

IBM XL C/C++ for Linux, V13.1.1



Getting Started with XL C/C++ for Little Endian Distributions

Version 13.1.1

IBM XL C/C++ for Linux, V13.1.1



Getting Started with XL C/C++ for Little Endian Distributions

Version 13.1.1

Note

Before using this information and the product it supports, read the information in “Notices” on page 25.

First edition

This edition applies to IBM XL C/C++ for Linux, V13.1.1 (Program 5765-J08; 5725-C73) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

© Copyright IBM Corporation 1996, 2014.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this document	v
Conventions	v
Related information	viii
IBM XL C/C++ information	viii
Standards and specifications	ix
Other IBM information	x
Other information	x
Technical support.	x
How to send your comments.	x

Chapter 1. Introducing XL C/C++	1
Commonality with other IBM compilers	1
Operating system and hardware support	1
A highly configurable compiler	2
Language standard compliance	3
Compatibility with GNU	3
Source-code migration and conformance checking	4
Libraries.	4
Tools, utilities, and commands	5
Advance Toolchain 8.0 support	6
Program optimization	6
Diagnostic listings	6
Symbolic debugger support	7

Chapter 2. Migrating to IBM XL C/C++ for Linux, V13.1.1.	9
Migrating from big endian Linux to little endian Linux.	9
Compiler options.	9
Supported GCC options	9

-q options not available	13
Template model	15
Changes in predefined macro support	15
Compiler pragmas	15

Chapter 3. Setting up and customizing XL C/C++.	17
Using custom compiler configuration files	17

Chapter 4. Developing applications with XL C/C++	19
The compiler phases	19
Editing C/C++ source files	19
Compiling with XL C/C++	20
Invoking the compiler	20
Specifying compiler options	20
XL C/C++ input and output files	21
Linking your compiled applications with XL C/C++	22
Dynamic and static linking	22
Running your compiled application	22
XL C/C++ compiler diagnostic aids	23
Debugging compiled applications	23
Determining which level of XL C/C++ is being used.	24

Notices	25
Trademarks and service marks	27

Index	29
------------------------	-----------

About this document

This document contains overview and basic usage information for the IBM® XL C/C++ for Linux, V13.1.1 compiler.

Who should read this document

This document is intended for C and C++ developers who are looking for introductory overview and usage information for XL C/C++. It assumes that you have some familiarity with command-line compilers, a basic knowledge of the C and C++ programming languages, and basic knowledge of operating system commands. Programmers new to XL C/C++ can use this document to find information on the capabilities and features unique to XL C/C++.

How to use this document

Unless indicated otherwise, all of the text in this reference pertains to both C and C++ languages. Where there are differences between languages, these are indicated through qualifying text and icons, as described in “Conventions.”

Throughout this document, the `xlc` and `xlc++` compiler invocations are used to describe the actions of the compiler. You can, however, substitute other forms of the compiler invocation command if your particular environment requires it, and compiler option usage will remain the same unless otherwise specified.

While this document covers information on configuring the compiler environment, and compiling and linking C or C++ applications using the XL C/C++ compiler, it does not include the following topics:

- Compiler installation: see the *XL C/C++ Installation Guide* for information on installing XL C/C++.
- Compiler options: see the *XL C/C++ Compiler Reference* for detailed information on the syntax and usage of compiler options.
- The C or C++ programming languages: see the *XL C/C++ Language Reference* for information on the syntax, semantics, and IBM implementation of the C or C++ programming languages.
- Programming topics: see the *XL C/C++ Optimization and Programming Guide* for detailed information on developing applications with XL C/C++, with a focus on program portability and optimization.

Conventions

Typographical conventions

The following table shows the typographical conventions used in the IBM XL C/C++ for Linux, V13.1.1 information.

