BORLAND® C++ 3.0

USER’S GUIDE

- INTEGRATED ENVIRONMENT
- OPTIMIZATION
- COMMAND-LINE COMPILER
- INSTALLATION

BORLAND
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>General hot keys</td>
<td>27</td>
</tr>
<tr>
<td>2.2</td>
<td>Menu hot keys</td>
<td>27</td>
</tr>
<tr>
<td>2.3</td>
<td>Editing hot keys</td>
<td>28</td>
</tr>
<tr>
<td>2.4</td>
<td>Window management hot keys</td>
<td>28</td>
</tr>
<tr>
<td>2.5</td>
<td>Online Help hot keys</td>
<td>28</td>
</tr>
<tr>
<td>2.6</td>
<td>Debugging/Running hot keys</td>
<td>29</td>
</tr>
<tr>
<td>2.7</td>
<td>Manipulating windows</td>
<td>32</td>
</tr>
<tr>
<td>2.8</td>
<td>General hot keys</td>
<td>40</td>
</tr>
<tr>
<td>2.9</td>
<td>Editing hot keys</td>
<td>40</td>
</tr>
<tr>
<td>2.10</td>
<td>Online Help hot keys</td>
<td>41</td>
</tr>
<tr>
<td>2.11</td>
<td>Compiling/Running hot keys</td>
<td>41</td>
</tr>
<tr>
<td>3.1</td>
<td>Information settings</td>
<td>66</td>
</tr>
<tr>
<td>3.2</td>
<td>Format specifiers recognized in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>debug器 expressions</td>
<td>72</td>
</tr>
<tr>
<td>5.1</td>
<td>Command-line options summary</td>
<td>143</td>
</tr>
<tr>
<td>A.1</td>
<td>Optimization options summary</td>
<td>174</td>
</tr>
<tr>
<td>A.2</td>
<td>Parameter types and possible registers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>used</td>
<td>184</td>
</tr>
<tr>
<td>B.1</td>
<td>Editing commands</td>
<td>185</td>
</tr>
<tr>
<td>B.2</td>
<td>Block commands in depth</td>
<td>188</td>
</tr>
<tr>
<td>B.3</td>
<td>Borland-style block commands</td>
<td>189</td>
</tr>
<tr>
<td>B.4</td>
<td>Other editor commands in depth</td>
<td>189</td>
</tr>
</tbody>
</table>
FIGURES

2.1: A typical window ......................... 31
2.2: A typical status line ................... 33
2.3: A sample dialog box .................... 34
3.1: The Open a File dialog box ............ 47
3.2: The Save File As dialog box .......... 49
3.3: The Change Directory dialog box .... 50
3.4: The Find Text dialog box .............. 56
3.5: The Replace Text dialog box .......... 58
3.6: The Breakpoints dialog box .......... 74
3.7: The Breakpoint Modify/New dialog box ........................................ 75
3.8: The Override Options dialog box ...... 77
3.9: The Include Files dialog box ......... 78
3.10: Set Application Options ............... 82
3.11: The Code Generation dialog box ...... 84
3.12: The Advanced Code Generation dialog box ........................................ 86
3.13: The Entry/Exit Code Generation dialog box ........................................ 88
3.14: The C++ options dialog box .......... 90
3.15: Advanced C++ Options ................. 92
3.16: The Turbo C++ for Windows
Optimization Options dialog box .......... 95
3.17: The Borland C++ Optimization
Options dialog box ........................ 96
3.18: The Transfer dialog box ............... 101
3.19: The Modify/New Transfer Item dialog box ........................................ 101
3.20: The Make dialog box .................. 103
3.21: The Linker dialog box ................. 104
3.22: The Libraries dialog box ............. 106
3.23: The Librarian Options dialog box ... 107
3.24: The Debugger dialog box ............. 108
3.25: The Startup Options dialog box ...... 116
3.26: The Colors dialog box ................. 118
Borland C++ is a professional optimizing compiler for C++ and C developers. With Borland C++, you get both C++ (AT&T v.2.1 compliant) and ANSI C. It is a powerful, fast, and efficient compiler with which you can create practically any application, including Microsoft Windows applications.

C++ is an object-oriented programming (OOP) language, and allows you to take advantage of OOP's advanced design methodology and labor-saving features. It's the next step in the natural evolution of C. It is portable, so you can easily transfer application programs written in C++ from one system to another. You can use C++ for almost any programming task, anywhere.

What's in Borland C++

Borland C++ includes the latest features programmers have asked for:

- **C and C++**: Borland C++ offers you the full power of C and C++ programming, with a complete implementation of the AT&T v. 2.1 specification as well as a 100% ANSI C compiler. Borland C++ 3.0 also provides a number of useful C++ class libraries, plus the first complete commercial implementation of templates, which allow efficient collection classes to be built using parameterized types.

- **Global Optimization**: a full suite of state-of-the-art optimization options gives you complete control over code generation, so you can program in the style you find most convenient, yet still produce small, fast, highly efficient code.

- **Faster compilation speed**: Borland C++ 3.0 cuts compilation time for C++ by up to half. Precompiled headers, a Borland exclusive, significantly shorten recompilation time.
Optimizations are also performed at high speed, so you don’t have to wait for high quality code.

■ **DPMI Compiler**: Compile huge programs limited only by the memory on your system. Borland C++ 3.0 now uses the industry-standard DPMI protected mode protocol that allows the compiler (as well as the IDE, the linker, and other programs) to be run in protected mode under DOS or Windows 386 enhanced mode.

■ **Microsoft Windows programming**: Borland C++ 3.0 provides complete support for developing Windows applications, including dynamic link libraries (DLLs) and EXEs. Added support for Windows programming includes the Resource Compiler, the Help Compiler, and the Resource Workshop. We’ve also included many sample C and C++ Windows applications to help get you going.

■ **EasyWin**: Automatic Windows-conversion feature 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 set a single compiler switch (or select “Windows application” in the IDE), and your DOS program runs in a window!

■ **Programmer’s Platform**: Borland C++ 3.0 comes with an improved version of the Programmer’s Platform, Borland’s open-architecture IDE that gives you access to a full range of programming tools and utilities, including

- a multi-file editor, featuring both an industry-standard Common User Access (CUA) interface and a familiar alternate interface that is compatible with previous versions of Borland C++
- advanced Turbo Editor Macro Language (TEML) and compiler
- multiple overlapping windows with full mouse support
- integrated resource linking, making it easy to develop Windows applications in a single environment
- fully integrated debugger running in DPMI, for debugging large applications
- a built-in assembler and support for inline assembler code
- complete undo and redo capability with an extensive buffer and much more.
- **Windows-hosted IDE:** The included Turbo C++ for Windows IDE lets you edit, compile, and run your programs under Windows, so you don’t have to task switch between Windows and a DOS compatibility box to create Windows programs. This greatly increases your productivity by allowing you to program, compile, link, debug and execute completely within 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 browse the source code of any element you select
  - visual SpeedBar for instant point-and-click access to frequently-used menu selections
- **WinSight:** Windows message-tracing utility lets you see inside your program’s interaction with Windows.
- **VROOMM:** Borland C++’s Virtual Run-time Object-Oriented Memory Manager lets you overlay your code without complexity. You select the code segments for overlaying; VROOMM takes care of the rest, doing the work needed to fit your code into 640K.
- **Help:** Online context-sensitive hypertext help, with copy-and-paste program examples for practically every function.
- **Streams:** Borland C++ includes full support for C++ iostreams, plus special Borland extensions to the streams library that allow you to position text, set screen attributes, and perform other manipulations to streams within the Windows environment.
- **Container classes:** Advanced container class libraries giving you sets, bags, lists, arrays, B-trees and other reusable data structures, implemented both as templates and as object-based containers for maximum flexibility.
- **Windows API:** The complete Windows API documentation in online help.

**Other features:**

- Over 200 new library functions for maximum flexibility and compatibility.
- Complex and BCD math, fast huge arithmetic.
- Heap checking and memory management functions, with **far** objects and **huge** arrays.
- Run-time library in a DLL for Windows applications.
- New BGI fonts and BGI support for the full ASCII character set.
- Shared project, configuration, and desktop files to let programmers work with the same environment whether they use the Programmer's Platform or the Windows-hosted IDE.
- Response files for the command-line compiler.
- NMAKE compatibility for easy transition from Microsoft C.

Hardware and software requirements

Borland C++ runs on the IBM PC compatible family of computers, including the AT and PS/2, along with all true IBM compatible 286, 386 or 486 computers. Borland C++ requires DOS 3.31 or higher, a hard disk, a floppy drive, and at least 640K plus 1MB of extended memory; it runs on any 80-column monitor. The Turbo C++ for Windows IDE requires protected mode Windows 3.0 or higher, at least 2MB of extended memory and a Windows-compatible monitor.

Borland C++ includes floating-point routines that let your programs make use of an 80x87 math coprocessor chip. It emulates the chip if it is not available. Though it is not required to run Borland C++, the 80x87 chip can significantly enhance the performance of your programs that use floating point math operations.

Borland C++ also supports (but does not require) any Windows-compatible mouse. The Resource Workshop requires a mouse.

The Borland C++ implementation

Borland C++ is a full implementation of the AT&T C++ version 2.1. It also supports the American National Standards Institute (ANSI) C standard. In addition, Borland 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 formal description of Borland C++.
Your Borland C++ package consists of a set of disks and nine manuals:

- Borland C++ User’s Guide (this manual)
- Borland C++ Tools and Utilities Guide
- Borland C++ Programmer’s Guide
- Borland C++ Library Reference
- Resource Workshop User’s Guide
- Turbo Debugger User’s Guide
- Turbo Profiler User’s Guide
- Turbo Assembler User’s Guide
- Turbo Assembler Quick Reference

In addition to these manuals, you’ll find a convenient Quick Reference card. The disks contain all the programs, files, and libraries you need to create, compile, link, and run your Borland C++ programs; they also contain sample programs, many standalone utilities, a context-sensitive help file, an integrated debugger, and additional C and C++ documentation not covered in these guides.

The User’s Guide

The User’s Guide introduces you to Borland C++ and shows you how to create and run both C and C++ programs. It consists of information you’ll need to get up and running quickly, and provides reference chapters on the features of Borland C++:

- Borland’s Programmer’s Platform, including the editor and Project Manager, as well as details on using the command-line compiler.
- These are the chapters in this manual:

**Introduction** introduces you to Borland C++ and tells you where to look for more information about each feature and option.

**Chapter 1: Installing Borland C++** tells you how to install Borland C++ on your system; it also tells you how to customize the colors, defaults, and many other aspects of Borland C++.

**Chapter 2: IDE Basics** introduces the features of the Programmer’s Platform, giving information and examples of how
to use the IDE to full advantage. It includes information on how to start up and exit from the IDE.

Chapter 3: Menus and options reference provides a complete reference to the menus and options in the Programmer's Platform.

Chapter 4: Managing multi-file projects introduces you to Borland C++'s built-in project manager and shows you how to build and update large projects from within the IDE.

Chapter 5: The command-line compiler tells how to use the command-line compiler. It also explains configuration files.

Appendix A: The Optimizer introduces the concepts of compiler optimization, and describes the specific optimization strategies and techniques available in Borland C++.

Appendix B: Editor reference provides a convenient command reference to using the editor with both the CUA command interface and the Borland C++ alternate interface.

Appendix C: Using EasyWin provides a guide to using the EasyWin functions to quickly and easily turn your DOS programs into applications that run under Microsoft Windows.

Appendix D: Precompiled headers tells you how to use Borland C++'s exclusive precompiled headers feature to save substantial time when recompiling large projects, especially Windows applications.

This volume introduces you to the many programming tools and utility programs provided with Borland C++. It contains information you'll need to make full use of the Borland C++ programming environment, including the Make utility, the Turbo Librarian and Linker, the WinSight program and special utilities for Microsoft Windows programming.

Chapter 1: Import library tools tells you how to use the IMPDEF, IMPLIB, and IMPLIBW utilities to define and specify import libraries.

Chapter 2: Make: The program manager introduces the Borland C++ MAKE utility, describes its features and syntax, and presents some examples of usage.
Chapter 3: TLIB: The Turbo librarian tells you how to use the Borland C++ Turbo Librarian to combine object files into integrated library (.LIB) files.

Chapter 4: TLINK: The Turbo linker is a complete reference to the features and functions of the Turbo Linker (TLINK).

Chapter 5: Using WinSight provides instructions for using WinSight to inspect your programs running under Microsoft Windows.

Chapter 6: RC: The Windows resource compiler tells you how to use the Resource Compiler to compile .RC scripts into .RES resource files for your Windows programs.

Chapter 7: HC: The Windows Help compiler contains instructions for using the Help Compiler to create help systems for your Microsoft Windows programs.

Chapter A: Error messages lists and explains run-time, compile-time, linker, librarian, and Help compiler errors and warnings, with suggested solutions.

The Programmer's Guide provides useful material for the experienced C user: a complete language reference for C and C++, writing Windows applications, a cross-reference to the run-time library, C++ streams, memory models, mixed-model programming, video functions, floating-point issues, and overlays, plus error messages.

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

Chapter 5: Using C++ streams tells you how to use the C++ iostreams library, as well as special Borland C++ extensions for Windows.

Chapter 6: The container class library tells you how to use the Borland C++ container class library in your programs.

Chapter 7: Converting from Microsoft C provides some guidelines on converting your Microsoft C programs to Borland C++.
Chapter 8: Building a Windows application introduces you to the concepts and techniques of writing applications for Microsoft Windows using Borland C++.

Chapter 9: DOS memory management covers memory models, mixed-model programming, and overlays.

Chapter 10: Math covers floating-point and BCD math.

Chapter 11: Video functions is devoted to handling text and graphics in Borland C++.

Chapter 12: BASM and inline assembly tells how to write inline assembly language functions that can be assembled with the built-in assembler (BASM) and used within your Borland C++ program.

Appendix A: ANSI implementation-specific standards describes those aspects of the ANSI C standard that have been left loosely defined or undefined by ANSI, and how Borland has chosen to implement them.

The Library Reference contains a detailed list and explanation of Borland C++’s extensive library functions and global variables.

Chapter 1: The main function describes the main function.

Chapter 2: The run-time library is an alphabetically arranged reference to all Borland C++ library functions.

Chapter 3: Global variables defines and discusses Borland C++’s global variables.

Appendix A: Library cross-reference provides a complete indexed locator reference to all Borland C++ library functions.

Using the manuals

The manuals are arranged so that you can pick and choose among the books and chapters to find exactly what you need to know at the time you need to know it. The User's Guide provides information on how to use Borland C++ as a product; the Programmer's Guide and the Library Reference provide material on programming issues in C and C++.
Chapter 1 of this manual (the User's Guide) tells you how to install Borland C++ and how to customize Borland C++'s defaults. The remaining chapters of the User's Guide are for use as reference chapters to using Borland C++'s IDE, editor, project manager, command-line compiler, precompiled headers, and online utilities.

If you don't know C or C++, there are many good products on the market that can get you going in these languages. You can use Chapters 1 through 5 in the Programmer's Guide for reference on specific technical aspects of Borland C++.

Your next step is to start programming in C and C++. You'll find Chapter 2, "The run-time library" in the Library Reference to be a valuable reference on how to use each function. Chapter 1, "The main function," provides information on aspects of the main function that is seldom found elsewhere. Or, you might prefer to use the online help; it contains much of the same information as the Library Reference, and includes programming examples that you can copy into your own programs. Once you have grown comfortable with programming, you may want to move into the more advanced issues covered in the Programmer's Guide.

If you are an experienced C or C++ programmer and you've already installed Borland C++, you'll probably want to jump immediately to the Programmer's Guide and to the Library Reference.

The Programmer's Guide covers certain useful programming issues, such as C++ streams, assembly language interface, memory models, video functions, overlays, and far and huge pointers. If you are interested in writing a Windows application in C++, Chapter 8, "Building a Windows application," provides an overview.

Typefaces and icons used in these books

All typefaces and icons used in this manual were produced by Borland's Sprint: The Professional Word Processor, on a PostScript laser printer.
This typeface represents text as it appears onscreen or in a program. It is also used for anything you must type literally (such as `BC` to start up Borland C++).

**ALL CAPS**

We use all capital letters for the names of constants and files.

()``

Square brackets `[ ]` in text or DOS command lines enclose optional items that depend on your system. 

*Text of this sort should not be typed verbatim.*

<->

Angle brackets in the function reference section enclose the names of include files.

**Boldface**

Borland C++ function names (such as `printf`), class, and structure names are shown in boldface when they appear in text (but not in program examples). This typeface is also used in text for Borland C++ reserved words (such as `char`, `switch`, `near`, and `cdecl`), for format specifiers and escape sequences (`%d`, `	`), 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, such as new terms.

**Keycaps**

This typeface indicates a key on your keyboard. 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*.

This icon indicates material that applies to Turbo C++ for Windows, or which relates specifically to writing a Windows program.

**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.

Borland resources

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.

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.

<table>
<thead>
<tr>
<th>Service</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompuServe</td>
<td>GO BORLAND</td>
</tr>
<tr>
<td>BIX</td>
<td>JOIN BORLAND</td>
</tr>
<tr>
<td>GEnie</td>
<td>BORLAND</td>
</tr>
</tbody>
</table>

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.

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.

Borland Customer Service is available weekdays from 7:00 a.m. to 5:00 p.m. Pacific time to answer any non-technical questions you have about Borland products, including pricing information, upgrades, and order status.
Your Borland C++ package includes two different versions of Borland C++: the IDE (Programmer's Platform) and the DOS command line version. It also includes Turbo C++ for Windows, which runs as a true Windows application.

If you don’t already know how to use DOS commands, refer to your DOS reference manual before setting up Borland C++ on your system.

Borland C++ comes with an automatic installation program called INSTALL. Because we used file-compression techniques, you must use this program; you can’t just copy the Borland C++ files onto your hard disk. Instead, INSTALL automatically copies and uncompresses the Borland C++ and Turbo C++ for Windows files. For reference, the README file on the installation disk includes a list of the distribution files.

We assume you are already familiar with DOS commands. For example, you’ll need the DISKCOPY command to make backup copies of your distribution disks. Make a complete working copy of your distribution disks when you receive them, then store the original disks away in a safe place.

None of Borland’s products use copy protection schemes. If you are not familiar with Borland’s No-Nonsense License Statement, read the agreement included with your Borland C++ package. Be sure to mail us your filled-in product registration card; this guarantees that you’ll be among the first to hear about the hottest new upgrades and versions of Borland C++.

This chapter contains the following information:

- installing Borland C++ and Turbo C++ for Windows on your system
- accessing the README file
- accessing the HELPME! file
- a pointer to more information on Borland’s example programs
Using INSTALL

Among other things, INSTALL detects what hardware you are using and configures Borland C++ appropriately. It also creates directories as needed and transfers files from your distribution disks (the disks you bought) to your hard disk. Its actions are self-explanatory; the following text tells you all you need to know.

To install Borland C++:

1. Insert the installation disk (disk 1) into drive A. Type the following command, then press Enter.
   
   A: INSTALL

2. Press Enter at the installation screen.

3. Follow the prompts.

4. At the end of installation, you may want to add this line to your CONFIG.SYS file:

   FILES = 20

   and this line to your AUTOEXEC.BAT file (or modify your existing PATH statement, if you already have one):

   PATH = C:\BORLANDC\BIN

Important!

When it is finished, INSTALL allows you to read the latest about Borland C++ in the README file, which contains important, last-minute information about Borland C++. The HELPME!.DOC file also answers many common technical support questions.

The next time you start Microsoft Windows (after you exit from the README file viewer) a Borland C++ program group will be created and installed in Program Manager. The program group will contain icons for the following Borland C++ programs and utilities:

- Borland C++
Important!

INSTALL assumes that Microsoft Windows is installed in the directory you specified as your Windows directory during installation. It also assumes that the Program Manager starts up automatically as your Windows “shell” when you start Windows. If you normally use a different command shell from Program Manager, should edit the SYSTEM.INI file in your Windows directory to include the line

```
SHELL=PROGMAN.EXE
```

otherwise you may get a message saying “cannot communicate with Program Manager” when you first open Windows and Borland C++ tries to create a new Program Manager group. Once Turbo C++ for Windows and the other tools are installed in a Program Manager group, you can examine their settings, then reinstall them in your alternate command shell if you want.

Protected mode and memory

Borland C++ utilizes the DPMI (Dos Protected Mode Interface) to run the compiler in protected mode, giving you access to all your computer’s memory without swapping. The protected mode interface is completely transparent to the user, and you should never have to even think about it, with a few possible exceptions.

DPMIINST

Once such exception may be when you run Borland C++ for the very first time. Borland C++ uses an internal database of various machine characteristics to determine how to enable protected mode on your machine, and configures itself accordingly. If your machine is not recognized by Borland C++, you will receive an error message saying

```
Machine not in database (RUN DPMIINST)
```

If you get this message, simply run the DPMIINST program by typing (at the DOS prompt)
DPMIINST

and following the program's instructions. DPMIINST runs your machine through a series of tests to determine the best way of enabling protected mode, and automatically configures Borland C++ accordingly. Once you have run DPMIINST, you will not have to run it again.

DPMIMEM

By default, the Borland C++ DPMI interface will allocate all available extended and expanded memory for its own use. If you don't want all of the available memory to be taken by the DPMI kernel, an environment variable must be set which specifies a maximum amount of memory to use. This variable can be entered directly at the DOS prompt or inserted as a line in your AUTOEXEC.BAT file, using the syntax

DPMIMEM=MAXMEM nnnn

where nnnn is the amount of memory in kilobytes.

For example, if a user has a system with 4MB and wants the DPMI kernel to use 2MB of it, leaving the other 2MB alone, the DPMIMEM variable would be set as follows:

c:> set DPMIMEM=MAXMEM 2000

When running under Windows 3.0 in 386 enhanced mode, it is not necessary to set the DPMIMEM variable; instead, you should use a Windows PIF file to configure the memory usage of Borland C++.

Under Windows standard mode, we suggest that the Borland DPMI kernel be pre-loaded prior to running windows. This is done by running DPMIRES.EXE (see the discussion of DPMIRES which follows). When using DPMIRES in conjunction with Windows, you should always set the DPMIMEM variable to less than the maximum available memory to insure that Windows will have enough physical memory to operate.

DPMIRES

DPMIRES is a Borland utility that can be used with BC 3.0 to increase performance of some of the Borland language tools under certain conditions. In particular, the performance of the following tools can be enhanced through its use:

- BCC
- TASMX
**TLINK**

When run, DPMIRES will enable the Dos Protected Mode interface and spawn a DOS command shell. The applications mentioned above will load faster into this shell. Typing 'EXIT' to the shell will remove it.

DPMIRES is especially useful if you are compiling with MAKER (the real mode MAKE) or with batch files, instead of using the protected mode MAKE. In this situation, it will be more efficient to run DPMIRES and then run MAKER or the batch file, since the compiler will load faster on each invocation.

**NOTE:** If you are running under DPMIRES, you may not run Windows 3.0 in enhanced mode. You must first exit to DOS and then run Windows 3.0.

Once the DPMI kernel is loaded (either by running BC or through the DPMIRES utility), the Borland C++ integrated development environment interacts directly with the DPMI server to allocate its memory, both to load and while operating. By default, the IDE will use all the extended memory reserved by the DPMI kernel and all available EMS (expanded) memory, the EMS memory being used as a swap device.

The Options I Environment I Startup... dialog and the /X and /E command line switches can be used to change this behavior. These settings do not affect the memory reserved by the kernel itself, only how much of it is used by the IDE.

The Use Extended Memory dialog item (and the /X command line option) can be used to tell BC how much of the memory reserved by the DPMI kernel to use. The main reason for limiting BC's use of the kernel's memory is to allow running of other DPMI applications from within the IDE's (using the Transfer capability), or from a DOS shell opened from the IDE.

The Use EMS Memory dialog item (and the /E command line option) are used to tell the IDE how many 16K EMS pages to use as a swap device. Unless the kernel has been instructed to leave aside some available memory, there will be no EMS pages available to the IDE.
Once you have installed Borland C++, and if you’re anxious to get up and running, change to the Borland C++ \BIN directory, type BC and press Enter. Or, you may wish to run Turbo C++ for Windows, by clicking on the Turbo C++ for Windows icon in the Program Manager. Otherwise, continue reading this chapter and the next for important start-up information.

After you have tried out the IDE, you may want to permanently customize some of the options. The Options | Environment | Startup and Options | Environment | Colors selections in the IDE make this easy to do; see page 19 for more information.

If you have a laptop computer (one with an LCD or plasma display), in addition to carrying out the procedures given in the previous sections, you need to set your screen parameters before using Borland C++. The IDE works best if you type MODE BW80 at the DOS command line before running Borland C++.

Although you could create a batch file to take care of this for you, you can also easily install Borland C++ for a black-and-white screen from within the IDE, using the Options | Environment | Startup option. Choose “Black and White / LCD” from the Video options group.

The README file contains last-minute information that may not be in the manuals.

Borland C++ automatically places you in the README file when you run the INSTALL program. To access the README file at a later time you can use the Borland C++ README program by typing at the DOS command line:

```dos
README
```
The HELPME!.DOC file

Your installation disk also contains a file called FILELIST.DOC, which lists every file on the distribution disks, with a brief description of what each one contains, and HELPME!.DOC, which contains answers to problems that users commonly run into. Consult it if you find yourself having difficulties. You can use the README program to look at HELPME!.DOC. Type this at the command line:

README HELPME!.DOC

Example programs

Your Borland C++ package includes the source code for a large number of example programs in C and C++ for both DOS and Windows, including a complete spreadsheet program called Turbo Calc. These programs are located in the ..\EXAMPLES directory (and subdirectories) created by INSTALL. The ..\EXAMPLES directory also contains subdirectories for examples of the other tools and utilities that come with Borland C++ (like the Turbo Assembler, Debugger and Resource Workshop). Before you compile any of these example programs, you should read the printed or online documentation for them.

Customizing the IDE

Borland C++ version 3.0 allows you completely customize your installation from within the IDE itself, using the various options that appear under the Options | Environment menu. These options allow you to specify the video mode, editing modes, menu colors, and default directories, among others.
Chapter 2, IDE basics

Borland's Programmer's Platform, also known as the integrated development environment or IDE, has everything you need to write, edit, compile, link, and debug your programs. It provides

- multiple, movable, resizable windows
- mouse support
- dialog boxes
- cut, paste, and copy commands that use the Clipboard
- full editor undo and redo
- examples ready to copy and paste from Help
- a built-in assembler
- quick transfer to other programs (like Turbo Assembler) and back again
- an editor macro language

This chapter explains how to start up and exit the Borland C++ IDE, discusses its generic components, and explains how configuration and project files work. Since the Turbo C++ for Windows IDE comes in this package, the last section describes its environment. Most of the features of the Borland C++ IDE are in the Turbo C++ for Windows IDE also.
Starting and exiting

---

**Borland C++ runs only in protected mode.**

To start the IDE, type `bc` at the DOS prompt. You can follow it with one or more command-line options.

---

**Command-line options**

The command-line options for Borland C++'s IDE are `/b, /d, /e, /h, /l, /m, /p, /rx, /s, and /x` which use this syntax:

```
BC [option [option...]] [sourcename \ projectname [sourcename]]
```

where `option` can be one or more of the options, `sourcename` is any ASCII file (default extension assumed), and `projectname` is your project file (it must have the .PRJ extension).

To turn an option off, follow the option with a minus sign. For example,

```
BC /e-
```

turns off the default swap to expanded memory option.

---

**The /b option**

The `/b` option causes Borland C++ to recompile and link all the files in your project, print the compiler messages to the standard output device, and then return to the operating system. This option allows you to start Borland C++ from a batch file so you can automate project builds. Borland C++ determines what .EXE to build based on the project file you specified on the command line or the file loaded in the active edit window if no project file is found.

To specify a project file, enter the `bc` command followed by `/b` and then the project file name. For example,

```
BC /b myproj.prj
```

This command loads a file in the editor and then compiles and links it:

```
BC myprog /b
```

---

**The /d option**

The `/d` option causes Borland C++ to work in dual monitor mode if it detects appropriate hardware (for example, a monochrome card and a color card); otherwise, the `/d` option is ignored. Using dual monitor mode makes it easier to watch a program's output while you are debugging the program.
If your system has two monitors, DOS treats one monitor as the active monitor. Use the DOS MODE command to switch between the two monitors (MODE CO80, for example, or MODE MONO). In dual monitor mode, the normal Borland C++ screen appears on the inactive monitor, and program output will go to the active monitor. So when you type `BC /d` at the DOS prompt on one monitor, Borland C++ comes up on the other monitor. When you want to test your program on a particular monitor, exit Borland C++, switch the active monitor to the one you want to test with, and then issue the `BC /d` command again. Program output then goes to the monitor where you typed the `BC` command.

Keep the following in mind when using the `/d` option:

- Don't change the active monitor (by using the DOS MODE command, for example) while you are in a DOS shell (File | DOS Shell).
- User programs that directly access ports on the inactive monitor's video card are not supported, and can cause unpredictable results.
- When you run or debug programs that explicitly make use of dual monitors, do not use the Borland C++ dual monitor option `/d`.

The `/e` option
The `/e` option tells Borland C++ to swap to expanded memory if necessary; it is on by default. The syntax for this option is as follows:

```
/e[=n]
```

where `n` equals the number of pages of expanded memory that you want the IDE to use for swapping. A page is 16K.

The `/h` option
If you type `BC /h` on the command line, you get a list of all the command-line options available. Their default values are also shown.

The `/l` option
Use the `/l` option if you're running Borland C++ on an LCD screen.

The `/m` option
The `/m` option lets you do a make rather than a build (that is, only outdated source files in your project are recompiled and linked). Follow the instructions for the `/b` option, but use `/m` instead.
The /p option
If your program modifies the EGA palette registers, use the /p option, which controls palette swapping on EGA video adapters. The EGA palette is restored each time the screen is swapped.

In general, you don’t need to use this option unless your program modifies the EGA palette registers or unless your program uses BGI to change the palette.

The /r option
/rx specifies the swap drive. If all your virtual memory fills up, you can have Borland C++ swap to a drive you specify, usually a RAM disk. The x in /rx is the letter of the fast swap drive. For example, /rd will use drive D as the swap drive.

The /s option
Using the /s option, the compiler allows the majority of available memory to be allocated for its internal tables while compiling. If it is compiling large modules, little memory may remain for the needed overlays; therefore, the compiler may spend a long time “thrashing,” that is, swapping overlays in and out of memory.

If you specify /s-, the compiler won’t permit its internal tables to severely restrict the overlay space in memory. As a result, if you are compiling very large modules, the compilation may fail and you’ll get an out-of-memory error, but the compiler won’t thrash excessively.

The /x option
Use the /x switch to tell Borland C++ how much of the available extended memory to use for its heap space.

/x
uses all available memory.

/x[=n]
where n equals the amount of memory in kilobytes, let’s you specify how much extended memory should be used.

Exiting Borland C++
There are three ways to leave the IDE.

- Choose File | Exit to leave the IDE completely; you have to type BC again to reenter it. You’ll be prompted to save your programs before exiting, if you haven’t already done so.
You return to the IDE after you exit the program you transferred to.

- Choose File | DOS Shell to shell out from the IDE to enter commands at the DOS command line. When you're ready to return to the IDE, type `EXIT` at the command line and press `Enter`. The IDE reappears just as you left it.
- Choose a program from the System menu (=) to temporarily transfer to another program without leaving the IDE. You can add new Transfer programs with the Options | Transfer command.

The components

There are three visible components to the IDE: the menu bar at the top, the window area in the middle, and the status line at the bottom. Many menu items also offer dialog boxes. Before we describe each menu item in the IDE, we'll explain these more generic components.

The menu bar

The menu bar is your primary access to all the menu commands. The menu bar is always visible except when you're viewing your program's output or transferring to another program.

If a menu command is followed by an ellipsis (...), choosing the command displays a dialog box. If the command is followed by an arrow (•), the command leads to another menu (a pop-up menu). If the command has neither an ellipsis nor an arrow, the action occurs as soon as you choose the command.

Here is how you choose menu commands using the keyboard:

1. Press `F10`. This makes the menu bar active; the next thing you type will relate to the items on the menu bar.
2. Use the arrow keys to select the menu you want to display. Then press `Enter`.

   As a shortcut for this step, you can just press the highlighted letter of the menu title. For example, from the menu bar, press `E` to move to and display the Edit menu. From anywhere, press `Alt` and the highlighted letter (such as `Alt+E`) to display the menu you want.
3. Use the arrow keys again to select a command from the menu you've opened. Then press `Enter`.
At this point, Borland C++ either carries out the command, displays a dialog box, or displays another menu.

There are two ways to choose commands with a mouse:

- Click the desired menu title to display the menu and click the desired command.
- Or, drag straight from the menu title down to the menu command. Release the mouse button on the command you want. (If you change your mind, just drag off the menu; no command will be chosen.)

Note that some menu commands are unavailable when it would make no sense to choose them. However, you can always get Online Help about currently unavailable commands.

Shortcuts

Borland C++ offers a number of quick ways to choose menu commands. The click-drag method for mouse users is an example. From the keyboard, you can use a number of keyboard shortcuts (or hot keys) to access the menu bar and choose commands. Shortcuts for dialog boxes work just as they do in a menu. (But be aware that you need to hold down Alt while pressing the highlighted letter when moving from an input box to a group of buttons or boxes.) Here's a list of the shortcuts available:

<table>
<thead>
<tr>
<th>Do this...</th>
<th>To accomplish this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Press Alt plus the highlighted letter of the command (just press the highlighted letter in a dialog box). For the menu, press Alt+Spacebar.</td>
<td>Display the menu or carry out the command.</td>
</tr>
<tr>
<td>Type the keystrokes next to a menu command.</td>
<td>Carry out the command.</td>
</tr>
</tbody>
</table>

For example, to cut selected text, press Alt+E T (for Edit | Cut) or you can just press Shift+Del, the shortcut displayed next to it.

Many menu items have corresponding hot keys; one- or two-key shortcuts that immediately activate that command or dialog box.

Command sets

Borland C++ has two command sets: the Common User Access (CUA) command set, the standard used by most Windows programs and the Alternate command set popularized in previous Borland products. The shortcuts available to you differ depending on which command set you use. You can select a
command set by choosing Options | Environment | 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 Borland C++ hot keys in both command sets.

