all breakpoints	No equivalent	Breakpoint view local menu  All Breakpoints Enable/ Disable All Breakpoints.
Enable/Disable breakpoint	Not available	With cursor on breakpoint position in Source or Disassembly view, choose local menu Enable/Disable breakpoint.
Evaluate or modify data	Data Evaluate/Modify (Ctrl+F4)	View Evaluator ( <i>Ctrl+F4</i> ).
Execute to a specified location	Run Execute to (Alt+F9)	Run Execute To ( <i>Ctrl+F9</i> ).
Execute until current routine returns	Run Until Return ( <i>Alt+F8</i> )	Run Return from Function ( <i>Ctrl+F8</i> ).
Exit program	File Exit ( <i>Alt+X</i> )	File Exit ( <i>Alt+X</i> ).
Inspect a variable	Position cursor in Module window, then press <i>Ctrl+I</i> or choose local menu Inspect Or, choose View Inspect	Position cursor in Source view, then double-click, press <i>Ctrl+I</i> , or choose local menu Inspect Value. Or, choose View Inspector.
Look at window messages returned to application	Windows Messages window, lower pane View Log (if sent to Log window)	In Messagepoint view details pane, choose Log Expression. Then View Log to see window messages.
Look at Windows local heap, module list, or global heap	View Log (local menu) Display Windows Info	No exact equivalent. View Heap and View Module are similar to Windows local heap and module list. No equivalent for global heap.

#### Table A.1: Turbo Debugger GX task list (continued)

Open a file Pick a module to view

Reload application program

Run application program

Run to current location

Set arguments before running application

Set characteristics of breakpoint

Set message breakpoint

Set watchpoint, tracepoint, or hardware breakpoint

Step into routine by instructions

Step into routine by statements

Step over routine by instructions

Step over routine by statements

Toggle breakpoint on and off at cursor in source or assembly language view

Watch data

File|Open View|Module (*F3*) Run|Program Reset (*Ctrl+F2*)

Run|Run (F9)

Run|Go To Cursor (F4)

Run|Arguments

Breakpoints|At (*Ctrl+F2*) View|Breakpoints|Set options

View Windows Messages

Breakpoints|Changed Mem. Glb. Breakpoints|Expn. True Global Breakpoints|Hdw. Breakpoint

Run|Trace Into (F7) in CPU window

Run|Trace Into (F7) in Module window

Run|Step Over (F8) in CPU window

Run|Step Over (F8) in Module window

Breakpoints|Toggle (F2)

Data|Add Watch (*Ctrl+F7*) View|Watches Or, in Module window, press *Ctrl+W* or choose Watch from local menu File|Load Process.

View|Module (F3).

Run|Reset (*Ctrl+F2*). Press SpeedBar button.

Run|Run (*F9*). Press SpeedBar button.

Source View local menu| Run to Here (*F4*).

Run|Arguments.

Set|Breakpoint. Pick it from list of breakpoints.

Set|Messagepoint.

Set|Datapoint.

Run|Instruction Into (*F11*). Press SpeedBar button.

Run|Statement Into (*F7*). Press SpeedBar button.

Run|Instruction Over (*F12*). Press SpeedBar button.

Run|Statement Over (*F8*). Press SpeedBar button.

F2 or double-click (if on left line marker). Local menu|Set/Remove Breakpoint.

Set|Add Watch (*Ctrl+F7*). View|Watch or, in Source view, press *Ctrl+W* or choose Add Watch from local menu.