Table 1. *Typographical conventions*

Typeface	Indicates	Example
bold	Lowercase commands, executable names, compiler options, and directives.	The compiler provides basic invocation commands, xlc and xlc (xlc++), along with several other compiler invocation commands to support various C/C++ language levels and compilation environments.
<i>italics</i>	Parameters or variables whose actual names or values are to be supplied by the user. Italics are also used to introduce new terms.	Make sure that you update the <i>size</i> parameter if you return more than the <i>size</i> requested.
<u>underlining</u>	The default setting of a parameter of a compiler option or directive.	nomaf <u>maf</u>
monospace	Programming keywords and library functions, compiler builtins, examples of program code, command strings, or user-defined names.	To compile and optimize myprogram.c, enter: xlc myprogram.c -O3.

Qualifying elements (icons)

Most features described in this information apply to both C and C++ languages. In descriptions of language elements where a feature is exclusive to one language, or where functionality differs between languages, this information uses icons to delineate segments of text as follows:

Table 2. *Qualifying elements*

















Qualifier/Icon	Meaning
C only, or C only begins   C only ends	The text describes a feature that is supported in the C language only; or describes behavior that is specific to the C language.
C++ only, or C++ only begins   C++ only ends	The text describes a feature that is supported in the C++ language only; or describes behavior that is specific to the C++ language.
IBM extension, or IBM extension begins   IBM extension ends	The text describes a feature that is an IBM extension to the standard language specifications.

Table 2. Qualifying elements (continued)


Qualifier/Icon	Meaning
C11, or C11 begins   C11 ends	The text describes a feature that is introduced into standard C as part of C11.
C++11, or C++11 begins   C++11 ends	The text describes a feature that is introduced into standard C++ as part of C++11.

Syntax diagrams


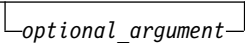
Throughout this information, diagrams illustrate XL C/C++ syntax. This section will help you to interpret and use those diagrams.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.
The  symbol indicates the beginning of a command, directive, or statement.
The  symbol indicates that the command, directive, or statement syntax is continued on the next line.
The  symbol indicates that a command, directive, or statement is continued from the previous line.
The  symbol indicates the end of a command, directive, or statement.
Fragments, which are diagrams of syntactical units other than complete commands, directives, or statements, start with the  symbol and end with the  symbol.


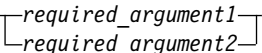
- Required items are shown on the horizontal line (the main path):

—keyword—required_argument—————>

- Optional items are shown below the main path:

—keyword——————>

- If you can choose from two or more items, they are shown vertically, in a stack. If you *must* choose one of the items, one item of the stack is shown on the main path.

—keyword——————>

If choosing one of the items is optional, the entire stack is shown below the main path.



- An arrow returning to the left above the main line (a repeat arrow) indicates that you can make more than one choice from the stacked items or repeat an item. The separator character, if it is other than a blank, is also indicated:



- The item that is the default is shown above the main path.



- Keywords are shown in nonitalic letters and should be entered exactly as shown.
- Variables are shown in italicized lowercase letters. They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Examples in this information

The examples in this information, except where otherwise noted, are coded in a simple style that does not try to conserve storage, check for errors, achieve fast performance, or demonstrate all possible methods to achieve a specific result.

The examples for installation information are labelled as either *Example* or *Basic example*. *Basic examples* are intended to document a procedure as it would be performed during a basic, or default, installation; these need little or no modification.

Related information

The following sections provide related information for XL C/C++:

IBM XL C/C++ information

XL C/C++ provides product information in the following formats:

- README files
README files contain late-breaking information, including changes and corrections to the product information. README files are located by default in the XL C/C++ directory, and in the root directory and subdirectories of the installation DVD.
- Installable man pages
Man pages are provided for the compiler invocations and all command-line utilities provided with the product. Instructions for installing and accessing the man pages are provided in the *IBM XL C/C++ for Linux, V13.1.1 Installation Guide*.
- Online product documentation

The fully searchable HTML-based documentation is viewable in IBM Knowledge Center at http://www.ibm.com/support/knowledgecenter/SSXVZZ_13.1.1/com.ibm.compilers.linux.doc/welcome.html.

- PDF documents

PDF documents are available on the web at <http://www.ibm.com/support/docview.wss?uid=swg27036675>.