Table 2.1: General hot keys

<table>
<thead>
<tr>
<th>CUA</th>
<th>Alternate</th>
<th>Menu item</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>F1</td>
<td>Help</td>
<td>Displays a help screen.</td>
</tr>
<tr>
<td>F2</td>
<td>F2</td>
<td>File</td>
<td>Save</td>
</tr>
<tr>
<td>F3</td>
<td>F3</td>
<td>File</td>
<td>Open</td>
</tr>
<tr>
<td>F4</td>
<td>F4</td>
<td>Run</td>
<td>Go to Cursor</td>
</tr>
<tr>
<td>Ctrl+F6</td>
<td>F6</td>
<td>Window</td>
<td>Zoom</td>
</tr>
<tr>
<td>F7</td>
<td>F7</td>
<td>Run</td>
<td>Trace Into</td>
</tr>
<tr>
<td>F8</td>
<td>F8</td>
<td>Run</td>
<td>Step Over</td>
</tr>
<tr>
<td>F9</td>
<td>F9</td>
<td>Compile</td>
<td>Make</td>
</tr>
<tr>
<td>F10</td>
<td>F10</td>
<td>(none)</td>
<td>Takes you to the menu bar.</td>
</tr>
</tbody>
</table>

Table 2.2: Menu hot keys

<table>
<thead>
<tr>
<th>CUA</th>
<th>Alternate</th>
<th>Menu item</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+Spacebar</td>
<td>Alt+Spacebar</td>
<td>= menu</td>
<td>Takes you to the = (System) menu</td>
</tr>
<tr>
<td>Alt+C</td>
<td>Alt+C</td>
<td>Compile menu</td>
<td>Takes you to the Compile menu</td>
</tr>
<tr>
<td>Alt+D</td>
<td>Alt+D</td>
<td>Debug menu</td>
<td>Takes you to the Debug menu</td>
</tr>
<tr>
<td>Alt+E</td>
<td>Alt+E</td>
<td>Edit menu</td>
<td>Takes you to the Edit menu</td>
</tr>
<tr>
<td>Alt+F</td>
<td>Alt+F</td>
<td>File menu</td>
<td>Takes you to the File menu</td>
</tr>
<tr>
<td>Alt+H</td>
<td>Alt+H</td>
<td>Help menu</td>
<td>Takes you to the Help menu</td>
</tr>
<tr>
<td>Alt+O</td>
<td>Alt+O</td>
<td>Options menu</td>
<td>Takes you to the Options menu</td>
</tr>
<tr>
<td>Alt+P</td>
<td>Alt+P</td>
<td>Project menu</td>
<td>Takes you to the Project menu</td>
</tr>
<tr>
<td>Alt+R</td>
<td>Alt+R</td>
<td>Run menu</td>
<td>Takes you to the Run menu</td>
</tr>
<tr>
<td>Alt+S</td>
<td>Alt+S</td>
<td>Search menu</td>
<td>Takes you to the Search menu</td>
</tr>
<tr>
<td>Alt+W</td>
<td>Alt+W</td>
<td>Window menu</td>
<td>Takes you to the Window menu</td>
</tr>
<tr>
<td>Alt+F4</td>
<td>Alt+F</td>
<td>File</td>
<td>Exit</td>
</tr>
</tbody>
</table>
### Table 2.3: Editing hot keys

<table>
<thead>
<tr>
<th>CUA</th>
<th>Alternate</th>
<th>Menu item</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Ins</td>
<td>Ctrl+Ins</td>
<td>Edit</td>
<td>Copy</td>
</tr>
<tr>
<td>Shift+Del</td>
<td>Shift+Del</td>
<td>Edit</td>
<td>Cut</td>
</tr>
<tr>
<td>Shift+Ins</td>
<td>Shift+Ins</td>
<td>Edit</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+Del</td>
<td>Ctrl+Del</td>
<td>Edit</td>
<td>Clear</td>
</tr>
<tr>
<td>Alt+Bkspc</td>
<td>Alt+Bkspc</td>
<td>Edit</td>
<td>Undo</td>
</tr>
<tr>
<td>Alt+Shift+Bksp Alt+Shift+Bksp</td>
<td>Alt+Shift+Bksp Alt+Shift+Bksp</td>
<td>Edit</td>
<td>Redo</td>
</tr>
<tr>
<td>F3</td>
<td>Ctrl+L</td>
<td>Search</td>
<td>Search Again</td>
</tr>
<tr>
<td>F2</td>
<td>File</td>
<td>Save</td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>F3</td>
<td>File</td>
<td>Open</td>
</tr>
</tbody>
</table>

### Table 2.4: Window management hot keys

<table>
<thead>
<tr>
<th>CUA</th>
<th>Alternate</th>
<th>Menu item</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+#</td>
<td>Alt+#</td>
<td></td>
<td>Displays a window, where # is the number of the window you want to view</td>
</tr>
<tr>
<td>Alt+0</td>
<td>Alt+0</td>
<td>Window</td>
<td>List</td>
</tr>
<tr>
<td>Ctrl+F4</td>
<td>Alt+F3</td>
<td>Window</td>
<td>Close</td>
</tr>
<tr>
<td>Shift+F5</td>
<td>Alt+F5</td>
<td>Window</td>
<td>Tile</td>
</tr>
<tr>
<td>Alt+F5</td>
<td>Alt+F4</td>
<td>Debug</td>
<td>Inspect</td>
</tr>
<tr>
<td>Shift+F5</td>
<td>Alt+F5</td>
<td>Window</td>
<td>User Screen</td>
</tr>
<tr>
<td>F5</td>
<td>F5</td>
<td>Window</td>
<td>Zoom</td>
</tr>
<tr>
<td>Ctrl+F6</td>
<td>F6</td>
<td>Window</td>
<td>Next</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Ctrl+F5</td>
<td></td>
<td>Changes size or position of active window</td>
</tr>
</tbody>
</table>

### Table 2.5: Online Help hot keys

<table>
<thead>
<tr>
<th>CUA</th>
<th>Alternate</th>
<th>Menu item</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>F1</td>
<td>Help</td>
<td>Contents</td>
</tr>
<tr>
<td>F1 F1</td>
<td>F1 F1</td>
<td></td>
<td>Brings up Help on Help. (Just press F1 when you’re already in the help system.)</td>
</tr>
<tr>
<td>Shift+F1</td>
<td>Shift+F1</td>
<td>Help</td>
<td>Index</td>
</tr>
<tr>
<td>Alt+F1</td>
<td>Alt+F1</td>
<td>Help</td>
<td>Previous Topic</td>
</tr>
<tr>
<td>Ctrl+F1</td>
<td>Ctrl+F1</td>
<td>Help</td>
<td>Topic Search</td>
</tr>
</tbody>
</table>
### Table 2.6: Debugging/Running hot keys

<table>
<thead>
<tr>
<th>CUA</th>
<th>Alternate</th>
<th>Menu Item</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt+F5</td>
<td>Alt+F4</td>
<td>Debug</td>
<td>Inspect</td>
</tr>
<tr>
<td>Alt+F7</td>
<td>Alt+F7</td>
<td>Search</td>
<td>Previous Error</td>
</tr>
<tr>
<td>Alt+F8</td>
<td>Alt+F8</td>
<td>Search</td>
<td>Next Error</td>
</tr>
<tr>
<td>Alt+F9</td>
<td>Alt+F9</td>
<td>Compile</td>
<td>Compile</td>
</tr>
<tr>
<td>Ctrl+F2</td>
<td>Ctrl+F2</td>
<td>Run</td>
<td>Program Reset</td>
</tr>
<tr>
<td>Ctrl+F3</td>
<td>Ctrl+F3</td>
<td>Debug</td>
<td>Call Stack</td>
</tr>
<tr>
<td>Ctrl+F4</td>
<td>Ctrl+F4</td>
<td>Debug</td>
<td>Evaluate/Modify</td>
</tr>
<tr>
<td>Ctrl+F5</td>
<td>Ctrl+F5</td>
<td>Debug</td>
<td>Add Watch</td>
</tr>
<tr>
<td>Ctrl+F6</td>
<td>Ctrl+F6</td>
<td>Debug</td>
<td>Toggle Breakpoint</td>
</tr>
<tr>
<td>Ctrl+F7</td>
<td>Ctrl+F7</td>
<td>Run</td>
<td>Run</td>
</tr>
<tr>
<td>Ctrl+F8</td>
<td>Ctrl+F8</td>
<td>Run</td>
<td>Go To Cursor</td>
</tr>
<tr>
<td>Ctrl+F9</td>
<td>Ctrl+F9</td>
<td>Run</td>
<td>Step Over</td>
</tr>
<tr>
<td>F7</td>
<td>F7</td>
<td>Run</td>
<td>Trace Into</td>
</tr>
<tr>
<td>F8</td>
<td>F8</td>
<td>Run</td>
<td>Make</td>
</tr>
</tbody>
</table>

Native makes the Alternate command set the default for Borland C++, the DOS-hosted IDE, and the CUA command set the default for Turbo C++ for Windows.

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, which determines which command set is in effect. 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 the Borland C++.

But maybe this is not what you want. When you are 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 this happen.

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

If you change the command set in either Borland C++ or Turbo C++, you change it for both products.

While Native seems to imply that the default command set for Borland C++ is Alternate, we recommend you choose the CUA command set.
Which command set you choose also determines which keys you use within the editor, and, to some extent, how the editor works. See more about using command sets in the editor in Appendix B.

Most of what you see and do in the IDE happens in a window. A window is a screen area that you can open, close, move, resize, zoom, tile, and overlap.

You can have many windows open in the IDE, but only one window can be active at any time. The active window is the one that you’re currently working in. Any command you choose or text you type generally applies only to the active window. (If you have the same file open in several windows, the action will apply to the file everywhere that it’s open.)

You can spot the active window easily: It’s the one with the double-lined border around it. The active window always has a close box, a zoom box, and scroll bars. If your windows are overlapping, the active window is always the one on top of all the others (the frontmost one).

There are several types of windows, but most of them have these things in common:

- a title bar
- a close box
- scroll bars
- a zoom box
- a window number (1 to 9)

A edit window also displays the current line and column numbers in the lower left corner. If you’ve modified your file, an asterisk (*) will appear to the left of the column and line numbers.

The following figure shows a typical window:
The close box of a window is the box in the upper left corner. Click this box to quickly close the window. (Or choose Window | Close.) The Inspector and Help windows are considered temporary; you can close them by pressing Esc.

The title bar, the topmost horizontal bar of a window, contains the name of the window and the window number. Double-clicking the title bar zooms the window. You can also drag the title bar to move the window around.

The zoom box of a window appears in the upper right corner. If the icon in that corner is an up arrow (†), you can click the arrow to enlarge the window to the largest size possible. If the icon is a double-headed arrow (‡), the window is already at its maximum size. In that case, clicking it returns the window to its previous size. To zoom a window from the keyboard, choose Window | Zoom.

The first nine windows you open in Borland C++ have a window number in the upper right border. You can make a window active
(and thereby bring it to the top of the heap) by pressing \textit{Alt} in combination with the window number. For example, if the Help window is \#5 but has gotten buried under the other windows, \textit{Alt}+5 brings it to the front.

\textit{Scroll bars} are horizontal or vertical bars that look like this:

```
<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

You use these bars with a mouse to scroll the contents of the window. Click the arrow at either end to scroll one line at a time. (Keep the mouse button pressed to scroll continuously.) You can click the shaded area to either side of the scroll box to scroll a page at a time. Finally, you can drag the scroll box to any spot on the bar to quickly move to a spot in the window relative to the position of the scroll box.

You can drag any corner to make a window larger or smaller. To resize using the keyboard, choose Size/Move from the Window menu.

Table 2.7 gives you a quick rundown of how to handle windows in Borland C++. Note that you don’t need a mouse to perform these actions—a keyboard works just fine.

<table>
<thead>
<tr>
<th>To accomplish this:</th>
<th>Use one of these methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open an edit window</td>
<td>Choose File</td>
</tr>
<tr>
<td>Open other windows</td>
<td>Choose the desired window from the Window menu</td>
</tr>
<tr>
<td>Close a window</td>
<td>Choose Close from the Window menu or click the close box of the window.</td>
</tr>
<tr>
<td>Activate a window</td>
<td>Click anywhere in the window, or Press \textit{Alt} plus the window number (1 to 9, in the upper right border of the window), or Choose Window</td>
</tr>
<tr>
<td>Move the active window</td>
<td>Drag its title bar. Or choose Window</td>
</tr>
</tbody>
</table>
Table 2.7: Manipulating windows (continued)

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resize the active window</td>
<td>Drag any corner. Or choose Window</td>
</tr>
<tr>
<td>Zoom the active window</td>
<td>Click the zoom box in the upper right corner of the window, or Double-click the window’s title bar, or Choose Window</td>
</tr>
</tbody>
</table>

The status line

The status line appears at the bottom of the screen; it

- reminds you of basic keystrokes and shortcuts (or hot keys) applicable at that moment in the active window.
- lets you click the shortcuts to carry out the action instead of choosing the command from the menu or pressing the shortcut keystroke.
- tells you what the program is doing. For example, it displays Saving filename... when an edit file is being saved.
- offers one-line hints on any selected menu command and dialog box items.

The status line changes as you switch windows or activities. One of the most common status lines is the one you see when you’re actually writing and editing programs in an edit window. Here is what it looks like:

```
F1 Help   F2 Save   F3 Open   F7 Trace   F8 Step   F9 Make   F10 Menu
```

When you’ve selected a menu title or command, the status line changes to display a one-line summary of the function of the selected item.

Dialog boxes

A menu command with an ellipsis after it (...) leads to a dialog box. Dialog boxes offer a convenient way to view and set multiple options. When you’re making settings in dialog boxes, you work with five basic types of onscreen controls: radio buttons, check
boxes, action buttons, input boxes, and list boxes. Here's a sample
dialog box that illustrates some of these items:

This dialog box has three standard buttons: OK, Cancel, and Help.
If you choose OK, the choices in the dialog box are made; if you
choose Cancel, nothing changes and no action is made, but the
dialog box is put away. Choose Help to open a Help window
about this dialog box. Esc is always a keyboard shortcut for
Cancel (even if no Cancel button appears).

If you're using a mouse, click the button you want. When you're
using the keyboard, press Alt and the highlighted letter of an item
to activate it. For example, Alt+K selects the OK button. Press Tab or
Shift+Tab to move forward or back from one item to another in a
dialog box. Each element is highlighted when it becomes active.

In this dialog box, OK is the default button, which means you need
only press Enter to choose that button. (On monochrome systems,
arrows indicate the default; on color monitors, default buttons are
highlighted.) Be aware that tabbing to a button makes that button
the default.

When you select a check box, an x appears in it to show you it's
on. An empty box indicates it's off. To change the status of a check
box, click it or its text, press Tab until the check box is highlighted
and then press Spacebar, or select Alt and the highlighted letter. You
can have any number of check boxes checked at any time.

If several check boxes apply to a topic, they appear as a group. In
that case, tabbing moves to the group. Once the group is selected,
use the arrow keys to select the item you want, and then press Spacebar
to check or uncheck it. On monochrome monitors, the active
check box or group of check boxes will have a chevron symbol (») to the left and right. When you press Tab, the chevrons
move to the next group of checkboxes or radio buttons.
Radio buttons are so called because they act just like the buttons on a car radio. There is always one—and only one—button pushed in at a time. Push one in, and the one that was in pops out.

Radio buttons differ from check boxes in that they present mutually exclusive choices. For this reason, radio buttons always come in groups, and only one radio button can be on in any one group at any one time. To choose a radio button, click it or its text. From the keyboard, select Alt and the highlighted letter, or press Tab until the group is highlighted and then use the arrow keys to choose a particular radio button. Press Tab or Shift+Tab again to leave the group with the new radio button chosen. The column to the left gives an example of a set of radio buttons.

Input boxes let you type in text. Most basic text-editing keys work in the text box (for example, arrow keys, Home, End, and insert/overwrite toggles by Ins). If you continue to type once you reach the end of the box, the contents automatically scroll. If there's more text than what shows in the box, arrowheads appear at the end (• and •). You can click the arrowheads to scroll or drag the text. If you need to enter control characters (such as ^L or ^M) in the input box, then prefix the character with a ^P. So, for example, to enter ^L into the input box, hold down the Ctrl key and press PL. (This capability is useful for search strings.)

If an input box has a down-arrow icon to its right, there is a history list associated with that input box. Press Enter to select an item from this list. In the list you'll find text you typed into this box the last few times you used this dialog box. The Find box, for example, has such a history list, which keeps track of the text you searched for previously. If you want to reenter text that you already entered, press ↓ or click the ↓ icon. You can also edit an entry in the history list. Press Esc to exit from the history list without making a selection.

Here is what a history list for the Find text box might look like if you had used it six times previously:
A final component of many dialog boxes is a list box, which lets you scroll through and select from variable-length lists (often file names) without leaving a dialog box. If a blinking cursor appears in the list box and you know what you’re looking for, you can type the word (or the first few letters of the word) and Borland C++ will search for it.

You make a list box active by clicking it or by choosing the highlighted letter of the list title (or press Tab until it’s highlighted). Once a list box is displayed, you can use the scroll box to move through the list or press ↑ or ↓ from the keyboard.

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, etc.)
- mouse preferences
- auto-save flags

The configuration file is not required to build programs defined by a project.

When you start a programming session, Borland C++ looks for TCCONFIG.TC first in the current directory and then in the
directory that contains BC.EXE. Turbo C++ also looks in the current directory but, if it doesn’t find TCCONFIG.TC, it looks in the directory that contains 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:

- compiler, linker, make and librarian options
- directory paths
- list of all files that make up the project
- special translators (such as Turbo Assembler)

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.

Project files for the IDE correspond to the .CFG configuration files that you supply to the command-line compiler (the default command-line compiler configuration file is TURBOC.CFG). The PRJCFG utility can convert .PRJ files to .CFG files and .CFG files to .PRJ files.

You can load project files in any of three ways:

1. When starting Borland C++, give the project name with the .PRJ extension after the BC command; for example,
   
   BC myproj.PRJ
   
2. You must use the .PRJ extension to differentiate it from source files.

3. 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. Thus, typing BC alone while the current directory contains one project file causes that project file to be loaded.

4. From within the IDE, you load a project file using the Project I Open Project command.
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. Note, however, that changing directories after loading a project may make the relative paths incorrect and your project unbuildable. If this happens, change the current directory back to where the project was loaded from.

Desktop files

Each project file has an associated desktop file (.DSK) that file contains state information about the associated project. While none of its information is needed to build the project, all of the information is directly related to the project. The desktop file includes

- 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
- watch expressions
- breakpoints

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

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: TCDEF.DPR and TCDEF.DSK files, collectively referred to as the default project.
These files are usually stored in the same directory as BC.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).

When you start a new project, the options you set in your previous project will be in effect.

The Turbo C++ for Windows IDE

The Turbo C++ for Windows IDE has everything you need to write, edit, compile, and link your programs in a Windows-hosted environment. You can even start up the powerful Turbo Debugger for Windows without leaving the IDE.

The Turbo C++ IDE is 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.

Starting Turbo C++ for Windows

As you do with other Windows products, 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.
- 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 choose OK.

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

Command sets

Just as Borland C++ does, 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.

Here are the menu shortcuts in the Turbo C++ IDE:

Table 2.8: General hot keys

<table>
<thead>
<tr>
<th>CUA</th>
<th>Alternate</th>
<th>Menu item</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td></td>
<td>File</td>
<td>Save</td>
</tr>
<tr>
<td>F3</td>
<td></td>
<td>File</td>
<td>Open</td>
</tr>
<tr>
<td>Alt+F4</td>
<td>Alt+X</td>
<td>File</td>
<td>Exit</td>
</tr>
<tr>
<td>Alt+Space</td>
<td>Alt+Space</td>
<td>(none)</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

Table 2.9: Editing hot keys

<table>
<thead>
<tr>
<th>CUA</th>
<th>Alternate</th>
<th>Menu item</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Ins</td>
<td>Ctrl+Ins</td>
<td>Edit</td>
<td>Copy</td>
</tr>
<tr>
<td>Shift+Del</td>
<td>Shift+Del</td>
<td>Edit</td>
<td>Cut</td>
</tr>
<tr>
<td>Shift+Ins</td>
<td>Shift+Ins</td>
<td>Edit</td>
<td>Paste</td>
</tr>
<tr>
<td>Ctrl+Del</td>
<td>Ctrl+Del</td>
<td>Edit</td>
<td>Clear</td>
</tr>
</tbody>
</table>
Table 2.9: Editing hot keys (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>CUA</th>
<th>Alternate</th>
<th>Menu item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restores the text in the active window to a previous state.</td>
<td>Alt+Bksp</td>
<td>Alt+Bksp</td>
<td>Edit</td>
</tr>
<tr>
<td>“Undoes” the previous Undo.</td>
<td>Alt+Shift+Bksp Alt+Shift+Bksp</td>
<td>Edit</td>
<td>Redo</td>
</tr>
<tr>
<td>Repeats last Find or Replace command</td>
<td>F3 Ctrl+L</td>
<td>Search</td>
<td>Search Again</td>
</tr>
</tbody>
</table>

Table 2.10: Online Help hot keys

<table>
<thead>
<tr>
<th>Function</th>
<th>CUA</th>
<th>Alternate</th>
<th>Menu item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brings up Help index</td>
<td>Shift+F1</td>
<td>Shift+F1</td>
<td>Help</td>
</tr>
<tr>
<td>Calls up language-specific help in the active edit window</td>
<td>Ctrl+F1</td>
<td>Ctrl+F1</td>
<td>Help</td>
</tr>
</tbody>
</table>

Table 2.11: Compiling/Running hot keys

<table>
<thead>
<tr>
<th>Function</th>
<th>CUA</th>
<th>Alternate</th>
<th>Menu item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Takes you to previous error</td>
<td>Alt+F7</td>
<td>Alt+F7</td>
<td>Search</td>
</tr>
<tr>
<td>Takes you to next error</td>
<td>Shift+F4</td>
<td>Alt+F8</td>
<td>Search</td>
</tr>
<tr>
<td>Runs program</td>
<td>Ctrl+F9</td>
<td>Ctrl+F9</td>
<td>Run</td>
</tr>
<tr>
<td>Invokes Project Manager to make an .EXE, .DLL, or .LIB file</td>
<td>F9</td>
<td>F9</td>
<td>Compile</td>
</tr>
<tr>
<td>Compiles file in active edit window</td>
<td>Alt+F9</td>
<td>Alt+F9</td>
<td>Compile</td>
</tr>
</tbody>
</table>

Although there are only two command sets, there is a third command set option: Native. Its purpose is to make switching between the Borland C++ and the Turbo C++ IDEs easier. See page 29 for information about the Native option.

Which command set you choose also determines which keys you use within the editor, and, to some extent, how the editor works. See more about using command sets in the editor in Appendix B.

Configuration and project files

Turbo C++ handles project management just as it does for Borland C++. See page 36 for information about configuration, project, and desktop files.
Using the SpeedBar

Turbo C++ for Windows has a SpeedBar you can use as a quick way to choose menu commands and other actions with your mouse. The first time you start Turbo C++ for Windows, the SpeedBar will be a horizontal grouping of buttons just under the menu bar. You can use it as it is, change it to be a vertical bar that appears on the left side of the Turbo C++ desktop window, or change it to be a pop-up palette you can move anywhere on your screen. You can also turn it off. To reconfigure the SpeedBar, choose Options | Environment | Desktop and select the option you want.

The buttons on the SpeedBar represent menu commands. They are shortcuts for your mouse, just as certain key combinations are shortcuts when you use 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 the Open command on the File menu.

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

These are the buttons that appear on the SpeedBar:

- ![Help](help_icon.png)
  - Help
- ![Search again](search_icon.png)
  - Search again
- ![Open a file](open_icon.png)
  - Open a file
- ![Cut to Clipboard](cut_icon.png)
  - Cut to Clipboard
- ![Save file](save_icon.png)
  - Save file
- ![Copy to Clipboard](copy_icon.png)
  - Copy to Clipboard
- ![Search for text](search_icon.png)
  - Search for text
- ![Paste from Clipboard](paste_icon.png)
  - Paste from Clipboard
Some of the buttons on the SpeedBar are occasionally dimmed, just as some of the menu commands are. This means that, in the current context, the command the button represents is not available to you. For example, the Paste from Clipboard button will be dimmed if there is nothing in your Clipboard.
Menus and options reference

This chapter provides a reference to each menu option in the IDE. It is arranged in the order that the menus appear on the screen. For information on starting and exiting the IDE, using the IDE command-line options, and general information on how the IDE works, see Chapter 2.

Next to some of the menu option descriptions in this reference you’ll see keyboard shortcuts, or hot keys. If a command set appears above the hot key, the hot key is valid only in that command set. If no command set appears, the hot key works in both command sets. For example,

- **CUA**
  - this means `Alt+F4` is a hot key in the CUA command set,

- **Alternate**
  - this means `Alt+X` is a hot key in the Alternate command set,

- **Ctrl+Ins**
  - and this means `Ctrl+Ins` is a hot key in both command sets.

If you are also using Turbo C++ for Windows, you’ll find the IDE very similar to the Borland C++ IDE. Throughout this menu reference, we’ve noted the major differences between the two IDEs:

- **Borland C++ only**
  - This note indicates the feature occurs only in Borland C++.
  - The Windows icon indicates the discussion is relevant only to Turbo C++ for Windows.
If neither of these items appear next to the text, the text is relevant to both IDEs.

≡ (System) menu

Borland C++ only

The ≡ menu appears on the far left of the menu bar. Alt+Spacebar is the fastest way to get there. When you pull down this menu, you see the Repaint Desktop command and the names of programs you've installed with the Options | Transfer command.

Turbo C++ for Windows has a Control menu on the far left of the Title bar. Alt+Spacebar is the shortcut key. The Control menu primarily lets you manage windows through menu commands instead of using a mouse. It is the standard Windows Control menu.

Repaint Desktop

Borland C++ only

Choose ≡ | Repaint Desktop to have Borland C++ redraw the screen. You may need to do this, for example, if a memory-resident program has left stray characters on the screen, or possibly if you have screen-swapping turned off (Options | Debugger and you've selected None for the Display swapping option) and you're stepping through a program.

Transfer items

Borland C++ only

A program that appears here on the ≡ menu can be run directly from the IDE. You install programs here with the Options | Transfer command. To run one of these programs, choose its name from the ≡ menu.

If you have more than one program installed with the same shortcut letter on this menu, the first program listed with that shortcut will be selected. You can select the second item by clicking it or by using the arrow keys to move to it and then pressing Enter.
The File menu lets you open and create program files in edit windows. The menu also lets you save your changes, perform other file functions, and quit the IDE.

**New**

The File | New command lets you open a new edit window with the default name NONAME{xx}.CPP (the xx stands for a number from 00 to 31). These NONAME files are used as a temporary edit buffer; the IDE prompts you to name a NONAME file when you save it.

**Open**

The File | Open command displays a file-selection dialog box for you to select a program file to open in an edit window. Here is what the box looks like:

![Open a File dialog box](image)

The dialog box contains an input box, a file list, buttons labeled Open, Replace, Cancel, and Help, and an information panel that describes the selected file. Now you can do any of these actions:

- Type in a full file name and choose Replace or Open. Open loads the file into a new edit window. Replace saves the file in the active window and replaces it with the contents of the selected file. An edit window must be active if you choose Replace.
- Type in a file name with wildcards, which filters the file list to match your specifications.
Press ↓ to choose a file specification from a history list of file specifications you’ve entered earlier.

- View the contents of different directories by selecting a directory name in the file list.

The input box lets you enter a file name explicitly or lets you enter a file name with standard DOS wildcards ( * and ?) to filter the names appearing in the history list box. If you enter the entire name and press Enter, Borland C++ opens it. (If you enter a file name that Borland C++ can’t find, it automatically creates and opens a new file with that name.)

If you press ↓ when the cursor is blinking in the input box, a history list drops down below the box. This list displays the last 15 file names or file name masks you’ve entered. Choose a name from the list by double-clicking it or selecting it with the arrow keys and pressing Enter.

Once you’ve typed in or selected the file you want, choose the Open button (choose Cancel if you change your mind). You can also just press Enter once the file is selected, or you can double-click the file name in the file list.

The Turbo C++ File Open dialog box doesn’t have the Replace button; therefore, you can only open another edit window rather than replace the contents of the file in the window with the contents of another file.

Using the File list box

In Borland C++, you can also type a lowercase letter to search for a file name or an uppercase letter to search for a directory name.

If you choose Replace instead of Open, the selected file replaces the file in the active edit window instead of opening up a new window.

Borland C++ only

The File list box displays all file names in the current directory that match the specifications in the input box, displays the parent directory, and displays all subdirectories. Click the list box or press Tab until the list box name is highlighted. You can now press ↓ or ↑ to select a file name, and then press Enter to open it. You can also double-click any file name in the box to open it. You might have to scroll the box to see all the names. If you have more than one pane of names, you can also use → and ←.

The file information panel at the bottom of the Open a File dialog box displays path name, file name, date, time, and size of the file you’ve selected in the list box. As you scroll through the list box, the panel is updated for each file.
The File | Save command saves the file in the active edit window to disk. (This menu item is disabled if there’s no active edit window.) If the file has a default name (NONAME00.CPP, or the like), the IDE opens the Save File As dialog box to let you rename and save it in a different directory or on a different drive. This dialog box is identical to the one opened for the Save As command, described next.

The File | Save As command lets you save the file in the active edit window under a different name, in a different directory, or on a different drive. When you choose this command, you see the Save File As dialog box:

Enter the new name, optionally with drive and directory, and click or choose OK. All windows containing this file are updated with the new name.

The File | Save All command works just like the Save command except that it saves the contents of all modified files, not just the file in the active edit window. This command is disabled if no edit windows are open.
The File | Change Dir command lets you specify a drive and a directory to make current. The current directory is the one Borland C++ uses to save files and to look for files. (When using relative paths in Options | Directories, they are relative to this current directory only.)

Here is what the Change Directory dialog box looks like:

![Change Directory dialog box]

There are two ways to change directories:

- Type in the path of the new directory in the input box and press Enter, or
- Choose the directory you want in the Directory tree (if you’re using the keyboard, press Enter to make it the current directory), then choose OK or press Esc.

If you choose the OK button, your changes will be made and the dialog box put away. If you choose the Chdir button, the Directory Tree list box changes to the selected directory and displays the subdirectories of the currently highlighted directory (pressing Enter or double-clicking on that entry gives you the same result). If you change your mind about the directory you’ve picked and you want to go back to the previous one (and you’ve yet to exit the dialog box), choose the Revert button.

Opening a project in another directory automatically changes directories, so you don’t have to change directories before you open another project.
Print

In Borland C++, you can also print the contents of the Output window.

The File | Print command lets you print the contents of the active edit window or the Message window. This command is disabled if the active window can’t be printed.

Printer Setup

Use this option if you want to change your printer setup from its normal configuration.

The Printer Setup command displays a Windows dialog box you can use to set up your printer. When you installed Windows on your system, you probably also installed one or more printer drivers so you could print from Windows. The Printer Setup command lets you select which printer you want to use for printing from Turbo C++.

If you choose Setup in the Printer Setup dialog box, another dialog box appears allowing you to select a paper size, specify a particular font, and so forth. The options available to you will depend on the capabilities of your printer.

DOS Shell

Borland C++ only

The File | DOS Shell command lets you temporarily exit Borland C++ to enter a DOS command or program. To return to Borland C++, type EXIT and press Enter.

You may find that when you’re debugging there’s not enough memory to execute this command. If that’s the case, terminate the debug session by choosing Run | Program Reset.

Don’t install any TSR programs (like SideKick) or print a file with the DOS print command while you’ve shelled to DOS, because memory may get misallocated.

Note: In dual monitor mode, the DOS command line appears on the Borland C++ screen rather than the User Screen. This allows you to switch to DOS without disturbing the output of your program.

You can also use the transfer items on the ⏯ (System) menu to quickly switch to another program without leaving Borland C++.
Exit

The File | Exit command exits the IDE and removes it from memory. If you have made any changes that you haven’t saved, the IDE asks you if you want to save them before exiting.

Closed File Listing

If you have opened files and then closed them, you’ll see the last five files listed at the bottom of the File menu. If you select the file name on the menu, the file will open. When you work with many open files, you can close some, yet open them again quickly using the list and reduce the clutter on your desktop.

Edit menu

The Edit menu lets you cut, copy, and paste text in edit windows. If you make mistakes, you can undo changes and even reverse the changes you’ve just undone. You can also open a Clipboard window to view or edit its contents, and copy text from the Message and Output windows.

Before you can use most of the commands on this menu, you need to know about selecting text (because most editor actions apply to selected text). Selecting text means highlighting it. You can select text either with keyboard commands or with a mouse; the principle is the same even though the actions are different.

From the keyboard:

- Press Shift while pressing any key that moves the cursor.

See page 187 in Appendix B for additional text selection commands.

With a mouse:

- To select text with a mouse, drag the mouse pointer over the desired text. If you need to continue the selection past a window’s edge, just drag off the side and the window will automatically scroll.
To select a single word, double-click it.
To extend or reduce the selection, Shift-click anywhere in the document (that is, hold Shift and click).

Once you have selected text, the Cut and Copy commands in the Edit menu become available.

The Clipboard is the magic behind cutting and pasting. It's a special window that holds text that you have cut or copied, so you can paste it elsewhere. The Clipboard works in close concert with the commands in the Edit menu.

Here's an explanation of each command in the Edit menu.

**Undo**

The Edit | Undo command restores the file in the current window to the way it was before the most-recent edit or cursor movement. If you continue to choose Undo, the editor continues to reverse actions until your file returns to the state it was in when you began your current editing session.

Undo inserts any characters you deleted, deletes any characters you inserted, replaces any characters you overwrote, and moves your cursor back to a prior position. If you undo a block operation, your file appears as it did before you executed the block operation.

Undo doesn't change an option setting that affects more than one window. For example, if you use the Ins key to change from Insert to Overwrite mode, then choose Undo, the editor won't change back to Insert mode.

The Group Undo option in the Editor Options dialog box (Options | Environment | Editor) affects Undo and Redo. See page 113 for information on Group Undo.

**Redo**

The Edit | Redo command reverses the effect of the most recent Undo command. The Redo command only has an effect immediately after an Undo command or after another Redo command. A series of Redo commands reverses the effects of a series of Undo commands.
The Edit | Cut command removes the selected text from your document and places the text in the Clipboard. You can then paste that text into any other document (or somewhere else in the same document) by choosing Paste. The text remains selected in the Clipboard so that you can paste the same text many times.

The Edit | Copy command leaves the selected text intact but places an exact copy of it in the Clipboard. You can then paste that text into any other document by choosing Paste.

If the Output or Message window is the active window when you select Edit | Copy, the entire contents of the window buffer (including any nonvisible portion) is copied to the Clipboard.

You can also copy text from a Help window: With the keyboard, use Shift and the arrow keys; with the mouse, click and drag the text you want to copy.

To copy text from a Help window in Turbo C++, display the text you want to copy, then select Edit | Copy. The entire contents of the window is copied to the Clipboard.

The Edit | Paste command inserts text from the Clipboard into the current edit window at the cursor position. The text that is actually pasted is the currently marked block in the Clipboard window.

The Edit | Clear command removes the selected text but does not put it into the Clipboard. This means you cannot paste the text as you could if you had chosen Cut or Copy. The cleared text is not retrievable unless you use the Edit | Undo command. Clear is useful if you want to delete text, but you don’t want to overwrite text being held in the Clipboard. You can clear the Clipboard itself by selecting all the text in the Clipboard, then choosing Edit | Clear.
Copy Example

Borland C++ only

The Edit | Copy Example command copies the preselected example text in the current Help window to the Clipboard. The examples are already predefined as blocks you can paste, so you don’t need to bother marking the example.

To copy a Help example in Turbo C++ for Windows, follow these steps:

1. Display the example you want to copy in the Help window.
2. Choose Edit | Copy and all the text in the Help window is copied to the Clipboard.
3. Make the window you want the example copied to the active window.

Show Clipboard

Borland C++ only

The Edit | Show Clipboard command opens the Clipboard window, which stores the text you cut and copy from other windows. The text that’s currently selected (highlighted) is the text Borland C++ uses when you choose Paste.

You can think of the Clipboard window as a history list of your cuts and copies. And you can edit the Clipboard so that the text you paste is precisely the text you want. Borland C++ uses whatever text is selected in the Clipboard when you choose Paste.

The Clipboard window is just like other edit windows; you can move it, resize it, and scroll and edit its contents. The only difference you’ll find in the Clipboard window is when you choose to cut or copy text. When you select text in the Clipboard window and choose Cut or Copy, the selected text immediately appears at the bottom of the window. (Remember, any text that you cut or copy is appended to the end of the Clipboard and highlighted—so you can paste it later.)

You can save the Clipboard contents across sessions in Borland C++. Choose Options | Environment | Desktop command and select the Clipboard option.

The Edit | Show Clipboard option doesn’t appear in the Turbo C++ IDE. Of course, you can display the Clipboard at any time using the Program Manager.
Search menu

The Search menu lets you search for text, function declarations, and error locations in your files.

Find

The Search ! Find command displays the Find Text dialog box, which lets you type in the text you want to search for and set options that affect the search.

Figure 3.4
The Find Text dialog box

You can set up your right mouse button to Find Text. Choose Options I Environment I Mouse and select the Search option.

The Find Text dialog box contains several buttons and check boxes:

- **Case Sensitive**
- **Whole words only**
- **Regular expression**

Check the Case Sensitive box if you do want the IDE to differentiate uppercase from lowercase.

Check the Whole Words Only box if you want the IDE to search for words only (that is, the string must have punctuation or space characters on both sides).

Check the Regular Expression box if you want the IDE to recognize GREP-like wildcards in the search string. The wildcards are ^, $, ., *, +, [ ], and \. Here's what they mean:

- `^` A circumflex at the start of the string matches the start of a line.
- `$` A dollar sign at the end of the expression matches the end of a line.
- `.` A period matches any character.
- `*` A character followed by an asterisk matches any number of occurrences (including zero) of that character. For example, `bo*` matches `bot`, `b`, `boo`, and also `be`.
A character followed by a plus sign matches any number of occurrences (but not zero) of that character. For example, \textit{bo}+ matches \textit{bot} and \textit{boo}, but not \textit{be} or \textit{b}.

Characters in brackets match any one character that appears in the brackets but no others. For example \textit{[bot]} matches \textit{b}, \textit{o}, or \textit{t}.

A circumflex at the start of the string in brackets means \textit{not}. Hence, \textit{[^bot]} matches any characters except \textit{b}, \textit{o}, or \textit{t}.

A hyphen within the brackets signifies a range of characters. For example, \textit{[b-o]} matches any character from \textit{b} through \textit{o}.

A backslash before a wildcard character tells Borland C++ to treat that character literally, not as a wildcard. For example, \textit{\^} matches \textit{^} and does not look for the start of a line.

Enter the string in the input box and choose OK to begin the search, or choose Cancel to forget it. If you want to enter a string that you searched for previously, press ↓ (or \textit{Alt+↓} in Turbo C++) to show a history list to choose from.

You can also pick up the word that your cursor is currently on in the edit window and use it in the Find Text box by simply invoking Find from the Search menu. In Borland C++, you can take additional characters from the text by pressing \textit{→}.

Choose from the Direction radio buttons to decide which direction you want the IDE to search—starting from the origin (which you can set with the Origin radio buttons).

Choose from the Scope buttons to determine how much of the file to search in. You can search the entire file (Global) or only the text you’ve selected.

Choose from the Origin buttons to determine where the search begins. When Entire Scope is chosen, the Direction radio buttons determine whether the search starts at the beginning or the end of the scope. You choose the range of scope you want with the Scope radio buttons.
The Search | Replace command displays a dialog box that lets you type in text you want to search for and text you want to replace it with.

The Replace Text dialog box contains several radio buttons and check boxes—many of which are identical to the Find Text dialog box, discussed previously. An additional checkbox, Prompt on Replace, controls whether you’re prompted for each change.

Enter the search string and the replacement string in the input boxes and choose OK or Change All to begin the search, or choose Cancel to forget it. If you want to enter a string you used previously, press ↓ (or Alt+↓ in Turbo C++) to show a history list to choose from.

If the IDE finds the specified text and Prompt on Replace is on, it asks you if you want to make the replacement. If you choose OK, it will find and replace only the first instance of the search item. If you choose Change All, it replaces all occurrences found, as defined by Direction, Scope, and Origin.

The Search | Search Again command repeats the last Find or Replace command. All settings you made in the last dialog box used (Find or Replace) remain in effect when you choose Search Again.
Go to Line Number

The Search | Go to Line Number command prompts you for the line number you want to find.

The IDE displays the current line number and column number in the lower left corner of every edit window.

Previous Error

The Search | Previous Error command 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.

Next Error

The Search | Next Error command 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.

Locate Function

Borland C++ only

The Search | Locate Function command displays a dialog box for you to enter the name of a function to search for. This command is available only during a debugging session.

Enter the name of a function or press ↓ to choose a name from the history list. As opposed to the Find command, this command finds the declaration of the function, not instances of its use.

Run menu

Alt R

The Run menu's commands run your program, start and end debugging sessions in Borland C++ and start Turbo Debugger for Windows in the Turbo C++ IDE.

Run

Ctrl F9

The Run | Run command runs your program, using any arguments you pass to it with the Run | Arguments command. If the source code has been modified since the last compilation, it
Run | Run will also invoke the Project Manager to recompile and link your program. (The Project Manager is a program building tool incorporated into the IDE; see Chapter 3, “The Project menu,” for more on this feature.)

If you’re using Turbo C++ and aren’t planning to debug your program with Turbo Debugger for Windows, you can compile and link it with the Source Debugging unchecked in the Options | Linker dialog box. Your program will link faster.

The rest of this discussion about Run | Run applies only to Borland C++.

If you don’t want to debug your program in Borland C++, you can compile and link it with the Source Debugging radio button set to None (which makes your program link faster) in the Options | Debugger dialog box. If you compile your program with Source Debugging set to On, the resulting executable code will contain debugging information that will affect the behavior of the Run | Run command in the following ways:

Source code the same If you have not modified your source code since the last compilation,
- the Run | Run command causes your program to run to the next breakpoint, or to the end if no breakpoints have been set.

Source code modified If you have modified your source code since the last compilation,
- and if you’re already stepping through your program using the Run | Step Over or Run | Trace Into commands, Run | Run prompts you whether you want to rebuild your program:
  - If you answer yes, the Project Manager recompiles and links your program, and sets it to run from the beginning.
  - If you answer no, your program runs to the next breakpoint or to the end if no breakpoints are set.
- and if you are not in an active debugging session, the Project Manager recompiles your program and sets it to run from the beginning.

Pressing Ctrl+Break causes Borland C++ to stop execution on the next source line in your program. If Borland C++ is unable to find a source line, a second Ctrl+Break will terminate the program and return you to the IDE.
You can't run or debug Windows applications within the IDE. If you try to do so, you'll get an error dialog box to that effect.

**Program Reset**  
* Borland C++ only  

The Run | Program Reset command stops the current debugging session, releases memory your program has allocated, and closes any open files that your program was using. Use this command when you want to cancel a debugging session or if there's not enough memory to run transfer programs or invoke a DOS shell.

**Go to Cursor**  
* Borland C++ only  

The Run | Go to Cursor command runs your program from the beginning of the program (or the last executed statement if you're in the middle of a debugging session) to the line the cursor is on in the current edit window. If the cursor is at a line that does not contain an executable statement, the command displays a warning.

Go to Cursor does not set a permanent breakpoint, but it does allow the program to stop at a permanent breakpoint if it encounters one before the line the cursor is on. If this occurs, you must move the cursor back and choose the Go to Cursor command again.

Use Go to Cursor to advance the run bar (the highlighted line of code that represents the next statement to be executed) to the part of your program you want to debug. If you want your program to stop at a certain statement every time it reaches that point, set a breakpoint on that line.

Note that if you position the cursor on a line of code that is not executed, your program will run to the next breakpoint or the end if no breakpoints are encountered. You can always use Ctrl+Break to stop a running program.

**Trace Into**  
* Borland C++ only  

The Run | Trace Into command runs your program statement-by-statement. If you Trace Into a function call, the run bar stops on the first line of the function instead of executing the function as a single step (see Run | Step Over). If a statement contains no calls to functions accessible to the debugger, Trace Into stops at the next executable statement.
Use the Trace Into command to enter a function called by the function you are now debugging. The next section illustrates the differences between the Trace Into and Step Over commands.

If the statement contains a call to a function accessible to the debugger, Trace Into halts at the beginning of the function's definition. Subsequent Trace Into or Step Over commands run the statements in the function's definition. When the debugger leaves the function, it resumes evaluating the statement that contains the call; for example,

