

Turbo C[®]++ for Windows
Version 3.0

User's Guide

Copyright © 1991 by Borland International. All rights reserved. All Borland products are trademarks or registered trademarks of Borland International, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders. Windows, as used in this manual, shall refer to Microsoft's implementation of a windows system.

R1

10 9 8 7 6 5 4 3 2 1

C O N T E N T S

Introduction	1	Chapter 3 Managing multi-file projects	23
What's in Turbo C++ for Windows	1	Sampling the project manager	24
Hardware and software requirements	2	Error tracking	27
The Turbo C++ for Windows implementation	3	Stopping a make	27
The Turbo C++ for Windows package	3	Syntax errors in multiple source files	28
The User's Guide	3	Saving or deleting messages	28
The Programmer's Guide	4	Autodependency checking	29
Installing Turbo C++ for Windows	5	Overriding libraries	30
Starting Turbo C++ for Windows	5	More Project Manager features	30
Customizing Turbo C++	6	Looking at files in a project	32
The README file	6	Chapter 4 Options reference	33
Typefaces used in these books	7	The Application Options dialog box	33
How to contact Borland	8	Compiler	35
Resources in your package	8	Code Generation	35
Borland resources	8	Advanced Code Generation	37
Chapter 1 IDE basics	11	Entry/Exit Code Generation	38
Starting Turbo C++ for Windows	11	C++ Options	40
Command-line options	12	Optimizations	42
Command sets	12	Source	43
Using the SpeedBar	14	Messages	44
Configuration and project files	15	Names	45
The configuration file	15	Make	45
Project files	16	Linker	47
The project directory	16	Librarian	50
Desktop files	17	Directories	51
Changing project files	17	Environment	52
Default files	17	Preferences	52
Chapter 2 Using the ObjectBrowser	19	Editor	53
Browsing through classes	20	Mouse	55
Browsing through functions	21	Desktop	55
Browsing through variables	22	Save	56
Inspecting symbols in your code	22	Appendix A Converting from Microsoft C	57
		Environment and tools	57

Paths for .h and .LIB files	58	Appendix C Using EasyWin	69
Source-level compatibility	58	DOS to Windows made easy	69
__MSC macro	58	_InitEasyWin()	70
Header files	59	Appendix D Precompiled headers	71
Memory models	59	How they work	71
Keywords	59	Drawbacks	72
Floating-point return values	60	Using precompiled headers	72
Structures returned by value	60	Setting file names	72
Conversion hints	61	Establishing identity	73
Appendix B Editor reference	63	Optimizing precompiled headers	73
Block commands	65	Index	75
Other editing commands	67		

T A B L E S

1.1: General hot keys	12	B.1: Editing commands	63
1.2: Editing hot keys	13	B.2: Block commands in depth	66
1.3: Online Help hot keys	13	B.3: Borland-style block commands	67
1.4: Compiling/Running hot keys	13	B.4: Other editor commands in depth	67

F I G U R E S

4.1: The Application Options dialog box . . .	34	4.6: The Optimization Options dialog box	42
4.2: The Code Generation Options dialog box	35	4.7: The Make dialog box	46
4.3: The Advanced Code Generation dialog box	37	4.8: The Linker Settings dialog box	47
4.4: The Entry Exit Code Generation dialog box	39	4.9: The Link Libraries dialog box	49
4.5: The C++ Options dialog box	41	4.10: The Librarian Options dialog box . . .	50
		4.11: The Editor Options dialog box	53

One of the advantages of Turbo C++ is that it eases you into C++ and Windows programming.

Turbo C++ is highly compatible with existing Turbo C code.

Turbo C++ for Windows provides you with AT&T's C++ version 2.1, true ANSI C compatibility, and a rich programming environment in Windows. It's for Windows, C, and C++ programmers who want a fast, efficient compiler; for Turbo Pascal programmers who want to learn C or C++ with all the "Turbo" advantages; and for anyone just learning C or C++.

C++ is an object-oriented programming (OOP) language, the next step in the natural evolution of C. You can use C++ for almost any programming task, anywhere, but it's just right for Windows.

We're also including EasyWin, a brand new library that makes it easy to compile and run DOS applications under Windows with no changes. You can start learning how to program in C, and try out features of C++ as you learn. Or, you can jump into full C++ Windows programming right away.

What's in Turbo C++ for Windows

Turbo C++ includes the latest features you need:

To find out how to install Turbo C++ see page 5.

- **C++:** Turbo C++ offers you the full power of C++ programming (implementing C++ version 2.1 from AT&T). To help you get started, we're also including C++ class libraries. To keep you going, we've added templates.
- **Streams:** Special Borland extensions to the streams library functions allow you to position text, set screen attributes, and perform other manipulations to streams within the Windows environment.
- **IDE:** Turbo C++ *Windows-hosted* integrated development environment (IDE) provides completely integrated Windows programming without ever leaving the Windows environment. The Turbo C++ for Windows IDE also includes:

- built-in ObjectBrowser that lets you visually explore your class hierarchies, functions and variables, locate inherited function and data members, and instantly inspect any element you select.
 - visual SpeedBar for instant point-and-click access to the currently available tools and functions whenever you are in the IDE.
 - online context-sensitive hypertext Help for the IDE, the C/C++ language, the Windows API, and the complete library reference. Practically every function comes with a copy-and-paste program example.
 - many indispensable library functions, including heap checking and memory management functions and a complete set of complex and BCD math functions.
- **ANSI C:** Turbo C++ provides you with an up-to-date implementation of the latest ANSI C standard.
 - **EasyWin:** lets you turn standard DOS applications using printf, scanf and other standard I/O functions into Windows applications *without changing a single line of code*. Just select Windows application in the IDE, and your DOS program runs in a window!
 - **Precompiled headers:** speed up program compilation time.
 - **Protected-mode compilation:** gives you increased capacity with no swapping.
 - **Templates:** (parameterized types), allow you to create families of “generic” classes or functions with arguments whose types are not known in advance. This gives you much of the flexibility of so-called “pure” (untyped) OOPS languages without sacrificing the efficiency of C++.
 - **DLLs:** support for creation of Dynamic Link Libraries.
 - **Built-ins:** assembler, linker, and librarian.

Hardware and software requirements

Turbo C++ runs on the IBM PC family of computers, using the 286, 386 (386 SX or 386 DX), or higher processor. Turbo C++ requires Windows 3.0, DOS 3.0 or higher, 640K RAM and at least 2 MB of extended memory; it runs on any Windows-compatible

monitor (EGA or higher). The minimum requirement is a hard disk drive and one floppy drive.

We recommend using a Windows-compatible mouse, but Turbo C++ does not require one.

The Turbo C++ for Windows implementation

Turbo C++ is a full implementation of the AT&T C++ version 2.1 and supports templates. It is also American National Standards Institute (ANSI) C standard. In addition, Turbo C++ includes certain extensions for mixed-language and mixed-model programming that let you exploit your PC's capabilities. See Chapters 1 through 4 in the *Programmer's Guide* for a complete description of Turbo C++.

The Turbo C++ for Windows package

Your Turbo C++ package consists of a set of distribution disks and two manuals. This book, the *User's Guide*, tells you how to use the product. The second book, the *Programmer's Guide*, focuses on programming in C, C++, and Windows.

The distribution disks contain all the programs, files, and libraries you need to create, compile, link, and run your Turbo C++ programs; they also contain sample programs, a context-sensitive Help file, and additional C and C++ documentation not covered in the manuals.

The User's Guide

This volume introduces you to Turbo C++ for Windows and shows you how to create and run both C and C++ programs under Windows. There's information you'll need to get up and running quickly: about installation, basics of the IDE, the editor, and Project Manager. Also, we explore ways to convert existing code and your other options.

This introduction tells you how to install Turbo C++ for Windows on your system. You'll customize it online.

Chapter 1: IDE basics introduces the features of the IDE, giving information and examples of how to use it to the fullest.

Chapter 2: Using the browser guides you in exploring class hierarchies and properties.

Chapter 3: Managing multi-file projects shows the way to use the Project Manager to manage multi-file programming projects.

Chapter 4: Options reference details the Options dialog boxes for access to powerful programming choices.

Appendix A: Converting from Microsoft C gives hints about converting Microsoft C code to Turbo C++.

Appendix B: Editor reference provides a complete reference to the editor.

Appendix C: Using EasyWin explains the quick way to make DOS programs run under Windows.

Appendix D: Precompiled headers discusses the advantages of generating and using preserved header files.

The Programmer's Guide

The *Programmer's Guide* provides useful material for the experienced C user: a complete language reference for C and C++, C++ streams, memory models, mixed-model programming, floating-point issues, BASM and inline assembly concerns, error messages, and Help compiler.

Chapters 1 through 4: Lexical elements, Language structure, C++ specifics, and The preprocessor, describe the Turbo C++ for Windows language.

Chapter 5: Using C++ streams tells you how to use the C++ version 2.1 stream library.

Chapter 6: Math discusses long doubles parameters, floating-point, and BCD math.

Chapter 7: BASM and inline assembly introduces language-specific features of the assembler.

Chapter 8: Building a Windows application gets you started programming in Windows.

Chapter 9: Error messages lists and explains all run-time, compile-time, and help compiler errors and warnings, with suggested solutions.

Appendix A: HC: The Windows Help compiler introduces the Windows Help Compiler.

Installing Turbo C++ for Windows

Turbo C++ comes with an automated installation program called INSTALL. You should use INSTALL to load Turbo C++ onto your system, because it will ensure that you get all the files you need into the places where you need them. INSTALL will automatically create directories and Program Manager groups and copy files from the distribution disks to your hard disk.

Important!

INSTALL assumes that the Program Manager starts up automatically as your Windows "shell" when you open Windows. If you are using a shell other than Program Manager, be sure to turn off the Installation Option "Create TurboC Group."

If you are not using drive A, replace drive A and WIN A with drive B and WIN B (or another letter) in the following procedures:

If you have Windows in your path, you can start INSTALL from the DOS command line.

1. Insert your Turbo C++ installation disk into drive A.
2. Type `WIN A:INSTALL` and press *Enter*.
3. Set options in the dialog box that appears.
4. Choose Install and INSTALL begins copying files.

If Windows is not in your path, you can start Windows first and follow these steps:

1. Insert your Turbo C++ installation disk into drive A.
2. Choose File | Run in the Windows Program Manager.
3. Type `A:INSTALL` and choose OK.
4. Set options in the dialog box that appears.
5. Choose Install and INSTALL begins copying files.

Starting Turbo C++ for Windows

There are two ways to start Turbo C++. In Windows, you double-click the Turbo C++ icon in the Program Manager, or you can select it with your keyboard and press *Enter*.

Customizing Turbo C++

The integrated development environment (IDE) allows you to customize the way Turbo C++ for Windows behaves by selecting options and preferences without exiting the program to use external utilities.

The IDE saves the options you have set so that the IDE is just as you left it the next time you start up Turbo C++. If you don't want this to happen, choose Preferences from the Options menu to open the Preferences dialog box and uncheck Auto Save options, Configuration, and Desktop. Choose OK.

The README file

The README file contains last-minute information that may not be in these manuals. It guides you to such online information as FILELIST.DOC, which lists every file on the distribution disks and describes what each file contains.

Here's how to access the README file:

1. Insert your Turbo C++ for Windows installation disk into drive A.
2. At the DOS prompt, type `A:` and press *Enter*.
3. Type `README` and press *Enter*. Once you're in README, use the *PgUp*, *PgDn*, *↑* and *↓* keys to scroll through the file.
4. Press *Esc* to exit.

After you've installed Turbo C++, you can open README in an edit window by following these steps:

1. Start Turbo C++ for Windows.
2. Choose Open from the File menu. In the following instruction, *TCWIN* is the name of the base directory in which you installed TCW. Type `\TCWIN\README.` (don't forget the `.`) in the input box and choose OK. The README file opens in an edit window.
3. When you're done with the README file, choose Close from the Window menu or press *Alt+F4* to exit Turbo C++.

Typefaces used in these books

The typefaces are used as follows:

- Monospace** This typeface represents text as it appears onscreen or in a program. It is also used for anything you must type (such as `TCW` to start up Turbo C++).
- ALL CAPS** We use all capital letters for the names of constants and files, except for header files, which are traditionally represented in all lowercase letters.
- []** Square brackets in text or command lines enclose optional items that depend on your system. *Text of this sort should not be typed verbatim.*
- < >** Angle brackets in the code examples and reference sections enclose the names of include files (such as `<include strng.h>`) or template type declarations (such as `template <class T>`).
- Boldface** Turbo C++ function names (such as `printf`), reserved words (such as `char`, `switch`, `near`, and `cdecl`), and structure names are shown in boldface when they appear in text (but not in program examples). This typeface is also used for format specifiers and escape sequences (`%d`, `\t`), and for command-line options (`/A`).
- Italics** Italics indicate variable names (identifiers) that appear in text. They can represent terms that you can use as is, or that you can think up new names for (your choice, usually). They are also used to emphasize certain words (especially new terms).
- Keycaps** This typeface indicates a key on your keyboard. It is often used to describe a particular key you should press; for example, "Press *Esc* to exit a menu."



This icon indicates keyboard actions.



This icon indicates mouse actions.



This icon indicates language items that are specific to C++. It is used primarily in the *Programmer's Guide*.



Language items that are specific to Windows.

How to contact Borland

Borland offers a variety of services to answer your questions about this product. Be sure to send in the registration card; registered owners are entitled to technical support and may receive information on upgrades and supplementary products.

Resources in your package

This product contains many resources to help you:

- The manuals provide information on every aspect of the program. Use them as your main information source.
- While using the program, you can press *F1* for help.
- Many common questions are answered in the DOC files listed in the README file located in the program directory. FILELIST.DOC shows you what files you have and what directories they're in.

Borland resources

800-822-4269 (voice)
Techfax

Borland Technical Support publishes technical information sheets on a variety of topics and is available to answer your questions.

TechFax is a 24-hour automated service that sends free technical information to your fax machine. You can use your touch-tone phone to request up to three documents per call.

408-439-9096 (modem)
File Download BBS
2400 Baud

The Borland File Download BBS has sample files, applications, and technical information you can download with your modem. No special setup is required.

Subscribers to the CompuServe, GENie, or BIX information services can receive technical support by modem. Use the commands in the following table to contact Borland while accessing an information service.

Service	Command
CompuServe	GO BORLAND
BIX	JOIN BORLAND
GEnie	BORLAND

Address electronic messages to Sysop or All. Don't include your serial number; messages are in public view unless sent by a service's private mail system. Include as much information on the question as possible; the support staff will reply to the message within one working day (in most cases).

408-438-5300 (voice)
Technical Support
6 a.m. to 5 p.m. PST

Borland Technical Support is available weekdays from 6:00 a.m. to 5:00 p.m. Pacific time to answer any technical questions you have about Borland products. Please call from a telephone near your computer, and have the program running. Keep the following information handy to help process your call:

- Product name, serial number, and version number.
- The brand and model of any hardware in your system.
- Operating system and version number. (Use the DOS command `VER` to find the version number.)
- Contents of your `AUTOEXEC.BAT` and `CONFIG.SYS` files (located in the root directory (\) of your computer's boot disk).
- The contents of your `WIN.INI` and `SYSTEM.INI` files (located in your Windows directory).
- A daytime phone number where you can be contacted.
- If the call concerns a problem, the steps to reproduce the problem.

408-438-5300 (voice)
Customer Service
7 a.m. to 5 p.m. PST

Borland Customer Service is available weekdays from 7:00 a.m. to 5:00 a.m. Pacific time to answer any non-technical questions you have about Borland products, including pricing information, upgrades, and order status.

IDE basics

Turbo C++ for Windows integrated development environment, or the IDE for short, has everything you need to write, edit, compile, and link your programs. You can even start up the powerful Turbo Debugger for Windows without leaving the IDE.

Turbo C++ has a Windows-hosted IDE based on Windows Multiple Document Interface (MDI). If you are familiar with other Windows programs, you'll feel right at home with the Turbo C++ IDE.

This chapter tells you how to start Turbo C++, explains about the IDE's two command sets, tells you how to use the SpeedBar, and explains how the IDE manages projects with its configuration, project, and desktop files.