The following files comprise the full set of XL C/C++ product information:

Table 3. XL C/C++ PDF files

Document title	PDF file name	Description
<i>IBM XL C/C++ for Linux, V13.1.1 Installation Guide, GC27-6540-00</i>	install.pdf	Contains information for installing XL C/C++ and configuring your environment for basic compilation and program execution.
<i>Getting Started with IBM XL C/C++ for Linux, V13.1.1, GI13-2875-00</i>	getstart.pdf	Contains an introduction to the XL C/C++ product, with information on setting up and configuring your environment, compiling and linking programs, and troubleshooting compilation errors.
<i>IBM XL C/C++ for Linux, V13.1.1 Compiler Reference, SC27-6570-00</i>	compiler.pdf	Contains information about the various compiler options, pragmas, macros, environment variables, and built-in functions.
<i>IBM XL C/C++ for Linux, V13.1.1 Language Reference, SC27-6550-00</i>	langref.pdf	Contains information about language extensions for portability and conformance to nonproprietary standards.
<i>IBM XL C/C++ for Linux, V13.1.1 Optimization and Programming Guide, SC27-6560-00</i>	proguide.pdf	Contains information on advanced programming topics, such as application porting, interlanguage calls with Fortran code, library development, application optimization, and the XL C/C++ high-performance libraries.

To read a PDF file, use Adobe Reader. If you do not have Adobe Reader, you can download it (subject to license terms) from the Adobe website at <http://www.adobe.com>.

More information related to XL C/C++ including IBM Redbooks® publications, white papers, tutorials, documentation errata, and other articles, is available on the web at:

<http://www.ibm.com/support/docview.wss?uid=swg27036675>

For more information about boosting performance, productivity, and portability, see the C/C++ café at <https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=5894415f-be62-4bc0-81c5-3956e82276f3>.

Standards and specifications

XL C/C++ is designed to support the following standards and specifications. You can refer to these standards for precise definitions of some of the features found in this information.

- *Information Technology - Programming languages - C, ISO/IEC 9899:1990*, also known as C89.
- *Information Technology - Programming languages - C, ISO/IEC 9899:1999*, also known as C99.

- *Information Technology - Programming languages - C, ISO/IEC 9899:2011*, also known as C11. (Partial support)
- *Information Technology - Programming languages - C++, ISO/IEC 14882:1998*, also known as C++98.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2003*, also known as *Standard C++*.
- *Information Technology - Programming languages - C++, ISO/IEC 14882:2011*, also known as C++11. (Partial support)
- *Information Technology - Programming languages - Extensions for the programming language C to support new character data types, ISO/IEC DTR 19769*. This draft technical report has been accepted by the C standards committee, and is available at <http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1040.pdf>
- *Altivec Technology Programming Interface Manual*, Motorola Inc. This specification for vector data types, to support vector processing technology, is available at http://www.freescale.com/files/32bit/doc/ref_manual/ALTIVECPIM.pdf.
- *ANSI/IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985*.

Other IBM information

- *ESSL product documentation* available at http://www.ibm.com/support/knowledgecenter/SSFHY8/essl_welcome.html

Other information

- *Using the GNU Compiler Collection* available at <http://gcc.gnu.org/onlinedocs>

Technical support

Additional technical support is available from the XL C/C++ Support page at http://www.ibm.com/support/entry/portal/overview/software/rational/xl_c~c++_for_linux. This page provides a portal with search capabilities to a large selection of Technotes and other support information.

If you cannot find what you need, you can send email to compinfo@cn.ibm.com.

For the latest information about XL C/C++, visit the product information site at <http://www.ibm.com/software/products/us/en/xlcpp-linux/>.

How to send your comments

Your feedback is important in helping to provide accurate and high-quality information. If you have any comments about this information or any other XL C/C++ information, send your comments by email to compinfo@cn.ibm.com.

Be sure to include the name of the manual, the part number of the manual, the version of XL C/C++, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

Chapter 1. Introducing XL C/C++

IBM XL C/C++ for Linux, V13.1.1 is an advanced, high-performance compiler that can be used for developing complex, computationally intensive programs, including interlanguage calls with C and Fortran programs.