Turbo Debugger GX for OS/2 User's Guide

## Index

-? command-line option 9

### A

accessing Help 11 local menus 51 actions breakpoints 25 datapoints 26 exceptionpoints 27 addresses functions 23 variables 34 altering control flags 38 control word, numeric processor 38 flags 39 memory 37 NPX tag word 38 registers CPU 38 numeric processor 38 status flags 38 status word, numeric processor 38 variable values 16 animate 51 application window, activating with SpeedBar 20 applications arguments, setting 53 loading 52 reloading 53 running 53 unloading 51 arguments, setting, for application 53 arrays 47 changing elements 48 Evaluator view 33 inspecting, C tutorial 46 values, changing 33 Variable view 34 ASCII, memory representation 37

assembly code current location 31 program counter glyph 31 viewing 31

### B

Borland license agreement 7 Borland Assembler, versions compatible with Turbo Debugger GX 1 Borland C, versions compatible with Turbo Debugger GX 1 Breakpoint view 25 detail form (figure) 23 displaying 51 Help information 26 list and view forms 23 list form (figure) 24 breakpoints 25, See also control points disabling 45, 52 Disassembly view glyph 31 enabling 52 hardware, setting 53 Help information 26 logging 40 running programs to 45 setting 53 characteristics 53 executable line glyph 30 simple 25 tutorial 44 Source view glyph 30 TDDEMO (figure) 45 toggling 53 window message, setting 53 buttons See also SpeedBar context-sensitive Help 12

### С

C++ exception stack view 39 C++ exceptionpoint view 28 C++ exceptions 28 -c command-line option 9 Call Stack view 36 displaying 51, 52 Help information 36 changing array values 33 complex variables Evaluator view 48 Inspector view 48 control flags 38 control word 38 data 48 directories 51 flags 39 memory 37 modules 53 NPX tag word 38 registers CPU 38 numeric processor 38 status flags 38 status word 38 structure values 33 variable values 16 tutorial 47 check boxes, dialog boxes 21 clearing log 40 memory 37 registers, CPU 39 closing files 51 code current location assembly 31 source 30 files, viewing 39 source executable line glyph 30 viewing 30 startup debugger setting 42 running in tutorial 43 views 24 code pointer See program counter glyph command-line options table of 9

Turbo Debugger GX 8 utilities 4 comparing TD DOS and TD GX debuggers 2, 52 compatibility requirements 1 compiling programs 16 conditions, filter 25 configuring, debugger 21 contents Help, displaying with SpeedBar 20 summary 6 Contents, Help panel 12 Contents choice, Help menu 11 context-sensitive Help 12 control flags, numeric processor 38 Control Panel 17 File menu 19 Help menu 11, 19 Menu-bar choices (table) 19 Properties dialog box (figure) 21 Run menu 19 Set menu 19 SpeedBar (figure) 19 SpeedBar buttons (tables) 20 status line 21 TDDEMO (figure) 42 Threads pane (figure) 20 view (figure) 18 View menu 19 Window menu 19 control points 24, See also breakpoints; datapoints; exceptionpoints; messagepoints defined 24 logging 40 reusing 20 saving 43 control word, numeric processor 38 conventions, typographic 5 coprocessor, numeric See numeric processor CPU registers 38 CPU view, displaying 51 creating log file 40 *Ctrl+E* (disable breakpoint), Source view 45 Ctrl+F4 (evaluate/change) 48 Ctrl+F8 (return from function) 44 Ctrl+F9 (run to expression) 44 Ctrl+F7 (watch) 45 Ctrl+H (Help key) 10

Ctrl+l (change), Inspector view shortcut 47 Ctrl+l (inspect), Source view shortcut 45, 46 Ctrl+l key 12 Ctrl+O (display options), Source view 43 Ctrl+W (watch), Source view shortcut 45 current instruction 31 location, running to 53 statement 30

### D

data changing 48, 52 types, compound 47 watching 53 Datapoint view 26 Help information 27 datapoints 26, See also control points Help information 27 logging 40 Source view 31 Variable view 34 debugger configuring 21 exiting 19 properties, setting 21 settings, startup code 42 debugging 15 comparison, TD GX and TD 51 compound variables 47 defined 15 features 1 files required 1 modules 32 source files and 1 steps 15 task list (table) 51 tasks 15 threads 20 tools 15 tutorial, Help 43 decrementing CPU registers 39 deleting breakpoints Disassembly view 31 Source view 31 datapoints 26