```c
if (func1() && func2())
    do_something();
```

With the run bar on the `if` statement, F7 will trace into `func1`; when the run bar is on the return in `func1`, F7 will trace into `func2`. F8 will step over `func2` and stop on `do_something`.

**Note:** The Trace Into command recognizes only functions defined in a source file compiled with these two options on:

- In the Advanced Code Generation dialog box (Options | Compiler), the Debug Info in OBJs check box must be checked.
- The Source Debugging radio buttons must be set to On (in the Options | Debugger dialog box).

---

**Step Over**

*Borland C++ only*  

The Run | Step Over command executes the next statement in the current function. It does not trace into calls to lower-level functions, even if they are accessible to the debugger.

Use Step Over to run the function you are now debugging, one statement at a time without branching off into other functions.

Here is an example of the difference between Run | Trace Into and Run | Step Over. These are the first 12 lines of a program loaded into an edit window:

```c
int findit(void)  /* Line 1 */
{
    return(2);
}
```
void main(void) { /* Line 6 */
    int i, j;
    i = findit(); /* Line 10 */
    printf("%d\n", i); /* Line 11 */
    j = 0; . . . /* Line 12 */
}

`findit` is a user-defined function in a module that has been compiled with debugging information. Suppose the run bar is on line 10 of your program. To position the run bar on line 10, place the cursor on line 10 and either press F4 or select Run | Go to Cursor.

- If you now choose Run | Trace Into, the run bar will move to the first line of the `findit` function (line 1 of your program), allowing you to step through the function.
- If you choose Run | Step Over, the `findit` function will execute and the run bar will move to line 11.

If the run bar had been on line 11 of your program, it would have made no difference which command you chose; Run | Trace Into and Run | Step Over both would have executed the `printf` function and moved the run bar to line 12. This is because the `printf` function does not contain debug information.

**Arguments**

The Run | Arguments command allows you to give your running programs command-line arguments exactly as if you had typed them on the DOS command line. DOS redirection commands will be ignored.

When you choose this command, a dialog box appears with a single input box. You only need to enter the arguments here, not the program name. Arguments take effect when your program starts.

*Borland C++ only*

If you are already debugging and want to change the arguments, select Program Reset and Run | Run to start the program with the new arguments.

**Debugger**

The Run | Debugger command starts Turbo Debugger for Windows so you can debug your program. Turbo C++ tells Turbo
Debugger which program to debug. Before you can use Turbo Debugger for Windows to debug your program you must:

1. Choose Options | Compiler and in the Advanced Code Generation dialog box check the Debug Info in OBJs option.
2. Choose Options | Linker and set Source Debugging to on.

The Run | Debugger Options command lets you pass arguments to Turbo Debugger for Windows when you choose the Run | Debugger command. See the Turbo Debugger for Windows manual for a description of all options.

Compile menu

Use the commands on the Compile menu to compile the program in the active window or to make or build your project. To use the Compile, Make, Build, and Link commands, you must have a file open in an active edit window or a project defined.

The Compile | Compile command compiles the file in the active edit window. If the Project or Message Window is active, Compile | Compile compiles the highlighted file.

When the compiler is compiling, a status box pops up to display the compilation progress and results. When compiling is complete, press any key to remove this box. In Turbo C++, press Enter or choose OK. If any errors or warnings occurred, the Message window becomes active and displays and highlights the first error.

The Compile | Make command invokes the Project Manager to compile and link your source code to the target executable or library.

Compile | Make rebuilds only the files that aren't current.

The .EXE file name listed is derived from one of two names in the following order:
Compile | Make

- the project file (.PRJ) specified with the Project | Open Project command
- the name of the file in the active edit window. If no project is defined, you'll get the default project defined by the file TCDEF.DPR, or, if you're using Turbo C++, the default project defined by the file TCDEFW.DPR.

Link

The Compile | Link command takes the files defined in the current project file or the defaults and links them.

Build

This command is similar to Compile | Make except that it rebuilds all the files in the project whether or not they are current. It performs the following steps:

1. It deletes the appropriate precompiled header (.SYM) file, if it exists.
2. It deletes any cached autodependency information in the project.
3. It sets the date and time of all the project's .OBJ files to zero.
4. Finally, it does a make.

If you abort a Build command by pressing Ctrl+Break in Borland C++, pressing Esc or choosing Cancel in Turbo C++, or get errors that stop the build, you can pick up where it left off simply by choosing Compile | Make.

Information

The Compile | Information command displays a dialog box with information on the current file or project. The information is for display only; you can't change it in the dialog box. The following table tells you what each line in the File Information dialog box means and where you can go to change the settings if you want to.
Compile | Information

<table>
<thead>
<tr>
<th>Setting</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current directory</td>
<td>The default directory.</td>
</tr>
<tr>
<td>Current file</td>
<td>File in the active window.</td>
</tr>
<tr>
<td>Expanded memory in use</td>
<td>Amount of expanded memory reserved by Borland C++.</td>
</tr>
<tr>
<td>Lines compiled</td>
<td>Number of lines compiled.</td>
</tr>
<tr>
<td>Total warnings</td>
<td>Number of warnings issued.</td>
</tr>
<tr>
<td>Total errors</td>
<td>Number of errors generated.</td>
</tr>
<tr>
<td>Total time</td>
<td>Amount of time your program has run (debugger only).</td>
</tr>
<tr>
<td>Program loaded</td>
<td>Debugging status.</td>
</tr>
<tr>
<td>Program exit code</td>
<td>DOS termination code of last terminated program.</td>
</tr>
<tr>
<td>Available memory</td>
<td>Amount of memory available to Borland C++ in bytes.</td>
</tr>
</tbody>
</table>

You'll see only some of these settings in Turbo C++. The Compile | Remove Messages command removes all messages from the Message window.

Debug menu

The Debug menu appears in Borland C++ only. The commands on the Debug menu control all the features of the integrated debugger. You specify whether or not debugging information is generated in the Options | Debugger dialog box.

You can't run or debug Windows applications within the Borland C++ IDE. If you try to do so, you'll get an error dialog box to that effect. You must run them under Microsoft Windows and use Turbo Debugger for Windows.

To debug applications in the Turbo C++ IDE, use Turbo Debugger for Windows. Start Turbo Debugger with the Run | Debugger command.

Inspect

The Debug | Inspect command opens an Inspector window that lets you examine and modify values in a data element. The type of element you’re inspecting determines the type of information
You can set up your right mouse button to inspect. Choose Options | Environment | Mouse and select the Inspect option.

You can position the cursor on the data element you want to inspect, then choose Alt+F4.

You can also choose Debug | Inspect to bring up the Inspector dialog box, and then type in the variable or expression you want to inspect. Alternatively, you can position the cursor on an expression, select Debug | Inspect, and, while in this dialog box, press → to bring in more of the expression. Press Enter to inspect it.

To close an Inspector window, make sure the window is active (topmost) and press Esc or choose Window | Close.

Here are some additional inspection operations you can perform:

- **Sub-inspecting:** Once you’re in an Inspector window, you can inspect certain elements to isolate the view. When an inspector item is inspectable, the status line displays the message “J Inspect.” To sub-inspect an item, you move the inspect bar to the desired item and press Enter.

- **Modifying inspector items:** When an inspector item can be modified, the status line displays “Alt+M Modify Field.” Move the cursor to the desired item and press Alt+M; a dialog box will prompt you for the new value.

- **Inspect range:** When you are inspecting certain elements, you can change the range of values that is displayed. For example, you can range-inspect pointer variables to tell Borland C++ how many elements the pointer points to. You can range-inspect an inspector when the status line displays the message “Set index range” and with the command Alt+I.

The following sections briefly describe the eight types of Inspector windows possible.

**Ordinal Inspector windows**

Ordinal Inspector windows show you the value of simple data items, such as

```c
char x = 4;
unsigned long y = 123456L;
```

These Inspector windows only have a single line of information following the top line (which usually displays the address of the variable, though it may display the word “constant” or have other information in it, depending on what you’re inspecting). To the

Chapter 3, Menus and options reference

67
left appears the type of the scalar variable (char, unsigned long, and so forth), and to the right appears its present value. The value can be displayed as decimal, hex, or both. It's usually displayed first in decimal, with the hex values in parentheses (using the standard C hex prefix of 0x).

If the variable being displayed is of type char, the character equivalent is also displayed. If the present value does not have a printing character equivalent, the backslash (\) followed by a hex value displays the character value. This character value appears before the decimal or hex values.

**Pointer Inspector windows**

Pointer Inspector windows show you the value of data items that point to other data items, such as

```c
char *p = "abc";
int *ip = 0;
int **ipp = &ip;
```

Pointer Inspector windows usually have a top line that contains the address of the pointer variable and the address being pointed to, followed by a single line of information.

To the left appears [0], indicating the first member of an array. To the right appears the value of the item being pointed to. If the value is a complex data item such as a structure or an array, as much of it as possible is displayed, with the values enclosed in braces ({}).

If the pointer is of type char and appears to be pointing to a null-terminated character string, more information appears, showing the value of each item in the character array. To the left in each line appears the array index ([1], [2], and so on), and the value appears to the right as it would in a scalar Inspector window. In this case, the entire string is also displayed on the top line, along with the address of the pointer variable and the address of the string that it points to.

**Array Inspector windows**

Array Inspector windows show you the value of arrays of data items, such as

```c
long thread[3][4][5];
char message[] = "eat these words";
```

There is a line for each member of the array. To the left on each line appears the array index of the item. To the right appears the value of the item being pointed to. If the value is a complex data
item such as a structure or array, as much of it as possible is displayed, with the values enclosed in braces ({}).

Structure and union Inspector windows
Structure and union Inspector windows show you the value of the members in your structure, class, and union data items. For example,

```c
struct date {
    int year;
    char month;
    char day;
} today;

union {
    int small;
    long large;
} holder;
```

Structures and unions appear the same in Inspector windows. These Inspector windows have as many items after the address as there are members in the structure or union. Each item shows the name of the member on the left and its value on the right, displayed in a format appropriate to its C data type.

Function Inspector windows
Function Inspector windows show the return type of the function at the bottom of the inspector. Each parameter that a function is called with appears after the memory address at the top of the list.

Function Inspector windows give you information about the calling parameters, return data type, and calling conventions for a function.

Class Inspector windows
The Class (or object) Inspector window lets you inspect the details of a class variable. The window displays names and values for members and methods defined by the class.

The window can be divided into two panes horizontally, with the top pane listing the data fields or members of the class, and the bottom pane listing the member function names and the function addresses. Press Tab to move between the two panes of the Class Inspector window.

If the highlighted data field is a class or a pointer to a class, pressing Enter opens another Class Inspector window for the highlighted type. In this way, you can quickly inspect complex nested structures of classes with a minimum of keystrokes.
Debug | Inspect

Constant Inspector window

Constant Inspector windows are much like Ordinal Inspector windows, but they have no address and can never be modified.

Type Inspector window

The Type Inspector window lets you examine a type. There is a Type Inspector window for each kind of instance inspector described here. The difference between them is that instance inspectors display the value of a field and type inspectors display the type of a field.

Evaluate/Modify

The Evaluate button is the default button; when you tab to the New Value field, the Modify button becomes the default.

The Debug | Evaluate/Modify command evaluates a variable or expression, displays its value, and, if appropriate, lets you modify the value. The command opens a dialog box containing three fields: the Expression field, the Result field, and the New Value field.

The Expression field shows a default expression consisting of the word at the cursor in the Edit window. You can evaluate the default expression by pressing Enter, or you can edit or replace it first. You can also press → to extend the default expression by copying additional characters from the Edit window.

You can evaluate any valid C expression that doesn’t contain

- function calls
- symbols or macros defined with \#define
- local or static variables not in the scope of the function being executed

If the debugger can evaluate the expression, it displays the value in the Result field. If the expression refers to a variable or simple data element, you can move the cursor to the New Value field and enter an expression as the new value.

Press Esc to close the dialog box. If you’ve changed the contents of the New Value field but do not select Modify, the debugger will ignore the New Value field when you close the dialog box.

Use a repeat expression to display the values of consecutive data elements. For example, for an array of integers named xarray,

- xarray[0], 5 displays five consecutive integers in decimal.
- xarray[0], 5x displays five consecutive integers in hexadecimal.
An expression used with a repeat count must represent a single data element. The debugger views the data element as the first element of an array if it isn’t a pointer, or as a pointer to an array if it is.

The Debug | Evaluate/Modify command displays each type of value in an appropriate format. For example, it displays an int as an integer in base 10 (decimal), and an array as a pointer in base 16 (hexadecimal). To get a different display format, precede the expression with a comma followed by one of the format specifiers shown in Table 3.2 on page 72.

Call Stack

The Debug | Call Stack command opens a dialog box containing the call stack. The Call Stack window shows the sequence of functions your program called to reach the function now running. At the bottom of the stack is main; at the top is the function that’s now running.

Each entry on the stack displays the name of the function called and the values of the parameters passed to it.

Initially the entry at the top of the stack is highlighted. To display the current line of any other function on the call stack, select that function’s name and press Enter. The cursor moves to the line containing the call to the function next above it on the stack.

For example, suppose the call stack looked like this:

```
func2()
func1()
main()
```

This tells you that main called func1, and func1 called func2. If you wanted to see the line of func1 that called func2, you could select func1 in the call stack and press Enter. The code for func1 would appear in the Edit window, with the cursor positioned on the call to func2.

To return to the current line of the function now being run (that is, to the run position), select the topmost function in the call stack and press Enter.
### Table 3.2: Format specifiers recognized in debugger expressions

<table>
<thead>
<tr>
<th>Character</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td><strong>Character.</strong> Shows special display characters for control characters (ASCII 0 through 31); by default, such characters are shown using the appropriate C escape sequences (\n, \t, and so on). Affects characters and strings.</td>
</tr>
<tr>
<td>S</td>
<td><strong>String.</strong> Shows control characters (ASCII 0 through 31) as ASCII values using the appropriate C escape sequences. Since this is the default character and string display format, the S specifier is only useful in conjunction with the M specifier.</td>
</tr>
<tr>
<td>D</td>
<td><strong>Decimal.</strong> Shows all integer values in decimal. Affects simple integer expressions as well as arrays and structures containing integers.</td>
</tr>
<tr>
<td>H or X</td>
<td><strong>Hexadecimal.</strong> Shows all integer values in hexadecimal with the 0x prefix. Affects simple integer expressions as well as arrays and structures containing integers.</td>
</tr>
<tr>
<td>F</td>
<td><strong>Floating point.</strong> Shows ( n ) significant digits (( n ) is an integer between 2 and 18). The default value is 7. Affects only floating-point values.</td>
</tr>
<tr>
<td>M</td>
<td><strong>Memory dump.</strong> Displays a memory dump, starting with the address of the indicated expression. The expression must be a construct that would be valid on the left side of an assignment statement, that is, a construct that denotes a memory address; otherwise, the M specifier is ignored. By default, each byte of the variable is shown as two hex digits. Adding a D specifier with the M causes the bytes to be displayed in decimal. Adding an H or X specifier causes the bytes to be displayed in hex. An S or a C specifier causes the variable to be displayed as a string (with or without special characters). The default number of bytes displayed corresponds to the size of the variable, but a repeat count can be used to specify an exact number of bytes.</td>
</tr>
<tr>
<td>P</td>
<td><strong>Pointer.</strong> Displays pointers in seg:ofs format with additional information about the address pointed to, rather than the default hardware-oriented seg:ofs format. Specifically, it tells you the region of memory in which the segment is located, and the name of the variable at the offset address, if appropriate. The memory regions are as follows:</td>
</tr>
<tr>
<td>R</td>
<td><strong>Structure/Union.</strong> Displays field names as well as values, such as { X:1, Y:10, Z:5 }. Affects only structures and unions.</td>
</tr>
</tbody>
</table>

**Memory region**

- 0000:0000-0000:03FF: Interrupt vector table
- 0000:0400-0000:04FF: BIOS data area
- 0000:0500-Borland C++: MS-DOS/TSRs
- Borland C++—User Program PSP: Borland C++
- User Program PSP: User Process PSP
- User Program—top of RAM: Name of a static user variable if its address falls inside the variable's allocated memory; otherwise nothing
- A000:0000-AFFF:FFFF: EGA/VGA Video RAM
- B000:0000-B7FF:FFFF: Monochrome Display RAM
- B800:0000-BFFF:FFFF: Color Display RAM
- C000:0000-EFFF:FFFF: EMS Pages/Adaptor BIOS ROMs
- F000:0000-FFFF:FFFF: BIOS ROMs

**Evaluate message**

- BIOS ROMs
The Debug | Watches command opens a pop-up menu of commands that control the use of watch expressions. Watch expressions can be saved across sessions; see Options | Environment | Desktop. The following sections describe the commands in this pop-up menu.

Add Watch

The Add Watch command inserts a watch expression into the Watch window.

When you choose this command, the debugger opens a dialog box and prompts you to enter a watch expression. The default expression is the word at the cursor in the current Edit window. There's also a history list available if you want to quickly enter an expression you've used before.

When you type a valid expression and press Enter or click OK, the debugger adds the expression and its current value to the Watch window. If the Watch window is the active window, you can insert a new watch expression by pressing Ins.

Delete Watch

The Delete Watch command deletes the current watch expression from the Watch window. To delete a watch expression other than the current one, select the desired watch expression by highlighting it. Then choose Delete Watch. When the Watch Window is active, you can press Del or Ctrl+Y to delete a watch.

Edit Watch

The Edit Watch command allows you to edit the current watch expression in the Watch window. A history list is available to save you time retyping.

When you choose this command, the debugger opens a dialog box containing a copy of the current watch expression. Edit the expression and press Enter. The debugger replaces the original version of the expression with the edited one.

You can also edit a watch expression from inside the Watch window by selecting the expression and pressing Enter.
Remove All Watches

The Remove All Watches command deletes all watch expressions from the Watch window.

Toggle Breakpoint

The Debug | Toggle Breakpoint command lets you set or clear an unconditional breakpoint on the line where the cursor is positioned. When a breakpoint is set, it is marked by a breakpoint highlight. Breakpoints can be saved across sessions using Options | Environment | Desktop.

Figure 3.6

The Breakpoints dialog box

The Debug | Breakpoints command opens a dialog box that lets you control the use of breakpoints—both conditional and unconditional ones. Here is what the dialog box looks like:

The dialog box shows you all set breakpoints, their line numbers, and the conditions. The condition has a history list so you can select a breakpoint condition that you've used before.

The row of buttons at the bottom of the dialog box give you several options:

- Choose Delete to remove a highlighted breakpoint from your program.
- Choose View to display the source code where the selected breakpoint is set.
- Choose At to set a breakpoint at a particular function. You must be debugging to choose At.
- Choose Edit to add a new breakpoint or modify an existing one and the Breakpoint Modify/New dialog box appears:

Borland C++ User’s Guide
If you choose New, a breakpoint is set at the location of your cursor in the active edit window. You can modify a breakpoint by making changes in this dialog box.

The Condition text box accepts any expression that evaluates to either true or false. When your program reaches that condition while you’re debugging, it stops executing.

You can specify when the debugger should stop on the breakpoint. In the Pass Count text box, type in a number. If you enter a 1, the debugger stops the first time the breakpoint is reached. If you enter a 2, the debugger stops the second time the breakpoint is reached, and so on.

Generally you will not change the file name, but you can if you want. You can also specify a new line number. The primary purpose of these two options is to identify a breakpoint you have already set.

When you are done modifying your breakpoint, choose Modify and the IDE accepts the new settings.

When a source file is edited, each breakpoint “sticks” to the line where it is set. Breakpoints stay set until you

- delete the source line a breakpoint is set on
- clear a breakpoint with Toggle Breakpoint

Borland C++ will continue to track breakpoints until

- you edit a file containing breakpoints and then don’t save the edited version of the file.
- you edit a file containing breakpoints and then continue the current debugging session without remaking the program. (Borland C++ displays the warning prompt “Source modified, rebuild?”)
Before you compile a source file, you can set a breakpoint on any line, even a blank line or a comment. When you compile and run the file, Borland C++ validates any breakpoints that are set and gives you a chance to remove, ignore, or change invalid breakpoints. When you are debugging the file, Borland C++ knows which lines contain executable statements, and will warn you if you try to set invalid breakpoints.

You can set an unconditional breakpoint without going through the dialog box by choosing the Debug | Toggle Breakpoint command.

Project menu

The Project menu contains all the project management commands to

- create a project
- add or delete files from your project
- specify which program your source file should be translated with
- set options for a file in the project
- specify which command-line override options to use for the translator program
- specify what the resulting object module is to be called, where it should be placed, whether the module is an overlay, and whether the module should contain debug information
- view included files for a specific file in the project

Open Project

The Open Project command displays the Open Project File dialog box, which allows you to select and load a project or create a new project by typing in a name.

This dialog box lets you select a file name similar to the File | Open dialog box, discussed on page 47. The file you select will be used as a project file, which is a file that contains all the information needed to build your project's executable. Borland C++ uses the project name when it creates the .EXE, .DLL, or .LIB file and .MAP file. A typical project file has the extension .PRJ.
Choose Project I Close Project when you want to remove your project and return to the default project.

Choose Project I Add Item when you want to add a file to the project's file list. This brings up the Add to Project List dialog box.

This dialog box is set up much like the Open a File dialog box (File I Open). Choosing the Add button puts the currently highlighted file in the Files list into the Project window. The chosen file is added to the Project window File list immediately after the highlight bar in the Project window. The highlight bar is advanced each time a file is added. (When the Project Window is active, you can press Ins to add a file.)

Choose Project I Delete Item when you want to delete the highlighted file in the Project window. When the Project window is active, you can press Del to delete a file.

The Local Options command opens the following dialog box:
The Override Options dialog box lets you include command-line override options for a particular project-file module. It also lets you give a specific path and name for the object file and lets you choose a translator for the module.

Any program you installed in the Transfer dialog box with the Translator option checked appears in the list of Project File Translators (see page 100 for information on the Transfer dialog box).

Check the Overlay this Module option if you want the selected project item to be overlaid. This item is local to one file. It is ignored if the Overlaid DOS EXE option is not selected in the Output radio button in Options | Linker | Settings.

Check the Exclude Debug Information option to prevent debug information included in the module you've selected from going into the .EXE.

Use this switch on already debugged modules of large programs. You can change which modules have debug information simply by checking this box and then re-linking (no compiling is required).

Check the Exclude from Link option if you don’t want this module linked in.

Include Files

Choose Project | Include Files to display the Include Files dialog box or, if you’re in the Project window, press the Spacebar. If you haven’t built your project yet, the Project | Include Files command will be disabled.

The Include Files dialog box looks like this:
You can scroll through the list of files displayed. Select the file you want to view and press Enter.

**Browse menu**

To browse with your mouse, choose Options > Environment > Mouse and select the Browse Right Mouse Button option.

**Turbo C++ only**

The Browse menu in the Turbo C++ for Windows IDE gives you access to the ObjectBrowser so you can visually browse through your class hierarchies, functions, and variables.

Before you can use the ObjectBrowser, you must compile your program so that debugging information is included in your executable file. 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 access the ObjectBrowser either through the Browse menu or directly from your source code by clicking the right mouse button on the class, function or variable you wish to inspect.

The ObjectBrowser has buttons on the title bar of the ObjectBrowser window. Choose them by clicking them with your mouse or using specific key combinations. By choosing one of these buttons, you tell the ObjectBrowser to perform some action. Not all of the buttons are available at all times. 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.
The Browse | Classes command opens an ObjectBrowser window that displays all of the classes in your application, arranged as a horizontal “tree” to show parent-child relationships. The window is automatically sized to display as much of your class hierarchy as possible. If the entire image does not fit within the window, use the scroll bars to move the image to view hidden sections. You can highlight any class in the display by using the arrow cursor keys, or by clicking directly on the class name. Using the buttons at the top of the ObjectBrowser window, you can

- exit the ObjectBrowser.
- go to the source code that defines the highlighted class.
- inspect the functions and data elements of the highlighted class.

The Functions command opens a window that lists every function in your program, in alphabetical order. Class member functions are listed together by class (for example, MyClass::MyFunc). In addition, an incremental search field is provided at the bottom of the dialog that allows you to 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 have typed in. Using the buttons at the top of the ObjectBrowser window, you can

- exit the ObjectBrowser.
- go to the source code that defines the highlighted function.
- inspect the declaration of the highlighted function.

The Variables command opens a window that lists every global variable in your program, in alphabetical order. This dialog box also contains an incremental search field. Using the buttons at the top of the ObjectBrowser window, you can

- exit the ObjectBrowser.
- open an edit window on the source code that defines the highlighted variable.
- inspect the declaration of the highlighted variable.
Symbols

You can also inspect a symbol by clicking it in your source code with your right mouse button. Set up your mouse this way with Options I Environment I Mouse and select Browse.

The Symbol at Cursor command opens an ObjectBrowser window for the symbol the cursor is on in the active edit window. The symbol may be any class, function, or variable symbol that is defined in your source code.

Rewind

The Rewind command takes the ObjectBrowser back to the previous view. Choosing the Rewind command is the same as choosing the Rewind button.

Overview

The Overview command shows an overview. An overview of classes is the class hierarchy. An overview of functions is a list of all functions. An overview of variables is a list of all variables. Choosing the Overview command is the same as choosing the Overview button.

Inspect

The Inspect command displays the detail of the selected item. Choosing the Inspect command is the same as choosing the Inspect button.

Goto

The Goto command takes you to the source code for the selected item. Choosing the Goto command is the same as choosing the Goto button.

Options menu

The Options menu contains commands that let you view and change various default settings in Borland C++. 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 Borland 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 | Environment | Preferences menu, your changes will be automatically saved when you exit from Borland C++.

The Options | Application menu choice brings up the Set Application Options dialog box. This dialog box provides the easiest and safest way to set up compilation and linking for a DOS or Windows executable. To use this dialog box, simply push one of the buttons. Borland C++ will verify and, if necessary, change some of the settings in the Code Generation, Entry/Exit Code Generation, and Linker dialog boxes. See page 88 (Entry/Exit Code) for detailed information on the code generated. Use this dialog box for initial setup only.

In the Turbo C++ for Windows environment, only the Windows App and Windows DLL options are available. Standard DOS and DOS overlay applications must be compiled with the Borland C++ IDE (or using the Borland C++ command line compiler).

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.

**DOS Standard:**

- pushes the Small memory model radio button in the Code Generation dialog box
Options | Application

- sets Assume SS equals DS to Default for memory model in the Code Generation dialog box
- pushes the DOS Standard radio button in the Entry/Exit Code Generation dialog box
- pushes the Standard DOS .EXE radio button in the Linker | Settings dialog box

DOS Overlay:
- pushes the Medium memory model button in the Code Generation dialog box
- sets Assume SS equals DS to Default for memory model in the Code Generation dialog box
- pushes the DOS Overlay button in the Entry/Exit Code Generation dialog box
- pushes the Overlaid DOS .EXE button in the Linker | Settings dialog box

Windows App:
- pushes the Small memory model button in the Code Generation dialog box
- sets Assume SS equals DS to Default for memory model in the Code Generation 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
- unchecks the Graphics Library option in the Libraries dialog box

Windows DLL:
- pushes the Compact memory model button in the Code Generation dialog box
- sets Assume SS equals DS to Never in the Code Generation 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
- unchecks the Graphics Library option in the Libraries dialog box

Borland C++ only

Chapter 3, Menus and options reference
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:

![Code Generation dialog box](image)

Here are what the various buttons and check boxes mean:

- The Model buttons determine which memory model you want to use. The default memory model is Small. The memory model chosen determines the normal method of memory addressing. Refer to Chapter 9, "DOS memory management," in the Programmer's Guide for more information about memory models in general.

- There are some restrictions about which memory models you can use for Windows executables. The Turbo C++ for Windows IDE allows you to select Small, Medium, Compact and Large memory models. Tiny and Huge are not supported.

- 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 Borland C++ to align noncharacter data (within structures and unions only) at even addresses. When this option is off (unchecked), Borland 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 Borland 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 Borland 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`.

- Check Generate Assembler Source to tell Borland C++ to produce an `.ASM` assembly language source file as its output, rather than an `.OBJ` object module.

- Compile Via Assembler allows you to specify that the compiler should produce assembly language output, then invoke TASM to assemble the output.

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, tiny, 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.

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

The Floating Point buttons let you decide how you want Borland 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 8087 (Borland C++ only) or 80287 to generate direct 8087 or 80287 inline code.

The Instruction Set (Borland C++ only) radio buttons let you choose what instruction set to generate code for. The default instruction set, 8088/8086, works with all PCs.
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 in order to use either the integrated debugger or the standalone Turbo Debugger.

Turning this option off allows you to link and create larger object files. While this option doesn’t affect execution speed, it does affect compilation time.

- Browser Info in OBJs controls whether information needed by the Turbo C++ for Windows ObjectBrowser is included in object (.OBJ) files. The default for this check box is off (unchecked). If you want to use ObjectBrowser to inspect your program (from within the Turbo C++ for Windows IDE), you must turn this option on.

Leaving this option off saves space in your object files.

- 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, as 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 \texttt{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 predefined size into far data segments; the Far Data Threshold specifies the minimum size above which data objects will be automatically made far.

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 Windows, the compiler generates a different prolog and epilog than it would for DOS. Because of this, you must use the Entry/Exit Code Generation dialog box to set the appropriate application. If you use the Set Application Options dialog box (described on page 82), the settings in the Entry/Exit Code 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.

If you want to set the prolog/epilog code for a DOS application, you need to select DOS Standard or DOS Overlay.
Push the DOS Standard radio button to tell the compiler to generate code that may not be safe for overlays. If you don’t plan to create an overlaid application, use this option.

Push the DOS Overlay radio button to tell the compiler to generate overlay safe code. Use this option when you’re creating an overlaid application.

If you want to set the prolog/epilog code for a Windows application, you need to 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 by 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 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.

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.
In the Borland C++ IDE, you can also select Register, to specify the new fastcall parameter-passing convention. For more information about the fastcall convention, see Appendix A, "Optimization."

**Important!**

Do not change this option unless you’re an expert and have read Chapter 12, “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.

This option is automatically turned off when you turn optimizations on; a duplicate of the Standards Stack Frame option also appears on the Options | Compiler | Optimization dialog box.

- **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++.
The Use C++ Compiler radio buttons tell Borland 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.

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, but uses .OBJ (and .ASM) extensions only available with TLINK 3.0 and TASM 2.0 (or newer).

- 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 Template Generation options allow you to specify how Borland C++ generates template instances in C++. For more information about templates, see Chapter 3 “C++ specifics,” in the Programmer’s Guide.

- Smart generates public (global) definitions for all template instances, but if more than one module generates the same template instance the linker will automatically merge duplicates to produce a single definition. This is the default setting, and is normally the most convenient way of generating template instances.

- Global, like Smart, generates public definitions for all template instances. However, it does not merge duplicates, so if the same template instance is generated more than once the linker will report public symbol redefinition errors.

- External tells the compiler to generate external references to all template instances. If you use this option, you must make certain that the instances are publicly defined elsewhere in your code.
- 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 (the latter is done automatically if you are using the huge memory model).

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.

The Advanced C++ Options command displays a dialog box with settings that control advanced code generation options for C++. Since Borland C++ version 3.0 handles certain C++ features more efficiently (but differently) than previous versions of Borland C++, some of these options are intended primarily for backward compatibility, where it is necessary to link with object modules or libraries compiled with older versions.

Borland C++ supports three different kinds of member pointer types, which you can control with these options.
### C++ Member Pointers

*(* Support **all** cases
    * ) Support multiple inheritance
    * ) Support single inheritance
    * ) Smallest for class

- Support All Cases (the default) places no restrictions on what members can be pointed to. Member pointers will use the most general (but not always the most efficient) representation.
- Support Multiple Inheritance allows member pointers to point to members of multiple inheritance classes, with the exception of members of virtual base classes.
- Support Single Inheritance permits member pointers to point to members of base classes that use single inheritance only.
- Smallest for Class specifies that member pointers will use the smallest possible representation that allows member pointers to point to all members of their particular class.

### Virtual Base Pointers

*(* Always near
    * ) Same size as 'this' pointer

When a class inherits virtually from a base class, the compiler stores a hidden pointer in the class object to access the virtual base class sub-object. Borland C++ 3.0 always makes this hidden pointer a **near** pointer by default, to generate more efficient code. Previous versions of Borland C++ matched the size of this pointer to the size of the ‘this’ pointer used by the class itself.

- Always Near specifies that the hidden pointer should always be near, for the smallest and most efficient code.
- Same Size as ‘this’ Pointer tells the compiler to match the size of the hidden pointer to the size of the ‘this’ pointer in the instance class, for backward compatibility.

### Options

* ] 'deep' virtual bases
* ] True 'pascal' member functions
* ] Honor precision of member pointers
* ] Disable constructor displacements
* ] Pass class values via reference
* ] Vtable pointer follows data members

Borland C++ 3.0 sometimes handles pointers differently from previous versions, in order to permit greater efficiency and flexibility. In some cases, this results in behavior that is incompatible with previous versions. To permit complete compatibility, the following options are provided:
Options | Compiler | Advanced C++ Options

- 'Deep' Virtual Bases directs the compiler not to change the layout of any classes in order to relax the restrictions on pointers to members of base classes through multiple levels of virtual inheritance.

- True Pascal Member Functions directs the compiler to pass the 'this' pointer to 'pascal' member functions as the first parameter on the stack. By default, Borland C++ 3.0 passes the 'this' pointer as the last parameter, which permits smaller and faster member function calls.

- Honor Precision of Member Pointers tells the compiler to honor an explicit cast to a pointer to a member of a simpler base class, even though it is actually pointing to a derived class member.

- Disable Constructor Displacements instructs the compiler not to add hidden members and code to a derived class, which it does by default to prevent an erroneous value for the 'this' pointer in special cases where the constructor of a derived class containing an inherited virtual function that it overrides, calls that function using a pointer to the virtual base class. This option ensures compatibility with the behavior of previous versions.

- Pass Class Values Via Reference tells the compiler to use a reference to a temporary variable in order to pass arguments of type class to a function. By default Borland C++ 3.0 copy-constructs the argument values directly to the stack.

- Vtable Pointer Follows Data Members instructs the compiler to place virtual table pointers after any nonstatic data members of the class, for compatibility with previous versions of Borland C++. The default method for version 3.0 is to place these pointer before any nonstatic data members, to make virtual member function calls smaller and faster.

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 for size or speed.

The Borland C++ IDE supports a full range of professional optimization options, while the Turbo C++ for Windows environment provides a more limited subset of optimizations.

For Turbo C++ for Windows, the Optimizations dialog box looks like this:

Optimizations (Turbo C++ for Windows)
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.

  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 Chapter 9, “DOS memory management,” in the *Programmer’s Guide* 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 Borland 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. If you are creating Windows applications, normally you'll want to optimize for speed.

In the Borland C++ character-based IDE, you have full access to the professional optimization features introduced in Borland C++ 3.0. The dialog box presents you with three separate categories of options, to let you fully customize the way the compiler optimizes your code. These features are listed briefly below. For your convenience, the command-line compiler switches corresponding to each option are indicated. A more complete discussion of optimization, including a description of the use and functionality of each menu option, appears in Appendix A, "The Optimizer."

The Optimizations Options affect how optimization of your code occurs.

- **Global Register Allocation** corresponds to the `-Oe` switch on the command line compiler. It enables global register allocation and variable live range analysis.
- **Invariant Code Motion**, corresponding to the `-Om` command line switch, moves invariant code out of loops.
- **Induction Variables** corresponds to the `-Ov` command line switch. It enables loop induction variables and strength reduction optimizations.
Loop Optimizations corresponds to -O1 option, and compacts loops into REP/STOSx instructions.

Suppress Redundant Loads corresponds to the -Z command line switch. It suppresses reloads of values that are already in registers.

Copy Propagation, corresponding to the -Op command line switch, propagates copies of constants, variables, and expressions where possible.

Assume no pointer aliasing corresponds to the -Oa command line switch. It instructs the compiler to assume that pointer expressions are not aliased in common subexpression evaluation.

Dead Code Elimination corresponds to the -Ob command line switch, and eliminates stores into dead variables.

Jump Optimization, corresponding to the -O compiler switch, removes jumps to jumps, unreachable code, and unnecessary jumps.

Inline Intrinsic Functions, corresponding to the -Oi compiler switch, instructs the compiler to expand common functions like strcpy() inline.

Standard Stack Frame instructs the compiler to generate a standard function entry/exit code. Corresponds to the -k-compiler option.

The Register Variables selections affect how the compiler handles the use of register variables. For more information about register variables see Chapter 9, “DOS memory management,” in the Programmer’s Guide.

None instructs the compiler not to use register variables even if you have used the register keyword.

Register Keyword specifies that register variables will be used only if you use the register keyword and a register is available.

Automatic directs the compiler to automatically assign register variables for you.

The Common Subexpressions tells the compiler how to find and eliminate duplicate expressions in your code, to avoid reevaluating the same expression.

No Optimization instructs the compiler not to eliminate common subexpressions.
Options | Compiler | Optimizations

- Optimize Globally corresponds to the -Og command line switch, and instructs the compiler to eliminate common subexpressions within an entire function.

- Optimize Locally corresponds to the -Oc command line switch, and instructs the compiler to eliminate common subexpressions within basic blocks only.

The Optimize For options let you change Borland C++'s code generation strategy. For backward compatibility, these buttons correspond to the same buttons in the Turbo C++ for Windows environment, and in earlier versions of Borland C++. They are not identical to the “Smallest Code” and “Fastest Code” buttons that appear at the bottom of the Optimization dialog box.

The three buttons at the bottom of the Optimizations dialog box allow you to specify “groups” of settings by making a single selection.

- No Optimizing corresponds to the -Od command line switch. It automatically disables all of the optimization options.

- Fastest Code corresponds to the -O2 command line switch. It automatically sets all of the optimization options to generate the fastest possible code.

- Smallest Code corresponds to the -O1 command line switch. It automatically sets the optimization options to produce the smallest possible code.

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 Nested Comments check box allows you to nest comments in Borland C++ source files. Nested comments are not allowed in standard C implementations. They are not portable.

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

- Choosing Borland C++ tells the compiler to recognize the Borland 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 Borland C++ extension keywords as normal identifiers.

- Choosing UNIX V tells the compiler to recognize only UNIX V keywords and treat any Borland C++ extension keywords as normal identifiers.

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

Use the Identifier Length 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**

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.

- **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.

**Portability**

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.

The Names command brings up a dialog box which 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 read Chapter 9, "DOS memory management," in the Programmer's Guide.

The Options | Transfer command (available in the Borland C++ IDE only) lets you add or delete programs in the = menu. Once you've done so, you can run those programs without actually leaving Borland C++. You return to Borland C++ after you exit the program you transferred to. The Transfer command displays this dialog box:
The Transfer dialog box has two sections:

- the Program Titles list
- the Transfer buttons

The Program Titles section lists short descriptions of programs that have been installed and are ready to execute. You might need to scroll the list box to see all the programs available.

The Transfer buttons let you edit and delete the names of programs you can transfer to, as well as cancel any changes you've made to the transfer list. There's also a Help button to get more information about using the transfer dialog box.

Choose Edit to add or change the Program Titles list that appears in the => menu. The Edit button displays the Modify/New Transfer Item dialog box.

If you're positioned on a transfer item when you select Edit, the input boxes in the Modify/New dialog box are automatically filled in; otherwise they're blank.

Using the Modify/New dialog box, you take these steps to add a new file to the Transfer dialog box:

1. Type a short description of the program you're adding on the Program Title input box.
Note that if you want your program to have a keyboard shortcut (like the S in the Save command or the t in the Cut command), you should include a tilde (~) in the name. Whatever character follows the tilde appears in bold or in a special color in the = menu, indicating that you can press that key to choose the program from the menu.

2. Tab to Program Path and enter the program name and optionally include the full path to the program. (If you don’t enter an explicit path, only programs in the current directory or programs in your regular DOS path will be found.)

3. Tab to Command Line and type any parameters or macro commands you want passed to the program. Macro commands always start with a dollar sign ($) and are entered in uppercase. For example, if you enter $CAP EDIT, all output from the program will be redirected to a special Edit window in Borland C++.

4. If you want to assign a hot key, tab to the Hot Key options and assign a shortcut to this program. Transfer shortcuts must be Shift plus a function key. Keystrokes already assigned appear in the list but are unavailable.

5. Now click or choose the New button to add this program to the list.

To modify an existing transfer program, cursor to it in the Program Titles list of the Transfer dialog box and then choose Edit. After making the changes in the Modify/New Transfer dialog box, choose the Modify button.

The Translator check box lets you put the Transfer program into the Project File Translators list (the list you see when you choose Project | Local Options). Check this option when you add a transfer program that is used to build part of your project.

The Delete button removes the currently selected program from the list and the = menu.

Transfer macros The IDE recognizes certain strings of characters called transfer macros in the parameter string of the Modify/New Transfer Item dialog box. The transfer macros are fully documented in the online file UTIL.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:

![The Make dialog box](image)

Note that the Turbo C++ for Windows version of the Make dialog box is slightly different from the Borland C++ version. In Turbo C++ for Windows, neither the "Run librarian" nor the "Generate Import Library" options are available.

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 combine your projects .OBJ files into a .LIB (library) file. The default is to run the linker to generate an executable application.

The Generate Import Library buttons are available in the Borland C++ IDE only.

These buttons control when and how IMPLIB is executed during the MAKE process. The Use DLL File Exports option generates an import library that consists of the exports in the DLL. The Use DEF File Exports generates an import library of exports in the DEF file. If either of these options is checked, MAKE invokes IMPLIB after the linker has created the DLL. This option controls how the transfer macro $IMPLIB gets expanded.
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 both Borland C++ and Turbo C++ for Windows, as well as the command-line version of Borland C++ 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 recomplied 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.

Note that the Borland C++ and Turbo C++ for Windows environments provide slightly different linker options. This is because Turbo C++ for Windows is a "Windows only" programming environment; therefore, DOS-oriented options are not supported.

For Borland C++ the Settings command opens up this 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. Standard DOS EXE produces a normal executable that runs under DOS. Overlaid DOS EXE produces an executable that is capable of being overlaid. 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 Borland C++. When you're linking with modules created by a compiler other than Borland C++, you may wish to leave this option is off (unchecked).

This option applies only to Windows applications and DLLs. When this option 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 Warn Duplicate Symbols option affects whether the linker warns you of previously encountered symbols in .LIB files.

The “No Stack” Warning option affects whether the linker generates the “No stack” message. It's normal for a program generated under the tiny model to display this message if the message is not turned off.

The “No Stack” Warning option does not appear in the Turbo C++ for Windows IDE, since Windows does not support the tiny model.

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 this option. This option is probably only useful when you are trying to export non-callback functions from DLLs—as in exported C++ member functions. This option isn’t necessary for normal Windows callback functions (declared FAR PASCAL).

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. See Chapter 4 in the *Tools and Utilities Guide* for a more in-depth discussion of desirable sizes.

With Segment Alignment, you can 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.

The Libraries dialog box has several radio buttons that allow you to choose what libraries will automatically be linked into your application.

The Graphics Library option controls the automatic searching of the BGI graphics library. When this option is checked, it is possible to build and run single-file graphics programs without using a project file. Unchecking this option speeds up the link step a bit because the linker doesn’t have to search in the BGI graphics library file.

The BGI Graphics library is not windows-compatible, so this option does not appear in the Turbo C++ for Windows IDE.

---

**Figure 3.22**

The Libraries dialog box

The Libraries dialog box has several radio buttons that allow you to choose what libraries will automatically be linked into your application.

The Graphics Library option controls the automatic searching of the BGI graphics library. When this option is checked, it is possible to build and run single-file graphics programs without using a project file. Unchecking this option speeds up the link step a bit because the linker doesn’t have to search in the BGI graphics library file.

The BGI Graphics library is not windows-compatible, so this option does not appear in the Turbo C++ for Windows IDE.
**Note:** You can uncheck this option and still build programs that use BGI graphics, provided you add the name of the BGI graphics library (GRAPHICS.LIB) to your project list.

The Turbo Vision library option (Borland C++ only) instructs the linker to automatically include the Turbo Vision application framework library when linking your application.

Turbo Vision is a DOS character-mode application framework. It is not windows-compatible, and this option does not appear in the Turbo C++ for Windows IDE.

The Container class library option tells the linker to automatically link in the Borland 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 Borland C++ ObjectWindows library 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 Borland 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.

---

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.
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).

The Options | Debugger command lets you make several settings affecting the integrated debugger. (Turbo C++ for Windows does not contain an integrated debugger, so this option does not appear in the Turbo C++ for Windows IDE.) This command opens this dialog box:

The following sections describe the contents of this box.
The Source Debugging radio buttons determine whether debugging information is included in the executable file and how the .EXE is run under Borland C++.

Programs linked with this option set to On (the default) can be debugged with either the integrated debugger or the standalone Turbo Debugger. Set this to On when you want to debug in the IDE.

If you set this option to Standalone, programs can be debugged only with Turbo Debugger, although they can still be run in Borland C++.

If you set this option to None, programs cannot be debugged with either debugger, because no debugging information has been placed in the .EXE file.

The Display Swapping radio buttons let you set when the integrated debugger will change display windows while running a program.

If you set Display Swapping to None, the debugger does not swap the screen at all. You should only use this setting for debugging sections of code that you’re certain do not output to the screen.

When you run your program in debug mode with the default setting of Smart, the debugger looks at the code being executed to see whether it will generate output to the screen. If it does (or if it calls a function), the screen is swapped from the IDE screen to the User Screen long enough for output to be displayed, then is swapped back. Otherwise, no swapping occurs.

Be aware of the following with smart swapping:

- It swaps on any function call, even if the function does no screen output.
- In some situations, the IDE screen might be modified without being swapped; for example, if a timer interrupt routine writes to the screen.

If you set Display Swapping to Always, the debugger swaps screens every time a statement executes. You should choose this setting any time the IDE screen is likely to be overwritten by your running program.

Note: If you’re debugging in dual monitor mode (that is, you used the Borland C++ command-line /d option or specified Dual monitor mode on the Options | Environment | Startup dialog box), you can
see your program’s output on one monitor and the Borland C++
screen on the other. In this case, Borland C++ never swaps screens
and the Display Swapping setting has no effect.

In the Inspectors check boxes, when Show Inherited is checked, it
tells the integrated debugger to display all member functions and
methods—whether they are defined within the inspected class or
inherited from a base class. When this option is not checked, only
those fields defined in the type of the inspected object are
displayed.

When checked, the Show Methods option tells the integrated
debugger to display member functions when you inspect a class.

Check the Show Decimal, Show Hex, or Show Both radio buttons
when you want to control how the values in inspectors are
displayed. Show both is on by default.

You can use the Program Heap Size input box to input how much
memory Borland C++ should assign a program when you debug
it. The actual amount of memory that Borland C++ tries to give to
the program is equal to the size of the executable image plus the
amount you specify here.

The default value for the program heap size is 64 Kbytes. You
may want to increase this value if your program uses dynamically
allocated objects.

Directories

The Options | Directories command lets you tell Borland C++
where to find the files it needs to compile, link, output, and
debug. This command opens the following dialog box containing
four 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 Borland C++ startup object files (C07.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. Borland C++ looks for and writes files to 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.

Borland C++ only

The Source Directories input box specifies the directories where the Borland C++ integrated debugger will look for the source code to libraries that do not belong to the open project (for example, container class libraries). Multiple directories can be entered, separated by semicolons. If the entry is blank, the current directory is searched.

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,

\[ C:\C\LIB;C:\C\MYLIBS;A:\BORLANDC\MATHLIBS;A:..\VIDLIBS \]

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, Desktop, Startup, and Colors.

In the Turbo C++ for Windows environment, the Startup and Colors options are not available. In addition, some of the other selections are slightly different from their Borland C++ equivalents.

Preferences

The Screen Size radio buttons let you specify whether your IDE screen is displayed in 25 lines or 43/50 lines. One or both of these buttons will be available, depending on the type of video adapter in your PC.
When set to 25 lines (the default), Borland C++ uses 25 lines and 80 columns. This is the only screen size available to systems with a monochrome display or Color Graphics Adapter (CGA).

If your PC has EGA or VGA, you can set this option to 43/50 lines. The IDE is displayed in 43 lines by 80 columns if you have an EGA, or 50 lines by 80 columns if you have a VGA.

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.

The Command Set options allow you to 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 Borland C++ IDE. For more information about the CUA and alternate editor command sets, see Chapter 2, “IDE Basics.”

If Editor Files is checked in the Auto Save options, and if the file has been modified since the last time you saved it, Borland 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 Borland C++.

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

When the Project option is checked, Borland 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 Borland C++.

When Save Old Messages is checked, Borland 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, Borland C++ automatically clears messages before a compile, a make, or a transfer that uses the Message window.
If you choose Editor from the Environment menu, these are the options you can pick from:

- When Create Backup Files is checked (the default), Borland 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, Borland C++ inserts a true tab character (ASCII 9) when you press Tab. When this option is not checked, Borland 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, Borland C++ begins every autoindented 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. This option is only effective when Cursor Through Tabs is also selected.

- 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. (See page 53 for more information about Undo and Redo.)

- 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 clipboard) replaces the entire selected text with the inserted text.

If you check Use Tab Character in this dialog box and press Tab, Borland C++ inserts a tab character in the file and the cursor moves to the next tab stop. The Tab Size input box allows you to 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. Borland C++ redisplays 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 Borland 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
- Go to Cursor Same as Run | Go To Cursor
- Breakpoint Same as Debug | Toggle Breakpoint
- Inspect Same as Debug | Inspect
- Evaluate Same as Debug | Evaluate
- Add Watch Same as Debug | Watches | Add Watch

In the Turbo C++ for Windows IDE, which does not support integrated debugging, the options from “Go to Cursor” and below are not available. The Turbo C++ for Windows environment, however, has an additional option, “Browse,” which sets the right mouse button to have the same effect as selecting Browse | Symbol at cursor from the menu.

The remaining mouse options, Mouse Double Click and Reverse Mouse Buttons, are available in the Borland C++ IDE only.

In the Mouse Double Click box, you can change the slider control bar to adjust the double-click speed of your mouse by using the arrow keys or the mouse.

Moving the scroll box closer to Fast means Borland C++ requires a shorter time between clicks to recognize a double click. Moving the scroll box closer to Slow means Borland C++ will still recognize a double click even if you wait longer between clicks.

If you want to experiment with different settings, you can double-click the Test button above the scroll bar. When you successfully double-click the bar it becomes highlighted.

When Reverse Mouse Buttons is checked, the active button on your mouse is the rightmost one instead of the leftmost. Note,
however, that the buttons won’t actually be switched until you choose the OK button.

Depending on how you hold your mouse and whether you’re right- or left-handed, the right mouse button might be more comfortable to use than the left.

Desktop

The Desktop dialog box lets you set whether history lists, the contents of the Clipboard, watch expressions, breakpoints, open and closed windows are saved across sessions. History lists and open windows are saved by default; because watch expressions and breakpoints may not be meaningful across sessions, they are not saved by default, nor are windows that you have closed. You can change these defaults by unchecking or checking the respective options.

The Turbo C++ for Windows IDE does not offer the Watch expressions and Breakpoints options. However, it provides an additional set of radio buttons allowing you to control the appearance of the SpeedBar. 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 2, “IDE Basics.”

Startup

The Startup dialog box allows you to set various startup options for the Borland C++ IDE.

- When Borland C++ switches between graphics and text mode to (run or debug a graphics program), the video display may become corrupted unless the entire EGA or VGA video palette is saved in a separate buffer during the switch. Save entire palette should be left unchecked if you will not be running or debugging graphics programs, since saving the palette slows down execution speed.
Dual monitor mode lets you run your program on one monitor while debugging in the IDE on another monitor.

Snow checking tells Borland C++ to check for video “snow.” This usually occurs only on older CGA video adapters. You should disable this option if your display driver doesn’t have a “snow” problem.

Auto detect (the default) tells Borland C++ to check your hardware on startup and set its video mode automatically.

Color specifies that Borland C++ should always run CGA/EGA/VGA/XGA monitors in color mode.

Black & White / LCD tells Borland C++ to run CGA/EGA/VGA/XGA monitors always in black and white mode. This mode should be used for most laptop LCD monitors.

Monochrome tells Borland C++ to run always in monochrome mode.

Swap File Drive lets you specify a disk drive use as a swap file in the event that Borland C++ runs out of memory while compiling or linking your project. If you have a RAM drive, you should specify this as your swap drive, to improve speed.

Use Extended Memory allows you to tell Borland C++ how much extended memory (in Kilobytes) to reserve for its use.

Use EMS Memory allows you to tell Borland C++ how many expanded memory (EMS) pages to reserve for its use.

The Colors dialog box (Borland C++ ONLY) allows you to set your color preferences for each component of the Borland C++ IDE. Simply select any Group and any Item within that Group, and the available Foreground and Background colors will appear in the respective sections of the dialog box. A sample of the currently selected scheme will appear in the Item Color box. To change the default color for that item, select your preferences from the Foreground and Background palettes. When you have modified all the items you wish to change, select OK to exit and save your changes. To exit without recording any changes you have made, select Cancel.
Save

The Options I 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, Borland C++ looks in the Executable directory (from which BC.EXE is run) for the same file.

Window menu

The Window menu contains window management commands. Most of the windows you open from this menu have all the standard window elements like scroll bars, a close box, and zoom boxes. Refer to page 30 for information on these elements and how to use them.

The Turbo C++ IDE Window menu differs somewhat from that of Borland C++ IDE. Although not as many window management commands are on the Turbo C++ Window menu, the same functionality exists. If you know how to use Windows, you’ll know how to manage windows within the Turbo C++ IDE; just use the same commands as you would in other Windows applications.
Size/Move

Choose Window | Size/Move to change the size or position of the active window.

When you choose this command, the active window moves in response to the arrow keys. When the window is where you want it, press Enter. You can also move a window by dragging its title bar.

If you press Shift while you use the arrow keys, you can change the size of the window. When it's the size you want it, press Enter. If a window has a resize corner, you can drag that corner or any other corner to resize it.

Zoom

Choose Window | Zoom to resize the active window to the maximum size. If the window is already zoomed, you can choose this command again to restore it to its previous size. You can also double-click anywhere on the top line (except where an icon appears) of a window to zoom or unzoom it.

Tile

Choose Window | Tile to tile all your open windows.

Cascade

Choose Window | Cascade to stack all open windows.

Arrange Icons

Choosing Window | Arrange Icons will rearrange any icons on the Turbo C++ desktop so they are evenly spaced, beginning at the lower left corner of the desktop.
Next

Choose Window | Next to make the next window active, which makes it the topmost open window.

<table>
<thead>
<tr>
<th>Ctrl</th>
<th>F6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt</td>
<td>F6</td>
</tr>
</tbody>
</table>

Close

Choose Window | Close to close the active window. You can also click the close box in the upper left corner to close a window.

<table>
<thead>
<tr>
<th>Ctrl</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt</td>
<td>F3</td>
</tr>
</tbody>
</table>

Close All

In Borland C++, choose Close All to close all windows and clear all history lists. This command is useful when you’re starting a new project. In Turbo C++, Close All simply closes all open windows on the Turbo C++ desktop. History lists are not saved.

Message

In Borland C++, you can display transfer program output in the Message window.

Choose Window | Message to open the Message window and make it active. The Message window displays error and warning messages, which you can use for reference, or you can select them and have the corresponding location be highlighted in the edit window. When a message refers to a file that is not currently loaded, you can press the Spacebar to load that file. When an error is selected in the Message window, press Enter to show the location of the error in the edit window and make the edit window active at the point of error.

To close the window, click its close box or choose Window | Close.

Output

Choose Window | Output to open the Output window and make it active. The Output window displays text from any DOS command-line text and any text generated from your program (no graphics).
The Output window is handy while debugging because you can view your source code, variables, and output all at once. This is especially useful when you’ve set the Options | Environment dialog box to a 43/50 line display and you are running a standard 25-line mode program. In that case, you can see almost all of the program output and still have plenty of lines to view your source code and variables.

If you would rather see your program’s text on the full screen—or if your program generates graphics—choose the Window | User Screen command instead.

To close the window, click its close box or choose Window | Close.

Choose Window | Watch to open the Watch window and make it active. The Watch window displays expressions and their changing values so you can keep an eye on how your program evaluates key values.

You use the commands in the Debug | Watches pop-up menu to add or remove watches from this window. Refer to the section on this menu for information on how to use the Watch window (page 73).

To close the window, click its close box or choose Window | Close.

Choose Window | User Screen to view your program’s full-screen output. If you would rather see your program output in a Borland C++ window, choose the Window | Output command instead.

Clicking or pressing any key returns you to the IDE.

Choose Window | Register to open the Register window and make it active.

The Register window displays CPU registers and is used most often when debugging inline ASM and TASM modules in your project.
To close the window, click its close box or choose Window | Close.

Choose Window | Project to open the Project window, which lets you view the list of files you’re using to create your program.

Choose Window | Project Notes to write down any details, make to-do lists, or list any other information about your project files.

Choose Window | List All to get a list of all the windows you’ve opened. The list contains the names of all files that are currently open as well as any of the last eight files you’ve opened in an edit window but have since closed. A recently closed file appears in the list prefixed with the word closed.

When you choose an already open file from the list, Borland C++ brings the window to the front and makes it active. When you choose a closed file from the list, Borland C++ reopens the file in an edit window the same size and location as when the window was closed. The cursor is positioned at its last location.

In Turbo C++ for Windows, you can see a list of recently closed files at the bottom of the File menu, and you can see a list of open windows at the bottom of the Window menu. Choosing a closed file name reopens the file in a new edit window. Choosing an open window makes that window the active one.

The Help menu gives you access to online help in a special window. There is help information on virtually all aspects of the IDE and Borland C++. (Also, one-line menu and dialog box hints appear on the status line whenever you select a command.)

To open the Help window in Borland C++, do one of these actions:

- Press F1 at any time (including from any dialog box or when any menu command is selected).
When getting help in a dialog box or menu, you cannot resize the window or copy to the clipboard. In this instance, Tab takes you to dialog box controls, not the next keyword.

- When an edit window is active and the cursor is positioned on a word, press Ctrl+F1 to get language help on that word.
- Click Help whenever it appears on the status line or in a dialog box.

To close the Help window, press Esc, click the close box, or choose Window | Close. You can keep the Help window onscreen while you work in another window unless you opened the Help window from a dialog box or pressed F1 when a menu command was selected.

Help screens often contain keywords (highlighted text) that you can choose to get more information. Press Tab to move to any keyword; press Enter to get more detailed help. (As an alternative, move the cursor to the highlighted keyword and press Enter.) With a mouse, you can double-click any keyword to open the help text for that item.

You can also cursor around the Help screen and press Ctrl+F1 on any word to get help. If the word is not found, an incremental search is done in the index and the closest match displayed.

When the Help window is active, you can copy from the window and paste that text into an edit window. You do this just the same as you would in an edit window: Select the text first, choose Edit | Copy, move to an edit window, then choose Edit | Paste.

To select text in the Help window, drag across the desired text or, when positioned at the start of the block, press Shift+→, ←, ↑, ↓ to mark a block.

You can also copy preselected program examples from help screens by choosing the Edit | Copy Example command.

Turbo C++ for Windows uses the Windows Help system. If you know how to use Help in other Windows applications, you’ll know how to get help in Turbo C++.

Contents

The Help | Contents command opens the Help window with the main table of contents displayed. From this window, you can branch to any other part of the help system.

You can get help on Help by pressing F1 when the Help window is active. You can also reach this screen by clicking on the status line.
The Help | Index command displays a full list of help keywords (the special highlighted text in help screens that let you quickly move to a related screen).

You can scroll the list or you can incrementally search it by pressing letters from the keyboard. For example, to see what’s available under “printing,” you can type p r i. When you type p, the cursor jumps to the first keyword that starts with p. When you then type r, the cursor then moves to the first keyword that starts with pr. When you then type i, the cursor moves to the first keyword that starts with pri, and so on.

When you find a keyword that interests you, choose it by cursoring to it and pressing Enter. (You can also double-click it.)

The Help | Topic Search command displays language help on the currently selected item.

To get language help, position the cursor on an item in an edit window and choose Topic Search. You can get help on things like function names (printf, for example), header files, reserved words, and so on. If an item is not in the help system, the help index displays the closest match.

The Help | Previous Topic command opens the Help window and redisplays the text you last viewed.

Borland C++ lets you back up through 20 previous help screens. You can also click on the status line to view the last help screen displayed.

The Help | Help on Help command opens up a text screen that explains how to use the Borland C++ help system. If you’re already in help, you can bring up this screen by pressing F1.

The Help | Using Help option in Turbo C++ for Windows is very similar to the Help on Help command in Borland C++. 
Active File

The Help | Active Help command displays a dialog box that lets you select the help file you want the IDE to use. Theses are the topics you can get help on:

- IDE, C++ language, and Windows API
- ObjectWindows API
- Turbo Vision API

About

When you choose this command, a dialog box appears that shows you copyright and version information for Borland C++ or Turbo C++ for Windows. Press Esc or click OK (or press Enter) to close the box.
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. Borland 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 Borland C++ to Compile | Make

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

- add or delete files from your project
set options for a file in the 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:

```c
#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 « "
";
}
```

MYFUNCS.CPP looks like this:

```c
char ss[] = "The restaurant at the end of ";
char *GetString(void)
{
    return ss;
}
```

And myfuncs.h looks like this:

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

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

The first step is to tell Borland 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).

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.
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, Local Options, and Include Files options are enabled on the Project menu.

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 status line at the bottom of the screen shows which actions can be performed at this point: F1 gives you help, Ins adds files to the Project, Del deletes a file from the Project, Ctrl+O lets you select options for a file, Spacebar lets you view information about the include files required by a file in the Project, Enter opens an editor window for the currently selected file, and F10 takes you to the main menu. You can also click on any of these items with the mouse to take the appropriate action. Press Ins now to add a file to the project list.

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.

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 can also search for a file in the Files list box by typing the first few letters of the one you want. In this case, typing my should take you right to MYFUNCS.CPP. Press Enter. 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. Borland C++ will compile files in the exact order they appear in the project.

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, Borland 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 I Make to make your project and choose Run I Run to run it. To view your output, choose Window I User Screen, then press any key to return to the IDE.

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 I Environment).

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, make options, and transfer items. The desktop file contains the state of all windows at the last time you were using the project.

The next time you use Borland C++, you can jump right into your project by reloading the project file. Borland C++ automatically
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.

Since you want to see the effect of tracking in multiple files, you need to modify the criterion Borland 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 de-
monstrate errors in multiple files, choose Fatal Errors in the Make
dialog box.

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. Press any key when the Errors: Press any
key message flashes.

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, separat-
ing the various messages generated by each module and its in-
clude 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), Borland 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 and debug stepping.

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 I 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.

Using different file translators

So far you've built projects that use Borland C++ as the only language translator. Many projects consist of both C++ code and assembler code, and possibly code written in other languages. It would be nice to have some way to tell Borland C++ how to build such modules using the same dependency checks that we've just described. With the Project Manager, you don't need to worry about forgetting to rebuild those files when you change some of the source code, or about whether you've put them in the right directory, and so on.

For every source file that you have included in the list in the Project window, you can specify

- which program (Borland C++, TASM, and so on) to use as its target file
- which command-line options to give that program
- whether the module is an overlay
- what to call the resulting module and where it will be placed (this information is used by the project manager to locate files needed for linking)
- whether the module contains debug information
- whether the module gets included in the link

By default, the IDE's compiler is chosen as the translator for each module, using no command-line override options, using the
Output directory for output, assuming that the module is not an overlay, and assuming that debug information is not to be excluded.

Let's look at a simple example. Go to the Project window and move to the file MYFUNCS.CPP. Now press Ctrl+O to bring up the Override Options dialog box for this file:

Except for Borland C++, each of the names in the Project File Translators list box is a reference to a program defined in the Transfer dialog box (Options | Transfer).

Press Esc, then F10 to return to the main menu, then choose Options | Transfer. The Transfer dialog box that appears contains a list of all the transfer programs currently defined. Use the arrow keys to select Turbo Assembler and press Enter. (Since the Edit button is the default, pressing Enter brings up the Modify/New Transfer Item dialog box.) Here you see that Turbo Assembler is defined as the program TASM in the current path. Notice that the Translator check box is marked with an X; this translator item is then displayed in the Override Options dialog box. Press Esc to return to the Transfer dialog box.

Suppose you want to compile the MYFUNCS module using the Borland C++ command-line compiler instead of the IDE's compiler. To do so, you would perform the following steps:

1. First, you need to define BCC as one of the Project File Translators in the Transfer dialog box. Cursor past the last entry in the Program Titles list, then press Enter to bring up the Modify/New Transfer Item dialog box. In the Program Title
input box, type Borland C++ command-line compiler; in the Program Path input box, type BCC; and in the command line, type $EDNAME.

2. Then check Translator by pressing Spacebar and press Enter (New is the default action button). Back at the Transfer dialog box, you see that Borland C++ command-line compiler is now in the Program Titles list box (the last part doesn’t show). Choose OK and press Enter.

3. Back in the Project window, press Ctrl+O to go to the Override Options dialog box again. Notice that Borland C++ command-line compiler is now a choice on the Project File Translators list for MYFUNCS.CPP (as well as for all of your other files).

Tab to the Project File Translators list box and highlight Borland C++ command-line compiler (at this point, pressing Enter or tabbing to another group will choose this entry). Use the Command-line Options input box to add any command-line options you want to give BCC when compiling MYFUNCS.CPP.

MYFUNCS.CPP now compiles using BCC.EXE, while all of your other source modules compile with BC.EXE. The Project Manager will apply the same criteria to MYFUNCS.CPP when deciding whether to recompile the module during a make as it will to all the modules that are compiled with BC.EXE.

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, and be able to record notes about what you're doing as you're working. 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:

```c
#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

```c
#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:

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

int GetMyTime(int which) {
    struct tm *timeptr;
    time_t secsnow;
    time(&secsnow);
    timeptr = localtime(&secsnow);
    // actual code goes here
}
```
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. Press Ins (or choose Project | Add Item) 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. 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. Simply choose the file using the arrow keys or the mouse, then 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.

Notes for your project

Now that you’ve had a chance to see the code in MYMAIN.CPP and mytime.h, you might decide to make some changes at a later time. Choose Window | Project Notes to bring up a new edit window that is kept as part of your project file. Type in any comments you want to remember about your project.

Each project maintains its own notes file, so that you can keep notes that go with the project you’re currently working on; they’re available at the touch of a button when you select the project file.
The command-line compiler

As an alternative to using the IDE, you can compile and run your programs with the command-line compiler (BCC.EXE). Almost anything you can do within the IDE can also be done using the command-line compiler. You can set warnings on or off, invoke TASM (or another assembler) to assemble .ASM source files, invoke the linker to generate executable files, and so on. In fact, if you only want to compile your C or C++ source file(s), you must use the -c option at the command line.

This chapter is organized into two parts. The first describes how to use the command-line compiler and provides a summary table of all the options. The second part, starting on page 174, presents the options organized functionally (with groups of related options).

The summary table, Table 5.1 (starting on page 143), summarizes the command-line compiler options and provides a page-number cross-reference to where you can find more detailed information about each option.

Using the command-line compiler

The command-line compiler uses DPMI (Dos Protected Mode Interface) to run in protected mode on 286, 386, or i486 machines with at least 640K conventional RAM and at least 1MB extended memory.
Note that, although Borland C++ runs in protected mode, it still generates applications that run in real mode. The advantage to using Borland C++ in protected mode is that the compiler has much more room to run than if you were running it in real mode, so it can compile larger projects faster and without extensive disk-swapping.

---

DPMIINST

For more information about running DPMIINST, see Chapter 1, Installing Borland C++.

---

The protected mode interface is completely transparent to the user. Borland C++ uses an internal database of various machine characteristics to determine how to enable protected mode on your machine, and configures itself accordingly. The only time you may need to be aware of it is when running the compiler for the first time. If your machine is not recognized by Borland C++, you will need to run the DPMIINST program by typing (at the DOS prompt)

```
DPMIINST
```

and following the program's instructions. DPMIINST runs your machine through a series of tests to determine the best way of enabling protected mode.

---

Running BCC

You can also use a configuration file. See page 147 for details.

---

To invoke Borland C++ from the command line, type `BCC` at the DOS prompt and follow it with a set of command-line arguments. Command-line arguments include compiler and linker options and file names. The generic command-line format is

```
BCC [option [option...]] filename [filename...]
```

Each command-line option may be preceded by either a hyphen (-) or slash (/), whichever you prefer. Each option must be separated from the BCC command, other options, and following file names by at least one space. 

---

Using the options

Compiler options are further divided into ten groups.

---

The options are divided into three general types:

- compiler options, described starting on page 148
- linker options, described starting on page 167
- environment options, described starting on page 167
To see an onscreen list of the options, type `bcc` (without any options or file names) at the DOS prompt. Then press Enter.

In order to select command-line options, enter a hyphen (`-`) or slash (`/`) immediately followed by the option letter (for example, `-l` or `/l`). To turn an option off, add a second hyphen after the option letter. This is true for all toggle options (those that turn an option on or off): A trailing hyphen (`-`) turns the option off, and a trailing plus sign (`+`) or nothing turns it on. So, for example, `-C` and `-C+` both turn nested comments on, while `-C-` turns nested comments off.

**Option precedence rules**

The option precedence rules are simple; command-line options are evaluated from left to right, and the following rules apply:

- For any option that is not an `-l` or `-L` option, a duplication on the right overrides the same option on the left. (Thus an off option on the right cancels an on option to the left.)
- The `-l` and `-L` options on the left, however, take precedence over those on the right.

**Table 5.1: Command-line options summary**

<table>
<thead>
<tr>
<th>Option</th>
<th>Page</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>@filename</td>
<td>147</td>
<td>Read compiler options from the response file <code>filename</code></td>
</tr>
<tr>
<td>+filename</td>
<td>147</td>
<td>Use the alternate configuration file <code>filename</code></td>
</tr>
<tr>
<td>-l</td>
<td>151</td>
<td>Generate 80186 instructions</td>
</tr>
<tr>
<td>-l-</td>
<td>151</td>
<td>Generate 8088/8086 instructions (default)</td>
</tr>
<tr>
<td>-2</td>
<td>151</td>
<td>Generate 80286 protected-mode compatible instructions</td>
</tr>
<tr>
<td>-A</td>
<td>156</td>
<td>Use only ANSI keywords</td>
</tr>
<tr>
<td>-A-, -AT</td>
<td>156</td>
<td>Use Borland C++ keywords (default)</td>
</tr>
<tr>
<td>-AK</td>
<td>156</td>
<td>Use only Kernighan and Ritchie keywords</td>
</tr>
<tr>
<td>-AU</td>
<td>156</td>
<td>Use only UNIX keywords</td>
</tr>
<tr>
<td>-a</td>
<td>151</td>
<td>Align word</td>
</tr>
<tr>
<td>-a-</td>
<td>151</td>
<td>Align byte (default)</td>
</tr>
<tr>
<td>-B</td>
<td>161</td>
<td>Compile and call the assembler to process inline assembly code</td>
</tr>
<tr>
<td>-b</td>
<td>151</td>
<td>Make enums always word-sized (default)</td>
</tr>
<tr>
<td>-b-</td>
<td>151</td>
<td>Make enums byte-sized when possible</td>
</tr>
<tr>
<td>-C</td>
<td>156</td>
<td>Nested comments on</td>
</tr>
<tr>
<td>-C-</td>
<td>156</td>
<td>Nested comments off (default)</td>
</tr>
<tr>
<td>-c</td>
<td>161</td>
<td>Compile to .OBJ but do not link</td>
</tr>
<tr>
<td>-Dname</td>
<td>150</td>
<td>Define <code>name</code> to the null string</td>
</tr>
<tr>
<td>-Dname=string</td>
<td>150</td>
<td>Define <code>name</code> to <code>string</code></td>
</tr>
<tr>
<td>-d</td>
<td>151</td>
<td>Merge duplicate strings on</td>
</tr>
<tr>
<td>-d-</td>
<td>151</td>
<td>Merge duplicate strings off (default)</td>
</tr>
<tr>
<td>-efilename</td>
<td>161</td>
<td>Use <code>filename</code> as the assembler to use</td>
</tr>
<tr>
<td>-efilename</td>
<td>167</td>
<td>Link to produce <code>filename.EXE</code></td>
</tr>
<tr>
<td>-Fc</td>
<td>151</td>
<td>Generate COMDEFS</td>
</tr>
<tr>
<td>-Ff</td>
<td>151</td>
<td>Create far variables automatically</td>
</tr>
<tr>
<td>Option</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>(-Ff=)size</td>
<td>151</td>
<td>Create far variables automatically; sets the threshold to size</td>
</tr>
<tr>
<td>(-Fm)</td>
<td>151</td>
<td>Enables the (-Fc), (-Ff), and (-Fs) options</td>
</tr>
<tr>
<td>(-Fs)</td>
<td>151</td>
<td>Assume DS = SS in all memory models</td>
</tr>
<tr>
<td>(-f)</td>
<td>152</td>
<td>Emulate floating point (default)</td>
</tr>
<tr>
<td>(-f=)</td>
<td>152</td>
<td>Don't do floating point</td>
</tr>
<tr>
<td>(-ff)</td>
<td>152</td>
<td>Fast floating point (default)</td>
</tr>
<tr>
<td>(-ff=)</td>
<td>152</td>
<td>Strict ANSI floating point</td>
</tr>
<tr>
<td>(-f87)</td>
<td>153</td>
<td>Use 8087 hardware instructions</td>
</tr>
<tr>
<td>(-f287)</td>
<td>153</td>
<td>Use 80287 hardware instructions</td>
</tr>
<tr>
<td>(-G)</td>
<td>175</td>
<td>Select code for speed</td>
</tr>
<tr>
<td>(-G=)</td>
<td>175</td>
<td>Select code for size (default)</td>
</tr>
<tr>
<td>(-gn)</td>
<td>157</td>
<td>Warnings: stop after (n) messages</td>
</tr>
<tr>
<td>(-H)</td>
<td>161</td>
<td>Causes the compiler to generate and use precompiled headers</td>
</tr>
<tr>
<td>(-H=)</td>
<td>161</td>
<td>Turns off generation and use of precompiled headers (default)</td>
</tr>
<tr>
<td>(-Hu)</td>
<td>161</td>
<td>Tells the compiler to use but not generate precompiled headers</td>
</tr>
<tr>
<td>(-H=filename)</td>
<td>161</td>
<td>Sets the name of the file for precompiled headers</td>
</tr>
<tr>
<td>(-h)</td>
<td>153</td>
<td>Use fast huge pointer arithmetic</td>
</tr>
<tr>
<td>(-Ipath)</td>
<td>168</td>
<td>Directories for include files</td>
</tr>
<tr>
<td>(-in)</td>
<td>156</td>
<td>Make significant identifier length to be (n)</td>
</tr>
<tr>
<td>(-Jg)</td>
<td>166</td>
<td>Generate definitions for all template instances and merge duplicates</td>
</tr>
<tr>
<td>(-Jgd)</td>
<td>166</td>
<td>Generate public definitions for all template instances; duplicates will result in redefinition errors</td>
</tr>
<tr>
<td>(-Jgx)</td>
<td>166</td>
<td>Generate external references for all template instances</td>
</tr>
<tr>
<td>(-jn)</td>
<td>157</td>
<td>Errors: stop after (n) messages</td>
</tr>
<tr>
<td>(-K)</td>
<td>153</td>
<td>Default character type unsigned</td>
</tr>
<tr>
<td>(-K=)</td>
<td>153</td>
<td>Default character type signed (default)</td>
</tr>
<tr>
<td>(-k)</td>
<td>153</td>
<td>Standard stack frame on (default)</td>
</tr>
<tr>
<td>(-Lpath)</td>
<td>168</td>
<td>Directories for libraries</td>
</tr>
<tr>
<td>(-lx)</td>
<td>167</td>
<td>Pass option (x) to the linker (can use more than one (x))</td>
</tr>
<tr>
<td>(-l-x)</td>
<td>167</td>
<td>Suppress option (x) for the linker</td>
</tr>
<tr>
<td>(-M)</td>
<td>167</td>
<td>Instruct the linker to create a map file</td>
</tr>
<tr>
<td>(-mc)</td>
<td>149</td>
<td>Compile using compact memory model</td>
</tr>
<tr>
<td>(-mh)</td>
<td>149</td>
<td>Compile using huge memory model</td>
</tr>
<tr>
<td>(-ml)</td>
<td>149</td>
<td>Compile using large memory model</td>
</tr>
<tr>
<td>(-mm)</td>
<td>149</td>
<td>Compile using medium memory model</td>
</tr>
<tr>
<td>(-mm=)</td>
<td>149</td>
<td>Compile using medium model; assume DS != SS</td>
</tr>
<tr>
<td>(-ms)</td>
<td>149</td>
<td>Compile using small memory model (default)</td>
</tr>
<tr>
<td>(-ms=)</td>
<td>149</td>
<td>Compile using small model; assume DS != SS</td>
</tr>
<tr>
<td>(-mt)</td>
<td>149</td>
<td>Compile using tiny memory model</td>
</tr>
<tr>
<td>(-mt=)</td>
<td>149</td>
<td>Compile using tiny model; assume DS != SS</td>
</tr>
<tr>
<td>(-N)</td>
<td>154</td>
<td>Check for stack overflow</td>
</tr>
<tr>
<td>(-npath)</td>
<td>168</td>
<td>Set the output directory</td>
</tr>
<tr>
<td>(-O1)</td>
<td>174</td>
<td>Generate smallest possible code</td>
</tr>
<tr>
<td>(-O2)</td>
<td>174</td>
<td>Generate fastest possible code</td>
</tr>
<tr>
<td>(-Od)</td>
<td>174</td>
<td>Disable all optimizations</td>
</tr>
<tr>
<td>(-On)</td>
<td>174</td>
<td>(Oa-Ox) Optimization options</td>
</tr>
<tr>
<td>(-ofilename)</td>
<td>161</td>
<td>Compile source file to filename.obj</td>
</tr>
<tr>
<td>(-P)</td>
<td>161</td>
<td>Perform a C++ compile regardless of source file extension</td>
</tr>
<tr>
<td>(-Pext)</td>
<td>161</td>
<td>Perform a C++ compile and set the default extension to (ext)</td>
</tr>
<tr>
<td>(-P=)</td>
<td>161</td>
<td>Perform a C++ or C compile depending on source file extension (default)</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><code>-P-ext</code></td>
<td>Perform a C++ or C compile depending on extension; set default extension to <code>ext</code></td>
<td></td>
</tr>
<tr>
<td><code>-p</code></td>
<td>Use Pascal calling convention</td>
<td></td>
</tr>
<tr>
<td><code>-pr</code></td>
<td>Usefastcall calling convention for passing parameters in registers</td>
<td></td>
</tr>
<tr>
<td><code>-p-</code></td>
<td>Use C calling convention (default)</td>
<td></td>
</tr>
<tr>
<td><code>-Qe</code></td>
<td>Instructs the compiler to use all available EMS memory (default)</td>
<td></td>
</tr>
<tr>
<td><code>-Qe-</code></td>
<td>Instructs the compiler to not use any EMS memory</td>
<td></td>
</tr>
<tr>
<td><code>-Qx</code></td>
<td>Instructs the compiler to use extended memory</td>
<td></td>
</tr>
<tr>
<td><code>-r</code></td>
<td>Use register variables on (default)</td>
<td></td>
</tr>
<tr>
<td><code>-r-</code></td>
<td>Suppresses the use of register variables.</td>
<td></td>
</tr>
<tr>
<td><code>-rd</code></td>
<td>Only allow declared register variables to be kept in registers</td>
<td></td>
</tr>
<tr>
<td><code>-R</code></td>
<td>Generate ObjectBrowser information</td>
<td></td>
</tr>
<tr>
<td><code>-S</code></td>
<td>Produce .ASM output file</td>
<td></td>
</tr>
<tr>
<td><code>-Tstring</code></td>
<td>Pass string as an option to TASM or assembler specified with <code>-E</code></td>
<td></td>
</tr>
<tr>
<td><code>-T-</code></td>
<td>Remove all previous assembler options</td>
<td></td>
</tr>
<tr>
<td><code>-tDe</code></td>
<td>Make the target a DOS .EXE file</td>
<td></td>
</tr>
<tr>
<td><code>-tDc</code></td>
<td>Make the target a DOS .COM file</td>
<td></td>
</tr>
<tr>
<td><code>-tW</code></td>
<td>Make the target a Windows module, using the same options as <code>-W</code></td>
<td></td>
</tr>
<tr>
<td><code>-Uname</code></td>
<td>Undefine any previous definitions of name</td>
<td></td>
</tr>
<tr>
<td><code>-u</code></td>
<td>Generate underscores (default)</td>
<td></td>
</tr>
<tr>
<td><code>-u-</code></td>
<td>Disables underscores</td>
<td></td>
</tr>
<tr>
<td><code>-v,-v-</code></td>
<td>Source debugging on</td>
<td></td>
</tr>
<tr>
<td><code>-vi,-vi-</code></td>
<td>Controls expansion of inline functions</td>
<td></td>
</tr>
<tr>
<td><code>-V</code></td>
<td>Smart C++ virtual tables</td>
<td></td>
</tr>
<tr>
<td><code>-Va</code></td>
<td>Pass class arguments by reference to a temporary variable</td>
<td></td>
</tr>
<tr>
<td><code>-Vb</code></td>
<td>Make virtual base class pointer same size as ‘this’ pointer of the class</td>
<td></td>
</tr>
<tr>
<td><code>-Vc</code></td>
<td>Do not add the hidden members and code to classes with pointers to virtual base class members</td>
<td></td>
</tr>
<tr>
<td><code>-Vf</code></td>
<td>Far C++ virtual tables</td>
<td></td>
</tr>
<tr>
<td><code>-Vmv</code></td>
<td>Member pointers have no restrictions (most general representation)</td>
<td></td>
</tr>
<tr>
<td><code>-Vmm</code></td>
<td>Member pointers support multiple inheritance</td>
<td></td>
</tr>
<tr>
<td><code>-Vms</code></td>
<td>Member pointers support single inheritance</td>
<td></td>
</tr>
<tr>
<td><code>-Vmd</code></td>
<td>Use the smallest representation for member pointers</td>
<td></td>
</tr>
<tr>
<td><code>-Vmp</code></td>
<td>Honor the declared precision for all member pointer types</td>
<td></td>
</tr>
<tr>
<td><code>-Vo</code></td>
<td>Enable all of the ‘backward compatibility’ -V switches (-Va, -Vb, -Vc, -Vp, -Vt, -Vv)</td>
<td></td>
</tr>
<tr>
<td><code>-Vp</code></td>
<td>Pass the ‘this’ parameter to ‘pascal’ member functions as the first parameter on the stack</td>
<td></td>
</tr>
<tr>
<td><code>-Vs</code></td>
<td>Local C++ virtual tables</td>
<td></td>
</tr>
<tr>
<td><code>-Vt</code></td>
<td>Place the virtual table pointer after non-static data members</td>
<td></td>
</tr>
<tr>
<td><code>-Vv</code></td>
<td>Do not change the layout of classes to relax restrictions on member pointers</td>
<td></td>
</tr>
<tr>
<td><code>-V0, -V1</code></td>
<td>External and Public C++ virtual tables</td>
<td></td>
</tr>
<tr>
<td><code>-W</code></td>
<td>Creates an .OBJ for Windows with all functions exportable</td>
<td></td>
</tr>
<tr>
<td><code>-WD</code></td>
<td>Creates an .OBJ for Windows to be linked as a .DLL with all functions exportable</td>
<td></td>
</tr>
<tr>
<td><code>-WDE</code></td>
<td>Creates an .OBJ for Windows to be linked as a .DLL with explicit export functions</td>
<td></td>
</tr>
<tr>
<td><code>-WE</code></td>
<td>Creates an .OBJ for Windows with explicit export functions</td>
<td></td>
</tr>
<tr>
<td><code>-WS</code></td>
<td>Creates an .OBJ for Windows that uses smart callbacks</td>
<td></td>
</tr>
<tr>
<td><code>-w</code></td>
<td>Display warnings on</td>
<td></td>
</tr>
<tr>
<td><code>-wxxx</code></td>
<td>Enable xxx warning message</td>
<td></td>
</tr>
</tbody>
</table>
Table 5.1: Command-line options summary (continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-w-xxx</td>
<td>Disable xxx warning message</td>
</tr>
<tr>
<td>-X</td>
<td>Disable compiler autodependency output</td>
</tr>
<tr>
<td>-Y</td>
<td>Enable overlay code generation</td>
</tr>
<tr>
<td>-Yo</td>
<td>Overlay the compiled files</td>
</tr>
<tr>
<td>-y</td>
<td>Line numbers on</td>
</tr>
<tr>
<td>-Z</td>
<td>Enable register load suppression optimization</td>
</tr>
<tr>
<td>-zAname</td>
<td>Code class</td>
</tr>
<tr>
<td>-zBname</td>
<td>BSS class</td>
</tr>
<tr>
<td>-zCname</td>
<td>Code segment</td>
</tr>
<tr>
<td>-zDname</td>
<td>BSS segment</td>
</tr>
<tr>
<td>-zEname</td>
<td>Far segment</td>
</tr>
<tr>
<td>-zFname</td>
<td>Far class</td>
</tr>
<tr>
<td>-zGname</td>
<td>BSS group</td>
</tr>
<tr>
<td>-zHname</td>
<td>Far group</td>
</tr>
<tr>
<td>-zPname</td>
<td>Code group</td>
</tr>
<tr>
<td>-zRname</td>
<td>Data segment</td>
</tr>
<tr>
<td>-zSname</td>
<td>Data group</td>
</tr>
<tr>
<td>-zTname</td>
<td>Data class</td>
</tr>
<tr>
<td>-zX*</td>
<td>Use default name for X. (default)</td>
</tr>
</tbody>
</table>

Syntax and file names

Borland C++ compiles files according to the following set of rules:

- filename.asm: Invoke TASM to assemble to .OBJ
- filename.obj: Include as object at link time
- filename.lib: Include as library at link time
- filename: Compile FILENAME.CPP
- filename.cpp: Compile FILENAME.CPP
- filename.c: Compile FILENAME.C
- filename.xyz: Compile FILENAME.XYZ

For example, given the following command line

```
BCC -a -f -C -01 -emyexe oldfile1 oldfile2 nextfile
```

Borland C++ compiles OLDFILE1.CPP, OLDFILE2.CPP, and NEXTFILE.CPP to an .OBJ, linking them to produce an executable program file named MYEXE.EXE with word alignment (-a), floating-point emulation (-f), nested comments (-C), and generate smallest code (-01) selected.

Borland C++ invokes TASM if you give it an .ASM file on the command line or if a .C or .CPP file contains inline assembly.

Here are the options that the command-line compiler gives to TASM:

```
/D_ _MODEL_ _ /D_ _LANG_ _ /ml /FLOATOPT
```
Response files

Response files allow you to have longer command strings than DOS normally allows.

See page 143 for what those rules are.

Configuration files

TURBOC.CFG is not the same as TCConfig.TC, which is the default IDE version of a configuration file.

where MODEL is one of: TINY, SMALL, MEDIUM, COMPACT, LARGE, or HUGE. The /mi option tells TASM to assemble with case sensitivity on. LANG is CDECL or PASCAL; FLOATOPT is r when you’ve specified -f87 or -f287; e otherwise.

If you need to specify many options or files on the command line, you can place them in an ASCII text file, called a response file (you can of course name it anything you like). You can then tell the command-line compiler to read its command line from this file by including the appropriate file name prefixed with @. You can specify any number of such files, and you can mix them freely with other options and file names.

For example, suppose the file MOON.RSP contains STARS.C and RAIN.C. This command

BCC SUN.C @MOON.RSP ANYONE.C

will cause Borland C++ to compile the files SUN.C, STARS.C, RAIN.C, and ANYONE.C in real mode. It expands to

BCC SUN.C STARS.C RAIN.C ANYONE.C

Any options included in a response file are evaluated just as though they had been typed in on the command line.

If you find you use a certain set of options over and over again, you can list them in a configuration file, called TURBOC.CFG by default. If you have a TURBOC.CFG configuration file, you don’t need to worry about using it. When you run BCC, it automatically looks for TURBOC.CFG in the current directory. If it doesn’t find it there, Borland C++ then looks in the startup directory (where BCC.EXE or BCCX.EXE resides).

You can create more than one configuration file; each must have a unique name. To specify the alternate configuration file name, include its file name, prefixed with +, anywhere on the BCC command line. For example, to read the option settings from the file D:\ALT.CFG, you could use the following command line:

BCC +D:\ALT.CFG .......

Your configuration file can be used in addition to or instead of options entered on the command line. If you don’t want to use
certain options that are listed in your configuration file, you can override them with options on the command line.

You can create the TURBOC.CFG file (or any alternate configuration file) using any standard ASCII editor or word processor, such as Borland C++'s integrated editor. You can list options (separated by spaces) on the same line or list them on separate lines.

Option precedence rules

In general, you should remember that command-line options override configuration file options. If, for example, your configuration file contains several options, including the \(-a\) option (which you want to turn off), you can still use the configuration file but override the \(-a\) option by listing \(-a-\) in the command line. However, the rules are a little more detailed than that. The option precedence rules detailed on page 143 apply, with these additional rules:

1. When the options from the configuration file are combined with the command-line options, any \(-I\) and \(-L\) options in the configuration file are appended to the right of the command-line options. This means that the include and library directories specified in the command line are the first ones that Borland C++ searches (thereby giving the command-line \(-I\) and \(-L\) directories priority over those in the configuration file).

2. The remaining configuration file options are inserted immediately after the BCC command (to the left of any command-line options). This gives the command-line options priority over the configuration file options.

Compiler options

Borland C++'s command-line compiler options fall into ten groups; the page references to the left of each group tell where you can find a discussion of each kind of option:

- See page 149. 1. Memory model options let you tell Borland C++ which memory model to use when compiling your program.
- See page 150. 2. Macro definitions let you define and undefine macros on the command line.
- See page 151. 3. Code-generation options govern characteristics of the generated code, such as the floating-point option, calling convention, character type, or CPU instructions.
4. **Optimization options** let you specify how the object code is to be optimized; a more detailed discussion of optimization options appears in Appendix A, "The Optimizer."

See page 156.

5. **Source code options** cause the compiler to recognize (or ignore) certain features of the source code; implementation-specific (non-ANSI, non-Kernighan and Ritchie, and non-UNIX) keywords, nested comments, and identifier lengths.

See page 157.

6. **Error-reporting options** let you tailor which warning messages the compiler will report, and the maximum number of warnings and errors that can occur before the compilation stops.

See page 159.

7. **Segment-naming control options** allow you to rename segments and to reassign their groups and classes.

See page 161.

8. **Compilation control options** let you direct the compiler to

- compile to assembly code (rather than to an object module)
- compile a source file that contains inline assembly
- compile without linking
- compile for Windows applications
- use precompiled headers or not

See page 163.

9. **EMS options** let you control how much expanded or extended memory Borland C++ uses.

See page 164.

10. **C++ virtual table options** let you control how virtual tables are handled.

See page 165.

11. **C++ member pointer options** let you control how member pointers are used.

See page 166.

12. **Template generation options** let you control how the compiler generates definitions or external declarations for template instances.

See page 168.

13. **Backward compatibility options** let you tell the compiler to use particular code generation strategies to insure backward compatibility with earlier versions of Borland C++.

---

**Memory model**

Memory model options let you tell Borland C++ which memory model to use when compiling your program. The memory models are tiny, small, medium, compact, large, and huge.

- `-mc` Compile using compact memory model
- `-mh` Compile using huge memory model
- `-ml` Compile using large memory model
- `-mm` Compile using medium memory model

---

See Chapter 9 in the Programmer's Guide for in-depth information on the memory models (what they are, how to use them).
Macro definitions

Macro definitions let you define and undefine macros (also called manifest or symbolic constants) on the command line. The default definition is the null string. Macros defined on the command line override those in your source file.

-`-Dname` Defines the named identifier `name` to the null string.

-`-Dname=string` Defines the named identifier `name` to the string `string` after the equal sign. `string` cannot contain any spaces or tabs.

-`-Uname` Undefines any previous definitions of the named identifier `name`.

Borland C++ lets you make multiple `#define` entries on the command line in any of the following ways:

- You can include multiple entries after a single `-D` option, separating entries with a semicolon (this is known as “ganging” options):
  
  ```
  BCC -Dxxx;yyy=1;zzz=NO MYFILE.C
  ```

- You can place more than one `-D` option on the command line:
  
  ```
  BCC -Dxxx -Dyyy=1 -Dzzz=NO MYFILE.C
  ```
You can mix ganged and multiple -D listings:
BCC -Dxxx -Dyyy=1;zzz=NO MYFILE.C

Code-generation options

Code-generation options govern characteristics of the generated code, such as the floating-point option, calling convention, character type, or CPU instructions.

-1 This option causes Borland C++ to generate extended 80186 instructions. It also generates 80286 programs running in real mode, such as with the IBM PC/AT under DOS.

-1- Tells the compiler to generate 8088/8086 instructions (the default).

-2 This option causes Borland C++ to generate 80286 protected-mode compatible instructions.

-a This option forces integer size and larger items to be aligned on a machine-word boundary. Extra bytes are inserted in a structure to ensure member alignment. Automatic and global variables are aligned properly. char and unsigned char variables and fields can be placed at any address; all others are placed at an even-numbered address. This option is off by default (-a-), allowing bytewise alignment.

-b This option (which is on by default) tells the compiler to always allocate a whole word for enumeration types.

-b- This option tells the compiler to allocate a signed or unsigned byte if the minimum and maximum values of the enumeration are both within the range of 0 to 255 or -128 to 127, respectively.

-d This option tells the compiler to merge literal strings when one string matches another, thereby producing smaller programs. This option is off by default (-d-).

-Fc This generates communal variables (COMDEFs) for global "C" variables that are not initialized and not declared as static or extern. The advantage of using this option is that header files that are included in several source files can contain declarations of global variables. So long as a given variable doesn’t need to be initialized
to a nonzero value, you don't need to include a
definition for it in any of the source files. You can use
this option when porting code that takes advantage of a
similar feature with another implementation.

-Ff When you use this option, global variables greater than
or equal to the threshold size are automatically made far
by the compiler. The threshold size defaults to 32,767;
you can change it with the -Ff=size option. This option
is useful for code that doesn't use the huge memory
model but declares enough large global variables that
their total size exceeds (or is close to) 64K. For tiny,
small, and medium models this option has no effect.

If you use this option in conjunction with -Fc, the
generated COMDEFs will be far in the compact, large,
and huge models.

-Ff=size Use this option to change the threshold size used by the
-Ff option.

-Fm This option enables all the other -F options (-Fc, -Ff and
-Fs). You can use it as a handy shortcut when porting
code from other compilers.

-Fs This option tells the compiler to assume that DS is equal
to SS in all memory models; you can use it when porting
code originally written for an implementation that
makes the stack part of the data segment. When you
specify this option, the compiler will link in an alternate
startup module (C0Fx.OBJ) that will place the stack in
the data segment.