Starting Turbo C++ for Windows

As with other Windows products, you double-click the Turbo C++ icon in the Program Manager to start Turbo C++.

If you have more than one project, you might want to create an icon for each project. Here's how to create a project icon:

- Choose File | New in the Programmer Manager.
- Select Program Item and the New Program Object dialog box appears.

- Type in a description for your project, and, in the command-line text box, type `TCW` followed by the project file name including the full path.

Now when you double-click the icon in the Program Manager, your project will load into Turbo C++.

Command-line options

You can specify two command-line options when you start Turbo C++: `/b` for building a project or `/m` for doing a make on a project. To specify either of these options,

- Select the Turbo C++ icon in the Program Manager.
- Choose File | Run.
- Add the command-line option you want to the command line in the command-line text box and include the full path. Choose OK.

When you use either of these options, your messages are written to a file named the same as your project file except it carries the extension `.MSG`. For example, if your project file is `MYPROJ.PRG`, the message file is `MYPROJ.MSG`.

Command sets

Turbo C++ has two command sets: the Common User Access (CUA) command set used by most Windows programs and the Alternate command set. The menu shortcuts available to you differ depending on which command set you use. You can select a command set by choosing Options | Preferences and then selecting the command set you prefer in the Preferences dialog box.

If you are a long-time Borland language user, you may prefer the Alternate command set.

The following tables list the most-used Turbo C++ hot keys in both command sets.

Table 1.1: General hot keys

CUA	Alternate	Menu item	Function
	<i>F2</i>	<u>F</u> ile <u>S</u> ave	Saves the file that's in the active edit window.
	<i>F3</i>	<u>F</u> ile <u>O</u> pen	Brings up a dialog box so you can open a file.
<i>Alt+F4</i>	<i>Alt+X</i>	<u>F</u> ile <u>E</u> xit	Exits Turbo C++.
<i>Alt+Space</i>	<i>Alt+Space</i>	(none)	Takes you to the Control menu.

Table 1.2: Editing hot keys

CUA	Alternate	Menu item	Function
<i>Ctrl+Ins</i>	<i>Ctrl+Ins</i>	<u>E</u> dit <u>C</u> opy	Copies selected text to Clipboard.
<i>Shift+Del</i>	<i>Shift+Del</i>	<u>E</u> dit <u>C</u> ut	Places selected text in the Clipboard, deletes selection.
<i>Shift+Ins</i>	<i>Shift+Ins</i>	<u>E</u> dit <u>P</u> aste	Pastes text from the Clipboard into the active window.
<i>Ctrl+Del</i>	<i>Ctrl+Del</i>	<u>E</u> dit <u>C</u> lear	Removes selected text from the window and doesn't put it in the Clipboard.
<i>Alt+Bksp</i>	<i>Alt+Bksp</i>	<u>E</u> dit <u>U</u> ndo	Restores the text in the active window to a previous state.
<i>Alt+Shft+Bksp</i> <i>F3</i>	<i>Alt+Shft+Bksp</i> <i>Ctrl+L</i>	<u>E</u> dit <u>R</u> edo <u>S</u> earch <u>S</u> earch Again	"Undoes" the previous Undo. Repeats last Find or Replace command.

Table 1.3: Online Help hot keys

CUA	Alternate	Menu item	Function
<i>Shift+F1</i>	<i>Shift+F1</i>	<u>H</u> elp <u>I</u> ndex	Brings up Help index.
<i>Ctrl+F1</i>	<i>Ctrl+F1</i>	<u>H</u> elp <u>T</u> opic Search	Calls up language-specific help in the active edit window.

Table 1.4: Compiling/Running hot keys

CUA	Alternate	Menu item	Function
<i>Alt+F7</i>	<i>Alt+F7</i>	<u>S</u> earch <u>P</u> revious Error	Takes you to previous error.
<i>Alt+F8</i>	<i>Alt+F8</i>	<u>S</u> earch <u>N</u> ext Error	Takes you to next error.
<i>Ctrl+F9</i>	<i>Ctrl+F9</i>	<u>R</u> un <u>R</u> un	Runs program.
<i>F9</i>	<i>F9</i>	<u>C</u> ompile <u>M</u> ake	Invokes Project Manager to make an .EXE, .DLL, or .LIB file.
<i>Alt+F9</i>	<i>Alt+F9</i>	<u>C</u> ompile <u>C</u> ompile	Compiles file in active edit window.

Native makes the Alternate command set the default for Borland C++ and the CUA command set the default for Turbo C++.

If you choose Options | Preferences to display the Preferences dialog box, you'll notice a third command set option: Native. This is the default setting.

If you write applications for Windows, you may do some of your development with Borland C++ and some with Turbo C++ for Windows. Both IDEs use the same configuration file, TCCONFIG.TC. Therefore, if you have selected the CUA command set for Turbo C++, that will be the one in effect the next time you start up Borland C++.

But maybe this isn't what you want. When you're working with the DOS product, Borland C++, you might prefer the Alternate command set, and when you use Turbo C++ for Windows, you

might want to use the CUA command set. The Native option lets you do this.

With Native selected, Borland C++ uses the Alternate command set automatically, and Turbo C++ uses the CUA command set.

Using the SpeedBar

To reconfigure the SpeedBar, choose Options | Environment | Desktop and select the option you want.

Turbo C++ for Windows has a SpeedBar you can use as a quick way to choose menu commands with your mouse. The first time you start Turbo C++ for Windows, the SpeedBar is a horizontal group of buttons beneath the menu bar. You can use it as it is, make it a vertical bar on the left side of the Turbo C++ desktop window, or make it a pop-up palette you can move anywhere on your screen. You can also turn it off.

The buttons on the SpeedBar represent menu commands. They are shortcuts for your mouse, just as certain key combinations are shortcuts on your keyboard. To choose a command, click a button with your mouse. If you click the File | Open button, for example, Turbo C++ responds just as if you chose Open on the File menu.

The SpeedBar is context sensitive. The buttons that appear on it depend on which window is active: the Turbo C++ desktop window, an edit window, the Project window, or the Message window.

These are the buttons on the SpeedBar:



Help



Search again



Open a file



Cut to Clipboard



Save file



Copy to Clipboard



Search for text



Paste from Clipboard



Undo



View include files



Compile



Add item to project



Make



Delete item from project



Edit source file



View file with error



Exit Turbo C++

Some buttons on the SpeedBar are occasionally dimmed and, just like some of the menu commands, are unavailable to you. For example, the Paste button is dimmed if there is nothing in your Clipboard.

Configuration and project files

With configuration files, you can specify how you want to work within the IDE. Project files contain all the information necessary to build a project, but don't affect how you use the IDE.

The configuration file

The configuration file, TCCONFIG.TC, contains only environmental (or global) information. The information stored in TCCONFIG.TC file includes

- editor key binding and macros
- editor mode setting (such as autoindent, use tabs, and so on)
- mouse preferences
- auto-save flags
- command set

When you start a programming session, Turbo C++ looks for TCCONFIG.TC first in the current directory and then in the directory that contains TCW.EXE. If Turbo C++ doesn't find TCCONFIG.TC, it creates it in the directory containing TCW.EXE.

Project files

The IDE places all information needed to build a program into a binary project file, a file with a .PRJ extension. Project files contain information on all other settings and options, including

- a list of all files that make up the project
- compiler, linker, make, and librarian options
- directory paths

In addition, the project file contains other general information on the project, such as compilation statistics (shown in the project window), and cached autodependency information.

You can load project files in several ways:

- From within the IDE, you load a project file using the Project | Open Project command.
- When starting Turbo C++ with the Program Manager File | Run command, enter the project name with the .PRJ extension along with the full path after you type the TCW command; for example,

```
TCW c:\tcwin\examples\myproj.PRJ
```

You must use the .PRJ extension to differentiate it from source files.

- If there is only one .PRJ file in the current directory, the IDE assumes that this directory is dedicated to this project and automatically loads it.

If you start Turbo C++ from the Windows 3.0 Program Manager, the Program Manager will make the directory where WIN.COM is kept the current directory. You can work around this by creating an icon for each project. Then when you choose a project icon, the directory the project is in becomes the current directory.

See page 11 for details about creating project icons.

The project directory

When a project file is loaded from a directory other than the current directory, the current DOS directory is set to where the project is loaded from. This allows your project to be defined in terms of relative paths in the Options | Directories dialog box and

also allows projects to move from one drive to another or from one directory branch to another.

Desktop files Each project file has an associated desktop file (*prjname.DSK*) that contains state information about the associated project. The desktop file includes

You can set some of these options on or off using Options | Environment | Desktop.

- the context information for each file in the project (for example, the position in the file)
- the history lists for various input boxes (for example, search strings, file masks, and so on)
- the layout of the windows on the desktop
- the contents of the Clipboard

Changing project files Because each project file has its own desktop file, changing to another project file causes the newly loaded project's desktop to be used, which can change your entire window layout. When you create a new project (by using Project | Open Project and typing in a new .PRJ file), the new project's desktop inherits the previous desktop. When you select Project | Close Project, the default project is loaded and you get the default desktop and project settings.

Default files When no project file is loaded, there are two default files that serve as global place holders for project- and state-related information: TCDEFW.DPR and TCDEFW.DSK files, collectively referred to as the *default project*.

Default files are updated only if a project is not loaded.

These files are usually stored in the same directory as TCW.EXE, and are created if they are not found. When you run the IDE from a directory without loading a project file, you get the desktop and settings from these files. These files are updated when you change any project-related options (for example, compiler options) or when your desktop changes (for example, the window layout) and no project is open.

Using the ObjectBrowser

The ObjectBrowser is a programming tool that takes full advantage of the Windows graphical environment. It lets you visually browse through your class hierarchies, functions, and variables. With the ObjectBrowser, you can

- graphically view the class hierarchies in your application, then select the class of your choice and view the functions and data elements it contains.
- list all the functions in your application, then select a function to view its declaration or go to where it is defined in your source code.
- list all the variables in your application, then select a variable to view its declaration or go to where it is defined in the source code.
- select a class, function, or variable in your source code, then view its details.



Before you use the ObjectBrowser, you *must* compile your program so that debugging information is included in your executable file. If you include any libraries or object files in your application, they must be compiled with debugging information on also.

If your executable is composed of more than one source code file, open the related project file in the IDE before using the ObjectBrowser.



You can also activate the ObjectBrowser by choosing an option on the Browse menu.

If you have a mouse, you'll find browsing through your code most convenient if you set up your right mouse button to activate the ObjectBrowser. Then you can click a class, function, or variable right in your source code and inspect it (view its details). To set up your mouse this way, choose Options | Environment | Mouse and select the Browse option.

The ObjectBrowser has buttons on the bar at the top of the ObjectBrowser window. Choose them by clicking them with your mouse or by using specific key combinations. By choosing one of these buttons, you tell the ObjectBrowser to perform some action. These are the buttons you will see, their keyboard equivalents, and the action they perform:



F1 Help.



Ctrl+G Go to the source code for the selected item.



Ctrl+I Inspect (view the details of) the selected item.



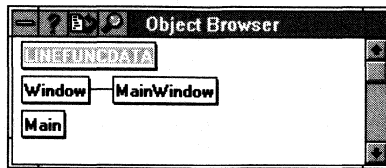
Ctrl+R Rewind the ObjectBrowser to the previous view.



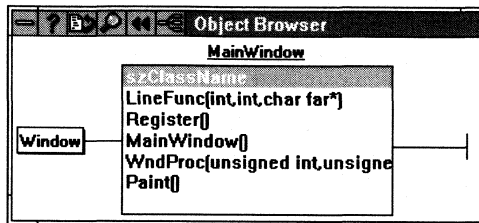
Ctrl+O Show an overview of the class hierarchy.

Browsing through classes

The ObjectBrowser lets you see the “big picture,” the class hierarchies in your application, as well as the small details. To activate the ObjectBrowser and see your classes displayed graphically, choose Browse | Classes. The ObjectBrowser draws your classes and shows their parent-child relationships in a horizontal tree.



To see more detail about a particular class, double-click it. If you aren't using a mouse, select the class by using your arrow cursor keys, and press *Enter*. The ObjectBrowser lists the symbols (the functions and variables) used in the class.



To see the declaration of a particular listed symbol, you can

- double-click the symbol.
- select it and click the Inspect button or choose Browse | Inspect (*Ctrl+I*).
- select it and press *Enter*.

To go to how the class is defined in your source code, choose the Go To Source Code button or Browse | Goto (*Ctrl+G*).

Any time you want to return to a higher level, click the Rewind button or choose Browse | Rewind (*Ctrl+R*).

Browsing through functions

Choosing Browse | Functions opens a window that lists every function in your application in alphabetical order. Class member functions are listed together by class (for example, MyClass::Myfunc).

Click the function you want more information about or use your cursor keys to select it. A text box at the bottom of the window lets you quickly search through the function list by typing the first few letters of the function name. As you type, the selections in the list change to match the characters you type.

Once you select the function you are interested in, you can

- choose the Inspect button or Browse | Inspect (*Ctrl+I*) to see the declaration of the function.
 - choose the Go To Source Code button or Browse | Goto (*Ctrl+G*) to see how the function is defined in the source code.
-

Browsing through variables

Just as you can list all your functions, you can list all your variables. Choose Browse | Variables and a window listing all the variables in your application opens.

Click the variable you want more information about or use your cursor keys to select it. Just as with the function list, you can search through the variable list by typing the first few letters of the variable name in the text box at the bottom of the window.

Once you have selected the variable you are interested in, you can

- choose the Inspect button or Browse | Inspect (*Ctrl+I*) to see the declaration of the variable.
 - choose the Go To Source Code button or Browse | Goto (*Ctrl+G*) to see how the variable is defined in the source code.
-

Inspecting symbols in your code

You can also inspect any symbol in your code without wading through class hierarchies or function and variable lists. There are two ways to inspect a symbol:

- Highlight the symbol in your code and choose Browse | Symbol at Cursor.
- Click the symbol in your code.

If the symbol you highlight or click is a class, the ObjectBrowser shows you all the functions and variables in that class. You can then choose to inspect any of these further.

Managing multi-file projects

Since most programs consist of more than one file, having a way to automatically identify those that need to be recompiled and linked would be ideal. Turbo C++'s built-in Project Manager does just that and more.

The Project Manager allows you to specify the files belonging to the project. Whenever you rebuild your project, the Project Manager automatically updates the information kept in the project file. This project file includes

- all the files in the project
- where to find them on the disk
- the header files for each source module
- which compilers and command-line options need to be used when creating each part of the program
- where to put the resulting program
- code size, data size, and number of lines from the last compile

Using the Project Manager is easy. To build a project,

- pick a name for the project file (from Project | Open Project)
- add source files using the Project | Add Item dialog box
- tell Turbo C++ to Compile | Make

Then, with the project-management commands available on the Project menu, you can

- add or delete files from your project

■ view included files for a specific file in the project

All the files in this chapter are in the Examples directory.

Let's look at an example of how the Project Manager works.

Sampling the project manager

Suppose you have a program that consists of a main source file, MYMAIN.CPP, a support file, MYFUNCS.CPP, that contains functions and data referenced from the main file, and myfuncs.h. MYMAIN.CPP looks like this:

This program runs in Windows because it uses the EasyWin library automatically.

```
#include <iostream.h>
#include "myfuncs.h"

main(int argc, char *argv[])
{
    char *s;
    if (argc > 1)
        s=argv[1];
    else
        s="the universe";
    cout << GetString() << s << "\n";
}
```

MYFUNCS.CPP looks like this:

```
char ss[] = "The restaurant at the end of ";

char *GetString(void)
{
    return ss;
}
```

And myfuncs.h looks like this:

```
extern char *GetString(void);
```

These files make up the program that we'll now describe to the Project Manager.

These names can be the same (except for the extensions), but they don't have to be. The name of your executable file (and any map file produced by the linker) is based on the project file's name.

The first step is to tell Turbo C++ the name of the project file that you're going to use: Call it MYPROG.PRJ. Notice that the name of the project file is not the same as the name of the main file (MYMAIN.CPP). And in this case, the executable file will be MYPROG.EXE (and if you choose to generate it, the map file will be MYPROG.MAP).

Go to the Project menu and choose Open Project. This brings up the Open Project File dialog box, which contains a list of all the files in the current directory with the extension .PRJ. Since you're starting a new file, type in the name MYPROG in the Open Project File input box.