This section contains information about the features of the XL C/C++ compiler at a high level. It is intended for people who are evaluating the compiler, and for new users who want to find out more about the product.

Commonality with other IBM compilers

IBM XL C/C++ for Linux, V13.1.1 is part of a larger family of IBM C, C++, and Fortran compilers. XL C/C++, together with XL Fortran, comprises the family of XL compilers.

These compilers are derived from a common code base that shares compiler function and optimization technologies for a variety of platforms and programming languages. Programming environments include IBM AIX®, IBM Blue Gene®/P, IBM Blue Gene®/Q, IBM i, selected Linux distributions, IBM z/OS®, and IBM z/VM®. The common code base, along with compliance with international programming language standards, helps support consistent compiler performance and ease of program portability across multiple operating systems and hardware platforms.

In addition, IBM XL C/C++ for Linux, V13.1.1 leverages the Clang infrastructure from the open source community for a portion of its compiler front end. Clang is a component of the LLVM open source compiler and toolchain project and provides the C and C++ language family front end for LLVM. XL C/C++ combines the Clang front-end infrastructure with the advanced optimization technology from IBM. For additional information about Clang, see the LLVM web site at: <http://clang.llvm.org/>

Operating system and hardware support

This section describes the operating systems and hardware that IBM XL C/C++ for Linux, V13.1.1 supports.

IBM XL C/C++ for Linux, V13.1.1 supports the following operating systems:

- Ubuntu Server V14.04
- Ubuntu Server V14.10
- SUSE Linux Enterprise Server 12

See the README file and "Before installing XL C/C++" in the *XL C/C++ Installation Guide* for a complete list of requirements.

The compiler, its libraries, and its generated object programs run on any IBM Power Systems™ server supported by your operating system distribution with the required software and disk space.

To exploit the various supported hardware configurations, the compiler provides options to tune the performance of applications according to the hardware type that runs the compiled applications.

A highly configurable compiler

You can use a variety of compiler invocation commands and options to tailor the compiler to your unique compilation requirements.

Compiler invocation commands

XL C/C++ provides several commands to invoke the compiler, for example, `xlC`, `xlC++`, and `xlcl`. Compiler invocation commands are provided to support all standardized C/C++ language levels, and many popular language extensions as well. These invocation commands allow for threadsafe compilation, and you can use them to link the programs that use multithreading.

For more information about XL C/C++ compiler invocation commands, see "Invoking the compiler" in the *XL C/C++ Compiler Reference*.

Compiler options

You can choose from a large selection of compiler options to control compiler behavior. You can benefit from using different options for the following tasks:

- Debugging your applications
- Optimizing and tune application performance
- Selecting language levels and extensions for compatibility with nonstandard features and behaviors that are supported by other C or C++ compilers
- Performing many other common tasks that would otherwise require changing the source code

You can specify compiler options through a combination of environment variables, compiler configuration files, command line options, and compiler directive statements embedded in your program source.

For more information about XL C/C++ compiler options, see "Compiler options reference" in the *XL C/C++ Compiler Reference*.

Custom compiler configuration files

The installation process creates a default compiler configuration file containing stanzas that define compiler option default settings.

If you frequently specify compiler option settings other than the default settings of XL C/C++, you can use makefiles to define your settings. Alternatively, you can create custom configuration files to define your own frequently used option settings.

For more information about using custom compiler configuration files, see "Using custom compiler configuration files" on page 17.

Language standard compliance

IBM XL C/C++ for Linux, V13.1.1 supports the following C/C++ programming language specifications.

C language specifications

- Partial support for ISO/IEC 9899:2011 (referred to as C11)
- ISO/IEC 9899:1999 (referred to as C99)
- ISO/IEC 9899:1990 (referred to as C89)

C++ language specifications

- Partial support for ISO/IEC 14882:2011 (referred to as C++11)
- ISO/IEC 14882:2003 (referred to as Standard C++)
- ISO/IEC 14882:1998, the first official specification of the language (referred to as C++98)

In addition to the standard language levels, XL C/C++ supports the following language extensions:

- Language extensions to support vector programming
- A subset of GNU C and C++ language extensions

See "Language levels and language extensions" in the *XL C/C++ Language Reference* for more information about C/C++ language specifications and extensions.