log 40 messagepoints 29 demo programs 4, See also tutorial Help 43 reloading 43 source files 41 starting 42 TDDEMO 41 demos, starting (figure) 42 detail form 22 Datapoint view 26 Exceptionpoint view 27, 28 list of views 23 Variable view 34 Watch view 35 dialog boxes 21 check boxes, using 21 context-sensitive Help 12 defined 21 entry fields, using 21 Properties 21 Properties (figure) 21 radio buttons, using 21 diamond, Source view glyph 30 directories changing 51 source file search order 2 disabling breakpoints 45, 52 Disassembly view 31 Source view 31 datapoints 26 logging 40 messagepoints 29 Disassembly pane, Disassembly view (figure) 32 Disassembly view 31 breakpoint glyph 31 displaying 51 Help information 32 panes 31 panes (figure) 32 program counter glyph 31 setting breakpoints 25 disks, distribution 3 display form Memory view 37 Numeric Processor view 38

displaying local menus 22, 51 views 51 window messages 52 distribution disks 3 .DLL files reading with TDUMP 4 viewing 32 DLLs, viewing 39 Dump view, displaying 51 dumping files 4

### E

editing text 16 elements, structure, changing 48 enabling breakpoints 52 Disassembly view 31 Source view 31 datapoints 26 logging 40 messagepoints 29 entering Turbo Debugger GX 8 entry fields context-sensitive Help 12 dialog boxes 21 erasing log 40 registers, CPU 39 error messages, status line 21 Essentials choice, Help menu 11 evaluating, data 48 Evaluator view 33 changing values, complex variables 48 Help information 33 Exceptionpoint view 27 exceptionpoints 27, See also control points Help information 27, 28 logging 40 .EXE files reading with TDUMP 4 viewing 39 exiting debugger 8, 19, 43, 52 expressions See also variables addresses 34 datapoints 26 entering in dialog boxes 48

inspecting 33 lists 34 messagepoints 28 modifying 52 types 34 values, changing Evaluator view 33 Inspector view 34 Watch view 35 views 24 watching 35

### F

F1 (Help) 10 F9 (run application) 45 F4 (run to here) 43 F2 (set breakpoint) 44 F7 (statement step into) 43 F8 (statement step over) 44 F1 help key 10 features comparison with DOS Turbo Debugger 2 Turbo Debugger GX 1 File menu 19 File | Properties 21 File view 39 displaying 52 Help information *39* FILELIST.DOC 3 files closing 51 debugging, required for 1 demo programs source 41 TDDEMO 41 dumping 4 File view 39 FILELIST.DOC 3 **INSTALL.EXE 7** log 40 MANUAL.TD 4 memory, reading into 37 memory, writing from 37 Online Help 6 online text 3 opening 52 program 3

reading, TDUMP 4 README.TD 3 source, search order 1 TDDEMO.C 41 TDDEMO.EXE 4 TDDEMOPM.EXE 4 TDUMP.EXE 4 UTILS.TD 4 viewing 39 filters breakpoints 25 conditions 25 datapoints 26 exceptionpoints 27 messagepoints 28 flags, toggling 39 Flags pane, Disassembly view (figure) 32 forms C++ exceptionpoint view list form 28 Datapoint view detail form 26 list form 26 display, Numeric Processor view 38 Exceptionpoint view detail form 27, 28 list form 27 Inspector view, list form 33 list and detail 22 Memory view display 37 Messagepoint view detail form 28 list form 29 Variable view detail form 34 list form 34 views 22 breakpoint example 23 default 23 Watch view detail form 35 list form 35 functions return, running until 52 returning from 44 stepping into 15 SpeedBar 20

stepping over 15 SpeedBar 20 window, tracking messages 28

### G

Glossary choice, Help menu 11

### Η

-h command-line option 9 hardware breakpoints, setting 53 Call Stack view 36 requirements computer 1 numeric processors 1 views 24 header files, viewing 39 heap, viewing 52 Heap view 36 Help information 36 Help breakpoints 26 Call Stack view 36 command-line options, Turbo Debugger GX utilities datapoints 27 demo programs 43 Disassembly view 32 Evaluator view 33 exceptionpoints 27, 28 File view 39 Heap view 36 Inspector view 34 local menu choices 26 Log view 40 Memory view 38 messagepoints 29 Modules view 33 Numeric Processor view 38 Register view 39 Source view 31 SpeedBar 20 Variable view 35 Watch view 35 Help menu 11, 19