Notice that once a project is opened, the Add Item, Delete Item, and Include Files options are enabled on the Project menu.

If the project file you load is in another directory, the current directory will be set to where the project file is loaded.

You can keep your project file in any directory; to put it somewhere other than the current directory, just specify the path as part of the file name. (You must also specify the path for source files if they're in different directories.) Note that all files and corresponding paths are relative to the directory where the project file is loaded from. After you enter the project file name, you'll see a Project window.

The Project window contains the current project file name (MYPROG). Once you indicate which files make up your project, you'll see the name of each file and its path. When the project file is compiled, the Project window also shows the number of lines in the file and the amount of code and data in bytes generated by the compiler.

The SpeedBar shows which actions can be performed at this point: you can get help, add a file to the Project, delete a file from the Project, view include files required by a file in the Project, open an editor window for the currently selected file, compile a selected file in the project or build the entire project. Press the Add Item to Project button now to add a file to the project list.

*You can change the file-name specification to whatever you want with the Name input box; *.CPP is the default.*

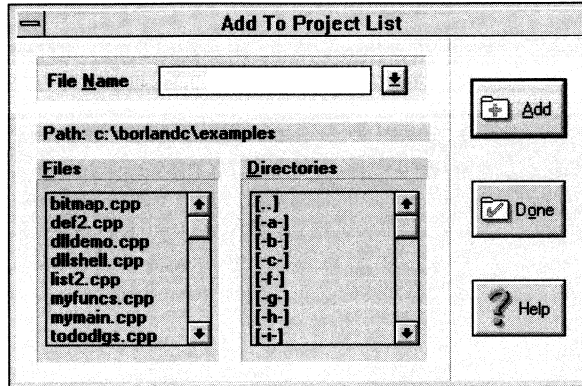
The Add to Project List dialog box appears; this dialog box lets you select and add source files to your project. The Files list box shows all files with the .CPP extension in the current directory. (MYMAIN.CPP and MYFUNCS.CPP both appear in this list.) Three action buttons are available: Add, Done, and Help.

If you copy the wrong file to the Project window, press Esc to return to the Project window, then Del or press the Delete Item button on the SpeedBar to remove the currently selected file.

Since the Add button is the default, you can place a file in the Project window by typing its name in the Name input box and pressing *Enter* or by choosing it in the Files list box and choosing OK. You'll see that MYFUNCS gets added to the Project window and then you're returned to the Add Item dialog box to add another file. Go ahead and add MYMAIN.CPP. Turbo C++ will compile files in the exact order they appear in the project.

Note that the Add button commits your change; pressing Esc when you're in the dialog box just puts the dialog box away.

Close the dialog box and return to the Project window. Notice that the Lines, Code, and Data fields in the Project window show n/a. This means the information is not available until the modules are actually compiled.



After all compiler options and directories have been set, Turbo C++ will know everything it needs to know about how to build the program called MYPROG.EXE using the source code in MYMAIN.CPP, MYFUNCS.CPP, and myfuncs.h. Now you'll actually build the project.

Choose Compile | Make or press the Make button on the SpeedBar to make your project and choose Run | Run to run it. When you are done viewing your output, close the window.

When you leave the IDE, the project file you've been working on is automatically saved on disk; you can disable this by unchecking Project in the Preferences dialog box (Options | Environment).

For more information on .PRJ and .DSK files, refer to the section, "Project and configuration files," in Chapter 1.

The saved project consists of two files: the project file (.PRJ) and the desktop file (.DSK). The project file contains the information required to build the project's related executable. The build information consists of compiler options, INCLUDE/LIB/OUTPUT paths, linker options, and make options. The desktop file contains the state of all windows at the last time you were using the project.

The next time you use Turbo C++, you can jump right into your project by reloading the project file. Turbo C++ automatically loads a project file if it is the only .PRJ file in the current directory; otherwise the default project and desktop (TCDEFW.*) are loaded. Since your program files and their corresponding paths are

relative to the project file's directory, you can work on any project by moving to the project file's directory and bringing up Turbo C++. The correct file will be loaded for you automatically. If no project file is found in the current directory, the default project file is loaded.

Error tracking

Syntax errors that generate compiler warning and error messages in multifile programs can be selected and viewed from the Message window.

To see this, let's introduce some syntax errors into the two files, MYMAIN.CPP and MYFUNCS.CPP. From MYMAIN.CPP, remove the first angle bracket in the first line and remove the `c` in **char** from the fifth line. These changes will generate five errors and two warnings in MYMAIN.

In MYFUNCS.CPP, remove the first `r` from `return` in the fifth line. This change will produce two errors and one warning.

Changing these files makes them out of date with their object files, so doing a make will recompile them.

Since you want to see the effect of tracking in multiple files, you need to modify the criterion Turbo C++ uses to decide when to stop the make process. This is done by setting a radio button in the Make dialog box (Options | Make).

Stopping a make

You can choose the type of message you want the make to stop on by setting one of the Break Make On options in the Make dialog box (Options | Make). The default is Errors, which is normally the setting you'd want to use. However, you can have a make stop after compiling a file with warnings, with errors, or with fatal errors, or have it stop after all out-of-date source modules have been compiled.

The usefulness of each of these modes is really determined by the way you like to fix errors and warnings. If you like to fix errors and warnings as soon as you see them, you should set Break Make On to Warnings or maybe to Errors. If you prefer to get an entire list of errors in all the source files before fixing them up, you should set the radio button to Fatal Errors or to Link. To demonstrate errors in multiple files, choose Fatal Errors in the Make dialog box.

Syntax errors in multiple source files

Since you've already introduced syntax errors into MYMAIN.CPP and MYFUNCS.CPP, go ahead and choose Compile | Make to "make the project." The Compiling window shows the files being compiled and the number of errors and warnings in each file and the total for the make. Choose OK when compiling stops.

Your cursor is now positioned on the first error or warning in the Message window. If the file that the message refers to is in the editor, the highlight bar in the edit window shows you where the compiler detected a problem. You can scroll up and down in the Message window to view the different messages.

Note that there is a "Compiling" message for each source file that was compiled. These messages serve as file boundaries, separating the various messages generated by each module and its include files. When you scroll to a message generated in a different source file, the edit window will only track in files that are currently loaded.

Thus, moving to a message that refers to an unloaded file causes the edit window's highlight bar to turn off. Press *Spacebar* to load that file and continue tracking; the highlight bar will reappear. If you choose one of these messages (that is, press *Enter* when positioned on it), Turbo C++ loads the file it references into an edit window and places the cursor on the error. If you then return to the Message window, tracking resumes in that file.

The Source Tracking options in the Preferences dialog box (Options | Environment) help you determine which window a file is loaded into. You can use these settings when you're message tracking.

Note that Previous message and Next message are affected by the Source Tracking setting. These commands will always find the next or previous error and will load the file using the method specified by the Source Tracking setting.

Saving or deleting messages

Normally, whenever you start to make a project, the Message window is cleared out to make room for new messages. Sometimes, however, it is desirable to keep messages around between makes.

Consider the following example: You have a project that has many source files and your program is set to stop on Errors. In this case, after compiling many files with warnings, one error in one file stops the make. You fix that error and want to find out if the compiler will accept the fix. But if you do a make or compile again, you lose your earlier warning messages. To avoid this, check Save Old Messages in the Preferences dialog box (Options | Environment). This way the only messages removed are the ones that result from the files you *recompile*. Thus, the old messages for a given file are replaced with any new messages that the compiler generates.

You can always get rid of all your messages by choosing Compile | Remove Messages, which deletes all the current messages. Unchecking Save Old Messages and running another make will also get rid of any old messages.

Autodependency checking

When you made your previous project, you dealt with the most basic situation: a list of C++ source file names. The Project Manager provides you with a lot of power to go beyond this simple situation.

The Project Manager collects autodependency information at compile time and caches these so that only files compiled outside the IDE need to be processed. The Project Manager can automatically check dependencies between source files in the project list (including files they themselves include) and their corresponding object files. This is useful when a particular C++ source file depends on other files. It is common for a C++ source to include several header files (.h files) that define the interface to external routines. If the interface to those routines changes, you'll want the file that uses those routines to be recompiled.

If you've checked the Auto-Dependencies option (Options | Make), Make obtains time-date stamps for all .CPP files and the files included by these. Then Make compares the date/time information of all these files with their date/time at last compile. If any date/time is different, the source file is recompiled.

If the Auto-Dependencies option is unchecked, the .CPP files are checked against .OBJ files. If earlier .CPP files exist, the source file is recompiled.

When a file is compiled, the IDE's compiler and the command-line compiler put dependency information into the .OBJ files. The Project Manager uses this to verify that every file that was used to build the .OBJ file is checked for time and date against the time and date information in the .OBJ file. The .CPP source file is recompiled if the dates are different.

That's all there is to dependencies. You get the power of more traditional makes while avoiding long dependency lists.

Overriding libraries

In some cases, it's necessary to override the standard startup files or libraries. You override the startup file by placing a file called C0x.OBJ as the *first* name in your project file, where *x* stands for any DOS name (for example, C0MINE.OBJ). It's critical that the name start with C0 and that it is the first file in your project.

To override the standard library, choose Options | Linker and, in the Libraries dialog box, select None for the Standard Run-time Library. Then add the library you want your project to use to the project file just as you would any other item.

More Project Manager features

Let's take a look at some of the other features the Project Manager has to offer. When you're working on a project that involves many source files, you want to be able to easily view portions of those files. You'll also want to be able to quickly access files that are included by others.

For example, expand MYMAIN.CPP to include a call to a function named **GetMyTime**:

```
#include <iostream.h>
#include "myfuncs.h"
#include "mytime.h"

main(int argc, char *argv[])
{
    char *s;

    if(argc > 1)
        s=argv[1];
```

```

else
    s="the universe";
cout << GetString() << s << "\n";
}

```

This code adds one new include file to MYMAIN: mytime.h. Together myfuncs.h and mytime.h contain the prototypes that define the **GetString** and **GetMyTime** functions, which are called from MYMAIN. The mytime.h file contains

```

#define HOUR 1
#define MINUTE 2
#define SECOND 3
extern int GetMyTime(int);

```

Go ahead and put the actual code for **GetMyTime** into a new source file called MYTIME.CPP:

```

#include <time.h>
#include "mytime.h"

int GetMyTime(int which)
{
    struct tm *timeptr;
    time_t secsnow;

    time(&secsnow);
    timeptr = localtime(&secsnow);
    switch (which) {
        case HOUR:
            return (timeptr -> tm_hour);
        case MINUTE:
            return (timeptr -> tm_min);
        case SECOND:
            return (timeptr -> tm_sec);
    }
}

```

MYTIME includes the standard header file time.h, which contains the prototype of the **time** and **localtime** functions, and the definition of *tm* and *time_t*, among other things. It also includes mytime.h in order to define HOUR, MINUTE, and SECOND.

Create these new files, then use Project | Open Project to open MYPROG.PRJ. The files MYMAIN.CPP and MYFUNCS.CPP are still in the Project window. Now to build your expanded project, add the file name MYTIME.CPP to the Project window. Choose Project | Add Item or press the Add Item button on the SpeedBar to bring up the Add Item dialog box. Use the dialog box to specify the name of the file you are adding and choose Done.

Now choose **Compile | Make** to make the project. MYMAIN.CPP will be recompiled because you've made changes to it since you last compiled it. MYFUNCS.CPP won't be recompiled, because you haven't made any changes to it since the make in the earlier example. MYTIME.CPP will be compiled for the first time.

In the MYPROG project window, move to MYMAIN.CPP and press *Spacebar* (or **Project | Include Files**) to display the Include Files dialog box. You can also choose the View Includes button on the SpeedBar. This dialog box contains the name of the selected file, several buttons, and a list of include files and locations (paths). The first file in the Include Files list box is highlighted; the list box lists all the files that were included by the file MYMAIN.CPP. If any of the include files is located outside of the current directory, the path to the file is shown in the Location field of the list box.

As each source file is compiled, the information about which include files are included by which source files is stored in the source file's .OBJ file. If you access the Include Files dialog box before you perform a make, it might contain no files or it might have files left over from a previous compile (which may be out of date). To load one of the include files into an edit window, highlight the file you want and press *Enter* or click the View button.

Looking at files in a project

Let's take a look at MYMAIN.CPP, one of the files in the Project. Double-click the file or select it and press *Enter*. This brings up an edit window with MYMAIN.CPP loaded. Now you can make changes to the file, scroll through it, search for text, or whatever else you need to do. When you are finished with the file, save your changes if any, then close the edit window.

Suppose that after browsing around in MYMAIN.CPP, you realize that what you really wanted to do was look at mytime.h, one of the files that MYMAIN.CPP includes. Highlight MYMAIN.CPP in the Project window, then press *Spacebar* to bring up the Include Files dialog box for MYMAIN. (Alternatively, while MYMAIN.CPP is the active edit window, choose **Project | Include Items**. Now choose mytime.h in the Include Files box and press the View button. This brings up an edit window with mytime.h loaded. When you're done, close the mytime.h edit window.

Options reference



Many of the features of Turbo C++ are controlled by various default settings that appear on the Options menu. You can invoke the Options menu either by clicking Options on the main menu, or by pressing *Alt-O*. Most of the commands in this menu lead to a dialog box.

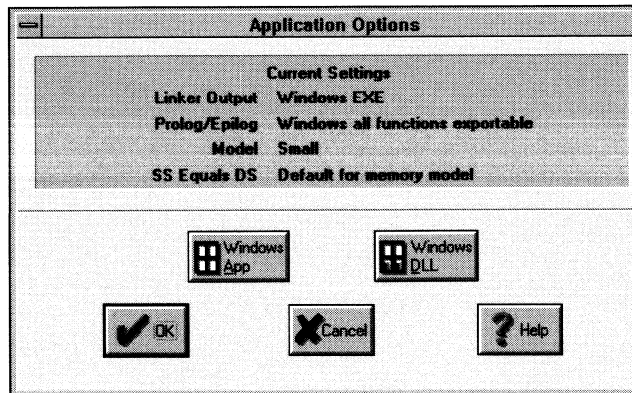
When you first view the settings in any of the options dialog boxes, you will see certain settings are already selected. These are the *default* settings, which Turbo C++ will use if you do not make any changes. These default settings are illustrated in the screen diagrams in this chapter. You can change any of the default settings by making the desired changes and selecting save project on the Options | Save dialog box. Alternatively, if you check the Project box in the Autosave group on the Options | Preferences menu, your changes will be automatically saved when you exit from Turbo C++.

The Application Options dialog box

The Options | Application menu choice brings up the Application Options Dialog box. This dialog box provides the easiest and safest way to set up compilation and linking for a Windows executable. To use this dialog box, simply push one of the buttons. Turbo C++ will verify and, if necessary, change some of the settings in the Code Generation Options, Entry Exit Code Generation, and Linker Settings dialog boxes. See page 38

(Entry/Exit Code Generation) for detailed information on the code generated. Use this dialog box for initial setup only.

Figure 4.1
The Application Options
dialog box



The standard options for applications and libraries each accomplish a set of tasks. You can choose only one button at a time. The current settings fields are updated when you press the button.

Windows App:

- pushes the Small memory model button in the Code Generation Options dialog box
- sets Assume SS equals DS to Default for memory model in the Code Generation Options dialog box
- pushes the Windows All Functions Exportable button in the Entry Exit Code Generation dialog box
- pushes the Windows .EXE button in the Linker Settings dialog box

Windows DLL:

- pushes the Compact memory model button in the Code Generation Options dialog box
- sets Assume SS equals DS to Never in the Code Generation Options dialog box
- pushes the Windows DLL All Functions Exportable button in the Entry Exit Code Generation dialog box
- pushes the Windows .DLL button in the Linker Settings dialog box

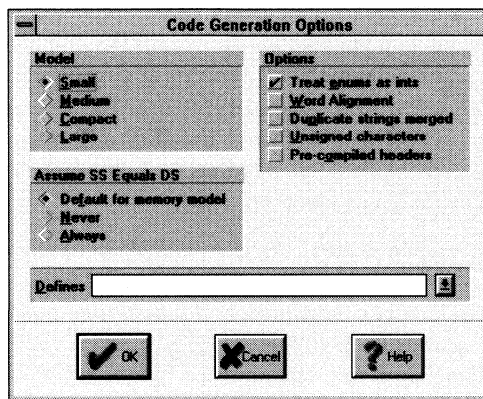
Compiler

The Options | Compiler command displays a pop-up menu that gives you several options to set that affect code compilation. The following sections describe these commands.