Compatibility with GNU

XL C/C++ supports a subset of the GNU compiler command options to facilitate porting applications that are developed with the **gcc** and **g++** compilers.

In addition, IBM XL C/C++ for Linux, V13.1.1 leverages the Clang infrastructure from the open source community for a portion of its compiler front end. Clang is a component of the LLVM open source compiler and toolchain project and provides the C and C++ language family front end for LLVM. For additional information about Clang, see the LLVM web site at: <http://clang.llvm.org/>.

XL C/C++ uses GNU C and GNU C++ header files together with the GNU C and C++ runtime libraries to produce code that is binary-compatible with that produced by GCC. Portions of an application can be built with XL C/C++ and combined with portions built with GCC to produce an application that behaves as if it had been built solely with GCC.

To achieve binary compatibility with GCC-compiled code, a program compiled with XL C/C++ includes the same headers as those used by a GNU compiler residing on the same system. To ensure that the proper versions of headers and runtime libraries are present on the system, the prerequisite GCC compiler must be installed before installing XL C/C++.

Some additional noteworthy points about this relationship are:

- IBM built-in functions coexist with GNU C built-ins.
- Compilation of C and C++ programs uses the GNU C and GNU C++ header files.
- Compilation uses the GNU assembler for assembler input files.
- Compiled C code is linked to the GNU C runtime libraries.

- Compiled C++ code is linked to the GNU C and GNU C++ runtime libraries.
- Code compiled with XL C/C++ can be debugged with the GNU debugger, **`gdb`**.

Source-code migration and conformance checking

XL C/C++ provides compiler invocation commands that instruct the compiler to compile your application code to a specific language level.

You can also use the **`-qlanglvl`** or the **`-std`** compiler option to specify a language level. If the language or language extension elements in your program source do not conform to the specified language level, the compiler issues diagnostic messages.

See `-qlanglvl` in the *XL C/C++ Compiler Reference* for more information.

Libraries

XL C/C++ includes a runtime environment containing a number of libraries.

Mathematical Acceleration Subsystem library

The Mathematical Acceleration Subsystem (MASS) library consists of scalar and vector mathematical built-in functions tuned specifically for optimum performance on supported processor architectures. You can choose a MASS library to support high-performance computing on a broad range of processors, or you can select a library tuned to support a specific processor family.

The MASS library functions offer improved performance over the default `libm` math library routines. These libraries are threadsafe, and are called automatically when you request specific levels of optimization for your application. You can also make explicit calls to MASS library functions, regardless of whether optimization options are in effect or not.

For more information, see "Using the Mathematical Acceleration Subsystem" in the *XL C/C++ Optimization and Programming Guide*.

Basic Linear Algebra Subprograms

The Basic Linear Algebra Subprograms (BLAS) set of high-performance algebraic functions are shipped in the `libxlopt` library. You can use these functions to:

- Compute the matrix-vector product for a general matrix or its transpose.
- Perform combined matrix multiplication and addition for general matrices or their transposes.

For more information about using the BLAS functions, see "Using the Basic Linear Algebra Subprograms" in the *XL C/C++ Optimization and Programming Guide*.

Other libraries

The following libraries are also shipped with XL C/C++:

- XL C++ Runtime Library contains support routines needed by the compiler.

Support for Boost libraries

IBM XL C/C++ for Linux, V13.1.1 provides partial support for the Boost V1.55.0 libraries. A patch file is available that modifies the Boost V1.55.0 libraries so that they can be built and used with XL C/C++ applications. The patch or modification file does not extend nor provide additional functionality to the Boost libraries.

To access the patch file for building the Boost libraries, go to Boost Library Regression Test Summaries and select download required Boost modification file for your compiler release and platform.

You can download the latest Boost libraries at <http://www.boost.org/>.