-f This option tells the compiler to emulate 80x87 calls at
run time if the run-time system does not have an 80x87;
if it does have one, the compiler calls the 80x87 chip for
floating-point calculations (the default).

-f- This option specifies that the program contains no
floating-point calculations, so no floating-point libraries
will be linked at the link step.

-ff With this option, the compiler optimizes floating-point
operations without regard to explicit or implicit type
conversions. Answers can be faster than under ANSI
operating mode. See Chapter 10, "Math," in the
-ff- This option turns off the fast floating-point option. The compiler follows strict ANSI rules regarding floating-point conversions.

-f87 This option tells the compiler to generate floating-point operations using inline 80x87 instructions rather than using calls to 80x87 emulation library routines. It specifies that a math coprocessor will be available at run time; therefore, programs compiled with this option will not run on a machine that does not have a math coprocessor.

-f287 This option is similar to -f87, but uses instructions that are only available with an 80287 (or higher) chip.

-h This option offers an alternative way of calculating huge pointer expressions; a way which is much faster but must be used with caution. When you use this option, huge pointers are normalized only when a segment wraparound occurs in the offset part. This will cause problems for huge arrays if any array elements cross a segment boundary. This option is off by default.

Normally, Borland C++ normalizes a huge pointer whenever adding to or subtracting from it. This ensures that, for example, if you have a huge array of structs that's larger than 64K, indexing into the array and selecting a struct field will always work with structs of any size. Borland C++ accomplishes this by always normalizing the results of huge pointer operations, so that the offset part contains a number that's no higher than 15. That way, a segment wraparound never occurs with huge pointers. The disadvantage of this approach is that it tends to be quite expensive in terms of execution speed. This option is automatically selected when compiling for Windows.

-K This option tells the compiler to treat all char declarations as if they were unsigned char type. This allows for compatibility with other compilers that treat char declarations as unsigned. By default, char declarations are signed (-K-).

-k This option generates a standard stack frame, which is useful when using a debugger to trace back through the stack of called subroutines. This option is on by default.
-N  This option generates stack overflow logic at the entry of each function, which causes a stack overflow message to appear when a stack overflow is detected. This is costly in terms of both program size and speed but is provided as an option because stack overflows can be very difficult to detect. If an overflow is detected, the message “Stack overflow!” is printed and the program exits with an exit code of 1.

-p  This option forces the compiler to generate all subroutine calls and all functions using the Pascal parameter-passing sequence. The resulting function calls are smaller and faster. Functions must pass the correct number and type of arguments, unlike normal C use, which permits a variable number of function arguments. You can use the cdecl statement to override this option and specifically declare functions to be C-type. This option is off by default (-p-).

-pr  This option forces the compiler to generate all subroutine calls and all functions using the newfastcall parameter-passing convention. With this option enabled, functions expect parameters to be passed in registers. You can also individually override the cdecl or pascal calling conventions by using the _fastcall modifier in declaring a function. For more information about _fastcall, see Appendix A, “The Optimizer.”

-u  With -u selected, when you declare an identifier, Borland C++ automatically puts an underscore (_) in front of the identifier before saving the identifier in the object module.

Borland C++ treats Pascal-type identifiers (those modified by the pascal keyword) differently—they are uppercase and are not prefixed with an underscore.

Underscores for C and C++ identifiers are optional, but on by default. You can turn them off with -u-. However, if you are using the standard Borland C++ libraries, you will encounter problems unless you rebuild the libraries. (To do this, you will need the Borland C++ run-time library source code; contact Borland for more information.)

-X  This option disables generation of autodependency information in the output file. Modules compiled with this
The -v and -vi options

Turbo Debugger is both a source level (symbolic) and assembly level debugger.

-v This option tells the compiler to include debugging information in the .OBJ file so that the file(s) being compiled can be debugged with either Borland C++'s integrated debugger or the standalone Turbo Debugger. The compiler also passes this option on to the linker so it can include the debugging information in the .EXE file.

To facilitate debugging, this option also causes C++ inline functions to be treated as normal functions. If you want to avoid that, use -vi.

-vi With this option enabled, C++ inline functions will be expanded inline.

In order to control the expansion of inline functions, the operation of the -v option is slightly different for C++. When inline function expansion is not enabled, the function will be generated and called like any other function. Debugging in the presence of inline expansion can be extremely difficult, so we provide the following options:

-v This option turns debugging on and inline expansion off.

-vi This option turns debugging off and inline expansion on.
-vi  This option turns inline expansion on.
-vi-  This option turns inline expansion off.

So, for example, if you want to turn both debugging and inline expansion on, you must use -v -vi.

Optimization options

Borland C++ is a professional optimizing compiler, featuring a number of options that let you specify how the object code is to be optimized; for size or speed, and utilizing (or not) a wide range of specific optimization techniques. Appendix A, "The Optimizer," discusses these options in detail.

Source code options

Source code options cause the compiler to recognize (or ignore) certain features of the source code; implementation-specific (non-ANSI, non-Kernighan and Ritchie, and non-UNIX) keywords, nested comments, and identifier lengths. These options are most significant if you plan to port your code to other systems.

-A  This option compiles ANSI-compatible code: Any of the Borland C++ extension keywords are ignored and can be used as normal identifiers. These keywords include

\begin{verbatim}
asm    _es    interrupt    _ss
cdecl  _export  _loadds  _saverregs
_cs    far     near    _fastcall
_ds    huge    pascal  _seg
\end{verbatim}

and the register pseudovariables, such as _AX, _BX, _SI, and so on.

-A-  This option tells the compiler to use Borland C++ keywords. -AT is an alternate version of this option.

-AK  This option tells the compiler to use only Kernighan and Ritchie keywords.

-AU  This option tells the compiler to use only UNIX keywords.

-C  This option allows you to nest comments. Comments may not normally be nested.

-In  This option causes the compiler to recognize only the first \( n \) characters of identifiers. All identifiers, whether vari-
Error-reporting options

Error-reporting options let you tailor which warning messages the compiler will report, and the maximum number of warnings and errors that can occur before the compilation stops.

- `gn` This option tells Borland C++ to stop compiling after \( n \) warning messages.
- `jn` This option tells the compiler to stop compiling after \( n \) error messages.
- `w` This option causes the compiler to display warning messages. You can turn this off with `-w-`. You can enable or disable specific warning messages with `-wxxx`, described in the following paragraphs.
- `-wxxx` This option enables the specific warning message indicated by `xxx`. The option `-w-xxx` suppresses the warning message indicated by `xxx`. The possible options for `-wxxx` are listed here and divided into four categories: ANSI violations, frequent errors (including more frequent errors), portability warnings, and C++ warnings. You can also use the pragma `warn` in your source code to control these options. See Chapter 4, "The preprocessor," in the Programmer's Guide.

### ANSI violations

The asterisk (*) indicates that the option is on by default. All others are off by default.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-wbbf</td>
<td>Bit fields must be <code>signed</code> or <code>unsigned int</code>.</td>
</tr>
<tr>
<td>-wbig*</td>
<td>Hexadecimal value contains more than three digits.</td>
</tr>
<tr>
<td>-wdpu*</td>
<td>Declare <code>type</code> prior to use in prototype.</td>
</tr>
<tr>
<td>-wdup*</td>
<td>Redefinition of <code>macro</code> is not identical.</td>
</tr>
</tbody>
</table>
-weas Assigning type to enumeration.
-wext* Identifier is declared as both external and static.
-wpin Initialization is only partially bracketed.
-wret* Both return and return with a value used.
-wstu* Undefined structure structure.
-wsus* Suspicious pointer conversion.
-wvoi* Void functions may not return a value.
-wzdi* Division by zero.

Frequent errors
-wamb Ambiguous operators need parentheses.
-wamp Superfluous & with function or array.
-wasm Unknown assembler instruction.
-waus* Identifier is assigned a value that is never used.
-wcce* Condition is always true/false.
-wdef Possible use of identifier before definition.
-weff* Code has no effect.
-wias* Array variable identifier is near.
-will* Ill-formed pragma.
-wnod No declaration for function function.
-wpar* Parameter parameter is never used.
-wpia* Possibly incorrect assignment.
-wpro Call to function with no prototype.
-wrch* Unreachable code.
-wrvl* Function should return a value.
-wstv Structure passed by value.
-wuse Identifier is declared but never used.

Portability warnings
-wcin Constant is long.
-wcpt* Nonportable pointer comparison.
-wrng* Constant out of range in comparison.
-wrpt* Nonportable pointer conversion.
-wsig Conversion may lose significant digits.
-wucp Mixing pointers to signed and unsigned char.

C++ warnings
-wbei* Initializing enumeration with type.
-wdsz* Array size for ‘delete’ ignored.
-whid* Function1 hides virtual function function2.
Segment-naming control

Don't use these options unless you have a good understanding of segmentation on the 8086 processor. Under normal circumstances, you will not need to specify segment names.

- **wibc** Base class base1 is inaccessible because also in base2.
- **winl** Functions containing identifier are not expanded inline.
- **wlin** Temporary used to initialize identifier.
- **wlvc** Temporary used for parameter in call to identifier.
- **wmpc** Conversion to type will fail for members of virtual base class base.
- **wmpd** Maximum precision used for member pointer type type.
- **wncf** Non-const function function called const object.
- **wnci** Constant member identifier is not initialized.
- **wnst** Use qualified name to access nested type type.
- **wnvf** Non-volatile function function called for volatile object.
- **wobi** Base initialization without a class name is now obsolete.
- **wofp** Style of function definition is now obsolete.
- **wovl** Overload is now unnecessary and obsolete.
- **wpre** Overloaded prefix operator ++/— used as a postfix operator.

Segment-naming control options allow you to rename segments and to reassign their groups and classes.

- **zAname** This option changes the name of the code segment class to name. By default, the code segment is assigned to class CODE.

- **zBname** This option changes the name of the uninitialized data segment class to name. By default, the uninitialized data segments are assigned to class BSS.

- **zCname** This option changes the name of the code segment to name. By default, the code segment is named _TEXT, except for the medium, large and huge models, where the name is filename_TEXT. (filename here is the source file name.)

- **zDname** This option changes the name of the uninitialized data segment to name. By default, the uninitialized data segment is named _BSS, except in the huge model, where no uninitialized data segment is generated.

-\texttt{-zEname}\ This option changes the name of the segment where far objects are put to \texttt{name}. By default, the segment name is the name of the far object followed by \_FAR. A name beginning with an asterisk (*) indicates that the default string should be used.

-\texttt{-zFname}\ This option changes the name of the class for far objects to \texttt{name}. By default, the name is FAR\_DATA. A name beginning with an asterisk (*) indicates that the default string should be used.

-\texttt{-zGname}\ This option changes the name of the uninitialized data segment group to \texttt{name}. By default, the data group is named DGROUP, except in the huge model, where there is no data group.

-\texttt{-zHname}\ This option causes far objects to be put into group \texttt{name}. By default, far objects are not put into a group. A name beginning with an asterisk (*) indicates that the default string should be used.

-\texttt{-zPname}\ This option causes any output files to be generated with a code group for the code segment named \texttt{name}.

-\texttt{-zRname}\ This option sets the name of the initialized data segment to \texttt{name}. By default, the initialized data segment is named _DATA, except in the huge model, where the segment is named \texttt{filename\_DATA}.

-\texttt{-zSname}\ This option changes the name of the initialized data segment group to \texttt{name}. By default, the data group is named DGROUP, except in the huge model, where there is no data group.

-\texttt{-zTname}\ This option sets the name of the initialized data segment class to \texttt{name}. By default the initialized data segment class is named DATA.

-\texttt{-zVname}\ This option sets the name of the far virtual table segment to \texttt{name}. By default far virtual tables are generated in the code segment.

-\texttt{-zWname}\ This option sets the name of the far virtual table class segment to \texttt{name}. By default far virtual table classes are generated in the CODE segment.
Compilation control options

Compilation control options allow you to control compilation of source files, such as whether your code is compiled as C or C++, whether to use precompiled headers, and what kind of Windows executable file is created. For more detailed information on how to create an Windows application, see Chapter 8, “Building a Windows application” in the Programmer’s Guide.

-zX* This option uses the default name for X. For example, -zA* assigns the default class name CODE to the code segment.

This option compiles and calls the assembler to process inline assembly code.

This option compiles and assembles the named .C, .CPP, and .ASM files, but does not execute a link command.

This option uses name as the name of the assembler to use. By default, TASM is used.

This option causes the compiler to generate and use precompiled headers, using the default filename TCDEF.SYM.

This option turns off generation and use of precompiled headers (this is the default).

This option tells the compiler to use but not generate precompiled headers.

This option sets the name of the file for precompiled headers, if you wish to save this information in a file other than TCDEF.SYM. This option also turns on generation and use of precompiled headers; that is, it also has the effect of -H.

This option compiles the named file to the specified filename.obj.

This option causes the compiler to compile your code as C++ always, regardless of extension. The compiler will assume that all files have .CPP extensions unless a different extension is specified with the code.
These five options (-W, -WD, -WDE, -WE, and -WS) relate to creating Windows applications. Note that you cannot use any of these options if you are using the -Y option (and vice versa).

-Pext
This option causes the compiler to compile all files as C++; it changes the default extension to whatever you specify with ext. This option is available because some programmers use .C or another extension as their default extension for C++ code.

-P-
This option tells the compiler to compile a file as either C or C++, based on its extension. The default extension is .CPP. This option is the default.

-P-ext
This option also tells the compiler to compile code based on the extension (.CPP as C++ code, all other file-name extensions as C code). It further specifies what the default extension is to be.

-S
This option compiles the named source files and produces assembly language output files (.ASM), but does not assemble. When you use this option, Borland C++ will include the C or C++ source lines as comments in the produced .ASM file.

-Tstring
This option passes string as an option to TASM (or as an option to the assembler defined with -E).

-T-
This option removes all previously defined assembler options.

-W
This option creates the most general kind of Windows executable, although not necessarily the most efficient. The compiler makes every far function exportable. This does not mean that all far functions actually will be exported, it only means that each far function can be exported. In order to actually export one of these functions, you must either use the _export keyword or add an entry for the function name in the EXPORTS section of the module definition file.

-WD
This option creates a module for use in a .DLL with all functions exportable.

-WDE
This option creates a module for use in a .DLL with only functions explicitly designated with _export as exportable.
-WE

This option creates an object module with only functions explicitly designated with `_export` as exportable.

-WS

Don't use this option for modules that will be compiled under the huge memory model.

This option creates an .OBJ with functions using smart callbacks. This option is recommended if you are writing Windows applications (not DLLs) which can assume SS = DS (most can). This option simplifies Windows programming; for instance, using it, you no longer need `MakeProcInstance` or `FreeProcInstance`, nor do you need to export your `WndProc`; instead, you can directly call a `WndProc`. Enabling this option results in faster Windows executables.

---

EMS and expanded memory options

If you have expanded (EMS) memory, you may want to make this memory available to the compiler for “swap” space in the event that your computer's extended (protected mode) memory is exhausted during compilation. These options give you the ability to control the compiler's use of EMS memory. You can also control the amount of expanded (protected mode) memory Borland C++ uses.

- `–Qe`
  This option instructs the compiler to use all EMS memory it can find. This is on by default for the command-line compiler (BCC). It speeds up your compilations, especially for large source files.

- `–Qe=yyyy`
  This option instructs the compiler to use `yyyy` pages (in 16K page sizes) of EMS memory for itself.

- `–Qe-`
  This option instructs the compiler not to use any EMS memory.

- `–Qx=nnnn`
  This option instructs the compiler to use `nnnn` bytes of extended memory.

---
C++ virtual tables

The -V option controls the C++ virtual tables. There are five variations of the -V option:

- **-V** Use this option when you want to generate 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 executables, but uses .OBJ and .ASM extensions only available with TLINK 3.0 and TASM 2.0 (or newer).

- **-Vs** Use this option when you want Borland C++ to generate 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.

- **-V0, -V1** These options work together to create global virtual tables. If you don’t want to use the Smart or Local options (-V or -Vs), you can use -V0 and -V1 to produce and reference global virtual tables. -V0 generates external references to virtual tables; -V1 produces public definitions for virtual tables.

When using these two options, at least one of the modules in the program must be compiled with the -V1 option to supply the definitions for the virtual tables. All other modules should be compiled with the -V0 option to refer to that Public copy of the virtual tables.

- **-Vf** You can use this option independently of or in conjunction with any of the other virtual table options. It causes virtual tables to be created in the code segment instead of the data segment (unless changed using the -zV and -zW options), and makes virtual table pointers into full 32-bit pointers (the latter is done automatically if you are using the huge memory model).

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.

C++ member pointers

The -Vm options control C++ member pointer types. There are five variations of the -Vm option:

The Borland C++ compiler supports three different kinds of member pointer types, with varying degrees of complexity and generality. By default, the compiler will use the most general (but in some contexts also the least efficient) kind for all member pointer types; this default behavior can be changed via the -Vm family of switches.

- Vmv  Member pointers declared while this option is in effect will have no restriction on what members they can point to; they will use the most general representation.

- Vmm  Member pointers declared while this option is in effect will be allowed to point to members of multiple inheritance classes, except that members of virtual base classes cannot be pointed to.

- Vms  Member pointers declared while this option is in effect will not be allowed to point to members of some base classes of classes that use multiple inheritance (in general, they can be used with single inheritance classes only).

- Vmd  Member pointers declared while this option is in effect will use the smallest possible representation that allows member pointers to point to all members of their class. If the class is not fully defined at the point where the member pointer type is declared, the most general representation has to be chosen by the compiler (and a warning is issued about this).

- Vmp  Whenever a member pointer is dereferenced or called, the compiler will treat the member pointer as if it were of the least general case needed for that particular pointer type. For example, a call through a pointer to...
member of a class that is declared without any base classes will treat the member pointer as having the simplest representation, regardless of how it's been declared. This will work correctly (and produce the most efficient code) in all cases except for one: when a pointer to a derived class is explicitly cast to a pointer to member of a 'simpler' base class, when the pointer is actually pointing to a derived class member. This is a non-portable (and dubious) construct, but if you need to compile code that uses it, use the -Vmp option. It will force the compiler to honor the declared precision for all member pointer types.

Template generation options

The \texttt{-Jg} option controls the generation of template instances in C++. There are three variations of the \texttt{-Jg} option:

\textbf{-Jg} Public definitions of all template instances encountered when this switch value is in effect will be generated, and if more than one module generates the same template instance, the linker will merge them to produce a single copy of the instance. This option (the default) is the most convenient approach to generating template instances. In order to generate the instances, however, the compiler must have available the function body (in the case of a template function) or the bodies of member functions and definitions for static data members (in the case of a template class).

\textbf{-Jgd} This option tells the compiler to generate public definitions for all template instances encountered.

Unlike the \texttt{-Jg} option, however, duplicate instances will \textit{not} be merged, causing the linker to report public symbol redefinition errors if more than one module defines the same template instance.

\textbf{-Jgx} This option instructs the compiler to generate external references to template instances. If you use this option you must make sure that the instances are publicly defined in some other module (using the \texttt{-Jgd} option), so that the external references will be satisfied.

For more information about templates, see Chapter 3, "C++ specifics," in the Programmer's Guide.
Linker options

See the section on TLINK in the Tools and Utilities Guide for a list of linker options.

-efilename
This option derives the executable program’s name from filename by adding the file extension .EXE (the program name will then be filename.EXE). filename must immediately follow the -e, with no intervening whitespace. Without this option, the linker derives the .EXE file’s name from the name of the first source or object file in the file name list. The default extension is .DLL when you are using -WD or -WDE.

-tDe
This specifies that the target (output) file will be a DOS .EXE file.

-tDc
This specifies that the target (output) file will be a DOS .COM file.

-tW[nn]
This specifies that the target (output) file will be a Windows module. It is identical to the -W option(s) described on 162: -W, -WD, -WDE, -WE, -WS, where the optional nn may be equal to D, DE, E or S.

-Ix
This option (which is a lowercase l) passes option x to the linker. The option -l-x suppresses option x. More than one option can appear after the -l.

-M
This option forces the linker to produce a full link map. The default is to produce no link map.

Environment options

When working with environment options, bear in mind that Borland C++ recognizes two types of library files: implicit and user-specified (also known as explicit library files). These are defined and discussed on page 170.

-Ipath
This option (which is an uppercase I) causes the compiler to search path (the drive specifier or path name of a subdirectory) for include files (in addition to searching the standard places). A drive specifier is a single letter, either uppercase or lowercase, followed by a colon (:). A directory
is any valid directory or directory path. You can use more than one -I directory option.

- **Lpath**
  This option forces the linker to get the C0x.OBJ start-up object file and the Borland C++ library files (Cx.LIB, MATHx.LIB, EMU.LIB, and FP87.LIB) from the named directory. By default, the linker looks for them in the current directory.

- **npath**
  This option places any .OBJ or .ASM files created by the compiler in the directory or drive named by *path*.

**Backward compatibility options**

Borland C++ version 3.0 introduces a number of improvements in the way some C++ operations are implemented, resulting in smaller, faster code with fewer restrictions and less overhead. In some cases, the new implementation is not fully compatible with previous versions of Borland C++. Where such compatibility is needed, the following options are provided:

- **-Va**
  When an argument of type class with constructors is passed by value to a function, this option instructs the compiler to create a temporary variable at the calling site, initialize this temporary with the argument value, and pass a reference to this temporary to the function. This behavior is compatible with previous versions of Borland C++. By default, version 3.0 will copy-construct such argument values directly to the stack, thus avoiding the introduction of the temporary (and also making access to the argument value faster).

- **-Vb**
  When a class inherits virtually from a base class, the compiler stores a hidden pointer in the class object to access the virtual base class subobject. The Borland C++ 3.0 compiler makes this pointer always 'near', which allows it to generate more efficient code. For backward compatibility, the -Vb option directs the BC++ 3.0 compiler to match the hidden pointer to the size of the 'this' pointer used by the class itself.

- **-Vc**
  To correctly implement the case when a derived class overrides a virtual function that it inherits from a virtual base class, and a constructor or destructor for the derived
Searching for include and library files

class calls that virtual function using a pointer to the virtual base class, the compiler may add hidden members to the derived class, and add more code to its constructors and destructors. This option directs the compiler not to add the hidden members and code, so that class instance layout is same as with previous versions of Borland C++.

-Vp This option directs the compiler to pass the ‘this’ parameter to ‘pascal’ member functions as the first parameter on the stack, for compatibility with previous versions of Borland C++. By default, version 3.0 always pushes ‘this’ as the last parameter regardless of calling convention.

-Vt This option instructs the compiler to place the virtual table pointer after any non-static data members of the particular class, to ensure compatibility when class instances are to be shared with non-C++ code and when sharing classes with code compiled with previous versions of Borland C++. By default, version 3.0 adds this pointer before any non-static data members of the class, thus making virtual member function calls smaller and faster.

-Vv This option directs the compiler not to change the layout of any classes (which it may need to do in order to allow pointers to virtual base class members, which were not supported in previous versions of Borland C++). If this option is used, the compiler will not be able to create a pointer to a member of a base class that can only be reached from the derived class through two or more levels of virtual inheritance.

-VO This option is a “master switch” that turns on all of the backward-compatibility options listed in this section. It can be used as a handy shortcut when linking with libraries built with older versions of Borland C++.

Borland C++ can search multiple directories for include and library files. This means that the syntax for the library directories (-L) and include directories (-I) command-line options, like that of the #define option (-D), allows multiple listings of a given option.
Here is the syntax for these options:

**Library directories:**  
-Ldirname[;dirname;...]

**Include directories:**  
-Idirname[;dirname;...]

The parameter `dirname` used with `-L` and `-I` can be any directory or directory path.

You can enter these multiple directories on the command line in the following ways:

- You can "gang" multiple entries with a single `-L` or `-I` option, separating ganged entries with a semicolon, like this:
  
  ```
  BCC -Ldirname1;dirname2;dirname3 -Iinc1;inc2;inc3 myfile.c
  ```

- You can place more than one of each option on the command line, like this:
  
  ```
  BCC -Ldirname1 -Ldirname2 -Ldirname3 -Iinc1 -Iinc2 -Iinc3 myfile.c
  ```

- You can mix ganged and multiple listings, like this:
  
  ```
  BCC -Ldirname1;dirname2 -Ldirname3 -Iinc1;inc2 -Iinc3 myfile.c
  ```

If you list multiple `-L` or `-I` options on the command line, the result is cumulative: The compiler searches all the directories listed, in order from left to right.

**Note**

The IDE also supports multiple library directories through the "ganged entry" syntax.

---

**File-search algorithms**

The Borland C++ include-file search algorithms search for the `#include` files listed in your source code in the following way:

- If you put an `#include <somefile.h>` statement in your source code, Borland C++ searches for `somefile.h` only in the specified include directories.

- If, on the other hand, you put an `#include "somefile.h"` statement in your code, Borland C++ searches for `somefile.h` first in the current directory; if it does not find the header file there, it then searches in the include directories specified in the command line.

The library file search algorithms are similar to those for include files:
Your code written under any version of Turbo C or Turbo C++ will work without problems in Borland C++.

- **Implicit libraries:** Borland C++ searches for implicit libraries only in the specified library directories; this is similar to the search algorithm for `#include <somefile.h>`. [Implicit library files are the ones Borland C++ automatically links in. These are the Cx.LIB and CWx.LIB files, EMU.LIB or FP87.LIB, MATHx.LIB, IMPORT.LIB, OVERLAY.LIB, and the start-up object files (C0x.OBJ, C0Wx.OBJ, or C0Dx.OBJ).]

- **Explicit libraries:** Where Borland C++ searches for explicit (user-specified) libraries depends in part on how you list the library file name. (Explicit library files are the ones you list on the command line or in a project file; these are file names with a .LIB extension.)
  
  - If you list an explicit library file name with no drive or directory (like this: mylib.lib), Borland C++ searches for that library in the current directory first. Then (if the first search was unsuccessful), it looks in the specified library directories. This is similar to the search algorithm for `#include "somefile.h"`.
  
  - If you list a user-specified library with drive and/or directory information (like this: c:mystuff\mylib1.lib), Borland C++ searches only in the location you explicitly listed as part of the library path name and not in the specified library directories.

An annotated example

Here is an example of a Borland C++ command line that incorporates multiple library and include directory options.

1. Your current drive is C:, and your current directory is C:\BORLANDC, where BCC.EXE resides. Your A drive's current position is A:\ASTROLIB.

2. Your include files (.h or "header" files) are located in C:\BORLANDC\INCLUDE.

3. Your startup files (C0T.OBJ, C0S.OBJ, ..., C0H.OBJ) are in C:\BORLANDC.

4. Your standard Borland C++ library files (CS.LIB, CM.LIB, ..., MATHS.LIB, MATHM.LIB, ..., EMU.LIB, FP87.LIB, and so forth) are in C:\BORLANDC\LIB.

5. Your custom library files for star systems (which you created and manage with TLIB) are in C:\BORLANDC\STARLIB. One of these libraries is PARX.LIB.
6. Your third-party-generated library files for quasars are in the A drive in \ASTROLIB. One of these libraries is WARP.LIB.

Under this configuration, you enter the following command:

```
BCC -mm -Llib;starlib -Iinclude orion.c umaj.c parx.lib a:\astrolib\warp.l
```

Borland C++ compiles ORION.C and UMAJ.C to .OBJ files, searching C:\BORLANDC\INCLUDE for any #include files in your source code. It then links ORION.OBJ and UMAJ.OBJ with the medium model start-up code (COM.OBJ), the medium model libraries (CM.LIB, MATHM.LIB), the standard floating-point emulation library (EMU.LIB), and the user-specified libraries (PARX.LIB and WARP.LIB), producing an executable file named ORION.EXE.

It searches for the startup code in C:\BORLANDC (then stops because they’re there); it searches for the standard libraries in C:\BORLANDC\LIB (and stops because they’re there).

When it searches for the user-specified library PARX.LIB, the compiler first looks in the current directory, C:\BORLANDC. Not finding the library there, the compiler then searches the library directories in order: first C:\BORLANDC\LIB, then C:\BORLANDC\STARLIB (where it locates PARX.LIB).

Since an explicit path is given for the library WARP.LIB (A:\ASTROLIB\WARP.LIB), the compiler only looks there.
The Optimizer

What is optimization?

An optimizer is a tool for improving your application's speed or shrinking down the application's size. It is not likely that the optimizer will double or triple the speed of your application or cut its size in half. It will allow you to program in the style which you find most convenient, not in the style that your computer finds most convenient.

When should you use the optimizer?

There are several theories as to the best use of the optimizer. One theory is that you should never develop a new program with the optimizer. Instead, you should compile with optimizations when your application is in its final stages of development. This theory is based on the fact that most compilers, when performing full optimizations, take two to three times longer to compile than when they are not performing any optimizations. Borland C++'s optimizer, however, takes only 50% longer to compile when performing full speed optimizations and 20% longer when performing full size optimizations, so you don't have to worry about slow compilation times.

Another theory says that you should always use the optimizer, even in the early stages of development, since the optimizer may reveal bugs in your code that do not appear when it is not optimized. Opponents of this theory argue that debugging such optimized code is a horrendous task not easily undertaken.
Borland C++'s Turbo Debugger understands optimized code and allows you to easily debug your optimized application, giving you the best of both worlds.

Optimization options

The command-line compiler controls code optimizations through the –O command line option. The –O option may be followed by one or more of the suboption letters given in the list below. For example, –Oaxt would turn on all speed optimizations and assume no pointer aliasing. You can turn off optimizations on the command line by placing a minus before the optimization letter. For example, –O2-p would turn on all optimizations except copy propagation. In addition, some optimizations are controlled by means other than –O. For example, –Z controls redundant load suppression.

The optimizations options follow the same rules for precedence as all other Borland C++ options. For example, –Od appearing on the command line after a –O2 would disable all optimizations.

Table A.1: Optimization options summary

<table>
<thead>
<tr>
<th>Command-line</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>–O2 Options</td>
<td>Compiler</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>–O1 Options</td>
<td>Compiler</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>–O Options</td>
<td>Compiler</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>–Oa Options</td>
<td>Compiler</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>–Ob Options</td>
<td>Compiler</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>–Oc Options</td>
<td>Compiler</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>–Od Options</td>
<td>Compiler</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>–Oe Options</td>
<td>Compiler</td>
</tr>
</tbody>
</table>
Table A.1: Optimization options summary (continued)

<table>
<thead>
<tr>
<th>Options</th>
<th>Compiler</th>
<th>Optimizations</th>
<th>Common Subexpressions</th>
<th>Optimize globally</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Og</td>
<td>Options</td>
<td>Compiler</td>
<td>Optimizations</td>
<td>Enables global register allocation and variable live range analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Og Options</td>
<td></td>
<td>Enables common subexpression elimination within an entire function. The -Og option and the -Oc option are mutually exclusive</td>
</tr>
<tr>
<td>-Oi</td>
<td>Options</td>
<td>Compiler</td>
<td>Optimizations</td>
<td>Enables inlining of intrinsic functions such as memcpy, strlen, etc.</td>
</tr>
<tr>
<td>-Ol</td>
<td>Options</td>
<td>Compiler</td>
<td>Optimizations</td>
<td>Loop optimization</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Compacts loops into REP/STOSx instructions</td>
</tr>
<tr>
<td>-Om</td>
<td>Options</td>
<td>Compiler</td>
<td>Optimizations</td>
<td>Invariant code motion</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Moves invariants code out of loops</td>
</tr>
<tr>
<td>-Op</td>
<td>Options</td>
<td>Compiler</td>
<td>Optimizations</td>
<td>Copy propagation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Propagates copies of constants, variables, and expressions where possible</td>
</tr>
<tr>
<td>-Os</td>
<td>Options</td>
<td>Compiler</td>
<td>Optimizations</td>
<td>Optimize for Size</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Makes code selection choices in favor of smaller code</td>
</tr>
<tr>
<td>-Ot</td>
<td>Options</td>
<td>Compiler</td>
<td>Optimizations</td>
<td>Optimize for Speed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Selects code in favor of executable speed</td>
</tr>
<tr>
<td>-Ov</td>
<td>Options</td>
<td>Compiler</td>
<td>Optimizations</td>
<td>Induction Variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enables loop induction variable and strength reduction optimizations</td>
</tr>
<tr>
<td>-Ox</td>
<td>None</td>
<td></td>
<td></td>
<td>Enables most speed optimizations. This is provided for compatibility with Microsoft compilers.</td>
</tr>
<tr>
<td>-Z</td>
<td>Options</td>
<td>Compiler</td>
<td>Optimizations</td>
<td>Suppress redundant loads</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Suppresses reloads of values which are already in registers</td>
</tr>
<tr>
<td>-pr</td>
<td>Options</td>
<td>Compiler</td>
<td>Entry/Exit Code</td>
<td>Calling Convention</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enables the use of the _fastcall calling convention for passing parameters in registers</td>
</tr>
</tbody>
</table>

Backward compatibility

In addition to these new options, all the old code generator options are obeyed. Note, however, that there is some duplication in the new and old options. In particular, -G and -G- are -Os and -Ot. In previous revisions, -Z performed load suppression but was documented as enabling aliasing. The optimizer detects when one register contains two expressions and suppresses extraneous loads of expressions "aliases." Note that this action is not the same as the aliases controlled by -Oa.

For completeness, the old -r (register optimization options) are documented below.

- This option enables the use of register variables (the default).
Unless you are an expert, don't use \(-r-\).

\(-r-\)  This option suppresses the use of register variables. When you are using this option, the compiler won't use register variables, and it won't preserve and respect register variables (SI,DI) from any caller. For that reason, you should not have code that uses register variables call code which has been compiled with \(-r-\).

On the other hand, if you are interfacing with existing assembly-language code that does not preserve SI,DI, the \(-r-\) option allows you to call that code from Borland C++.

\(-rd\)  This option only allows declared register variables to be kept in registers.

---

A closer look at the Borland C++ Optimizer

Global register allocation  Because memory references are so expensive on these processors, it is extremely important to minimize those references through the intelligent use of registers. Global register allocation both increases the speed and and decrease the size of your application. You should always use global register allocation when compiling your application with optimizations on.

Dead code elimination  Although you may never intentionally write code to do things which are unnecessary, the optimizer may reveal possibilities to eliminate stores into variables which are not needed. In the following example, the optimizer creates a new variable to take the place of the expression \(a[j]\), thereby eliminating the need for the variable \(j\). Using \(-Ob\) will remove the code to store any result into variable \(j\).

```c
int goo(void), a[10];
int f(void){
  int i, j;
  j = i = goo();
  for( j = 0; j < 10; j++ )
    a[j] = goo();
  return i;
}
```
Since the optimizer must determine where variables are no longer used and where their values are needed (live range analysis), you must use -Oe before using -Ob. Use -Ob whenever you use -Oe, since -Ob will always result in smaller and faster code.