Code Generation

The Code Generation command displays a dialog box. The settings in this box tell the compiler to prepare the object code in certain ways. The dialog box looks like this:

Figure 4.2
The Code Generation
Options dialog box



Here are what the various buttons and check boxes mean:

The Model buttons determine which memory model you want to use. The memory model chosen determines the default method of memory addressing. The default memory model is Small. For more information about memory models, see the online file WINMEM.DOC.

The Options control various code generation defaults.

- When checked, Treat enums as ints causes the compiler to always allocate a whole word for variables of type **enum**. Unchecked, this option tells the compiler to allocate an unsigned or signed byte if the minimum and maximum values of the enumeration are both within the range of 0 to 255 or -128 to 127, respectively.
- Word Alignment (when checked) tells Turbo C++ to align non-character data (within structures and unions only) at even addresses. When this option is off (unchecked), Turbo C++ uses

byte-aligning, where data (again, within structures and unions only) can be aligned at either odd or even addresses, depending on which is the next available address.

Word alignment increases the speed with which 80x86 processors fetch and store the data.

- Duplicate Strings Merged (when checked) tells Turbo C++ to merge two strings when one matches another. This produces smaller programs, but can introduce bugs if you modify one string.
- Unsigned Characters (when checked) tells Turbo C++ to treat all **char** declarations as if they were **unsigned char** type.
- Check Precompiled Headers when you want the IDE to generate and use precompiled headers. Precompiled headers can dramatically increase compilation speeds, though they require a considerable amount of disk space. When this option is off (the default), the IDE will neither generate nor use precompiled headers. Precompiled headers are saved in *PROJECTNAME.SYM*.

See Appendix D for more on precompiled headers.

If the Default for Memory Model radio button is pushed, whether the stack segment (SS) is assumed to be equal to the data segment (DS) is dependent on the memory model used. Usually, the compiler assumes that SS is equal to DS in the small and medium memory models (except for DLLs).

When the Never radio button is pushed, the compiler will not assume SS is equal to DS.

The Always button tells the compiler to always assume that SS is equal to DS. It causes the IDE to substitute the C0Fx.OBJ startup module for C0x.OBJ to place the stack in the data segment.

Use the Defines input box to enter macro definitions to the preprocessor. You can separate multiple defines with semicolons (;) and assign values with an equal sign (=); for example,

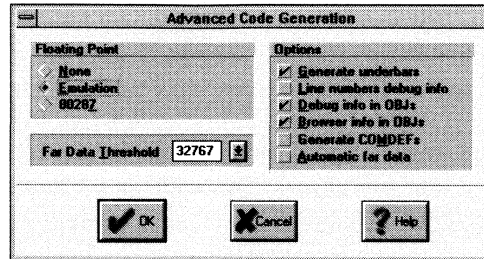
```
TESTCODE;PROGCONST=5
```

Leading and trailing spaces will be stripped, but embedded spaces are left intact. If you want to include a semicolon in a macro, you must place a backslash (\) in front of it.

Advanced Code Generation

The Advanced Code Generation menu choice takes you to the Advanced Code Generation dialog box. Here's what that dialog box looks like:

Figure 4.3
The Advanced Code Generation dialog box



The Floating Point buttons let you decide how you want Turbo C++ to generate floating-point code.

- Choose None if you're not using floating point. (If you choose None and you use floating-point calculations in your program, you get link errors.)
- Choose Emulation if you want your program to detect whether your computer has an 80x87 coprocessor (and to use it if you do). If it is not present, your program will emulate the 80x87.
- Choose 80287 to generate direct 80287 inline code.

The advanced options are shown to the left.

- When checked, the Generate Underbars option automatically adds an underbar, or underscore, character (`_`) in front of every global identifier (that is, functions and global variables). If you are linking with standard libraries, this box must be checked.
- Line Numbers Debug Info (when checked) includes line numbers in the object and object map files (the latter for use by a symbolic debugger). This increases the size of the object and map files but does not affect the speed of the executable program.

Since the compiler might group together common code from multiple lines of source text during jump optimization, or might reorder lines (which makes line-number tracking difficult), you might want to make sure the Jump Optimization check box (Options | Compiler | Optimizations) is off (unchecked) when this option is checked.

- **Debug Info in OBJs** controls whether debugging information is included in object (.OBJ) files. The default for this check box is on (checked), which you need to use either the integrated debugger or the standalone Turbo Debugger.
Turning this option off lets you link and create larger object files. While this option doesn't affect execution speed, it *does* affect compilation time.
- **Fast Floating Point** lets you optimize floating-point operations without regard to explicit or implicit type conversions. When this option is unchecked, the compiler follows strict ANSI rules regarding floating-point conversions.
- **The Fast Huge Pointers** option normalizes huge pointers only when a segment wrap-around occurs in the offset portion of the segment. This greatly speeds up the computation of huge pointer expressions, but must be used with caution because it can cause problems for huge arrays if array elements cross a segment boundary.
- **When checked, the Generate COMDEFs** option allows a communal definition of a variable to appear in header files as long as it is not initialized. Thus a definition such as `int SomeArray[256];` could appear in a header file that is then included in many modules, and the compiler will generate it as a communal variable rather than a public definition (a COMDEF record rather than a PUBDEF record). The linker will then only generate one instance of the variable so it will not be a duplicate definition linker error.
- **The Automatic Far Data** option and the **Far Data Threshold** type-in box work together. When checked, the Automatic Far Data option tells the compiler to automatically place data objects larger than a pre-defined size into far data segments; the Far Data Threshold specifies the minimum size above which data objects will be automatically made far.

This option is ignored if you're using the small or medium memory models.

Entry/Exit Code Generation

See Chapter 8, "Building a Windows application," in the Programmer's Guide for more on prolog and epilog code.

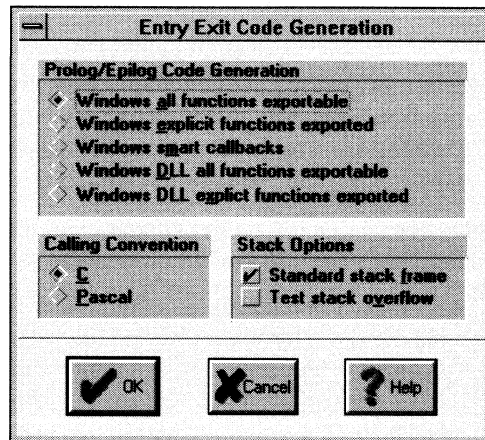
When you compile a C or C++ program for Windows or DOS, the compiler needs to know which kind of prolog and epilog to create for each of a module's functions.

If the program is intended for a Windows DLL, the compiler generates a different prolog and epilog than it would for an EXE. Because of this, you must use the Entry Exit Code Generation dialog box to set the appropriate application. If you use the

Application Options dialog box (described on page 33), the settings in the Entry Exit Code Generation dialog box will already be correct for the type of application you choose.

This dialog box also allows you to select the calling convention and to set a couple of stack options. All options affect what code is generated for function calls and returns.

Figure 4.4
The Entry Exit Code
Generation dialog box



To set the prolog/epilog code for a Windows application, you must select one of five options.

- Windows All Functions Exportable is the most general kind of Windows executable, although not necessarily the most efficient. It assumes that all functions are capable of being called by the Windows kernel or by other modules, and generates the necessary overhead information for every function, whether the function needs it or not. The module definition file will control which functions actually get exported.
- Use Windows Explicit Functions Exported if you have functions that will not be called the Windows kernel. It isn't necessary to generate export-compatible prolog/epilog code information for these functions. The `_export` keyword provides a way to tell the compiler which specific functions will be exported: Only those far functions with `_export` will be given the special Windows prolog/epilog code.
- Push the Windows Smart Callbacks button to select Turbo C++ smart callbacks. See Chapter 8, "Building a Windows application," in the *Programmer's Guide* for details on smart callbacks.

- Push the Windows DLL All Functions Exportable button to create an .OBJ file to be linked as a .DLL with all functions exportable.
- Push the Windows DLL Explicit Functions Exported button to create an .OBJ file to be linked as a .DLL with certain functions explicitly selected to be exported. Otherwise this is essentially the same as Windows Explicit Functions Exported; see that discussion for more information.

The Calling Convention options cause the compiler to generate either a C calling sequence or a Pascal calling sequence for function calls. The differences between C and Pascal calling conventions are in the way each handles stack cleanup, order of parameters, case, and prefix (underbar) of global identifiers.

Important! *Do not change this option unless you're an expert and have read Chapter 7, "BASM and inline assembly," in the Programmer's Guide.*

- Standard Stack Frame (when checked) generates a standard stack frame (standard function entry and exit code). This is helpful when debugging—it simplifies the process of tracing back through the stack of called subroutines.

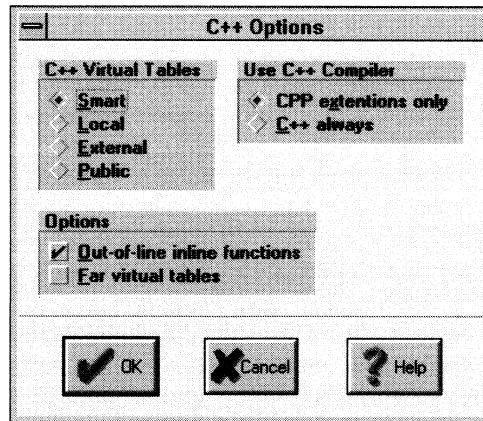
If you compile a source file with this option off (unchecked), any function that does not use local variables and has no parameters is compiled with abbreviated entry and return code. This makes the code shorter and faster, but prevents the Debug | Call Stack command from "seeing" the function. Thus, you should always check the option when you compile a source file for debugging.

- When checked, the Test Stack Overflow generates code to check for a stack overflow at run time. Even though this costs space and time in a program, it can be a real lifesaver, since a stack overflow bug can be difficult to track down.

C++ Options

The C++ Options command displays a dialog box that contains settings that tell the compiler to prepare the object code in certain ways when using C++.

Figure 4.5
The C++ Options dialog box



The C++ Virtual Tables radio buttons let you control C++ virtual tables and the expansion of inline functions when debugging.

- The Smart option generates C++ virtual tables (and inline functions not expanded inline) so that only one instance of a given virtual table or inline function will be included in the program. This produces the smallest and most efficient executables, not compatible with older linkers and assemblers.
- The Local option generates local virtual tables (and inline functions not expanded inline) such that each module gets its own private copy of each virtual table or inline function it uses; this option uses only standard .OBJ (and .ASM) constructs, but produces larger executables.
- The External option generates external references to virtual tables; one or more of the modules comprising the program must be compiled with the Public option to supply the definitions for the virtual tables.
- The Public option generates public definitions for virtual tables.

The Use C++ Compiler radio buttons tell Turbo C++ whether to always compile your programs as C++ code, or to always compile your code as C code except when the file extension is .CPP.

- Use Out-of-Line Inline Functions when you want to step through or set breakpoints on inline functions.
- The Far Virtual Tables option causes virtual tables to be created in the code segment instead of the data segment, and makes virtual table pointers into full 32-bit pointers.

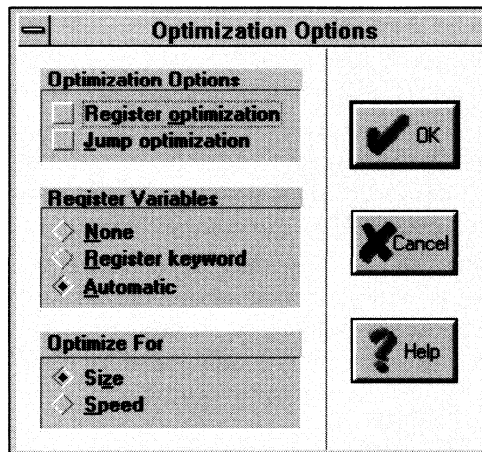
There are two primary reasons for using this option: to remove the virtual tables from the data segment, which may be getting

full, and to be able to share objects (of classes with virtual functions) between modules that use different data segments (for example, a DLL and an executable using that DLL). You must compile all modules that may share objects either entirely with or entirely without this option. You can achieve the same effect by using the **huge** or **_export** modifiers on a class-by-class basis.

Optimizations

The Optimizations command displays a dialog box. The settings in this box tell the compiler to prepare the object code in certain ways to optimize the size or speed.

Figure 4.6
The Optimization Options dialog box



The Optimizations Options affect how optimization of your code occurs.

- Register Optimization suppresses the reloading of registers by remembering the contents of registers and reusing them as often as possible.

Exercise caution when using this option. The compiler can't detect whether a value has been modified indirectly by a pointer.

- Jump Optimization reduces the code size by eliminating redundant jumps and reorganizing loops and switch statements.

Important!

When this option is checked, the sequences of tracing and stepping in the debugger can be confusing, since there might be

multiple lines of source code associated with a particular generated code sequence. For best stepping results, turn this option off (uncheck it) while you are debugging.

The Register Variables radio buttons suppress or enable the use of register variables.

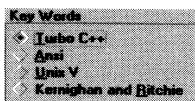
With Automatic chosen, register variables are automatically assigned for you. With None chosen, the compiler does not use register variables even if you've used the **register** keyword. With Register keyword chosen, the compiler uses register variables only if you use the **register** keyword and a register is available. See the online file WINMEM.DOC for more details.

Generally, you can keep this option set to Automatic unless you're interfacing with preexisting assembly code that does not support register variables.

The Optimize For buttons let you change Turbo C++'s code generation strategy. Normally the compiler optimizes for size, choosing the smallest code sequence possible. You can also have the compiler optimize for speed, so that it chooses the *fastest* sequence for a given task.

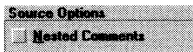
Source

The Source command displays a dialog box. The settings in this box tell the compiler to expect certain types of source code. The dialog box presents the following options:



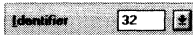
The Key Words radio buttons tell the compiler how to recognize keywords in your programs.

- Choosing Turbo C++ tells the compiler to recognize the Turbo C++ extension keywords, including **near**, **far**, **huge**, **asm**, **cdecl**, **pascal**, **interrupt**, **_es**, **_export**, **_ds**, **_cs**, **_ss**, and the register pseudovariables (**_AX**, **_BX**, and so on). For a complete list, refer to Chapter 1, "Lexical elements," in the *Programmer's Guide*.
- Choosing ANSI tells the compiler to recognize only ANSI keywords and treat any Turbo C++ extension keywords as normal identifiers.
- Choosing UNIX V tells the compiler to recognize only UNIX V keywords and treat any Turbo C++ extension keywords as normal identifiers.



- Choosing Kernighan and Ritchie tells the compiler to recognize only the K&R extension keywords and treat any Turbo C++ extension keywords as normal identifiers.

The Nested Comments checkbox allows you to nest comments in Turbo C++ source files. Nested comments are not allowed in standard C implementations. They are not portable.



Use the Identifier input box to specify the number (n) of significant characters in an identifier. Except in C++, which recognizes identifiers of unlimited length, all identifiers are treated as distinct only if their first n characters are distinct. This includes variables, preprocessor macro names, and structure member names. The number can be from 1 to 32; the default is 32.

Messages

The Messages command displays a submenu that lets you set several options that affect compiler error messages in the IDE.

Display presents a dialog box that allows you to specify how (and if) you want error messages to be displayed.

- **Errors: Stop After** causes compilation to stop after the specified number of errors have been detected. The default is 25, but you can enter any number from 0 to 255. (Entering 0 causes compilation to continue until the end of the file.)
- **Warnings: Stop After** causes compilation to stop after the specified number of warnings have been detected. The default is 100, but you can enter any number from 0 to 255. (Entering 0 causes compilation to continue until the end of the file or until the error limit entered above been reached, whichever comes first.)
- The **Display Warnings** options allow you to choose whether the compiler will display all warnings, only the warnings selected in the Messages submenu option, or to display no warnings.