For more information on support for libraries, search on the XL C/C++ Compiler support page at http://www.ibm.com/support/entry/portal/overview/software/rational/xl_c~c++_for_linux.

Tools, utilities, and commands

This topic introduces the main tools, utilities, and commands that are included with XL C/C++. It does not contain all compiler tools, utilities, and commands.

Utilities

install The **install** utility installs and configures IBM XL C/C++ for Linux, V13.1.1 for use on your system.

xl_configure

You can use the **xl_configure** utility to facilitate the use of XL C/C++ with IBM Advance Toolchain. For details, see "Using IBM XL C/C++ for Linux, V13.1.1 with the Advance Toolchain" in *XL C/C++ Compiler Reference*.

Commands

Profile-directed feedback (PDF) related commands

cleanpdf command

The **cleanpdf** command removes all the PDF files or the specified PDF files from the directory to which profile-directed feedback data is written.

mergepdf command

The **mergepdf** command provides the ability to weigh the importance of two or more PDF records when combining them into a single record. The PDF records must be derived from the same executable.

resetpdf command

The current behavior of the **resetpdf** command is the same as the **cleanpdf** command, and is retained for compatibility with earlier releases on other platforms.

showpdf command

The **showpdf** command displays the following types of profiling information for all the procedures executed in a PDF run (compilation under the **-qpdf1** option):

- Block-counter profiling
- Call-counter profiling
- Value profiling

- Cache-miss profiling, if you specified the `-qpdf1=level=2` option during the `-qpdf1` phase.

You can view the first two types of profiling information in either text or XML format. However, you can view value profiling and cache-miss profiling information only in XML format.

For more information, see `-qpdf1`, `-qpdf2` in the *XL C/C++ Compiler Reference*.

Advance Toolchain 8.0 support

IBM XL C/C++ for Linux, V13.1.1 fully supports IBM Advance Toolchain 8.0, which is a set of open source development tools and runtime libraries. With IBM Advance Toolchain, you can take advantage of the latest POWER® hardware features on Linux, especially the tuned libraries.

For more information, see "Using IBM XL C/C++ for Linux, V13.1.1 with the Advance Toolchain" in the *XL C/C++ Compiler Reference*.

Program optimization

XL C/C++ provides several compiler options that can help you control the optimization and performance of your programs.

With these options, you can perform the following tasks:

- Select different levels of compiler optimizations.
- Control optimizations for loops, floating point, and other types of operations.
- Optimize a program for a particular class of machines or for a very specific machine configuration, depending on where the program will run.

Optimizing transformations can give your application better overall execution performance. XL C/C++ provides a portfolio of optimizing transformations tailored to various supported hardware. These transformations offer the following benefits:

- Reducing the number of instructions executed for critical operations
- Restructuring generated object code to make optimal use of the Power Architecture
- Improving the usage of the memory subsystem

For more information, see these related topics:

- "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*
- "Optimizing and tuning options" in the *XL C/C++ Compiler Reference*
- "Compiler built-in functions" in the *XL C/C++ Compiler Reference*

Diagnostic listings

The compiler output listings provide important information to help you develop and debug your applications more efficiently.

For more information about the applicable compiler options and the listing itself, see "Compiler messages and listings" in the *XL C/C++ Compiler Reference*.

Symbolic debugger support

You can instruct XL C/C++ to include debugging information in your compiled objects by using different levels of the **-g** compiler option.

For details, see **-g** in *XL C/C++ Compiler Reference*.

The debugging information can be examined by **gdb** or any other symbolic debugger to help you debug your programs.

Chapter 2. Migrating to IBM XL C/C++ for Linux, V13.1.1

This section describes features of IBM XL C/C++ for Linux, V13.1.1 that you should consider when moving your application that was compiled with versions of XL C/C++ on other platforms.

Migrating from big endian Linux to little endian Linux

There are some factors that you should consider when moving your applications from POWER8® big endian systems.

- To help migrate programs from big endian systems, you can use the **-qaltivec=be** or **-qaltivec=le** option to toggle the vector element sequence in registers to big endian or little endian element order.