Help panels contents 12 printing 13 Help system 10 accessing 11 Contents panel 12 context-sensitive Help 12 Help window, using 11 icon in manual 5 index 12 printing 13 searching index 12 strategies for access 11 using 10 Help window, using 11 hiding views SpeedBar 20 Window menu 19 hot keys See shortcut keys

icons in manual 5 Help 5 notes 5 incrementing CPU registers 39 index, Help system 12 Index choice, Help menu 11 Inspect command 46 inspecting expressions 33 variables 33, 46-47, 52 compound 47 Inspector view 33 changing values complex variables 48 tutorial 47 Help information 34 scrolling 46 tutorial 46-47 inspectors Evaluator view 33 Inspector view 34 Source view 31 Variable view 34 Watch view 35 **INSTALL.EXE** 7 installing Turbo Debugger GX 7 instructions current 31 stepping by into functions 20 over functions 20

### Κ

keys hot See shortcut keys shortcut See shortcut keys

### L

labels, running programs to, tutorial 44 license agreement, Borland 7 line numbers, displaying current 43 list form 22 C++ exceptionpoint view 28 Datapoint view 26 Exceptionpoint view 27 Inspector view 33 list of views 23 Messagepoint view 29 Variable view 34 Watch view 35 listing window messages 52 loading modules 53 processes 52 TDDEMO (figure) 42 local menus 22 displaying 51 panes, Disassembly view 32 shortcut keys 22 locations current, running to 53 program, specifying 23 specified, running to 52 Log view 40 displaying 52 Help information 40 logging control points 40 datapoints 26 exceptionpoints 27, 28 messagepoints 29

### М

-m command-line option 9 MANUAL.TD 4 math coprocessor See numeric processor memory 37 changing 37 clearing 37 heap 36 moving 37 searching 37 viewing 37 writing 37 Memory pane See also Memory view Disassembly view (figure) 32 Memory view 36, 37, See also Memory pane displaying 51 Help information 38 Menu-bar choices (table) 19 menus context-sensitive Help 12 File 19 Help 11, 19 local 22 displaying 51 panes, Disassembly view 32 Run 19 Set 19 View 19 Window 19 Menus choice, Help menu 11 Messagepoint view 28 displaying 52 messagepoints 28, See also control points Help information 29 logging 40 setting 53 messages status line 21 window displaying 52 listing 52 tracking 28 modifying data 48 variable values, tutorial 47 variables 52

Module view displaying 52, 53 Help information 33 modules changing 53 multiple, viewing 33 viewing 32, 52 Modules view 32 mouse, setting breakpoints 44 moving memory 37

### N

NPX tag word, numeric processor 38 hardware requirements 1 Numeric Processor view 38 displaying 52 Help information 38

### 0

.OBJ files, reading with TDUMP 4 online text files 3 opening files 52 log file 40 TDDEMO (figure) 42 options, command-line Turbo Debugger GX 8 Turbo Debugger GX utilities 4 OS/2, versions, compatible 1

### Ρ

panels, Help contents 12 printing 13 panes Disassembly view 31 Disassembly view (figure) 32 local menu 32 performing tasks in 32 Register, changing values 32 passing exceptions 27, 28 PID 20 pointer, code See program counter popping up, local menus 22 printing Help information 13 process loading 19 resetting Run menu 19 SpeedBar 20 stopping Run menu 19 SpeedBar 20 unloading 19 process ID 20 processes loading 52 unloading 51 Product Information choice, Help menu 11 program arguments, setting 53 locations, specifying 23 resetting Run menu 19 SpeedBar 20 running Run menu 19 SpeedBar 20 stopping Run menu 19 SpeedBar 20 program counter glyph Disassembly view 31 Source view 30 program files 3 programs compiling 16 current location 43 debugging 15 exceptions, receiving 27, 28 loading 52 recompiling 16 reloading 53 running 53 to breakpoints 45 to labels 44 to text selector 43 startup code 42 stepping through, tutorial 44 unloading 51 properties, setting 19, 21 Properties dialog box (figure) 21