When you choose **Portability** on the Messages submenu, a dialog box appears that lets you specify which types of portability problems you want to be warned about.

Check the warnings you want to be notified of and uncheck the ones you don't. Choose **OK** to return to the Compiler Messages dialog box.

When you choose ANSI Violations on the Messages submenu, a dialog box appears that lets you specify which, if any, ANSI violations you want to be warned about.

Check the warnings you want to be notified of and uncheck the ones you don't. Choose OK to return to the Compiler Messages dialog box.

When you choose the C++ Warnings button in the Messages submenu, another dialog box appears that lets you determine which specific C++ warnings you want to enable.

Check the warnings you want to be notified of and uncheck the ones you don't. Choose OK to return to the Compiler Messages dialog box.

When you choose the Frequent Errors button in the Compiler Messages dialog box, another dialog box appears that lets you specify which frequently-occurring errors you want to be warned about.

Check the errors you want to be notified of and uncheck the ones you don't. Choose OK to return to the Compiler Messages dialog box.

Choosing Less frequent errors lets you make the same choice, to be warned or not, about several less frequently occurring errors.

Check or uncheck these errors as in the previous dialog boxes, and choose OK to return to the Messages dialog box.

Names

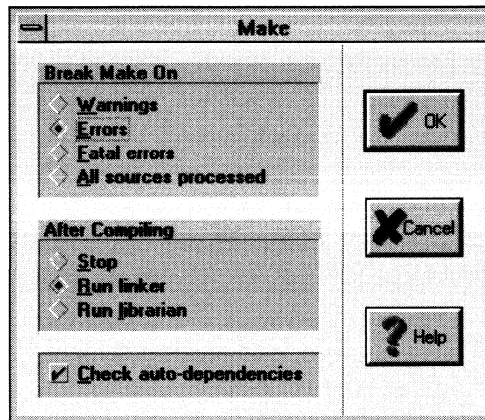
The Names command brings up a dialog box that lets you change the default segment, group, and class names for code, data, and BSS sections. *Do not change the settings in this command unless you are an expert and have viewed information about memory models in the online file WINMEM.DOC.*

Make

The Options | Make command displays a dialog box that lets you set conditions for project management. Here's what the dialog box looks like:

Options I Make

Figure 4.7
The Make dialog box



Use the Break Make On radio buttons to set the condition that will stop the making of a project. The default is to stop after compiling a file with errors.

Use the After Compiling radio buttons to specify what to do after all the source code modules defined in your project have been compiled. You can choose to Stop (leaving .OBJ files), Run linker to generate an .EXE file, or Run librarian to generate a .LIB file. The default is to run the linker to generate an executable application.

When the Check Auto-dependencies option is checked, the Project Manager automatically checks dependencies for every .OBJ file on disk that has a corresponding .C source file in the project list.

The Project Manager opens the .OBJ file and looks for information about files included in the source code. This information is always placed in the .OBJ file by Turbo C++ for Windows when the source module is compiled. Then every file that was used to build the .OBJ file is checked for time and date against the time and date information in the .OBJ file. The source file is recompiled if the dates are different. This is called an *autodependency check*. If this option is off (unchecked), no such file checking is done.

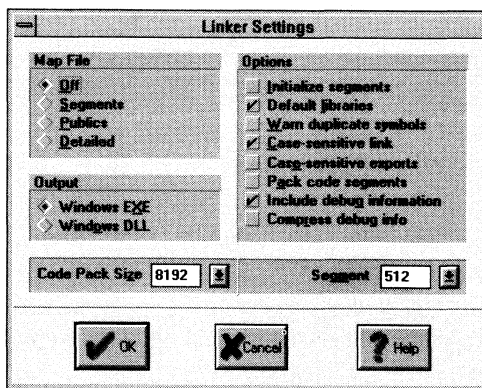
After the C source file is successfully compiled, the project file contains valid dependency information for that file. Once that information is in the project file, the Project Manager uses it to do its autodependency check. This is much faster than reading each .OBJ file.

Linker

The Options | Linker command lets you make several settings that affect linking. The Linker command opens a submenu containing the choices Settings and Libraries.

The Settings command opens up this dialog box:

Figure 4.8
The Linker Settings dialog box



This dialog box has several check boxes and radio buttons. The following sections contain short descriptions of what each does.

Use the Map File radio buttons to choose the type of map file to be produced. For settings other than Off, the map file is placed in the output directory defined in the Options | Directories dialog box. The default setting for the map file is Off.

Use these radio buttons to set your application type. Windows EXE produces a Windows application, while Windows DLL produces a Windows dynamic link library.

If checked, Initialize Segments tells the linker to initialize uninitialized segments. (This is normally not needed and will make your .EXE files larger.)

Some compilers place lists of default libraries in the .OBJ files they produce. If the Default Libraries option is checked, the linker tries to find any undefined routines in these libraries as well as in the default libraries supplied by Turbo C++. When you're linking with modules created by a compiler other than Turbo C++, you may wish to leave this option off (unchecked).

The Warn Duplicate Symbols option affects whether the linker warns you of previously encountered symbols in .LIB files.

The Case-sensitive Link option affects whether the linker is case-sensitive. Normally, this option should be checked, since C and C++ are both case-sensitive languages.

By default, the linker ignores case with the names in the IMPORTS and EXPORTS sections of the module definition file. If you want the linker be case-sensitive in regard to these names, check the Case-sensitive Exports option. This option is probably only useful when you are trying to export non-callback functions from DLLs—as in exported C++ member functions. It isn't necessary for normal Windows callback functions (declared FAR PASCAL).

When Pack code segments is checked, the linker tries to minimize the number of code segments by packing multiple code segments together; typically, this will improve performance. This option will never create segments greater than 64K.

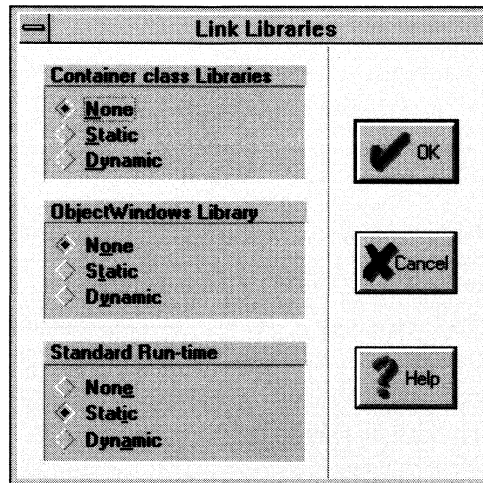
The Include debug info option instructs the linker to include information needed in order to debug your application using the Turbo Debugger.

The Compress debug info option instructs the linker to compress the debugging information in the output file. This option will slow down the linker, and should only be checked in the event of a "Debugger information overflow" error when linking.

You can change the default code packing size to anything between 1 and 65,536 with Code Pack Size.

The Segment box allows you to set the segment alignment. Note that the alignment factor will be automatically rounded up to the nearest power of two (meaning that if you enter 650, it will be rounded up to 1,024). The possible numbers you can enter must fall in the range of 2 to 65,535.

Figure 4.9
The Link Libraries dialog box



The Link Libraries dialog box has several radio buttons, that allow you to choose what libraries will automatically be linked into your application. For more information about using libraries in your applications, see Chapter 3, “Managing multi-file projects.”

The Container class Libraries option tells the linker to automatically link in the Turbo C++ container class library, which is available in both static (.LIB) and dynamic (.DLL) form. These radio buttons tell the linker which, if either, form of the Container class library you want to automatically link in with your application.

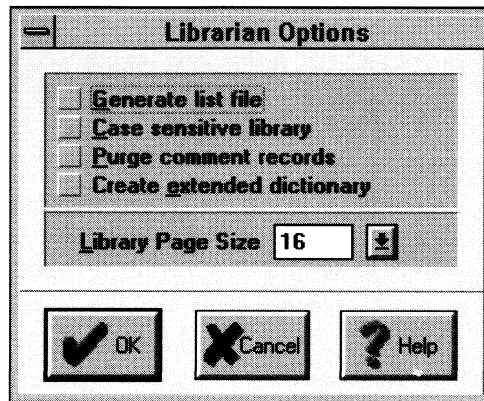
The Turbo C++ ObjectWindows Library option is a Windows application framework that is available in both static (.LIB) and dynamic (.DLL) form. These radio buttons tell the linker which, if either, form of the ObjectWindows library you want to automatically link in with your application.

In Turbo C++ 3.0, the standard run-time libraries are available in both static (.LIB) and dynamic (.DLL) form. Choosing the dynamic form can help to reduce the size of your Windows executable file, and can also reduce the overhead of loading these libraries more than once if they will be called by more than one application running simultaneously.

Librarian

The Options | Librarian command lets you make several settings affecting the use of the built in Librarian. Like the command-line librarian (TLIB), the built-in Librarian combines the .OBJ files in your project into a .LIB file. In order for any of these settings to take effect, you must have checked Run librarian in the Options | Make dialog box.

Figure 4.10
The Librarian Options dialog box



- The Generate list file check box determines whether the Librarian will automatically produce a list file (.LST) listing the contents of your library when it is created.
- The Case sensitive library check box tells the Librarian to treat case as significant in all symbols in the library (this means that CASE, Case, and case, for example, would all be treated as different symbols).
- The Purge comment records check box tells the Librarian to remove all comment records from modules added to the library.
- The Create extended dictionary check box determines whether the Librarian will include in compact form, additional information that will help the linker to process library files faster.

The Library Page Size option allows you to set the number of bytes in each library “page” (dictionary entry). The page size determines the maximum size of the library: it cannot exceed 65,536 pages. The default page size, 16, allows a library of about 1

MB in size. To create a larger library, change the page size to the next higher value (32).

Directories

The Options | Directories command lets you tell Turbo C++ where to find the files it needs to compile, link and output. This command opens the following dialog box containing three input boxes:

- The Include Directories input box specifies the directory that contains your include files. Standard include files are those given in angle brackets (<>) in an **#include** statement (for example, **#include <myfile.h>**). These directories are also searched for quoted includes not found in the current directory. Multiple directory names are allowed, separated by semicolons.
- The Library Directories input box specifies the directories that contain your Turbo C++ startup object files (C0?.OBJ) and run-time library files (.LIB files) and any other libraries that your project may use. Multiple directory names are allowed, separated by semicolons.
- The Output Directory input box specifies the directory that stores your .OBJ, .EXE, and .MAP files. Turbo C++ looks for that directory when doing a make or run, and to check dates and times of .OBJs and .EXEs. If the entry is blank, the files are stored in the current directory.

Use the following guidelines when entering directories in these input boxes:

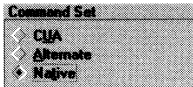
- You must separate multiple directory path names (if allowed) with a semicolon (;). You can use up to a maximum of 127 characters (including whitespace).
- Whitespace before and after the semicolon is allowed but not required.
- Relative and absolute path names are allowed, including path names relative to the logged position in drives other than the current one. For example,

```
C:\LIB;C:\MYLIBS;A:\TURBOC\MATHLIBS;A:..\VIDLIBS
```

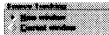
Environment

The Options | Environment command lets you make environment-wide settings. This command opens a menu that lets you choose settings from Preferences, Editor, Mouse and Desktop.

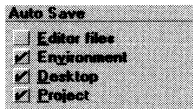
Preferences



The Command Set options let you choose either the CUA or the alternate command set as your editor interface. You can also select "Native," which specifies that the CUA interface will be used for the Turbo C++ for Windows IDE, and Alternate will be used for the Turbo C++ DOS IDE. For more information about the CUA and alternate editor command sets, see Chapter 1, "IDE basics."



When stepping source or viewing the source from the Message window, the IDE opens a new window whenever it encounters a file that is not already loaded. Selecting Current Window causes the IDE to replace the contents of the topmost Edit window with the new file instead of opening a new Edit window.

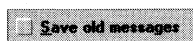


If Editor Files is checked in the Auto Save options, and if the file has been modified since the last time you saved it, Turbo C++ automatically saves the source file in the Edit window whenever you run your program.

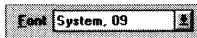
When the Environment option is checked, all the settings you made in this dialog box will be saved automatically when you exit Turbo C++.

When Desktop is checked, Turbo C++ saves your desktop when you close a project or exit, and restores when you reopen the project or return to Turbo C++.

When the Project option is checked, Turbo C++ saves all your project, autodependency, and module settings when you close your project or exit, and restores them when you reopen the project or return to Turbo C++.



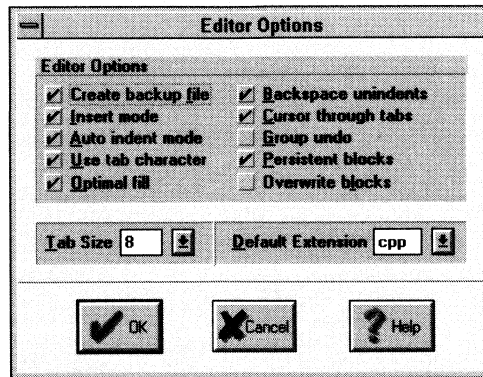
When Save Old Messages is checked, Turbo C++ saves the error messages currently in the Message window, appending any messages from further compiles to the window. Messages are not saved from one session to the next. By default, Turbo C++ automatically clears messages before a compile, a make, or a transfer that uses the Message window.



The Font input box allows you to specify a font for the editor window. Click on the arrow to open a list box displaying the available fonts, highlight the font you want, and click OK.

Editor

Figure 4.11
The Editor Options dialog box



If you choose Editor from the Environment menu, these are the options you can pick from:

- When Create backup files is checked (the default), Turbo C++ automatically creates a backup of the source file in the Edit window when you choose File | Save and gives the backup file the extension .BAK.
- When Insert mode is not checked, any text you type into Edit windows overwrites existing text. When the option is checked, text you type is inserted (pushed to the right). Pressing *Ins* toggles Insert mode when you're working in an Edit window.
- When Autoindent mode is checked, pressing *Enter* in an Edit window positions the cursor under the first nonblank character in the preceding nonblank line. This can be a great aid in typing readable program code.
- When Use tab character is checked, Turbo C++ inserts a true tab character (ASCII 9) when you press *Tab*. When this option is not checked, Turbo C++ replaces tabs with spaces. If there are any lines with characters on them prior to the current line, the cursor is positioned at the first corresponding column of characters following the next whitespace found. If there is no "next" whitespace, the cursor is positioned at the end of the line. After the end of the line, each *Tab* press is determined by the Tab Size setting.

- When you check Optimal fill, Turbo C++ begins every auto-indented line with the minimum number of characters possible, using tabs and spaces as necessary. This produces lines with fewer characters than when Optimal Fill is not checked.
- When Backspace unindents is checked (which is the default) and the cursor is on a blank line or the first non-blank character of a line, the *Backspace* key aligns (outdents) the line to the previous indentation level.
- When you check Cursor through tabs, the arrow keys will move the cursor space by space through tabs; otherwise the cursor jumps over tabs.
- When Group undo is unchecked, choosing Edit | Undo reverses the effect of a single editor command or keystroke. For example, if you type ABC, it will take three Undo commands to delete C, then B, then A.

If Group undo is checked, Undo reverses the effects of the previous command and all immediately preceding commands of the same type. The types of commands that are grouped are insertions, deletions, overwrites, and cursor movements. For example, if you type ABC, one Undo command deletes ABC.

For the purpose of grouping, inserting a carriage return is considered an insertion followed by a cursor movement. For example, if you press *Enter*, then type ABC, choosing Undo once will delete the ABC, and choosing Undo again will move the cursor to the new carriage return. Choosing Edit | Redo at that point would move the cursor to the following line. Another Redo would insert ABC.

- When Persistent blocks is checked (the default), marked blocks behave as they always have in Borland's C and C++ products; that is, they remain marked until deleted or unmarked (or until another block is marked). With this option unchecked, moving the cursor after a block is selected de-selects the entire block of text.
- When Overwrite blocks is checked *and* Persistent Blocks is *unchecked*, marked blocks behave differently in these instances:
 1. Pressing the *Del* key or the *Backspace* key clears the entire selected text.
 2. Inserting text (pressing a character or pasting from the Clipboard) replaces the entire selected text with the inserted text.