Related information



Program migration from big-endian systems



-maltivec (-qaltivec)

Compiler options

IBM XL C/C++ for Linux, V13.1.1 introduces support for many of the options supported by GCC.

If you have makefiles for programs that were previously compiled with versions of XL C/C++ on other platforms, you should also note that some of the options are not available.

Supported GCC options

The following GCC options are supported in IBM XL C/C++ for Linux, V13.1.1. For details about these options, see the GNU Compiler Collection online documentation at <http://gcc.gnu.org/onlinedocs/>.

- @file
- -###
- --help
- --sysroot
- --version
- -ansi
- -dD
- -dM
- -fansi-escape-codes
- -fasm, -fno-asm
- -fcolor-diagnostics
- -fcommon, -fno-common
- -fconstexpr-depth
- -fexceptions
- -ffast-math

- -fdiagnostic-parsable-fixits
- -fdiagnostics-fixit-info
- -fdiagnostics-format=[clang | msvc | vi]
- -fdiagnostic-show-category=[none | id | name]
- -fdiagnostic-show-template-tree
- -fdiagnostics-print-source-range-info
- -fdiagnostics-show-name
- -fdiagnostics-show-option
- -fdollars-in-identifiers, -fno-dollars-in-identifiers
- -fdump-class-hierarchy
- -ffreestanding
- -fgnu89-inline
- -fhosted
- -finline-functions
- -fmessage-length
- -fno-access-control
- -fno-assume-sane-operator-new
- -fno-builtin
- -fno-diagnostics-show-caret
- -fno-diagnostics-show-option
- -fno-elide-type
- -fno-gnu-keywords
- -fno-operator-names
- -fno-rtti
- -fno-show-column
- -fpack-struct
- -fpermissive
- -fPIC, -fno-PIC
- -fPIE, -fno-PIE
- -fshort-enums
- -fshort-wchar
- -fshow-column
- -fshow-source-location
- -fsigned-bitfields, -fno-signed-bitfields
- -fsigned-char, -fno-signed-char
- -fstrict-aliasing
- -fsyntax-only
- -ftabstop=*width*
- -ftemplate-backtrace-limit
- -ftemplate-depth
- -ftime-report
- -ftls-model, -fno-tls-model
- -ftrap-function=*name*
- -ftrapping-math, -fnotrapping-math
- -funsigned-bitfields, -fno-unsigned-bitfields

- -funsigned-char, -fno-unsigned-char
- -funroll-all-loops
- -funroll-loops
- -fvisibility
- -idirafter
- -imacros
- -include
- -iprefix
- -iquote
- -isysroot
- -isystem
- -iwithprefix
- -M
- -MD
- -MF
- -MG
- -MM
- -MMD
- -MP
- -MQ
- -MT
- -maltivec, -mno-altivec
- -mcpu
- -mtune
- -nodefaultlibs
- -nostartfiles
- -nostdinc
- -nostdinc++
- -Ofast
- -pedantic
- -pedantic-errors
- -pie
- -rdynamic
- -shared
- -shared-libgcc
- -static
- -static-libgcc
- -std
- -trigraphs
- -w
- -Wall
- -Wambiguous-member-template
- -Wbad-function-cast
- -Wbind-to-temporary-copy
- -Wc++11-compat

- -Wcast-align
- -Wchar-subscripts
- -Wcomment
- -Wconversion
- -Wdelete-non-virtual-dtor
- -Wempty-body
- -Wenum-compare
- -Werror
- -Werror=foo [specically, -Werror=unused-command-line-argument to switch between warning/error for invalid options]
- -Weverything
- -Wextra-tokens
- -Wfatal-errors
- -Wfloat-equal
- -Wfoo
- -Wformat-nonliteral
- -Wformat-security
- -Wformat-y2k
- -Wignored-qualifiers
- -Wimplicit
- -Wimplicit-function-declaration
- -Wimplicit-int
- -Wmain
- -Wmissing-braces
- -Wmissing-field-initializers
- -Wmissing-prototypes
- -Wnarrowing
- -Wno-attributes
- -Wno-builtin-macro-redefined
- -Wno-deprecated
- -Wno-deprecated-declarations
- -Wno-division-by-zero
- -Wno-endif-labels
- -Wno-format
- -Wno-format-extra-args
- -Wno-format-zero-length
- -Wno-int-conversion
- -Wno-int-to-pointer-cast
- -Wno-invalid-offsetof
- -Wno-multichar
- -Wno-unused-result
- -Wno-return-local-addr
- -Wno-virtual-move-assign
- -Wnon-virtual-dtor
- -Wnonnull