Common subexpression elimination is the process of finding duplicate expressions within the target scope and storing the calculated value of those expressions once so as to avoid recalculating the expression. Although in theory this optimization could reduce code size, in practice, it is a speed optimization and will only rarely result in size reductions. You should also use global common subexpression analysis if you like to reuse expressions rather than create explicit stack locations for them. For example, rather than code

```c
    temp = t->n.o.left;
    if(temp->op == O_ICON || temp->op == O_FCON)
        ...
```

you could code

```c
    if(t->n.o.left->op == O_ICON || t->n.o.left->op == O_FCON)
        ...
```

and let the optimizer take decide whether it is more efficient to create the temporary.

If you find that global common subexpression elimination is creating too many temporaries for you code size requirements, you can force common subexpression elimination to be done within groups of statements unbroken by jumps (basic blocks) by turning on local common subexpression elimination via the -Oc option on the command line.

Moving invariant code out of loops is a speed optimization. The optimizer uses the information about all the expressions in the function gathered during common subexpression elimination to find expressions whose values do not change inside a loop. To prevent the calculation from being done many times inside the loop, the optimizer moves the code outside the loop so that it is calculated only once. The optimizer then reuses the calculated value inside the loop. For example, in the code below, \( x \times y \times z \) is evaluated in every iteration of the loop.
int v[10];
void f(void){
    int i,x,y,z;
    for (i = 0; i < 10; i++)
        v[i] = x * y * z;
}

The optimizer rewrites the code for the loop so that it looks like:

int v[10];
void f(void){
    int i,x,y,z,tl;
    tl = x * y * z;
    for (i = 0; i < 10; i++)
        v[i] = tl;
}

You should use loop invariant code motion whenever you are compiling for speed and you have used global common subexpressions, since moving code out of loops can result in enormous speed gains.

Copy propagation

Propagating copies is primarily speed optimization, but since it never increases the size of your code, it is safe to use it if you have enabled -Og. Like loop invariant code motion, copy propagation relies on the analysis performed during common subexpression elimination. Copy propagation means that the optimizer remembers the values assigned to expressions and uses those values instead of loading the value of the assigned expressions. Copies of constants, expressions, and variables may be propagated. In the following code, for example, the constant value 5 is used in the second assignment instead of the expression on the right side.

    PtrParIn->IntComp = 5;
    ( *( PtrParIn->PtrComp ) ).IntComp = PtrParIn->IntComp;

Pointer aliasing

Pointer aliasing is not an optimization in itself, but it does affect the way the optimizer performs common subexpression elimination and copy propagation. When pointer aliasing is turned on, it allows the optimizer to maintain copy propagation information across function calls and to maintain common subexpression information across some stores. Otherwise, the optimizer must discard information about copies and subexpressions in these situations. Pointer aliasing might create bugs which are hard to spot, so it is only applied when you use -Oa.
-Oa controls how the optimizer treats expressions with pointers in them. When compiling with global or local common subexpressions and -Oa enabled, the optimizer will recognize

\[ *p \times x \]

as a common subexpression in function foo.

```c
int g, y;
int foo(int *p){
    int x=5;
    y = *p * x;
    g = 3;
    return (*p * x);
}
void goo(void){
g=2;
foo(&g); /* This is incorrect, since the assignment g = 3 invalidates the expression *p * x */
}
```

-Oa also controls how the optimizer treats expressions involving variables whose address has been taken. When compiling with -Oa, the compiler assumes that assignments via pointers will only affect those expressions involving variables whose addresses have been taken and which are of the same type as the left hand side of the assignment in question. To illustrate, consider the following function.

```c
int y, z;
int f(void){
    int x;
    char *p = (char *)&x;
    y = x * z;
    *p = 'a';
    return (x*z);
}
```

When compiled with -Oa, the assignment \(*p = 'a'\) will not prevent the optimizer from treating \(x*z\) as a common subexpression, since the destination of the assignment, \(*p\), is a char, whereas the addressed variable is an int. When compiled without -Oa, the assignment to \(*p\) will prevent the optimizer from creating a common subexpression out of \(x*z\).
Creating induction variables and performing strength reduction are speed optimizations performed on loops. The optimizer uses a mathematical technique called induction to create new variables out of expressions used inside a loop. These variables are called induction variables. The optimizer assures that the operations performed on these new variables are computationally less expensive (reduced in strength) than those used by the original variables.

Opportunities for these optimizations are common if you use array indexing inside loops, since a multiplication operation is required to calculate the position in the array which is indicated by the index. For example, the optimizer would create an induction variable out of the operation \( v[i] \) in the code below, since the \( v[i] \) operation would require a multiplication. This induction variable also eliminates the need to preserve the value of \( i \).

```c
int v[10];
void f(void){
    int i,x,y,z;
    for (i = 0; i < 10; i++)
        v[i] = x * y * z;
}
```

With -Ov enabled, the optimizer would change this code to the following:

```c
int v[10];
void f(void){
    int i,x,y,z, *p;
    for (p = v; p < &v[10]; p++)
        *p = x * y * z;
}
```

You should use -Ov whenever you are compiling for speed and your code contains loops.

Loop compaction takes advantage of the string move instructions on the 80x86 processors by replacing the code for a loop with such an instruction.

```c
int v[100];
void t(void){
    int i;
    for (i = 0; i < 100; i++)
```


v[i] = 0;
}

The optimizer will reduce this to the machine instructions:

```
mov cx,100
mov di,offset DGROUP:_v
push ds
pop es
mov ax,0
rep stosw
```

You should use -Ol to compact loops whenever you are generating code for speed.

Depending on the complexity of the operands, the compacted loop code may also be smaller than the corresponding non-compiled loop. You may wish to experiment with this optimization if you are compiling for size and have loops of this nature.

---

**Code size versus speed optimizations**

You can control the selection and compaction of instructions with the -Ot and the -Os options. These options work like -G and -G- in previous version of Borland C++ but they have been enhanced to do more. Most notable are the structure copy inlining and code compaction optimizations. Whether you use -Ot or -Os depends on what you are trying to achieve with your application.

**Structure copy inlining**

The most visible optimization performed when compiling for speed as opposed to size is that of inlining structure copies. When you enable -Ot, the compiler determines whether it can safely generate code to perform a rep movsw instruction instead of calling a helper function to do the copy. For structures and unions of over 8 bytes in length, performing this optimization produces faster structure copies than the corresponding helper function call.

**Code compaction**

The most visible optimization performed when compiling for size is code compaction. In code compaction, the optimizer scans the generated code for duplicate sequences. When such sequences warrant, the optimizer replaces one sequence of code with a jump to the other, thereby eliminating the first piece of code. SWITCH statements contain the most opportunities code compaction.
Load suppression is both a speed and size optimization. When \texttt{-Z} is enabled, the optimizer keeps track of the values it loads into registers and suppresses loads of values which it already has in a register. For example, when compiling the following code with \texttt{-Z} enabled (and with copy propagation turned off), the optimizer would push the value of \*x it loaded into ES:BX instead of reloading the value \*x.

\begin{verbatim}
void f(void)
{
    int *x = 5;
    goo(*x);
}
\end{verbatim}

You should always use this optimization whenever you are compiling with the optimizer enabled.

There are times when you would like to use one of the common string or memory functions, such as \texttt{strcpy()} or \texttt{memcmp()}, but you do not want to incur the overhead of a function call. By using \texttt{-Oi}, the compiler will generate the code for these functions within your function's scope, eliminating the need for a function call. The resulting code will execute faster than a call to the same function, but it will also be larger.

The following is a list of those functions which are inlined when \texttt{-Oi} is enabled.

\begin{verbatim}
memchr
memcmp
memcpy
memset
strcpy
strcat
strchr
strcmp
strcpy
strlen
strncat
strncpy
strncmp
strrchr
rotl
rotr
fabs
alloca
\end{verbatim}
You can control the inlining of each of these functions with the #pragma intrinsic. For example,

```c
#pragma intrinsic strcpy
```

would cause the compiler to generate code for strcpy in your function.

```c
#pragma intrinsic -strcpy
```

would prevent the compiler from inlining strcpy. Using these pragmas in a file will override the command-line switches or IDE options used to compile that file.

When inlining any intrinsic function, you must include a prototype for that function before you use it. This is because when inlining, the compiler actually creates a macro which renames the inlined function to a function which the compiler internally recognizes. In the above example, the compiler would create a macro

```c
#define strcpy __strcpy__
```

The compiler recognizes calls to functions with two leading and two trailing underscores and tries to match the prototype of that function against its own internally stored prototype. If you did not supply a prototype or the prototype you supplied does not match the compiler's internal prototype, the compiler will reject the attempt to inline that function and will generate an error.

The command line compiler included in the Borland C++ product introduces a new calling convention, called _fastcall. Functions declared using this modifier expect parameters to be passed in registers.

fastcall modifier

The compiler treats this calling convention as a new language specifier, along the lines of _cdecl and _pascal. Functions declared with either of these two languages modifiers cannot also have the _fastcall modifier since they use the stack to pass parameters. Likewise, the _fastcall modifier cannot be used together with _export, _loadds. The compiler generates a warning if you try to mix functions of these types or if you use the _fastcall modifier in a dangerous situation. You may, however, use functions using the _fastcall convention in overlaid modules, i.e. with modules that will use VROOMM.
Parameter rules

The compiler uses the rules given in table A.2 when deciding which parameters are to be passed in registers. A maximum of three parameters may be passed in registers to any one function. You should not assume that the assignment of registers will reflect the ordering of the parameters to a function.

Table A.2
Parameter types and possible registers used

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>character (signed and unsigned)</td>
<td>AL, DL, BL</td>
</tr>
<tr>
<td>integer (signed and unsigned)</td>
<td>AX, DX, BX</td>
</tr>
<tr>
<td>long (signed and unsigned)</td>
<td>DX:AX</td>
</tr>
<tr>
<td>near pointer</td>
<td>AX, DX, BX</td>
</tr>
</tbody>
</table>

Far pointer, union, structure, and floating point (float and double) parameters are pushed on the stack.

Function naming

Functions declared with the _fastcall modifier have different names than their non-_fastcall counterparts. The compiler prefixes the _fastcall function name with an “@”. This prefix applies to both unmangled C function names and to mangled C++ function names.
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>
<thead>
<tr>
<th>Command</th>
<th>Both modes</th>
<th>CUA</th>
<th>Alternate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cursor movement commands</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Character left</td>
<td>←</td>
<td>Ctrl+S</td>
<td></td>
</tr>
<tr>
<td>Character right</td>
<td>→</td>
<td>Ctrl+D</td>
<td></td>
</tr>
<tr>
<td>Word left</td>
<td>Ctrl+←</td>
<td>Ctrl+A</td>
<td></td>
</tr>
<tr>
<td>Word right</td>
<td>Ctrl+→</td>
<td>Ctrl+F</td>
<td></td>
</tr>
<tr>
<td>Line up</td>
<td>↑</td>
<td>Ctrl+E</td>
<td></td>
</tr>
<tr>
<td>Line down</td>
<td>↓</td>
<td>Ctrl+X</td>
<td></td>
</tr>
<tr>
<td>Scroll up one line</td>
<td>Ctrl+W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scroll down one line</td>
<td>Ctrl+Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page up</td>
<td>PgUp</td>
<td></td>
<td>Ctrl+R</td>
</tr>
<tr>
<td>Page down</td>
<td>PgDn</td>
<td></td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Beginning of line</td>
<td>Home</td>
<td></td>
<td></td>
</tr>
<tr>
<td>End</td>
<td>End</td>
<td>Ctrl+Q S</td>
<td></td>
</tr>
<tr>
<td>Top of window</td>
<td>Ctrl+Q E</td>
<td>Ctrl+Q D</td>
<td>Ctrl+Home</td>
</tr>
<tr>
<td>Bottom of window</td>
<td>Ctrl+Q X</td>
<td>Ctrl+X</td>
<td>Ctrl+End</td>
</tr>
<tr>
<td>Top of file</td>
<td>Ctrl+Q R</td>
<td>Ctrl+Home</td>
<td>Ctrl+PgUp</td>
</tr>
<tr>
<td>Bottom of file</td>
<td>Ctrl+Q C</td>
<td>Ctrl+End</td>
<td>Ctrl+PgDn</td>
</tr>
<tr>
<td>Move to previous position</td>
<td>Ctrl+P</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A word is defined as a sequence of characters separated by one of the following: space < > , ; . () {} ^ ` * - / $ # = ! ~ ? " % & : @ \, and all control and graphic characters.
### Table B.1: Editing commands (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Both modes</th>
<th>CUA</th>
<th>Alternate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Insert and delete commands</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete character</td>
<td>Del</td>
<td></td>
<td>Ctrl+G</td>
</tr>
<tr>
<td>Delete character to left</td>
<td>Backspace</td>
<td>Shift+Tab</td>
<td>Ctrl+H</td>
</tr>
<tr>
<td>Delete line</td>
<td>Ctrl+Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete to end of line</td>
<td>Ctrl+Q Y</td>
<td>Shift+Ctrl+Y</td>
<td></td>
</tr>
<tr>
<td>Delete word</td>
<td>Ctrl+T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert line</td>
<td>Ctrl+N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert mode on/off</td>
<td>Ins</td>
<td></td>
<td>Ctrl+V</td>
</tr>
<tr>
<td><strong>Block commands</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move to beginning of block</td>
<td>Ctrl+Q B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move to end of block</td>
<td>Ctrl+Q K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set beginning of block</td>
<td>Ctrl+K B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set end of block</td>
<td>Ctrl+K K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exit to menu bar</td>
<td>Ctrl+K D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hide/Show block</td>
<td>Ctrl+K H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mark line</td>
<td>Ctrl+K L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Print selected block</td>
<td>Ctrl+K P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mark word</td>
<td>Ctrl+K T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete block</td>
<td>Ctrl+K Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copy block</td>
<td>Ctrl+K C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move block</td>
<td>Ctrl+K V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copy to Clipboard</td>
<td>Ctrl+Ins</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cut to Clipboard</td>
<td>Shift+Del</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete block</td>
<td>Ctrl+Del</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indent block</td>
<td>Ctrl+K I</td>
<td>Shift+Ctrl+I</td>
<td></td>
</tr>
<tr>
<td>Paste from Clipboard</td>
<td>Shift+Ins</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read block from disk</td>
<td>Ctrl+K R</td>
<td>Shift+Ctrl+R</td>
<td></td>
</tr>
<tr>
<td>Unindent block</td>
<td>Ctrl+K U</td>
<td>Shift+Ctrl+U</td>
<td></td>
</tr>
<tr>
<td>Write block to disk</td>
<td>Ctrl+K W</td>
<td>Shift+Ctrl+W</td>
<td></td>
</tr>
<tr>
<td><strong>Extending selected blocks</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left one character</td>
<td>Shift+←</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right one character</td>
<td>Shift+→</td>
<td></td>
<td></td>
</tr>
<tr>
<td>End of line</td>
<td>Shift+End</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beginning of line</td>
<td>Shift+Home</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same column on next line</td>
<td>Shift+↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same column on previous line</td>
<td>Shift+↑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One page down</td>
<td>Shift+PgDn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One page up</td>
<td>Shift+PgUp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left one word</td>
<td>Shift+Ctrl+←</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right one word</td>
<td>Shift+Ctrl+→</td>
<td></td>
<td></td>
</tr>
<tr>
<td>End of file</td>
<td>Shift+Ctrl+End</td>
<td></td>
<td>Shift+Ctrl+PgDn</td>
</tr>
<tr>
<td>Beginning of file</td>
<td>Shift+Ctrl+Home</td>
<td></td>
<td>Shift+Ctrl+PgUp</td>
</tr>
</tbody>
</table>
Table B.1: Editing commands (continued)

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

* 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

<table>
<thead>
<tr>
<th>Command</th>
<th>CUA</th>
<th>Alternate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy block</td>
<td>Ctrl+Ins, Shift+Ins</td>
<td>Ctrl+Ins, Shift+Ins</td>
<td>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.</td>
</tr>
<tr>
<td>Copy text</td>
<td>Ctrl+Ins</td>
<td>Ctrl+Ins</td>
<td>Copies selected text to the Clipboard.</td>
</tr>
<tr>
<td>Cut text</td>
<td>Shift+Del</td>
<td>Shift+Del</td>
<td>Cuts selected text to the Clipboard.</td>
</tr>
<tr>
<td>Delete block</td>
<td>Ctrl+Del</td>
<td>Ctrl+Del</td>
<td>Deletes a selected block. You can “undelete” a block with Undo.</td>
</tr>
<tr>
<td>Move block</td>
<td>Shift+Del, Shift+Ins</td>
<td>Shift+Del, Shift+Ins</td>
<td>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.</td>
</tr>
<tr>
<td>Paste from Clipboard</td>
<td>Shift+Ins</td>
<td>Shift+Ins</td>
<td>Pastes the contents of the Clipboard.</td>
</tr>
<tr>
<td>Read block from disk</td>
<td>Shift+Ctrl+R, Ctrl+K R</td>
<td>Ctrl+K R</td>
<td>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.</td>
</tr>
<tr>
<td>Write block to disk</td>
<td>Shift+Ctrl+W, Ctrl+K W</td>
<td>Ctrl+K W</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

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

### 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

<table>
<thead>
<tr>
<th>Command</th>
<th>CUA</th>
<th>Alternate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoindent</td>
<td>Ctrl+O I</td>
<td>Ctrl+O I</td>
<td>Toggles the automatic indenting of successive lines. You can also use Options</td>
</tr>
<tr>
<td>Cursor through tabs</td>
<td>Ctrl+O R</td>
<td>Ctrl+O R</td>
<td>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. Ctrl+O R is a toggle.</td>
</tr>
<tr>
<td>Find place marker</td>
<td>Ctrl+n*</td>
<td>Ctrl+Q n*</td>
<td>Finds up to ten place markers (n can be any number in the range 0 to 9) in text. Move the cursor to any previously set marker by pressing Ctrl+Q and the marker number.</td>
</tr>
<tr>
<td>Open file</td>
<td>F3</td>
<td></td>
<td>Lets you load an existing file into an edit window.</td>
</tr>
<tr>
<td>Optimal fill</td>
<td>Ctrl+O F</td>
<td>Ctrl+O F</td>
<td>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.</td>
</tr>
<tr>
<td>Save file</td>
<td>F2</td>
<td></td>
<td>Saves the file and returns to the editor.</td>
</tr>
<tr>
<td>Command</td>
<td>CUA</td>
<td>Alternate</td>
<td>Function</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Set place</td>
<td>Shift+Ctrl n*</td>
<td>Ctrl+K n*</td>
<td>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.</td>
</tr>
<tr>
<td>Show previous error</td>
<td>Alt+F7</td>
<td>Alt+F7</td>
<td>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.</td>
</tr>
<tr>
<td>Show next error</td>
<td>Alt+F8</td>
<td>Alt+F8</td>
<td>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.</td>
</tr>
<tr>
<td>Tab mode</td>
<td>Ctrl+O T</td>
<td>Ctrl+O T</td>
<td>Toggles Tab mode. You can specify the use of true tab characters in the IDE with the Options</td>
</tr>
<tr>
<td>Unindent</td>
<td>Ctrl+O U</td>
<td>Ctrl+O U</td>
<td>Toggles Unindent. You can turn Unindent on and off from the IDE with the Options</td>
</tr>
</tbody>
</table>

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

EasyWin is an exciting new feature of Borland 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 compile your program with the Windows compiler switch (-W), or select Windows .EXE from the Options | Compiler | Application menu in the IDE. Borland 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:

```c
#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.
Added functions

For your convenience, EasyWin also includes five additional functions that allow you to specify the X and Y window coordinates for input and output, clear the window or clear to the end of the current line. These functions are