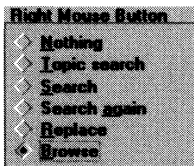
If you check Use tab character in this dialog box and press *Tab*, Turbo C++ inserts a tab character in the file and the cursor moves to the next tab stop. The Tab size input box lets you dictate how many characters to move for each tab stop. Legal values are 2 through 16; the default is 8.

To change the way tabs are displayed in a file, just change the tab size value to the size you prefer. Turbo C++ redisplayes all tabs in that file in the size you chose. You can save this new tab size in your configuration file by choosing Options | Save Options.

The Default Extension input box lets you tell Turbo C++ which extension to use as the default when compiling and loading your source code. Changing this extension doesn't affect the history lists in the current desktop.

Mouse

When you choose Mouse from the Environment menu, the Mouse Options dialog box is displayed, which contains all the settings for your mouse. These are the options available to you:

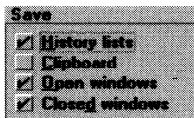


The Right Mouse Button radio buttons determine the effect of pressing the right button of the mouse (or the left button, if the reverse mouse buttons option is checked). Topic Search is the default.

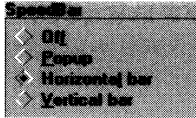
Here's a list of what the right button would do if you choose something other than Nothing:

Topic Search	Same as Help Topic Search
Search	Same as Search Find
Search again	Same as Search Search Again
Replace	Same as Search Replace
Browse	Same as Browse Symbol

Desktop



The Desktop dialog box lets you set whether history lists, the contents of the Clipboard, open windows and closed windows are saved across sessions. History lists, open windows and the Clipboard are saved by default; windows which you have closed are not. You can change these defaults by unchecking or checking the respective options.



The SpeedBar radio buttons allow you to specify how the SpeedBar will be displayed. You may choose to turn the SpeedBar off entirely, or to have it appear as a popup, a vertical bar, or a horizontal bar. For more information about the SpeedBar, see Chapter 1, "IDE basics."

Save

The Options | Save command brings up a dialog box that lets you save settings that you've made in both the Find and Replace dialog boxes (off the Search menu) and in the Options menu (which includes all the dialog boxes that are part of those commands) for IDE, Desktop, and Project items. Options are stored in three files, which represent each of these categories. If it doesn't find the files, Turbo C++ looks in the Executable directory (from which TCW.EXE is run) for the same file.

Converting from Microsoft C

If you're an experienced C or C++ programmer, but the Turbo C++ programming environment is new to you, then you should read this appendix before you do anything else. We appreciate that you want to be up and running fast with a new piece of software, and we know that you want to spend as little time as possible reading the manual. However, the time you spend reading this chapter will probably save you a lot of time later. Please read on.

Environment and tools

The Turbo C++ IDE (integrated development environment) is roughly the equivalent of the Programmer's Workbench, although naturally we think you'll find the IDE much easier to use. If you're interested in building Windows applications, see Chapter 8 in the *Programmer's Guide*.

You can find out more about configuration and project files in Chapters 1 and 3 in the User's Guide.

The IDE loads its settings from two files: TCCONFIG.TC, the default configuration file, and a project file (.PRJ). TCCONFIG.TC contains general environmental information. The current project file contains information more specific to the application you're building.

A project is the IDE's equivalent of a makefile. It includes the list of files to be built, as well as settings for the IDE options that control the compilation and linkage of that program. If you don't

specify a project file when you start the IDE, a nameless project is opened and set with default compiler and linker options, but no file name list.

Unlike Microsoft C, however, Turbo C++ does not automatically create and run a makefile based on settings and file names that you give it in the project.

Paths for .h and .LIB files

Microsoft C works with two environment variables, LIB and INCLUDE. The Microsoft linker uses the LIB variable to discover the location of the run-time libraries; similarly, INCLUDE is used to find standard header files. Turbo C++ does not use environment variables to store the path for the library or include files. Instead, you can easily set these paths in the IDE using the environment options.

When you install Turbo C++, you are asked to set paths for include files and library files. Those paths are then the default paths in the IDE.

Remember that even if you haven't opened a project, Turbo C++ will store the paths in its default project file.

In the IDE, reset default search paths for libraries and header files with the Options | Directories command. The settings in the Directories dialog box become a part of the current project.

Source-level compatibility

The following sections tell you how to make sure that your code is compatible with Turbo C++'s compiler and linker.

__MSC macro

The Turbo C++ libraries contain many functions to increase compatibility with applications originally written in Microsoft C. If you define the macro __MSC before you include the dos.h header file, the DOSERROR structure will be defined to match Microsoft's format.

Header files

Some nonstandard header files can be included by one of two names, as follows.

Original name	Alias
alloc.h	malloc.h
dir.h	direct.h
mem.h	memory.h

If you are defining data in header files in your program, you should use the Options | Compiler | Advanced code generation | Generate COMDEFs IDE option to generate COMDEFs. Otherwise you will get linker errors.

Memory models

Although the same names are used for the standard memory models, there are fairly significant differences for the large data models in the standard configuration.

In Microsoft C, all large data models have a default NEAR data segment to which DS is maintained. Data is allocated in this data segment if the data size falls below a certain threshold, or in a far data segment otherwise. You can set the threshold value with the `/Gtn` option, where *n* is a byte value. The default threshold is 32,767. If `/Gt` is given but *n* is not specified, the default is 256.

In all other memory models under Microsoft C, both a near and a far heap are maintained.

In Turbo C++, the large and compact models have a default NEAR data segment to which DS is maintained. All static data is allocated to this segment by default, limiting the total static data in the program to 64K, but making all external data references near.

Keywords

Turbo C++ supports the same set of keywords as Microsoft C 5.1 with the exception of **fortran**.

Turbo C++ supports the same set of keywords as Microsoft C 6.0 with the exception of:

- **_based**, **_self**, and **_segname**, because Turbo C++ does not support based pointers
- **_segment**; Turbo C++'s keyword **_seg** is the equivalent of **_segment**
- **_emit**; Turbo C++ uses the pseudofunction **__emit__**, because this style allows addresses of variables to be given as arguments, and allows multiple bytes to be output; **_emit**, by contrast, works like an assembly DB, allowing one immediate byte to be output
- **_fortran**; use the **_pascal** calling convention instead

Turbo C++ provides **_cs**, **_ds**, **_es**, and **_ss** pointer types. See the index in online Help for details.

Floating-point return values

In Microsoft C, **_cdecl** causes float and double values to be returned in the **__fac** (floating point accumulator) global variable. Long doubles are returned on the NDP stack. **_fastcall** causes floating point types to be returned on the NDP stack. **_pascal** causes the calling program to allocate space on the stack and pass address to function. The function stores the return value and returns the address.






In Turbo C++, floating point values are returned on the NDP stack.

Structures returned by value

In a Microsoft C-compiled function declared with **_cdecl**, the function returns a pointer to a static location. This static location is created on a per-function basis. For a function declared with **_pascal**, the calling program allocates space on the stack for the return value. The calling program passes the address for the return value in a hidden argument to the function.

Turbo C++ returns 1-byte structures in AL, 2-byte structures in AX and 4-byte structures in AX and DX. For 3-byte structures and structures larger than 4 bytes, the compiler passes a hidden argument (a far pointer) to the function that tells the function where to return the structure.

Conversion hints

-  Write *portable* code. Portable code is compatible with many different compilers and machines. Whenever possible, use only functions from the ANSI standard library (for example, use **time** instead of **gettime**). The portability information in online Help will tell you if a function is ANSI standard.
-  If you must use a function that's not in the ANSI standard library, use a Unix-compatible function, if possible (for example, use **chmod** instead of **_chmod**, or **signal** instead of **ctrlbrk**). Again, the portability information in online Help will tell you if a function is available on Unix machines.
-  Avoid the use of bit fields and code that depends on word size, structure alignment, or memory model. For example, Turbo C++ defines **ints** to be 16 bits wide, but a 32-bit C++ compiler would define 32-bit wide **ints**.
-  Insert the preprocessor statement **#define __MSC** in each module before **dos.h** is included.
-  If you were using the link option **/STACK:n** in your Microsoft application, initialize the global variable **_stklen** with the appropriate stack size.

Editor reference

The editor has two command sets: CUA and Alternate. The tables in this appendix list all the available commands. You can use some commands in both modes, while others are available in only one mode. Choose Options | Environment | Preferences and select the command set you want in the Preferences dialog box.

Most of these commands need no explanation. Those that do are described in the text following Table B.1.

Table B.1
Editing commands

*A word is defined as a sequence of characters separated by one of the following: space <> , ; . () ^ ` * + - / \$ # = | ~ ? ! " % & ` ; @ \, and all control and graphic characters.*

Command	Both modes	CUA	Alternate
Cursor movement commands			
Character left	←		Ctrl+S
Character right	→		Ctrl+D
Word left	Ctrl+←		Ctrl+A
Word right	Ctrl+→		Ctrl+F
Line up	↑		Ctrl+E
Line down	↓		Ctrl+X
Scroll up one line	Ctrl+W		
Scroll down one line	Ctrl+Z		
Page up	PgUp		Ctrl+R
Page down	PgDn		Ctrl+C
Beginning of line	Home		
	Ctrl+Q S		
End of line	End		
	Ctrl+Q D		
Top of window	Ctrl+Q E	Ctrl+E	Ctrl+Home
Bottom of window	Ctrl+Q X	Ctrl+X	Ctrl+End
Top of file	Ctrl+Q R	Ctrl+Home	Ctrl+PgUp
Bottom of file	Ctrl+Q C	Ctrl+End	Ctrl+PgDn
Move to previous position	Ctrl+P		

Table B.1: Editing commands (continued)

Command	Both modes	CUA	Alternate
Insert and delete commands			
Delete character	<i>Del</i>		<i>Ctrl+G</i>
Delete character to left	<i>Backspace</i> <i>Shift+Tab</i>		<i>Ctrl+H</i>
Delete line	<i>Ctrl+Y</i>		
Delete to end of line	<i>Ctrl+Q Y</i>	<i>Shift+Ctrl+Y</i>	
Delete word	<i>Ctrl+T</i>		
Insert line	<i>Ctrl+N</i>		
Insert mode on/off	<i>Ins</i>		<i>Ctrl+V</i>
Block commands			
Move to beginning of block	<i>Ctrl+Q B</i>		
Move to end of block	<i>Ctrl+Q K</i>		
Set beginning of block	<i>Ctrl+K B</i>		
Set end of block	<i>Ctrl+K K</i>		
Exit to menu bar	<i>Ctrl+K D</i>		
Hide/Show block	<i>Ctrl+K H</i>		
Mark line	<i>Ctrl+K L</i>		
Print selected block	<i>Ctrl+K P</i>		
Mark word	<i>Ctrl+K T</i>		
Delete block	<i>Ctrl+K Y</i>		
Copy block	<i>Ctrl+K C</i>		
Move block	<i>Ctrl+K V</i>		
Copy to Clipboard	<i>Ctrl+Ins</i>		
Cut to Clipboard	<i>Shift+Del</i>		
Delete block	<i>Ctrl+Del</i>		
Indent block	<i>Ctrl+K I</i>	<i>Shift+Ctrl+I</i>	
Paste from Clipboard	<i>Shift+Ins</i>		
Read block from disk	<i>Ctrl+K R</i>	<i>Shift+Ctrl+R</i>	
Unindent block	<i>Ctrl+K U</i>	<i>Shift+Ctrl+U</i>	
Write block to disk	<i>Ctrl+K W</i>	<i>Shift+Ctrl+W</i>	
Extending selected blocks			
Left one character	<i>Shift+ ←</i>		
Right one character	<i>Shift+ →</i>		
End of line	<i>Shift+End</i>		
Beginning of line	<i>Shift+Home</i>		
Same column on next line	<i>Shift+ ↓</i>		
Same column on previous line	<i>Shift+ ↑</i>		
One page down	<i>Shift+PgDn</i>		
One page up	<i>Shift+PgUp</i>		
Left one word	<i>Shift+Ctrl+ ←</i>		
Right one word	<i>Shift+Ctrl+ →</i>		
End of file	<i>Shift+Ctrl+End</i>		<i>Shift+Ctrl+PgDn</i>
Beginning of file	<i>Shift+Ctrl+Home</i>		<i>Shift+Ctrl+PgUp</i>

Table B.1: Editing commands (continued)

Command	Both modes	CUA	Alternate
Other editing commands			
Autoindent mode on/off	<i>Ctrl+O I</i>		
Cursor through tabs on/off	<i>Ctrl+O R</i>		
Exit the IDE		<i>Alt+F4</i>	<i>Alt+X</i>
Find place marker	<i>Ctrl+Q n *</i>	<i>Ctrl n *</i>	
Help	<i>F1</i>		
Help index	<i>Shift+F1</i>		
Insert control character	<i>Ctrl+P**</i>		
Maximize window			<i>F5</i>
Open file			<i>F3</i>
Optimal fill mode on/off	<i>Ctrl+O F</i>		
Pair matching	<i>Ctrl+Q [,</i> <i>Ctrl+Q]</i>	<i>Alt+[,Alt+]</i>	
Save file	<i>Ctrl+K S</i>		<i>F2</i>
Search	<i>Ctrl+Q F</i>		
Search again		<i>F3</i>	<i>Ctrl+L</i>
Search and replace	<i>Ctrl+Q A</i>		
Set marker	<i>Ctrl+K n *</i>	<i>Shift+Ctrl n *</i>	
Tabs mode on/off	<i>Ctrl+O T</i>		
Topic search help	<i>Ctrl+F1</i>		
Undo	<i>Alt+Backspace</i>		
Unindent mode on/off	<i>Ctrl+O U</i>		

* *n* represents a number from 0 to 9.

** Enter control characters by first pressing *Ctrl+P*, then pressing the desired control character.

Block commands

A block of text is any amount of text, from a single character to hundreds of lines, that is selected on your screen. There can be only one block in a window at a time. Select a block with your mouse or by holding down *Shift* while moving your cursor to the end of the block with the arrow keys. Once selected, the block can be copied, moved, deleted, or written to a file. You can use the Edit menu commands to perform these operations or you can use the keyboard commands listed in the following table.

When you choose Edit | Copy or press *Ctrl+Ins*, the selected block is copied to the Clipboard. When you choose Edit | Paste or *Shift+Ins*, the block held in the Clipboard is pasted at the current cursor position. The selected text remains unchanged and is no longer selected.

If you choose Edit | Cut or press *Shift+Del*, the selected block is moved from its original position and held in the Clipboard. It is

pasted at the current cursor position when you choose the Paste command.

The copying, cutting, and pasting commands are the same in both the CUA and Alternate command sets.

Table B.2: Block commands in depth

Command	CUA	Alternate	Function
Copy block	<i>Ctrl+Ins, Shift+Ins</i>	<i>Ctrl+Ins, Shift+Ins</i>	Copies a previously selected block to the Clipboard and, after you move your cursor to where you want the text to appear, pastes it to the new cursor position. The original block is unchanged. If no block is selected, nothing happens.
Copy text	<i>Ctrl+Ins</i>	<i>Ctrl+Ins</i>	Copies selected text to the Clipboard.
Cut text	<i>Shift+Del</i>	<i>Shift+Del</i>	Cuts selected text to the Clipboard.
Delete block	<i>Ctrl+Del</i>	<i>Ctrl+Del</i>	Deletes a selected block. You can “undelete” a block with Undo.
Move block	<i>Shift+Del, Shift+Ins</i>	<i>Shift+Del, Shift+Ins</i>	Moves a previously selected block from its original position to the Clipboard and, after you move your cursor to where you want the text to appear, pastes it to the new cursor position. The block disappears from its original position. If no block is marked, nothing happens.
Paste from Clipboard	<i>Shift+Ins</i>	<i>Shift+Ins</i>	Pastes the contents of the Clipboard.
Read block from disk	<i>Shift+Ctrl+R Ctrl+K R</i>	<i>Ctrl+K R</i>	Reads a disk file into the current text at the cursor position exactly as if it were a block. The text read is then selected as a block. When this command is issued, you are prompted for the name of the file to read. You can use wildcards to select a file to read; a directory is displayed. The file specified can be any legal file name.
Write block to disk	<i>Shift+Ctrl+W Ctrl+K W</i>	<i>Ctrl+K W</i>	Writes a selected block to a file. When you give this command, you are prompted for the name of the file to write to. The file can be given any legal name (the default extension is CPP). If you prefer to use a file name without an extension, append a period to the end of its name.