### Q

quitting, debugger 19, 52

### R

-r command-line option 10 radio buttons, dialog boxes 21 reading files 39 TDUMP 4 memory 37 README.TD file 3 recompiling programs 16 register, numeric processor 38 Register pane changing register values 32 Disassembly view (figure) 32 Register view 38 reloading programs 53 SpeedBar 20 removing breakpoints Disassembly view 31 Source view 31 datapoints 26 messagepoints 29 resetting, programs 53 Run menu 19 SpeedBar 20 reusing control points 20 watches 20 Run menu 19 running animate 51 to current location 53 until function return 52 programs 53 Run menu 19 setting arguments 53 SpeedBar 20 startup code 42 to specified location 52

### S

-s command-line option *10* sample programs *See* demo programs; tutorial

saving control points 20, 43 watches 20, 43 window positions, Window menu 19 scope, Watch view 46 scrolling, Inspector view 46 search order, source files 2 searching File view 39 Help system index 12 memory 37 Source view 31 Set menu 19 setting breakpoints 53 Disassembly view 31 hardware 53 Help information 26 simple 25 Source view 31 tutorial 44 window message 53 control points, Set menu 19 datapoints 26, 27 exceptionpoints 27, 28 messagepoints 29, 53 tracepoints 53 watches 35, 53 watchpoints 53 setting properties 21 Shift+F1 (Help key) 10 Shift+F1 key 12 shortcut keys 22 Ctrl+F4 (evaluate/change) 48 Ctrl+F8 (return from function) 44 Ctrl+F9 (run to expression) 44 Ctrl+F7 (watch) 45 Disassembly view 31 Evaluator view 33 F1 (Help) 10 F9 (run application) 45 F4 (run to here) 43 F2 (set breakpoint) 44 F7 (statement step into) 43 F8 (statement step over) 44 File view 39 Inspector view 34

Ctrl+G (change) 47 Log view 40 Memory view 37 Register view 39 Source view 31 Ctrl+E (disable breakpoint) 45 Ctrl+l (inspect) 45, 46 Ctrl+O (display options) 43 Ctrl+W (watch) 45 Variable view 34 Watch view 35 showing local menus 22 showing views SpeedBar 20 Window menu 19 software requirements 1 source code current location 30 executable line glyph 30 program counter glyph 30 viewing 30 source files, required for source debugging 1 source modules multiple, viewing 33 viewing 32 Source view 30 breakpoint glyph 30 displaying 52 executable line glyph 30 Help information *31* modules, opening 32 program counter 43 program counter glyph 30 setting breakpoints 25 stepping, tutorial 43 TDDEMO (figure) 42 specifying properties 21 specifying variable values 16 SpeedBar buttons (table) 20 (figure) 19 Statement Step Into button, tutorial 43 Stack pane, Disassembly view (figure) 32 starting programs 42 TDDEMO (figure) 42 starting Turbo Debugger GX 8

startup code, running debugger setting 42 tutorial 43 statement, current 30 Statement Step Into button, tutorial 43 statements stepping by into functions 20 over functions 20 tutorial 43 status flags, numeric processor 38 status line Control Panel 21 SpeedBar buttons 19 status word, numeric processor 38 stepping into functions 15 by instruction 20 by statement 20 over functions 15 by instruction 20 by statement 20 return from function 44 tutorial 44 Run menu 19 by statement, tutorial 43 stopping program Run menu 19 SpeedBar 20 stopping Turbo Debugger GX 8 structures changing elements Evaluator view 48 Inspector view 48 Evaluator view 33 values, changing 33 Variable view 34 summary of manual contents 6 switches, command-line Turbo Debugger GX 8 Turbo Debugger GX utilities 4 switching, application window, SpeedBar 20