```c
gotoxy()
wherex()
wherey()
clrscr()
clreol()
```

These functions have the same names (and uses) as functions in conio.h (see the Library Reference). Classes in constrea.h provide CONIO functionality for use with C++ streams (see Chapter 5, "Using C++ streams," in the Programmer's Guide for a complete discussion).
Precompiled headers

Borland 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, Borland 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 198.

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 (see page 85). The IDE bases the name of the precompiled header file on the project name, creating `PROJECT.SYM`
- from the command line using the `-H`, `-H=filename`, and `-Hu` options (see page 161)
- 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 \-H=filename command-line option or 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 (-mX)
- underscores on externs (-u)
- maximum identifier length (-IL)
- target DOS (default) or Windows (-W or -Wx)
- generate word alignment (-a)
- Pascal calls (-p)
- treat enums as integers (-b)
- default char is unsigned (-K)
- virtual table control (-Vx)
Optimizing precompiled headers

For Borland 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.
43/50-line display 111
<> (angle brackets) in #include directive 110
-2 BCC option (80286 instructions) 151
-1 BCC option (extended 80186 instructions) 151
» (chevron) in dialog boxes 34
~ (tilde) in transfer program names 101
$ editor macros See individual names of macros; transfer macros
+ operator
  overloading See overloaded operators, addition (+)
1's complement See operators, 1's complement
80x87 math coprocessors See numeric coprocessors
80x86 processors
  instruction set 86
  instructions 151
  extended 151
= (System) menu 46
> (arrows) in dialog boxes 34

A
-a BCC option (align integers) 151
-A BCC option (ANSI keywords) 156
About command 125
action symbols See TLIB (librarian)
activating
  menu bar 25
Active File command 125
active window See windows, active
Add button 77
Add Item command 77, 129
Add Watch command 73
  hot key 29
addresses, memory See memory, addresses
Advanced C++ Options
  command 92
Advanced Code Generation
  command 86
  dialog box 86
Advanced Code Generation dialog box 86
After Compiling
  option 103
alignment
  integers 151
  word 85
Alternate command set 40
American National Standards Institute See ANSI
ancestors See classes, base
angle brackets (<> in #include directive 110
ANSI
Borland C++ keywords and 156
C standard 4
compatible code 156
floating point conversion rules 152
keywords
  option 156
  using only 98
violations 157
ANSI Violations 99
applications
Microsoft Windows See Microsoft Windows applications
transferring to and from Borland C++ 100
Arguments
command 63
arguments
command-line compiler 141
passing to Turbo Debugger 64
variable list 154
Arrange Icons command 119
arrays
huge
fast huge pointer arithmetic and 87
arrays, inspecting values 68
arrows (+) in dialog boxes 34
.ASM files See assembly language
assembler
compile via 85
source option 85
assembly language
assembling from the command line 141
compiling 161
default assembler 161
directory 168
inline routines 161
options
passing 162
removing 162
output files 162
projects and 134
assembly level debugger See Turbo Debugger
Assume no pointer aliasing
option 97
Assume SS equals DS option 85
-AT BCC option (Borland C++ keywords) 156
-AU option (UNIX keywords) 156
Auto Save option 112
auto variables See variables, automatic
autodependencies See also dependencies
autoindent mode 187, 189
Autoindent Mode option 113
automatic dependencies 103
checking 133
information
disabling 154
Automatic Far Objects option 88
automatic variables See variables, automatic

B

-b BCC option (allocate whole word for enums) 151
-B BCC option (process inline assembler code) 161
/b IDE option (build) 22
Backspace Unindents option 113
backup files (.BAK) 113
backward
searching 57
Backward compatibility options 168
.BAK files 113
bar
execution See run bar
run See run bar
bar, title 31
base classes See classes, base
BBS segment See also segments
BC and BCC See Borland C++; command-line
compiler; integrated environment
BC.EXE See integrated environment
BCC.EXE See command-line compiler
BCINST See also BCINST menu and command
names
BGI See Borland Graphics Interface
BGIOBJ See The online document UTIL.DOC
binding See C++, binding
block
copy 186, 188
Borland-style 189
cut 188
delete 186, 188
extending 186
hide and show 186
hide/show
Borland-style 189
indent 186
move 186, 188
Borland-style 189
move to beginning of 186
move to end of 186
print 186
read from disk 186, 188
set beginning of 186
Borland-style 189
set end of 186
Borland-style 189
Index

unindent 186
write to disk 186, 188
block commands 187
block operations (editor) See editing, block operations
blocks, text See editing, block operations
Borland
contacting 10
Borland C++ See also C++; integrated environment
C and 154
calling convention 89
exiting 15
implementation data 4
installing 14-18
on laptops 18
keywords
as identifiers 98, 99, 156
Optimizations dialog box 96
project files and 134
quitting 24
starting 15
starting up 22
transferring from 100
Borland Graphics Interface (BGI) See also graphics
EGA palettes and 24
library 106, 107
Borland C++ See also C++; C language;
keywords
boxes See check boxes; dialog boxes; list boxes;
text, boxes
branching See if statements; switch statements
Break Make On
Make dialog box 131
option 103
Breakpoints
command 74
dialog box 74
breakpoints See debugging; watch expressions
clearing 75
controlling 74
deleting 74
editing 75
inline functions and 91
losing 75
saving across sessions 116
setting 74
viewing 74
Browse
menu 79
browser
information
storing 87
Browser Info in OBJs option 87
BSS names 100
bugs See also debugging, See debugging
Build command 65
build IDE option 22
buttons
Change All 58
choosing 34
in dialog boxes 34
mouse 115
ObjectBrowser 79
radio 34

C
C++ 91, See also Borland C++; C language
binding
late See also member functions, virtual
Borland C++ implementation 4
classes See classes
compiling 90
compiling files as 161
constructors See constructors
data members See data members
destructors See destructors
dynamic objects See also objects
formatting See formatting
functions See also member functions
inline
command-line option (-vi) 155
debugging and 91, 155
virtual tables and 164, 165
overloading See overloaded functions
help 124
hierarchies See classes
inheritance See inheritance
inline functions See C++, functions, inline
member functions See member functions
operators See operators, C++; overloaded
operators
polymorphism See polymorphism
streams See streams, C++
structures See structures
templates
  generating 166
types
  reference See reference types
virtual tables See virtual tables
warnings 100, 158
C++ Options
  command 90
dialog box 90
-C BCC option (compile but don't link) 161
-C BCC option (nested comments) 156
C language See also C++
  Borland C++ and 154
  help 124
Call Stack command 71
  hot key 29
callbacks
  smart See smart callbacks
    Windows applications and 89
calling
  conventions 89
calling convention
    _fastcall 183
Cancel button 34
$CAP EDIT macro 102
Cascade command 119
Case-Sensitive Exports option 105
case sensitive option
  librarian 108
case sensitivity
  in searches 56
  linking with 105
    module definition file and 105
case statements See switch statements
cdecl statement 154
.CFG files See configuration files
Change All button 58
Change Dir command 50
Change Directory dialog box 50
characters
  char data type See data types, char
  control
    IDE and 35
data type char See data types, char
delete 186
tab
  printing 51
charts See graphics, charts
Check Auto-dependencies option 103
check boxes 34
chevron symbol (») 34
class arguments
  passing by value 34
class hierarchy
  display of 80
Class Inspector window 69
classes See also structures
  browsing 80
debugging 69
  inspecting 69
  names 100
  sharing objects 92, 165
Clear command 54, 188
  hot key 28, 40
click speed (mouse) 115
Clipboard 53, 187
  clearing 54
copy to 186
cut to 186
  editing text in 55
  paste from 186, 188
  saving across sessions 116
  showing 55
Close All command 120
close boxes 31
Close command 120
  hot key 28
Close Project command 77
closed files listing 52
closing windows 120
Code Generation
  Advanced
    command 86
    command 84
dialog box 84

code-generation
  command-line compiler options 151
Code Pack Size option 106

code segment
  group 160
  names 100
  naming and renaming 159
  storing virtual tables in 92, 164
    -WD option and 162

colors  See graphics, colors

colors and palettes
  EGA 24

Colors dialog box 117

columns
  numbers 30

COMDEFS
  generating 151
  PUBDEFS versus 87

command line
  Borland C++  See command-line compiler
  options  See command-line compiler, options;
  integrated environment, command-line
  options
  viewing from IDE 120, 121

command-line
  specify project file on 22

command-line compiler 142
  arguments 141
  compiling and linking with 141
  configuration files  See configuration files,
  BCC
  directives  See directives
  options 143, 148, 174
    -2 (80286 instructions) 151
    80286 instructions (-2) 151
    -1 (extended 80186 instructions) 151
    -A and -AT (Borland C++ keywords) 156
    _fastcall
    conventions (--pr) 154
    -H (precompiled headers) 161
    -P (C++ and C compilation) 161
    -Wx (Windows applications) 162
    -X (disable autodependency information)
       154
    -Y (overlays) 155
    -Yo (overlays) 155
    -a (align integers) 151
    -AK (Kernighan and Ritchie keywords)
       156
    allocate whole word for enum (-b) 151
    ANSI
      compatible code 156
      keywords (--A) 156
      violations 157
    assembler code 161, 162
    assembler to use (-E) 161
    assume DS = SS (-Fs) 152
    -AU (UNIX keywords) 156
    autodependency information (-X) 154
    -b (allocate whole word for enums) 151
    -B (process inline assembler) 161
    Borland C++ keywords (--A- and --AT) 156
    C++ and C compilation (--P) 161
    C++ inline functions (--vi) 155
    -c (compile and assemble) 161
    -C (nested comments) 156
    code-generation 151
    code segment
      class name 159
        group 160
      .COM file names (-tDc) 167
    comments, nesting (--C) 156
    compilation control 161
    compile and assemble (--c) 161
    configuration files and 143
    -D (macro definitions) 150
    -d (merge literal strings) 151
    data segment
      class name 160
        group 160
      name 159, 160
    debugging information (--v) 155
    #defines 150
      ganging 150
    directory (--n) 168
    .DLLs with all exportables (-WD) 162
    .DLLs with explicit exports (-WDE) 162
    -E (assembler to use) 161
    -e (EXE program name) 167
    emulate 80x87 (-f) 152
    enable --F options (--Fm) 152
    environment 167
error reporting 157
.EXE file names (-e) 167
.EXE file names (-fDe) 167
expanded memory 163
extended 80186 instructions (-I) 151
-f287 (inline 80x87 code) 153
-f87 (inline 80x87 code) 153
-f (emulate 80x87) 152
far global variables (-Ff) 152
far objects (-zE, -zF, and -zH) 159, 160
far virtual table segment
  class name 160
fast floating point (-ff) 87, 152
fast huge pointers (-h) 153
-Fc (generate COMDEFS) 151
-Ff (far global variables) 152
-Fm (enable -F options) 152
frequent errors 158
Fs (assume DS = SS) 152
functions, void 157
-G (speed optimization) 156
generate COMDEFS (-Fc) 151
generate underscores (-u) 154
gn (stop on n warnings) 157
-h (fast huge pointers) 153
identifiers, length (-i) 156
include files 170
  directory (-I) 143, 167
inline 80x87 code (-f87) 153
integer alignment (-a) 151
-jn (stop on n errors) 157
-k (standard stack frame) 153
-K (unsigned characters) 153
Kernighan and Ritchie keywords (-AK) 156
-l (linker options) 167
-L (object code and library directory) 143, 168
libraries 170
  directory (-L) 143, 168
line numbers (-y) 155
link map (-M) 167
linker (-l) 167
-M (link map) 167
macro definitions (-D) 150
memory model (-mx) 149
memver pointers (-V and -VII) 165
merge literal strings (-d) 151
-n (.OBJ and .ASM directory) 168
-N (stack overflow logic) 154
nested comments (-C) 156
object code and library directory (-L) 143, 168
object files (-o) 161
.OBJs with explicit exports (-WE) 162
order of evaluation 148
  response files and 147
overlays (-Y) 155
overlays (-Yo) 155
Pascal
  conventions (-p) 154
  identifiers 154
pass options to assembler (-Tstring) 162
pointer conversion, suspicious 157
portability warnings 158
precedence 148
  response files and 147
precedence rules 143
precompiled headers (-H) 161
process inline assembler (-B) 161
produce .ASM but don’t assemble (-S) 162
project files and 77
-Q (expanded memory) 163
-rd (register variables) 176
register variables 175, 176
remove assembler options (-T-) 162
-S (produce .ASM but don’t assemble) 162
segment-naming control 159
smart callbacks (-WS) 163
speed optimization (-G) 156
stack overflow error message (-N) 154
standard stack frame (-k) 153
stop on n errors(-jn) 157
stop on n warnings (-gn) 157
structures and 157
symbolic debugger 155
syntax 146
-T- (remove assembler options) 162
-Tstring (pass options to assembler) 162
template (-Jg) 166
toggling 143
undefine (-U) 150
underscores (-u) 154
UNIX keywords (-AU) 156
using 142
- v (debugging information) 155
- vi (C++ inline functions) 155
virtual tables (-V and -Vn) 164
warnings (-wxxx) 157-159
warnings (-wxxx) 157
Windows applications (-W) 162
Windows target files (-tW) 167
- y (line numbers) 155
- Z (far virtual table segments) 160
- Z (code and data segments) 159, 160
response files 147
  option precedence 147
syntax 142
Turbo Assembler and 146
using 142
command-line options 22
  build (/b) 22, 40
dual monitors (/d) 22
EGA palette (/p) 24
expanded (/e) 23
extended memory for heap space 24
help (/h) 23
laptops (/l) 23
make (/m) 23, 40
RAM disk (/r) 24
trash control (/s) 24
Turbo C++ IDE 40
command set
  Alternate 26, 40
  Common User Access (CUA) 26, 40
selecting a 26
Command Set option 112
command sets 26, 40
Native option 29, 41
commands  See individual command names,
  See also command-line compiler, options;
  individual command names
choosing
  with a mouse 26
  with keyboard 25
  with SpeedBar 42
editor
  block operations 186, 187-188
cursor movement 185
insert and delete 186
comments
  nested 98, 156
Common subexpressions 97
  optimize globally 97
  optimize locally 98
Common User Access (CUA) command set 26, 40
communal variables 151
compilation 149,  See also compilers
  assembler source output 85
  command-line compiler options 161
  rules governing 146
  speeding up 85
  to .EXE file 64, 65
  to .OBJ file 64
compilation via assembler 85
Compile
  menu 64
Compile command 64
  hot key 29, 41
Compiler
  command 84
compiler directives  See directives
Compiler Messages submenu 99
compilers  See also compilation
  C++ 90
  code optimization 95, 96
  command line  See command-line compiler
collection files  See configuration files
memory models  See memory models
optimizations
  for speed or size 95, 98
  stopping after errors and warnings 99
compiling  See also compilers
Compress debug info option 106
conditional breakpoints  See breakpoints, conditional
configuration files 36
  command-line compiler 143, 147
    creating 148
    overriding 143, 148
    priority rules 148
contents of 36
IDE 36-39
  TCCONFIG.TC 36
saving 118
Turbo C++ for Windows 41
constants
documenting 69
hexadecimal
too large 157
manifest See macros
manifest or symbolic See macros
octal
too large 157
symbolic See macros
constructors See C++, constructors
Container class library 107
Contents command 123
hot key 28
control character
insert 187
control characters
entering in IDE 35
format specifier 72
Control menu 46
hot key 46
conventions
typographic 9
conversion specifications See format specifiers
conversions
floating point
ANSI rules 152
pointers
suspicious 157
specifications See format specifiers
coprocessors See numeric coprocessors
copy and paste See editing, copy and paste
copy block
Borland-style 189
Copy command 54
hot key 28, 40
Copy Example command 55, 123
Copy propagation
option 97
copy protection 13
copy to Clipboard 186
copying, and pasting See editing, copy and paste
copyright information 125
CPP (preprocessor) See The online document
UTILITY.DOC
.CPP files See C++
CPU registers 121
Create Backup Files option 113
creating new files See files, new Ctrl+Break 59, 60
CUA command set 26, 40
CUA option 112
Current window option 112
cursor See also editor, cursor movement
cursor through tabs 187, 189
Cursor Through Tabs option 113
customer assistance 10
customizing See also BCINST
IDE 111
Cut command 54
hot key 28, 40
cut to Clipboard 186
D
-D BCC option (macro definitions) 150
-d BCC option (merge literal strings) 151
/d IDE option (dual monitors) 22
data
aligning 85
hiding See access
structures See also arrays; structures
data members See C++, data members
data segment
group 160
names 100
naming and renaming 159, 160
removing virtual tables from 92, 164
-WD option and 162
data structures See also arrays; structures
data types See also arrays; structures
data types
char
default 85
changing 153
converting See conversions, See conversion
floating point See floating point
integers See integers
Dead code elimination
option 97
Debug Info in OBJS option 87
Trace into command and 62
Debug menu 66
Debugger
command 63
debugger, integrated See integrated debugger
Debugger command 108
Debugger Options
  command 64
Debugger Options dialog box 108
debugging  See also integrated debugger
  arrays 68
  breakpoints  See breakpoints
Browser Info in OBJs 87
call stack 71
classes 69
  constants 69
Debug Info in OBJs 87
dialog box choices 108
display swapping 109
  dual monitors and 109
excluding information 78
expressions 70
format specifiers 71
functions 69
heap size 110
hot keys 29
information 60, 108
  command-line compiler option 155
    in .EXE or OBJ files 155
    storing 87
inspecting values 66
line numbers information 87
pointers 68
stack overflow 90
starting a session 59
Step Over command 62
structures and unions 69
subroutines 90
Trace Into command 61
types 70
  variables 70
  watch expressions  See watch expressions
Windows applications 66
declarations
  data  See data, declaring
.DEF files
  import libraries and 103
default arguments  See arguments, default
default assembler 161
default buttons 34
Default Extension option 114
Default Libraries option 105
#define directive
  command-line compiler options 150
    ganging 150
Defines option 86
delete block 186
delete characters 186
Delete Item
  command 77
Delete Item command 129
delete lines 186
Delete Watch command 73
delete words 186
deleting text
  redoing 53
  undoing 53
dependencies 103
  automatic  See autodependencies
derived classes  See classes, derived
descendants  See classes, derived
desktop
  saving options in 116
desktop files
    contents of 38
    Turbo C++ for Windows 41
desktop files (.DSK)
    default 38
    projects and 38
Desktop option 112
Desktop Preferences dialog box 116
desktop window
  arranging icons in 119
dialog boxes  See also buttons; check boxes; list
    boxes; radio buttons
  arrows in 34
  defined 33
  entering text 35
  Preferences 189
directional delimiters  See delimiters
directives
  MAKE  See MAKE (program manager),
    directives
Directories
  command 110
directories
  .ASM and .OBJ
    command-line option 168
    changing 50
defining 110
include files 143, 167, 169
    example 171
libraries 170
    command-line option 143, 168
    example 171
output 110
project files 38
projects 130
semicolons in paths 111
source 111
disk space
    running out of 197
disks
distribution
    defined 14
display
    formats
        debugger 71
        repainting 46
        swapping 109
        dual monitors and 109
Display Warnings
    option 99
Display Warnings option 99
displays  See screens
distribution disks 5
    backing up 13
distributions disks, defined 14
division  See floating point, division; integers, division
DLLs  See also import libraries
    creating 89, 162
    import libraries and 103
    linker and 103
    MAKE and 103
    packing code segments 105
    setting 105
do while loops  See loops, do while
DOS
    output
        viewing from IDE 120, 121
        shelling to
            TSRs and 51
            wildcards 48
DOS MODE command 22
DOS Overlay command 89
DOS Shell command 24, 51
DOS Standard command 89
double (floating point)  See floating point, double
double-click speed (mouse) 115
DPMI
    use of extended and expanded memory 17
DPMIINST
    protected mode and 15, 142
DPMIMEM environment variable 16
DPMIRES protected mode utility 16
DS register (data segment pointer) 85
.DSK files
    default 38
    projects and 38
dual monitor mode 22, 23
dual monitors 22
    display swapping and 109
    DOS command line and 51
duplicate, strings, merging 85
Duplicate Strings Merged option 85
duplicate symbols 105
dynamic binding  See C++, binding, late
dynamic link libraries  See DLLs
dynamic objects  See objects, dynamic

E
-E BCC option (assembler to use) 161
-e BCC option (EXE program name) 167
/e IDE option (expanded memory) 23
eye early binding  See C++, binding
Edit  See also editing
    menu 52
    windows
        loading files into 132
Edit Watch command 73
Edit windows
    option settings 113
edit windows
cursor
    moving 185
editing  See also Edit,  See also editor; text
    block operations 186, 187-188
    deleting 188
    deleting text 114
    marking 114
    overwrite 114
Index
in response files 147
examples
  copying from Help 55, 123
library and include directories 171
.EXE files
  creating 29, 64, 65
directory 110
linking 65
making 27, 41
naming 64
user-selected name for 167
executable files  See .EXE files
execution
  bar  See run bar
Exit
  command 52
Exit command
  hot key 40
exit the IDE 187
exiting Borland C++ 24
expanded memory 17, See extended and
  expanded memory
  controlling use of 163
  IDE option 23
explicit
  library files 167
_export (keyword)
  Windows applications and 89, 163
exports
  case sensitive 105
expressions
  debugging 70
evaluating
    restrictions on 70
values
  displaying 70
extended 80186 instructions 151
extended and expanded memory
  RAM disk and 24
extended dictionary option
  librarian 108
extended memory 17, 18
  IDE and 24
extensibility  See also C++
extension keywords
  ANSI and 156
External option
  C++ Virtual Tables
    command-line option 164
  C++ Virtual tables 91
extraction operator (>>)  See overloaded
  operators
F
  –f287 option (inline 80x87 code) 153
  –f87 option (inline 80x87 code) 153
  -f BCC option (emulate 80x87) 152
  _fastcall
    calling convention 183
    command-line option 154
far
  variables 152
Far Data Threshold type-in box 88
far objects  See objects, far
Far option
  C++ Virtual tables 92
far virtual table segment
  naming and renaming 160
Fast Floating Point option 87
fast huge pointers 153
Fast Huge Pointers option 87
Fastest Code option 98
fatal errors  See errors
  –Fc BCC option (generate COMDEFS) 151
features
  IDE 21
features of Borland C++ 1
  –FF BCC option (far global variables) 152
  –ff option (fast floating point) 87, 152
file
  open 187, 189
  save 187, 189
File menu 47
files  See also individual file-name extensions
  assembly language  See assembly language
  backup (.BAK) 113
  batch  See batch files
  C++  See C++
  .CCP  See C++
closed
  reopening 122
compiling as C++ or C 161
configuration 36,  See configuration files
Index
inspecting 69
listing of 80
locating 59
member See member functions
ordinary member See member functions, ordinary
overloaded See overloaded functions
parameters See parameters, See arguments
searching for 59
stepping over 62
tracing into 61
virtual See member functions, virtual
void
returning a value 157
Windows 162

G
-G BCC option (speed optimization) 156
ganging
command-line compiler options
#define 150
macro definition 150
defined 150, 170
IDE 170
library and include files 170
Generate COMDEFS option 87
Generate Underbars option 87
get from (>>) See overloaded operators
global declarations See declarations, global
global menus See menus
Global register allocation
option 96
global variables
word-aligning 151
-gn BCC option (stop on n warnings) 157
Go Cursor command 61
Go to Cursor command
hot key 27, 29
Go to Line Number
command 59
Go to source
ObjectBrowser
hot key 79
Goto
ObjectBrowser 81
graphics See also graphics drivers
graphics drivers See also graphics
Graphics Library option 106, 107
GREP See The online document UTIL.DOC
GREP (file searcher)
wildcards in the IDE 56
group names 100
Group Undo option 113
Undo and Redo commands and 53

H
-h BCC option (fast huge pointers) 153
-H BCC option (precompiled headers) 161
/h IDE option (list options) 23
hardware
requirements
mouse 4
requirements to run Borland C++ 4
hdrfile pragma 196, 197
hdrstop pragma 196, 198
header files See also include files
help 124
precompiled See also precompiled headers
searching for 170
variables and 87
windows.h See windows.h
heap
size 110
heap space
extended memory for 24
Help
button 34
menu 122
ObjectBrowser
hot key 79
topic search 187
windows
closing 123
copying from 55, 123
keywords in 123
opening 122
selecting text in 123
help 187
accessing 122
active file 125
C and C++ 124
help on help 124
hot keys 27, 28, 41
IDE 23
index 124
keywords 123
language 124
previous topic 124
status line 33
table of contents 123
help index 187
Help on Help command 124
HELPME!.DOC file 14, 19
hexadecimal numbers See numbers, hexadecimal
hierarchies See classes
history lists 35
closing 120
saving across sessions 116
wildcards and 48
hot keys 45
debugging 29
editing 28, 40
help 27, 28, 41
make project 132
menus 26, 27
next error 132
previous error 132
transfer macros 102
transfer program names 101
using 26
huge pointers 153

-i BCC option (identifier length) 156
-I BCC option (include files directory) 143, 167
icons
arranging 119
icons used in books 9
IDE 21, See integrated environment
command-line options 22
dual monitors (/d) 22
EGA palette (/p) 24
expanded memory (/e) 23
help (/h) 23
laptops (/l) 23
make (/m) 23
/p EGA palette 24
RAM disk (/r) 24
syntax 22
trash control (/s) 24
control characters and 35
starting up 22
IDE features 21
identifiers
Borland C++ keywords as 98, 99, 156
duplicate 105
length 99
Pascal-type 154
significant length of 151, 156
undeфинing 150
underscore for 154
$IMPLIB See also import libraries
IMPLIB program See also import libraries
implicit
library files 167
import libraries See also DLLs
DLLs and 103
generating 103
#include directive See also include files
angled brackets and 170
directories 110
quotes and 170
Include Directories
input box 110
Include Files
command 78
include files See also header files
command-line compiler options 170
directories 143, 167, 169
multiple 171
help 124
projects 129
searching for 170
user-specified 143, 167
Include Files command 129
Include Files dialog box 78
incremental search 36
indent block 186
indenting automatically 113
Index command
hot key 28, 41
Index command (help) 124
indexes See arrays
Induction variables
option 96
Information command 65
information hiding See access
initialization  See specific type of initialization
initialized data segment  See data segment
inline assembly code 161
inline code  See assembly language, inline	routines; 80x87 math coprocessor
inline functions, C++  See C++, functions, inline
Inline intrinsic functions
   option 97
input boxes 35
insert lines 186
insert mode 186
Insert Mode option 113
insertion operator (<<)  See overloaded
   operators
Inspect
command 66
   ObjectBrowser 81
      hot key 79
Inspect command
      hot key 28, 29
inspecting symbols
   with ObjectBrowser 81
Inspector windows 66
   arrays 68
class 69
   classes 69
constant 69
   function 69
ordinal 67
   pointers 68
   structures and unions 69
   Type 70
installation 14-18
   on a laptop system 18
instances  See classes, instantiation and
   instantiation  See classes, instantiation and
Instruction Set radio buttons 86
integers  See also floating point; numbers
   aligned on word boundary 151
   assigning to enumeration 157
integrated debugger  See debugging,  See also
debugging
   breakpoints  See breakpoints
debugging information for 155
integrated development environment  See
   integrated environment
integrated environment
   command-line arguments and 63
configuration files  See configuration files,
   integrated environment
customizing 19, 111
debugging  See debugging
editing  See editing
ganging 170
makes 133
menus  See menus
multiple library directories 170
settings
   saving 118
intrinsic functions
   inline
      option 97
Invariant code motion
      option 96
I/O
   C++  See C++, I/O
J
   -jn BCC option (stop on n errors) 157
Jump Optimization
      option 95
Jump optimization
      option 97
K
   -k BCC option (standard stack frame) 153
   -K BCC option (unsigned characters) 153
K&R  See Kernighan and Ritchie
Keep Messages command
      toggle 133
Kernighan and Ritchie
   keywords 99, 156
keyboard
   choosing buttons with 34
   choosing commands with 25
   selecting text with 52
keys, hot  See hot keys
keywords
   ANSI
      command 156
Borland C++ 98
   using, as identifiers 156
help 124
Help windows 123
Kernighan and Ritchie
  using 156
options 98
register
  Register Variables option and 95
UNIX
  using 156

L
-1 BCC option (linker options) 167
-1 BCC option (object code and library directory) 143, 168
/l IDE option (LCD screen) 23
language help 124
laptop computers
  installing Borland C++ onto 18
laptops
  IDE option (/l) 23
late binding See C++, binding
LCD displays
  installing Borland C++ for 18
LCD screens 23
left-handed
  mouse support for 115
Less Frequent Errors dialog box 100
.LIB files See libraries
librarian See TLIB
  case sensitive option 108
  dialog box choices 107
  extended dictionary option 108
  list file option 107
  purge comments option 108
Librarian command 107
Librarian Options dialog box 107
libraries
  command-line compiler options 170
  container class 107
  default 105
directories 110, 169
  command-line option 143, 168
  multiple 171
dynamic link (DLL) See DLLs
explicit and implicit 167
files 110, 143, 168
floating point 152

Index

graphics 106, 107
import See import libraries
linking 65
overriding in projects 136
rebuilding 154
routines
  80x87 floating-point emulation 153
searching for 170
user-specified 167
utility See TLIB
library
  ObjectWindows 107
Library Directories
    input box 110
library files See libraries
license statement 13
line
  mark a 186
line numbers See lines, numbering
Line Numbers Debug Info option 87
lines
  delete 186
  filling with tabs and spaces 113
  insert 186
  moving cursor to 59
  numbering 30
    in object files 155
    information for debugging 87
    restoring (in editor) 53
Link command 65
link map, full 167
Linker
  command 104
  dialog box 104, 106
linker See also TLINK
  case sensitive linking 105
command-line compiler options 167
DLLs and 103
link map
  creating 167
  options
    from command-line compiler 167
Linker option
  container class library 107
linking
  excluding from 78
list boxes 36
file names 48
searching incrementally 124
List command
hot key 28
list file option
librarian 107
List window 122
literal strings See strings, literal
local menus See menus
Local option
C++ Virtual tables 91
Local Options
C++ Virtual Tables
command-line option 164
command 77, 129
Locate Function
command 59
long double (floating point) See floating point,
long double
long integers See integers, long
Loop Optimizations
option 96
.LST files See files; listfile (TLIB option)

M
-M BCC option (link map) 167
/m IDE option (make) 23
macros See also editor, macros; MAKE
(program manager), macros
command-line compiler 150
ganging 150
MAKE See MAKE (program manager),
micros
preprocessor 86
transfer See transfer macros
Turbo editor See The online document
UTIL.DOC
make
IDE option 23
MAKE (program manager)
After compiling, 103
DLLs and 103
explicit rules See MAKE (program manager),
rules
implicit rules See MAKE (program manager),
rules
integrated environment makes and 133
stopping makes 103, 131
Make command 64, 103
hot key 27, 29
makefiles See MAKE (program manager)
manifest constants See macros
manipulators See also formatting, C++;
individual manipulator names
manus
using 8
map files 167
directory 110
options 105
marker
find 187, 189
set 187, 190
math coprocessors See numeric coprocessors
maximize See zooming, See Zoom command
member functions See also C++, functions; data
members
virtual See also C++, binding, late
member pointers
controlling 165
members
data See data members
functions See member functions
memory
dump
format specifier 72
expanded 17
controlling 163
IDE and 23
extended 17
extended and expanded See extended and
expanded memory
RAM disk and 24
heap size 110
protected mode and 15
memory models
automatic far data and 88
changing 84
command-line options 85, 149
smart callbacks and 163
menu bar See also menus
activating 25
menu commands
choosing 26
with SpeedBar 42
dimmed 26
grayed 26
unavailable 26
menus See also individual menu names
   accessing 25
   commands See individual command names
hot keys 26, 27
opening 25
reference 45
   with an ellipsis (…) 25, 33
   with arrows (↑) 25
Message Tracking
toggle 132
Message window 120, 133
   copying text from 54
   removing messages 66
messages See errors; warnings, See also errors; warnings
   appending 112
   removing 66
Messages command 99
methods See member functions
mice See mouse
Microsoft Windows See also Microsoft Windows applications
   resources See resources
Microsoft Windows All Functions Exportable command 89
Microsoft Windows applications See also
   Microsoft Windows
code segments 105
   command-line compiler options 162, 163
debugging 66
   export functions and 89, 162
IDE options 89
   optimizing for 96, 98
   prolog and epilog code 88
   setting application type 105
   setting options for 82, 88
   smart callbacks and 89, 163
Microsoft Windows DLL All Functions Exportable command 89
Microsoft Windows DLL Explicit Functions Exported command 89
Microsoft Windows Explicit Functions Exported command 89
Microsoft Windows Smart Callbacks command 89
MODE command (DOS) 22
models, memory See memory models
modularity See encapsulation
module definition files
   exported functions and 89
   EXPORTS section
      case-sensitive 105
   IMPORTS section
      case-sensitive 105
monitors See also screens
dual 22, 51, 109
   number of lines 111
mouse
   buttons
      switching 115
   choosing commands with 26, 34
   compatibility 4
   double-click speed 115
   left-handed
      support for 115
   options 114
   reversing buttons 115
   right button
      browse with 79
   right button action 115
   selecting text with 52
   support for 21
mouse buttons
   right and left 26
Mouse Double Click option 115
Mouse Options dialog box 114
moving text See editing, moving text; editing, block operations
multi-source programs See projects
Multiple Document Interface (MDI) 39
multiple files See projects
multiple inheritance See inheritance
multiple listings
   command-line compiler options
      #define 150
      include and library 170
      macro definition 150
   -mx options (memory models) 149
N
–n BCC option (.OBJ and .ASM directory) 168
–N BCC option (stack overflow logic) 154
Names
command 100
names See identifiers
Native command set option 29, 41
nested
comments 156
delimiters See delimiters
Nested Comments option 98
New command 47
New Value field 70
New Window option 112
Next command 120
hot key 27, 28
next error
show 190
Next Error command 59
hot key 29, 41
No-Nonsense License Statement 13
NAME file name 47
nondirectional delimiters See delimiters
nonfatal errors See errors
null character See characters, null
numbers See also floating point; integers
decimal 110
format specifier 72
hexadecimal 110
constants
too large 157
format specifier 72
octal
constants
too large 157
real See floating point
numeric coprocessors See also floating point
emulating 152
generating code for 152, 153
inline instructions 86, 153
O
–o BCC option (object files) 161
.OBJ files
browser information 87
compiling 161
creating 64
debugging information 87
dependencies 103
directories 110, 168
line numbers in 155
object files See .OBJ files
object-oriented programming See C++
ObjectBrowser
choosing commands in 79
ObjectBrowser buttons 79
objects See also C++
far
class names 160
generating 88
group names 160
segment names 159
ObjectWindows library option 107
OBJXREF See The online document UTIL.DOC
citional numbers See numbers, octal
OK button 34
one’s complement See operators, 1’s
complement
online help See help
OOP See C++
Open a File dialog box 47, 189
Open command 47, 189
hot key 27, 28, 40
open file 187, 189
Open Project
command 76
opening a file 47
operators
associativity See associativity
C++ See also overloaded operators
delete See delete (operator)
get from (>>) See overloaded operators
new See new (operator)
put to (<<) See overloaded operators
one’s complement See operators, 1’s
complement
overloading See overloaded operators
precedence See precedence
Optimal Fill option 113, 187, 189
Optimization
what is 173
Optimizations
command 94
optimizations 95, 96
  command-line compiler options 156
Common subexpressions 97
fast floating point 87
Fastest Code 98
for speed or size 95, 98
No Optimizing 98
precompiled headers 198
register variables 97
  usage 175
Smallest Code 98
Windows applications and 95, 98
Optimizations dialog box
  Turbo C++ for Windows 95
  Turbo C++ for Windows 96
option
  Compress debug info 106
Options
  backward compatibility 168
  C++ template generation
    command-line option 166
options See integrated environment, See
  specific entries (such as command-line
  compiler, options)
Options menu 81
  settings
    saving 118
ordinals, inspecting 67
ordinary member functions See member
  functions, ordinary
Out-Line Inline Functions option 91
output
  to DOS
    viewing from IDE 120, 121
User Screen 121
Output command 120
Output Directory
  input box 110
Output window
  copying text from 54
overlays
  generating 155
  projects and 78
  supporting 89
Override Options dialog box 77

Overview
  ObjectBrowser 81
    hot key 79
overview
  in ObjectBrowser 79
Overwrite Blocks option 114
Overwrite Mode 113

P
  -P BCC option (C++ and C compilation) 161
  -p BCC option (Pascal conventions) 154
  /p IDE option (EGA palette) 24
  -pr BCC option (fastcall calling convention) 154
Pack Code Segments option 105
pair matching 187
parameter-passing sequence
  _fastcall 154
parameter-passing sequence, Pascal 154
parameter types
  register usage and 184
parameters See arguments
Pascal
  calling convention 89
  identifiers of type 154
  parameter-passing sequence 154
Paste command 54
  hot key 28, 40
paste from Clipboard 186, 188
pasting See editing, copy and paste
path names in Directories dialog box 111
Persistent Blocks option 114
place marker
  find 187, 189
  set 187, 190
plasma displays
  installing Borland C++ for 18
pointers
  fast huge 87, 153
  format specifier 72
  inspecting values 68
  memory regions 72
  to self  See this (keyword)
  suspicious conversion 157
  virtual table
    32-bit 92, 164
      -WD option and 162
polymorphism See C++
pop-up menus  See also  menus
portability warnings  99, 158
#pragma hdrfile  196, 197
#pragma hdrstop  196, 198
precedence
  command-line compiler options  143, 148
  response files and  147
precompiled headers  195-199
  command-line options  161
  controlling  196
  drawbacks  196
  how they work  195
  inline member functions and  196
  optimizing use of  198
  rules for  197
  using
    IDE  85
Preferences dialog box  189
preprocessor directives  See  directives
previous error
  show  190
Previous Error command  59
  hot key  29, 41
Previous Topic command  124
  hot key  28
Print command  51
printer
  setting up  51
printer drivers  51
Printer Setup command  51
PRJ2MAK  See The online document  UTIL.DOC
  .PRJ files  See projects
PRJCFG  See The online document  UTIL.DOC
PRJCNVT  See The online document  UTIL.DOC
  procedures  See functions
program manager (MAKE)  See MAKE
  (program manager)
Program Reset command  61
  hot key  29
Programmer's Platform  See integrated
  environment
programming
  with classes  See  C++
programs
  C++  See  C++
  ending  59
  heap size  110
multi-source  See  projects
rebuilding  60, 65
resetting  61
running  59
  arguments for  63
  to cursor  61
  Trace Into  61
transfer
  list  135
  transferring to external from  Borland  C++  100
Project
  command  122
  menu  76
project files  36
  contents of  37
  Turbo  C++ for Windows  41
Project Manager  59,  See also  projects
  closing projects  77
  Include files and  78
Project Name
  command  102
Project Notes command  122
Project Notes window  139
Project option  112
projects  See also  Project Manager
  autodependency checking  103
  speeding up  104
automatic dependency checking and  133
building  127
changing  38
closing  77
default  38
desktop files and  38, 36-39
directories  130
directory  38
error tracking  131, 132
excluding from  78
  .EXE file names and  64
files
  adding  129
  command-line options and  77
deleting  129
  include  129
  information  134
  list  129
  options  129
out of date 133
viewing 139
IDE configuration files and 37
include files 129
information in 127
libraries and
overriding 136
loading 37
makes and 133
making
hot key for 132
managing 122
meaning of 76
naming 128
new 129
notes 122, 139
opening 37
overlays and 78
saving 130
translator option 78
translators See also Transfer
default 134
example 135
multiple 134
specifying 135
prolog and epilog code
generating 88
protected mode 15
command-line compiler 142
DPMIMEM variable 16
DPMIRE5 utility 16
pseudovariables, register
using as identifiers 156
PUBDEFs
COMDEFs versus 87
Public option
C++ Virtual Tables
command-line option 164
C++ Virtual tables 91
pull-down menus See menus
purge comments option
librarian 108
put to (<<) See overloaded operators
put to operator (<<) See overloaded operators

Q

–Q BCC options (expanded memory) 163

Quit
command 24
quitting Borland C++ 52
quitting Turbo C++ 52

R

–r BCC option (register variables) 175
/rx IDE option (RAM disk) 24
radio buttons 34
RAM disk
IDE and 24
random numbers See numbers, random
.RC files See also Resource Compiler
–rd option (register variables) 176
read block 186
README 18
README.DOC 14
real numbers See floating point
rebuilding libraries 154
Redo command 53
Group Undo and 53, 113
hot key 28, 41
register (keyword)
Register Variables option and 95
Register command 121
Register keyword
option 97
Register Optimization option 95
register usage and parameter types 184
register variable optimization 97
Register Variables option 95
registers
allocating 96
Automatic
option 97
DS (data segment pointer) 85
None
option 97
pseudovariables
using as identifiers 156
reusing 95
SS (stack segment pointer) 85
variables
suppressed 175
toggle 175
windows 121
relational operators See operators, relational

Index
Remove All Watches command 74
Remove Messages command 66, 133
Repaint Desktop command 46
Replace
    command 58
Replace dialog box
    settings
        saving 118
Replace Text
    dialog box 58
replacing a file 47
.RES files See also resources
resetting programs 61
resize corner 31
Resource Compiler See also .RC files
resources See also .RES files
response files
    defined 147
    option precedence 147
Result field 70
Reverse Mouse Buttons option 115
Rewind
    ObjectBrowser 81
        hot key 79
Right Mouse Button option 115
Ritchie, Dennis See Kernighan and Ritchie
Run
    command 59
    menu 59
Run command
    hot key 29, 41
running programs 59
scope See also variables
Screen Size
    option 111
screens
    LCD
        IDE option 23
            installing Borland C++ for 18
        number of lines 111
        plasma
            installing Borland C++ for 18
        repainting 46
        two
            using 22
    scroll bars 31, 32
    scrolling windows 32
Search Again command 58
    hot key 28, 41
search and replace See also searching
search for text 187
Search menu 56
searching
    direction 57
    error and warning messages 59
    functions 59
    in list boxes 124
    include files 170
    libraries 170
    origin 57
    regular expressions 56
    repeating 58
    and replacing text 58
    scope of 57
    search and replace 58
Segment Alignment option 106
segment-naming control
    command-line compiler options 159
segments
    aligning 106
    code
        minimizing 105
        packing 105
    controlling 159
    initializing 105
    names 100
selecting text 187
self See this (keyword)
semicolons (;) in directory path names 111
Set Application Options dialog box 82
shortcuts See hot keys
keyboard 45
Show Clipboard command 55
Size/Move command 119
Smallest Code option 98
smart callbacks
memory models and 163
Windows applications and 163
Smart option
  C++ Virtual Tables
    command-line option 164, 165
  C++ Virtual tables 91
software See programs
software license agreement 13
software requirements to run Borland C++ 4
Source
  command 98
Source Debugging command 60
  and Trace Into command 62
Source Directory
  input box 111
source files
  .ASM
    command-line compiler and 141
directory 111
  multiple See projects
source-level debugger See Turbo Debugger
Source Options dialog box 98
Source Tracking option 112
Source Tracking options 132
spaces vs. tabs 113
speed
  optimization 156
SpeedBar 42
  configuring the 42
spreadsheets See Turbo Calc
SS register (stack segment pointer) 85
stack
  Call Stack command 71
  overflow 90, 154
  standard frame
    generating 153
  warnings 105
standalone debugging information 108
standalone librarian
  case sensitive 108
  extended dictionary 108
list file 107
  purge comments 108
standalone utilities See also MAKE (program
  manager); TLIB (librarian); TLINK (linker);
  TOUCH
standard library files See libraries
Standard stack frame
  option 97
standard stack frame
  generating 153
Standard Stack Frame command 90
Standard Stack Frame option 71
start-up and exit
  command-line compiler 142
  IDE 22
Startup Preferences dialog box 116
statements See break statements; if statements;
  switch statements
static binding See C++, binding, early
status line 33
staux, functions of See streams
stdin, functions of See streams
stdout, functions of See streams
Step Over command 62
  hot key 27, 29
sterr, functions of See streams
stprn, functions of See streams
streams
  C++
    manipulators and See manipulators
strings
  duplicate
    merging 85
  format specifier 72
    literal
      merging 151
structures
  ANSI violations 157
  C++ See also classes
    format specifier 72
    inspecting 69
    undefined 157
    zero length 157
Suppress redundant loads
  option 97
swapping displays 109
switch statements
  break See break statements
switch to another program 51
switches See command-line compiler, options;
  integrated environment, options
.SYM files 195, 196
  default names 196
  disk space and 197
  smaller than expected 197
symbolic
  constants See macros
destructor See Turbo Debugger
symbolic constants See macros
symbols
  action See TLIB
duplicate 105
syntax
  errors
    project files 131, 132
    IDE command line 22
System menu
  hot key 46
System menu := 25
system requirements 4

T
  --T-- BCC option (remove assembler options) 162
'this' pointer in 'pascal' member functions 169
Tab mode 190
Tab Size option 114
tables, virtual See virtual tables
tabs
  characters
    printing 51
  size of 114
  spaces vs. 113
  using in the editor 113
Tabs mode 187
TASM See Turbo Assembler
TCCONFIG.TC See configuration files,
  integrated environment
TCDEF.DPR files 38
TCDEF.DSK files 38
TCDEF.SYM 161, 195, 196, See also .SYM files
Turbo C++ for Windows
  Optimizations dialog box 95
technical support 10
TEML See The online document UTIL.DOC
Template Generation option 91
templates
  generation 166
terminate and stay resident See TSR programs
Test Stack Overflow command 90
text See also editing
  blocks See editing, block operations
copy and paste 54
cutting 54
deleting 54
entering
  in dialog boxes 35
inserting vs. overwriting 113
pasting 54
restoring (in editor) 53
screen display of 111
selecting 52
  Help window 123
text files See also editing
  THELP See The online document UTIL.DOC
thresh control
  IDE and 24
threshold size
  far global variables
    setting 152
thunks See smart callbacks
tilde (~) in transfer program names 101
Tile command 119
  hot key 28
title bars 31
Toggle Breakpoint command 74
  hot key 29
Topic Search command 124
  hot key 28, 41
Topic search in Help 187
Trace Into command 61
  Debug Info in OBJs option and 62
  hot key 27, 29
  Source Debugging command and 62
Transfer See also projects, translators
  command 25, 100
dialog box 101
  projects and 135
programs 46
   editing 101
transfer macros 102
defined 102
hot keys for 102
how expanded 102
transfer programs
   list 135
transfer to another program 51
Translator option 78, 102
translators See projects, translators
Treat enums as ints option 84
TRIGRAPh See The online document
   UTIL.DOC
TSR programs
   shelling to DOS and 51
-TSTRING BCC option (pass string to assembler) 162
Turbo Assembler
   Borland C++ command-line compiler and 146
   command-line compiler and 141
default 161
   invoking 146
 Turbo C++ for Windows IDE 39
 Turbo Debugger
described 155
 Turbo Debugger for Windows 63
   Windows applications and 66
 Turbo Editor Macro Language compiler See
   The online document UTIL.DOC
TURBOC.CFG 147
25-line display 111
typefaces used in these books 9
types See data types
default 70
typographic conventions 9

U
-U BCC option (undefine) 150
-u BCC option (underscores) 154
unary operators See operators, unary
unconditional breakpoints See breakpoints
underbars See underscores
underscores 154
generating automatically 87, 154
undo 187

Undo command 53
   Group Undo and 53, 113
   hot key 28, 40
unindent
   block 186
   mode 187, 190
uninitialized data segment See data segment unions
   format specifier 72
   inspecting 69
UNIX
   keywords 99
   using 156
   porting Borland C++ files to 157
Unsigned Characters option 85
Use Tab Character option 113
User Screen
   hot key 28
User Screen command 121
user-specified library files 167
utilities See also The online document
   UTIL.DOC

V
-V and -Vn BCC options (C++ virtual tables)
   164
-v BCC option (debugging information) 155
-Va BCC option (class argument compatibility)
   168
-Vb BCC option (virtual base class pointer
   compatibility) 168
-Vc BCC option (derived class with pointer to
   inherited virtual base class member function)
   168
-Vm BCC options (C++ member pointers) 165
-Vp BCC option ('this' pointer in 'pascal'
   member functions compatibility) 169
-Vt BCC option (virtual table pointers) 169
-Vv BCC option (pointers to virtual base class
   members) 169
variable
   inspecting a 80
variable argument list 154
variables See also scope
   automatic
   word-aligning 151
communal 151
debugging 70
global
  far 152
header files and 87
inspecting values of 67
list of 80
  register 95, 175
version number information 125
-vi option (C++ inline functions) 155
Video Graphics Array Adapter (VGA) 112
virtual access See also C++
virtual base class
  hidden pointer to 168
virtual base class members
  pointers to 169
virtual functions See member functions, virtual
  hidden members in derived classes with
  pointers to 168
virtual table pointers
  compatibility 169
virtual tables 91
  32-bit pointers and 92, 164
    -WD option and 162
  controlling 164
  storing in the code segment 92, 164
    -WD option and 162
visibility See scope

W
-W BCC options (Windows applications) 162
-wxxx BCC options (warnings) 157
Warnings
  Stop After 99
warnings See also errors
  ANSI Violations 99
  C++ 100, 158
  command-line options 157-159
  enabling and disabling 157
  frequent errors 100, 158
  messages 7
  options 157-159
  portability 99, 158
watch expressions See also debugging
  adding 73
  controlling 73
  deleting 73, 74
size
  hot key 28
source tracking 112
swapping in debug mode 109
dual monitors and 109
tiling 119
title bar 31
User Screen 121
Watch 121
window number 31
zooming 31, 33, 119
Windows (Microsoft) See Microsoft Windows
word
  delete 186
  mark 186
  word aligning
    integers 151
Word Alignment option 85
write block 186
-WS BCC options (smart callbacks) 163
-wxxx BCC option (warnings) 157
-wxxx BCC options (warnings) 157-159
X
  -X BCC option (disable autodependency information) 154
  /x IDE option (extended memory) 24
Y
  -y BCC option (line numbers) 155
  -Y BCC option (overlays) 155
  -Yo BCC option (overlays) 155
Z
  zoom box 31
  Zoom command 119
    hot key 27, 28
-2V options (far virtual table segments) 160
-2X options (code and data segments) 159, 160