If you have used Borland editors in the past, you may prefer to use the block commands listed in this table; they work in both command sets.

Table B.3
Borland-style block
commands

Selected text is highlighted only if both the beginning and end have been set and the beginning comes before the end.

Command	Keys	Function
Set beginning of block	<i>Ctrl+K B</i>	Begin selection of text.
Set end of block	<i>Ctrl+K K</i>	End selection of text.
Hides/shows selected text	<i>Ctrl+K H</i>	Alternately displays and hides selected text.
Copy selected text to the cursor.	<i>Ctrl+K C</i>	Copies the selected text to the position of the cursor. Useful only with the Persistent Block option.
Move selected text to the cursor.	<i>Ctrl+K V</i>	Moves the selected text to the position of the cursor. Useful only with the Persistent Block option.

Other editing commands

The next table describes certain editing commands in more detail. The table is arranged alphabetically by command name.

Table B.4: Other editor commands in depth

Command	CUA	Alternate	Function
Autoindent	<i>Ctrl+O I</i>	<i>Ctrl+O I</i>	Toggles the automatic indenting of successive lines. You can also use Options Environment Editor Autoindent in the IDE to turn automatic indenting on and off.
Cursor through tabs	<i>Ctrl+O R</i>	<i>Ctrl+O R</i>	The arrow keys will move the cursor to the middle of tabs when this option is on; otherwise the cursor jumps several columns when cursoring over multiple tabs. <i>Ctrl+O R</i> is a toggle.
Find place marker	<i>Ctrl+n*</i> <i>Ctrl+Q n*</i>	<i>Ctrl+Q n*</i>	Finds up to ten place markers (<i>n</i> can be any number in the range 0 to 9) in text. Move the cursor to any previously set marker by pressing <i>Ctrl+Q</i> and the marker number.
Open file		<i>F3</i>	Lets you load an existing file into an edit window.
Optimal fill	<i>Ctrl+O F</i>	<i>Ctrl+O F</i>	Toggles optimal fill. Optimal fill begins every line with the minimum number of characters possible, using tabs and spaces as necessary. This produces lines with fewer characters.
Save file		<i>F2</i>	Saves the file and returns to the editor.

Table B.4: Other editor commands in depth (continued)

Command	CUA	Alternate	Function
Set place	<i>Shift+Ctrl n*</i> <i>Ctrl+K n*</i>	<i>Ctrl+K n*</i>	Mark up to ten places in text. After marking your location, you can work elsewhere in the file and then easily return to your marked location by using the Find Place Marker command (being sure to use the same marker number). You can have ten places marked in each window.
Show previous error	<i>Alt+F7</i>	<i>Alt+F7</i>	Moves the cursor to the location of the previous error or warning message. This command is available only if there are messages in the Message window that have associated line numbers.
Show next error	<i>Alt+F8</i>	<i>Alt+F8</i>	Moves the cursor to the location of the next error or warning message. This command is available only if there are messages in the Message window that have associated line numbers.
Tab mode	<i>Ctrl+O T</i>	<i>Ctrl+O T</i>	Toggles Tab mode. You can specify the use of true tab characters in the IDE with the Options Environment Editor Use Tab Character option.
Unindent	<i>Ctrl+O U</i>	<i>Ctrl+O U</i>	Toggles Unindent. You can turn Unindent on and off from the IDE with the Options Environment Editor Backspace Unindents option.

* *n* represents a number from 0 to 9.

Using EasyWin

EasyWin is an exciting new feature of Turbo C++ that lets you compile standard DOS applications that use traditional “TTY style” input and output so that they will run as true Windows programs. Best of all, *you don’t have to change a single line of code to use EasyWin!*

DOS to Windows made easy

To convert your DOS applications that use standard FILES or IOSTREAM functions, simply select Windows .EXE from the Options | Compiler | Application menu in the IDE. Turbo C++ will note that your program does not contain a **WinMain** function (normally required for Windows applications) and *automatically* link in the EasyWin library. When you run your program in the Windows environment, a standard window will be created, and your program will take input and produce output for that window exactly as if it were the standard screen.

Here’s an example program:

```
#include <stdio.h>
main()
{
    printf("Hello, world\n");
    return 0;
}
```

or, for C++, you could write

```
#include <iostream.h>

main()
{
    cout << "Hello, world\n";
    return 0;
}
```

That's all there is to it. The EasyWin window is used anytime input or output is requested from or to a TTY device. This means that in addition to `stdin` and `stdout`, the `stderr`, `stdaux`, and `cerr` "devices" are all connected to this window.

_InitEasyWin()

EasyWin's reason for being is to convert DOS applications to Windows programs, quickly and easily. However, there may be reasons for using EasyWin from within a "true" Windows program. For example, you may want to add **printf** functions to your program code to help you debug your Windows program.

To use EasyWin from within a Windows program, simply make a call to **_InitEasyWin()** before doing any standard input or output.

For example:

```
#include <windows.h>
#include <stdio.h>

#pragma argsused
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
                  LPSTR lpszCmdLine, int cmdShow)
{
    _InitEasyWin();
    /* Normal windows setup */
    printf("Hello, world\n");
    return 0;
}
```

The prototype for **_InitEasyWin()** can be found in `stdio.h`, `io.h`, and `iostream.h`.

Precompiled headers

Turbo C++ can generate and subsequently use precompiled headers for your projects. Precompiled headers can greatly speed up compilation times.

How they work

When compiling large C and C++ programs, the compiler can spend up to half of its time parsing header files. When the compiler parses a header file, it enters declarations and definitions into its symbol table. If 10 of your source files include the same header file, this header file is parsed 10 times, producing the same symbol table every time.

Precompiled header files cut this process short. During one compilation, the compiler stores an image of the symbol table on disk in a file called TCDEF.SYM by default. (TCDEF.SYM is stored in the same directory as the compiler.) Later, when the same source file is compiled again (or another source file that includes the same header files), the compiler reloads TCDEF.SYM from disk instead of parsing all the header files again. Directly loading the symbol table from disk is over 10 times faster than parsing the text of the header files.

Precompiled headers will only be used if the second compilation uses one or more of the same header files as the first one, and if a

lot of other things, like compiler options, defined macros and so on, are also identical.

If, while compiling a source file, Turbo C++ discovers that the first **#includes** are identical to those of a previous compilation (of the same source or a different source), it will load the binary image for those **#includes**, and parse the remaining **#includes**.

Use of precompiled headers for a given module is an all or nothing deal: the precompiled header file is not updated for that module if compilation of any included header file fails.

Drawbacks

When using precompiled headers, TCDEF.SYM can become very big, because it contains symbol table images for all sets of includes encountered in your sources. You can reduce the size of this file; see “Optimizing precompiled headers” on page 73.

If a header contains any code, then it can't be precompiled. For example, while C++ class definitions may appear in header files, you should take care that only member functions that are inline are defined in the header; heed warnings such as “Functions containing for are not expanded inline”.

Using precompiled headers

You can control the use of precompiled headers in any of the following ways:

- from within the IDE, using the Options | Compiler | Code Generation dialog box. The IDE bases the name of the precompiled header file on the project name, creating *PROJECT.SYM*
- or from within your code using the pragmas **hdrfile** and **hdrstop** (see Chapter 4 in the *Programmer's Guide*)

Setting file names

The compiler uses just one file to store all precompiled headers. The default file name is TCDEF.SYM. You can explicitly set the name with the #pragma **hdrfile** directive.

Caution! You may notice that your .SYM file is smaller than it should be. If this happens, the compiler may have run out of disk space when writing to the .SYM file. When this happens, the compiler deletes the .SYM in order to make room for the .OBJ file, then starts creating a new (and therefore shorter) .SYM file. If this happens, just free up some disk space before compiling.

Establishing Identity

The following conditions need to be identical for a previously generated precompiled header to be loaded for a subsequent compilation.

The second or later source file must:

- have the same set of include files in the same order
- have the same macros defined to identical values
- use the same language (C or C++)
- use header files with identical time stamps; these header files can be included either directly or indirectly

In addition, the subsequent source file must be compiled with the same settings for the following options:

- memory model, including SS != DS
- underscores on externs
- maximum identifier length
- target DOS (default) or Windows
- generate word alignment
- Pascal calls
- treat enums as integers
- default char is unsigned
- virtual table control

Optimizing precompiled headers

For Turbo C++ to most efficiently compile using precompiled headers, follow these rules:

- Arrange your header files in the same sequence in all source files.
- Put the largest header files first.
- Prime TCDEF.SYM with often-used initial sequences of header files.

- Use `#pragma hdrstop` to terminate the list of header files at well-chosen places. This lets you make the list of header files in different sources look similar to the compiler. `#pragma hdrstop` is described in more detail in Chapter 4 in the *Programmer's Guide*.

For example, given the two source files ASOURCE.C and BSOURCE.C, both of which include windows.h and myhdr.h,

```
ASOURCE.C:  #include <windows.h>
             #include "myhdr.h"
             #include "xxx.h"
             <...>
```

```
BSOURCE.C:  #include "zz.h"
             #include <string.h>
             #include "myhdr.h"
             #include <windows.h>
             <...>
```

You would rearrange the beginning of BSOURCE.C to:

```
Revised BSOURCE.C: #include <windows.h>
                   #include "myhdr.h"
                   #include "zz.h"
                   #include <string.h>
                   <...>
```

Note that windows.h and myhdr.h are in the same order in BSOURCE.C as they are in ASOURCE.C. You could also make a new source called PREFIX.C containing only the header files, like this:

```
PREFIX.C     #include <windows.h>
             #include "myhdr.h"
```

If you compile PREFIX.C first (or insert a `#pragma hdrstop` in both ASOURCE.C and BSOURCE.C after the `#include "myhdr.h"` statement) the net effect is that after the initial compilation of PREFIX.C, both ASOURCE.C and BSOURCE.C will be able to load the symbol table produced by PREFIX.C. The compiler will then only need to parse xxx.h for ASOURCE.C and zz.h and string.h for BSOURCE.C.

<> (angle brackets) in #include directive 51
 ; (semicolons) in directory path names 51

A

Add Item command 25
 Advanced Code Generation
 command 37
 dialog box 37
 Advanced Code Generation dialog box 37
 After Compiling
 option 46
 alignment
 word 35
 alloc.h (header file)
 malloc.h and 59
 Alternate command set 12
 American National Standards Institute *See*
 ANSI
 angle brackets (<>) in #include directive 51
 ANSI
 C standard 3
 keywords
 using only 43
 ANSI Violations 44
 Application Options dialog box 33
 applications
 Microsoft Windows *See* Microsoft Windows
 applications
 arrays
 huge
 fast huge pointer arithmetic and 38
 Assume SS equals DS option 36
 Auto Save option 52
 autodependencies *See* automatic dependencies
 autoindent mode 65, 67
 Autoindent mode option 53
 automatic dependencies 46
 checking 29

Automatic Far Objects option 38

B

Backspace unindents option 54
 backup files (.BAK) 53
 .BAK files 53
 _based (keyword) 59
 block
 copy 64, 66
 Borland-style 67
 cut 66
 delete 64, 66
 extending 64
 hide and show 64
 hide/show
 Borland-style 67
 indent 64
 move 64, 66
 Borland-style 67
 move to beginning of 64
 move to end of 64
 print 64
 read from disk 64, 66
 set beginning of 64
 Borland-style 67
 set end of 64
 Borland-style 67
 unindent 64
 write to disk 64, 66
 block commands 65
 block operations (editor) *See* editing, block
 operations
 blocks, text *See* editing, block operations
 Borland
 contacting 8
 Break Make On
 Make dialog box 27
 option 46

breakpoints *See also* debugging; watch
expressions
inline functions and 41
saving across sessions 55
BSS names 45
bugs *See* debugging
buttons
ObjectBrowser 20

C

C++ 41, *See also* Turbo C++
Turbo C++ implementation 3
classes *See* classes
compiling 41
functions
inline
debugging and 41
hierarchies *See* class hierarchy
inline functions *See* C++, functions, inline
virtual tables *See* virtual tables
warnings 45
C++ Options
command 40
dialog box 41
C language *See* C++
callbacks
smart
Windows applications and 39
calling
conventions 40
Case-Sensitive Exports option 48
case sensitive option
librarian 50
case sensitivity
linking with 47
module definition file and 48
_cdecl (keyword)
Microsoft C 60
.CFG files *See* configuration files
characters
char data type *See* data types, char
data type char *See* data types, char
delete 64
Check Auto-dependencies option 46
class
view details of 21
class hierarchy
view 20
classes
names 45
sharing objects 42
Clear command 66
hot key 13
Clipboard 65
copy to 64
cut to 64
paste from 64, 66
saving across sessions 55
Code Generation
Advanced
command 37
command 35
dialog box 35
Code Pack Size option 48
code segment
names 45
storing virtual tables in 41
COMDEFs
generating 59
PUBDEFs versus 38
command-line options
build (/b) 12
make (/m) 12
Turbo C++ for Windows IDE 12
command set
Alternate 12
Common User Access (CUA) 12
Native option 13
Command Set option 52
command sets 12
commands *See also* options
choosing
with SpeedBar 14
editor
block operations 64, 65-66
cursor movement 63
insert and delete 64
comments
nested 44
Common User Access command set 12, *See also*
CUA
compatibility
with Microsoft C 57-61

- compilation *See also* compilers
 - speeding up 36
- Compile command
 - hot key 13
- Compiler
 - command 35
- Compiler Messages submenu 44
- compilers *See also* compilation
 - C++ 41
 - code optimization 42
 - configuration files *See* configuration files
 - memory models *See* memory models
 - optimizations
 - for speed or size 43
 - stopping after errors and warnings 44
- Compress debug info option 48
- configuration files
 - contents of 15
 - IDE
 - TCCONFIG.TC 15
 - saving 56
- Container class library 49
- control character
 - insert 65
- conventions
 - typographic 7
- copy and paste *See* editing, copy and paste
- copy block
 - Borland-style 67
- Copy command
 - hot key 13
- copy to Clipboard 64
- copying, and pasting *See* editing, copy and paste
- Create Backup Files option 53
- CUA *See also* Common User Access command
 - set
- CUA option 52
- Current window option 52
- Cursor through tabs 65, 67
- Cursor through tabs option 54
- customer assistance 8
- customizing
 - IDE 52
- customizing Turbo C++ 6
- Cut command
 - hot key 13

- cut to Clipboard 64

D

- data
 - aligning 35
- data segment
 - names 45
 - removing virtual tables from 41
- data structures *See also* arrays
- data types *See also* data
 - char
 - default 36
- Debug Info in OBJs option 37
- debugging
 - Debug Info in OBJs 37
 - information
 - storing 37
 - line numbers information 37
 - stack overflow 40
 - subroutines 40
 - watch expressions *See* watch expressions
- Default extension option 55
- Default Libraries option 47
- Defines option 36
- delete block 64
- delete characters 64
- Delete Item command 25
- delete lines 64
- delete words 64
- dependencies 46
 - automatic *See* automatic dependencies
- desktop
 - saving options in 55
- desktop files
 - contents of 17
- desktop files (.DSK)
 - default 17
 - projects and 17
- Desktop option 52
- Desktop Preferences dialog box 55
- dialog boxes
 - Preferences 67
- dir.h (header file)
 - direct.h and 59
- direct.h (header file)
 - dir.h and 59

- directives
 - Microsoft compatibility 58
- Directories
 - command 51
- directories
 - defining 51
 - output 51
 - project files 16
 - projects 26
 - semicolons in paths 51
- disk space
 - running out of 72
- Display Warnings
 - option 44
- Display Warnings option 44
- distribution disks 3
- DLLs
 - creating 39, 40
 - packing code segments 48
 - setting 47
- DS register (data segment pointer) 36
- .DSK files
 - default 17
 - projects and 17
- duplicate
 - symbols 47
- duplicate, strings, merging 36
- Duplicate Strings Merged option 36
- dynamic link libraries *See* DLLs