### Т

tag word, NPX 38 task list (table) 51 Tasks choice, Help menu 11 TDDEMO See also tutorial starting (figure) 42 TDDEMO.C 41 TDDEMO.EXE 4 TDDEMOPM.EXE 4 TDUMP.EXE 4 terminating Turbo Debugger GX 8 text editors, compatibility with Turbo Debugger GX 16 text files, online 3 text selector, running programs to, tutorial 43 thread ID 20 threads, debugging 20 Threads pane (figure) 20 TID 20 toggling breakpoints 53 flags 39 tracepoints, setting 53 Turbo Debugger for DOS, task comparison (table) 51 Turbo Debugger GX comparison with DOS Turbo Debugger 2 features 1 files 3 task comparison (table) 51 tutorial 41 changing nlines 48 totalcharacters 47 inspecting letterinfo 46 nwords 45.46 totalcharacters 47 labels, running programs to 44 return from function 44 stepping over functions 44 types, variables 34 typographic conventions 5

### U

unloading a process 51 user window, activating with SpeedBar 20 Using Help choice, Help menu 11 utilities disk-based documentation for 4 TDUMP 4

#### UTILS.TD 4

### ۷

values, changing expressions Evaluator view 33 Inspector view 34 Watch view 35 variables Evaluator view 33 Inspector view 34 Watch view 35 Variable view 34 displaying 52 Help information 35 variables See also expressions addresses 34 arrays 34 changing values 33 changing values 16 complex 34 changing values 33 compound, inspecting 47 datapoints 26 inspecting 33, 46-47, 52 lists 34 messagepoints 28 modifying 52 return values 47-48 scope, Watch view 46 structures 34 changing values 33 types 34 values, changing Evaluator view *33* Inspector view 34 Watch view 35 views 24 watching 35, 45, 53 View menu 19 viewing window messages 52 views 22 Breakpoint 25 detail form (figure) 23 displaying 51 list form (figure) 24

breakpoint, forms 23 C++ exception stack view 39 C++ exceptionpoint 28 Call Stack 36 displaying 51, 52 code information 24 Control Panel 17 Control Panel (figure) 18 control point 24 CPU, displaying 51 Datapoint 26 defined 22 Disassembly 31 displaying 51 Dump, displaying 51 Evaluator 33 tutorial 48 Exceptionpoint 27 File 39 displaying 52 forms breakpoint example 23 default 23 list and detail 22 hardware information 24 Heap 36 hiding SpeedBar 20 Window menu 19 Inspector 33 list of, two display forms 23 Log 40 displaying 52 Memory 36 displaying 51 Memory (figure) 37 Messagepoint 28 displaying 52 Module 32 displaying 52, 53 Numeric Processor 38 displaying 52 Register 38 shortcut keys 22 showing SpeedBar 20 Window menu 19
Source 30 displaying 52 modules, opening 32 TDDEMO (figure) 42 Source (figure) 30 Variable 34 displaying 52 variable information 24 Watch 35 displaying 52 VIEWs choice, Help menu 11

## W

Watch view 35 displaying 52 Help information 35 using, tutorial 45 watches Inspector view 34 reusing 20 saving 43 setting 53 Source view 31 Variable view 34

Watch view 35 watching expressions 35 variables 35 watchpoints setting 53 tutorial 45 Window menu 19 windows application, activating with SpeedBar 20 context-sensitive Help 12 Help, using 11 hiding SpeedBar 20 Window menu 19 messages breakpoints 53 displaying 52 listing 52 tracking 28 saving positions, Window menu 19 showing SpeedBar 20 writing memory 37

## Ζ

zeroing, registers, CPU 39





Corporate Headquarters: 100 Borland Way, Scotts Valley, CA 95066-3249, (408) 431-1000. Offices in: Australia, Belgium, Canada, Chile, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Latin America, Malaysia, Netherlands, New Zealand, Singapore, Spain, Sweden, Taiwan, and United Kingdom • Part #BCP1415WW21774 • BOR 7004