E

- Edit *See also* editing
 - windows
 - loading files into 28
- Edit windows
 - option settings 53
- edit windows
 - cursor
 - moving 63
- editing *See also* Edit, *See also* editor; text
 - block operations 64, 65-66
 - deleting 66
 - deleting text 54
 - marking 54
 - overwrite 54
 - reading and writing 66
 - selecting blocks 54
- commands
 - cursor movement 63
 - insert and delete 64
- copy and paste *See also* Clipboard
 - hot key 13
- hot keys 13
- insert mode
 - overwrite mode vs. 53
- matching pairs *See* pair matching
- miscellaneous commands 67-68
- options
 - setting 53
- pair matching *See* pair matching
- paste *See* editing, copy and paste
- selecting text 65
- setting defaults 53
- editor *See also* editing
 - changing fonts 52
 - options
 - setting 53
 - setting defaults 53
 - tabs in 53
- Editor Files option 52
- Editor Options 53
- `__emit__()` 60
- `_emit` (keyword) 60
- emulation, 80x87
 - floating point 37
- Entry Exit Code
 - Generation dialog box 39
- Entry/Exit Code
 - command 38
- enumerations (enum)
 - treating as integers 35
- Environment
 - command 52
- environment *See* integrated development
 - environment
 - variables 58
- Environment option
 - Auto Save 52
- error
 - show next 68
 - show previous 68
- Errors
 - Stop After 44

- errors *See also* warnings
 - Frequent 45
 - messages
 - compile time 27, 28
 - removing 29
 - saving 29
 - setting 44
 - which book to look in for 4
 - next
 - hot key 13, 28
 - previous
 - hot key 13, 28
 - stopping on *n* 44
 - syntax
 - project files 27, 28
 - tracking
 - project files 27, 28
- .EXE files
 - directory 51
 - making 13
- executable files *See* .EXE files
- Exit command
 - hot key 12
- exit the IDE 65
- _export (keyword)
 - Windows applications and 39
- exports
 - case sensitive 48
- expressions
 - watch *See* Watch, expressions
- extended dictionary option
 - librarian 50
- External option
 - C++ Virtual tables 41

F

- Far Data Threshold type-in box 38
- far objects *See* objects, far
- Far option
 - C++ Virtual tables 41
- Fast Floating Point option 38
- Fast Huge Pointers option 38
- fatal errors *See* errors
- features of Turbo C++ 1
- file
 - configuration 15

- open 65, 67
- save 65, 67
- files *See also* individual file-name extensions
 - backup (.BAK) 53
 - configuration *See* configuration files
 - desktop (.DSK)
 - default 17
 - projects and 17
 - editing *See* Editor Files option
 - executable *See* .EXE files
 - header *See* header files
 - include *See* include files
 - information in dependency checks 29
 - library *See* libraries, files
 - loading into editor 28
 - make *See* Make (program manager)
 - map *See* map files
 - multiple *See* projects
 - out of date, recompiled 29
 - project 16
 - project (.PRJ) *See* projects
 - saving
 - automatically 52
 - .TC *See* configuration files, IDE
- filling lines with tabs and spaces 53
- Find command *See* Search
- Find dialog box
 - settings
 - saving 56
- floating point
 - code generation 37
 - fast 38
 - Microsoft C and 60
- Font option 52
- fortran (keyword) 59
 - _pascal keyword and 60
- Frequent Errors
 - warnings 45
- frequent errors 45
- functions *See also* individual function names, *See also* scope
 - browsing through 21
 - calling conventions 40
 - defined in source 22
 - export
 - Windows applications and 39

inline

C++

precompiled headers and 72

view details of 22

virtual *See* virtual tables

G

Generate COMDEFs option 38

Generate Underbars option 37

Go to

ObjectBrowser

hot key 20

group names 45

Group undo option 54

H

hardware

requirements

mouse 3

requirements to run Turbo C++ 2

hdrfile pragma 72

hdrstop pragma 72, 74

header files *See also* include files

Turbo C++ versus Microsoft C 59

Microsoft C 59

precompiled *See also* precompiled headers
variables and 38

Help

ObjectBrowser

hot key 20

topic search 65

help 65

hot keys 13

help index 65

hierarchies *See* class hierarchy

history lists

saving across sessions 55

hot keys

editing 13

help 13

make project 28

next error 28

previous error 28

I

IDE 11, *See also* integrated development
environment, *See* integrated development
environment

identifiers

duplicate 47

length 44

Turbo C++ keywords as 43

import libraries *See* DLLs

Include debug info option 48

#include directive *See also* include files

directories 51

Include Directories

input box 51

INCLUDE environment variable 58

include files *See also* header files

paths 58

projects 25

Include Files command 25

indent block 64

indenting automatically 53

Index command

hot key 13

indexes *See* arrays

initialized data segment *See* data segment

inline functions, C++ *See* C++, functions, inline;
functions

insert lines 64

insert mode 64

Insert mode option 53

Inspect

ObjectBrowser

hot key 20

installing Turbo C++ 5

integrated debugger *See* debugging

breakpoints *See* breakpoints

integrated development environment (IDE) 1

integrated environment

customizing 52

INCLUDE environment variable and 58

LIB environment variable and 58

makes 29

Programmer's Workbench and 57

settings

saving 56

J

Jump Optimization
option 42

K

K&R *See* Kernighan and Ritchie
Keep Messages command
toggle 29
Kernighan and Ritchie
keywords 43
keys, hot *See* hot keys
keywords
Microsoft C 59
options 43
register
Register Variables option and 43
Turbo C++ 43

L

Less Frequent Errors dialog box 45
LIB environment variable 58
.LIB files *See* libraries
librarian
case sensitive option 50
dialog box choices 50
extended dictionary option 50
list file option 50
purge comments option 50
Librarian command 50
Librarian Options dialog box 50
libraries
container class 49
default 47
directories 51
dynamic link (DLL) *See* DLLs
files 51
import *See* DLLs
overriding in projects 30
paths 58
library
ObjectWindows 49
Library Directories
input box 51
library files *See* libraries

line
mark a 64
line numbers *See* lines, numbering
Line Numbers Debug Info option 37
lines

delete 64
filling with tabs and spaces 53
insert 64
numbering
information for debugging 37

Linker
command 47
dialog box 49
settings dialog box 47

linker
case sensitive linking 47

Linker option
container class library 49

list file option
librarian 50

Local option
C++ Virtual tables 41

Local Options
command 25

M

macros
preprocessor 36
Turbo editor *See* The online document
UTIL.DOC
MAKE (program manager)
After compiling 46
integrated environment makes and 29
stopping makes 27, 46
Make command 45
malloc.h (header file)
alloc.h and 59
map files
directory 51
options 47
marker
find 65, 67
set 65, 68
mem.h (header file)
memory.h and 59
memory.h (header file)
mem.h and 59

- memory models
 - automatic far data and 38
 - changing 35
 - command-line options 36
 - Microsoft C and 59
- menu bar *See* menus
- menu commands
 - choosing
 - with SpeedBar 14
- menus *See also* individual menu names
 - commands *See* individual command names
- Message Tracking
 - toggle 28
- Message window 28
- messages
 - appending 52
- Messages command 44
- mice *See* mouse
- Microsoft C
 - Turbo C++ projects and 57
 - _cdecl keyword 60
 - COMDEFs and 59
 - converting from 57-61
 - environment variables and 58
 - floating-point return values 60
 - header files 59
 - Turbo C++ header files versus 59
 - keywords 59
 - memory models and 59
 - structures 60
- Microsoft Windows All Functions Exportable
 - command 39
- Microsoft Windows applications
 - code segments 48
 - export functions and 39
 - IDE options 39
 - optimizing for 43
 - prolog and epilog code 38
 - setting application type 47
 - setting options for 33, 38
 - smart callbacks and 39
- Microsoft Windows DLL All Functions
 - Exportable command 39
- Microsoft Windows DLL Explicit Functions
 - Exported command 40
- Microsoft Windows Explicit Functions
 - Exported command 39

- Microsoft Windows Smart Callbacks command 39
- models, memory *See* memory models
- module definition files
 - exported functions and 39
 - EXPORTS section
 - case-sensitive 48
 - IMPORTS section
 - case-sensitive 48
- mouse
 - compatibility 3
 - options 55
 - right button action 55
- Mouse Options dialog box 55
- moving text *See* editing, block operations
- _MSC macro 58
- multi-source programs *See* projects
- Multiple Document Interface (MDI) 11
- multiple files *See* projects

N

- Names
 - command 45
- names *See* identifiers
- Native command set option 13
- Nested Comments option 44
- New Window option 52
- next error
 - show 68
- nonfatal errors *See* errors
- numeric coprocessors
 - inline instructions 37

O

- .OBJ files
 - debugging information 37
 - dependencies 46
 - directories 51
- object files *See* .OBJ files
- object-oriented programming (OOP) 1
- ObjectBrowser buttons 20
- objects
 - far
 - generating 38
- ObjectWindows library option 49
- online Help *See* Help

- OOP *See* Object-Oriented Programming
- Open a File dialog box 67
- Open command 67
 - hot key 12
- open file 65, 67
- Optimal Fill option 53, 65, 67
- Optimizations
 - command 42
- optimizations 42
 - fast floating point 38
 - for speed or size 43
 - precompiled headers 73
 - Windows applications and 43
- Optimizations dialog box
 - Turbo C++ for Windows 42
- option
 - Compress debug info 48
 - Include debug info 48
- Options menu 33
 - settings
 - saving 56
- Out-Line Inline Functions option 41
- Output Directory
 - input box 51
- Overview
 - ObjectBrowser
 - hot key 20
- Overwrite Blocks option 54
- Overwrite mode 53

P

- Pack Code Segments option 48
- pair matching 65
- Pascal
 - calling convention 40
- _pascal (keyword)
 - fortran keyword and 60
- Paste command
 - hot key 13
- paste from Clipboard 64, 66
- pasting *See* editing, copy and paste
- path names in Directories dialog box 51
- Persistent blocks option 54
- place marker
 - find 65, 67
 - set 65, 68

- pointers
 - fast huge 38
 - virtual table
 - 32-bit 41
- pop-up menus *See* menus
- portability warnings 44
- #pragma hdrfile 72
- #pragma hdrstop 72, 74
- precompiled headers 71-74
 - controlling 72
 - drawbacks 72
 - how they work 71
 - inline member functions and 72
 - optimizing use of 73
 - rules for 73
 - using
 - IDE 36
- Preferences dialog box 67
- previous error
 - show 68
- Previous Error command
 - hot key 13
- previous view
 - ObjectBrowser 20
- procedures *See* functions
- program manager (Make) *See* Make (program manager)
- Programmer's Platform *See* integrated development environment
- Programmer's Workbench
 - integrated environment and 57
- project files 16
 - contents of 16
- project icons 11
- Project option 52
- projects
 - autodependency checking 46
 - speeding up 46
 - automatic dependency checking and 29
 - building 23
 - changing 17
 - default 17
 - desktop files and 17
 - directories 26
 - directory 16
 - error tracking 27, 28

- files
 - adding 25
 - deleting 25
 - include 25
 - list 25
 - options 25
 - out of date 29
 - viewing 32
- IDE configuration files and 16
- include files 25
- information in 23
- libraries and
 - overriding 30
- loading 16
- makes and 29
- making
 - hot key for 28
- Microsoft C and 57
- naming 24
- new 25
- opening 16
- saving 26
- prolog and epilog code
 - generating 38
- PUBDEFs
 - COMDEFs versus 38
- Public option
 - C++ Virtual tables 41
- pull-down menus *See* menus
- purge comments option
 - librarian 50
- put to (<<) *See* overloaded operators
- put to operator (<<) *See* overloaded operators

R

- random numbers *See* numbers, random
- read block 64
- README file 6
- real numbers *See* floating point
- Redo command
 - Group Undo and 54
 - hot key 13
- register (keyword)
 - Register Variables option and 43
- Register Optimization option 42
- Register Variables option 43

- registers
 - DS (data segment pointer) 36
 - reusing 42
 - SS (stack segment pointer) 36
- Remove Messages command 29
- Replace dialog box
 - settings
 - saving 56
- Rewind
 - ObjectBrowser
 - hot key 20
- Right Mouse Button option 55
- Ritchie, Dennis *See* Kernighan and Ritchie
- Run command
 - hot key 13

S

- Save command 56
 - hot key 12
- save file 65, 67
- Save Old Messages option 52
- scope *See* variables
- Search Again command
 - hot key 13
- search for text 65
- search.h (header file) 59
- _seg (keyword)
 - _segment keyword and 60
- Segment Alignment option 48
- _segment (keyword) 60
- segments
 - aligning 48
 - code
 - minimizing 48
 - packing 48
 - initializing 47
 - names 45
 - _segname (keyword) 59
 - selecting text 65
 - _self (keyword) 59
- semicolons (;) in directory path names 51
- shortcuts *See* hot keys
- Smart option
 - C++ Virtual tables 41
- software requirements to run Turbo C++ 2
- Source
 - command 43

- source code
 - go to 22
 - source files
 - multiple *See* projects
 - Source Options dialog box 44
 - Source Tracking option 52
 - Source Tracking options 28
 - spaces vs. tabs 53
 - SpeedBar 14
 - configuring the 14
 - SS register (stack segment pointer) 36
 - stack
 - overflow 40
 - standalone librarian
 - case sensitive 50
 - extended dictionary 50
 - list file 50
 - purge comments 50
 - standard library files *See* libraries
 - Standard Stack Frame command 40
 - starting Turbo C++ for Windows 5, 11
 - strings
 - duplicate
 - merging 36
 - structures
 - Turbo C++ versus Microsoft C 60
 - .SYM files 71, 72
 - default names 72
 - disk space and 72
 - smaller than expected 72
 - symbol
 - view declaration of 21
 - symbols
 - duplicate 47
 - inspect 22
 - syntax
 - errors
 - project files 27, 28
 - system
 - requirements 2
- T**
- Tab mode 68
 - Tab size option 54
 - tables, virtual *See* virtual tables
 - tabs
 - size of 54
 - spaces vs. 53
 - using in the editor 53
 - Tabs mode 65
 - TCCONFIG.TC file 13, 15
 - TCDEF.SYM 71, 72, *See also* .SYM files
 - TCDEFW.DPR files 17
 - TCDEFW.DSK files 17
 - TCW *See* Turbo C++; integrated development environment
 - TCW.EXE *See* integrated development environment
 - Turbo C++ for Windows
 - Optimizations dialog box 42
 - technical support 8
 - TEML *See* The online document UTIL.DOC
 - Test Stack Overflow command 40
 - text *See also* editing
 - blocks *See* editing, block operations
 - files *See also* editing
 - inserting vs. overwriting 53
 - thunks *See* callbacks, smart
 - Topic Search command
 - hot key 13
 - Topic search in Help 65
 - Treat enums as ints option 35
 - Turbo C++ *See also* C++; integrated development environment; C++; keywords, *See also*
 - calling convention 40
 - converting to from Microsoft C 57-61
 - keywords
 - as identifiers 43
 - Turbo C++ for Windows
 - implementation data 3
 - Turbo Editor Macro Language compiler *See* The online document UTIL.DOC
 - typefaces used in these books 7
 - types *See* data types
 - typographic conventions 7
- U**
- unconditional breakpoints *See* breakpoints
 - underscores
 - generating automatically 37
 - undo 65
 - Undo command
 - Group Undo and 54

- hot key 13
- unindent
 - block 64
 - mode 65, 68
- uninitialized data segment *See* data segment
- UNIX
 - keywords 43
- Unsigned Characters option 36
- Use tab character option 53
- utilities *See also* The online document
- UTIL.DOC

V

- varargs.h (header file) 59
- variables
 - browse through 22
 - defined in source 22
 - header files and 38
 - register 43
 - view declarations of 22
- virtual tables 41
 - 32-bit pointers and 41
 - storing in the code segment 41

W

- Warnings
 - Stop After 44
- warnings *See also* errors
 - ANSI Violations 44
 - C++ 45
 - frequent errors 45
 - messages
 - which book to look in for 4
 - portability 44
- watch expressions *See also* debugging
 - saving across sessions 55
- windows
 - Edit *See* Edit, window, *See* Edit, windows
 - saving across sessions 55
 - source tracking 52
- Windows (Microsoft) *See* Microsoft Windows
- word
 - delete 64
 - mark 64
- Word Alignment option 35
- write block 64