- -Woverlength-strings
- -Woverloaded-virtual
- -Wpadded
- -Wparantheses
- -Wpedantic
- -Wpointer-arith
- -Wpointer-sign
- -Wreorder
- -Wreturn-type
- -Wsequence-point
- -Wshadow
- -Wsign-compare
- -Wsign-conversion
- -Wsizeof-pointer-memaccess
- -Wswitch
- -Wsystem-headers
- -Wtautological-compare
- -Wtrigraphs
- -Wtype-limits
- -Wundef
- -Wuninitialized
- -Wunknown-pragmas
- -Wunused
- -Wunused-label
- -Wunused-parameter
- -Wunused-value
- -Wunused-variable
- -Wvarargs
- -Wvariadic-macros
- -Wvla
- -Wwrite-strings
- -x
- -X

-q options not available

The following -q options are not available for use with IBM XL C/C++ for Linux, V13.1.1.

- -qabi_version
- -qalloca
- -qassert
- -qattr
- -qcinc
- -qcomplexgccincl
- -qcplucmt
- -qdbfmt

- -qdbxextra
- -qdigraph
- -qenum
- -qfdpr
- -qflag
- -qformat
- -qfuncsect
- -qgenproto
- -qhaltonmsg
- -qidirfirst
- -qignpragma
- -qinfo
- -qkeepinlines
- -qkeyword
- -qldbl28
- -qlibansi
- -qlibmpi
- -qlistfmt
- -qlistopt
- -qlonglit
- -qlonglong
- -qmaxerr
- -qmbcs, -qdbcs
- -qminimaltoc
- -qoptdebug
- -qpack_semantic
- -qppline
- -qprint
- -qproto
- -qproclocal, -procimported
- -qpr
- -qrestrict
- -qshowinc
- -qskipsrc
- -qsmp
- -qsource
- -qsrcmsg
- -qstackprotect
- -qstatsym
- -qsupress
- -qsyntab
- -qtabsize
- -qtempinc
- -qtemplaterecompile
- -qtemplateregistry

- -qtempmax
- -qthreaded
- -qtmplparse
- -qtrigraphs
- -qupconv
- -qutf
- -qvrsave
- -qwarn0x
- -qwarn64
- -qxcall
- -qxref

Template model

The template model used by IBM XL C/C++ for Linux, V13.1.1 is different from that used with previous versions of the compiler.

IBM XL C/C++ for Linux, V13.1.1 supports *Greedy instantiation*. The compiler generates a template instantiation in each compilation unit that uses it. The linker discards the duplicates.

For more information about the C++ template model, see "The C++ template model" in the *XL C/C++ Optimization and Programming Guide*

Changes in predefined macro support

The macros that are supported by IBM XL C/C++ for Linux, V13.1.1 are different from the macros that are supported by other versions of the XL C/C++ compiler.

Some macros that are supported by other versions of XL C/C++ for various platforms might be undefined in IBM XL C/C++ for Linux, V13.1.1. Refer to "Compiler predefined macros" in the *XL C/C++ Compiler Reference* for a full list of supported macros.

You can specify the **-Wunsupported-xl-macro** option to check whether any unsupported macro is used. If an unsupported macro is used in your code, the compiler issues a warning message.

Compiler pragmas

IBM XL C/C++ for Linux, V13.1.1 introduces support for many of the pragmas supported by GCC.

Supported GCC pragmas

The following GCC pragmas are supported in IBM XL C/C++ for Linux, V13.1.1. For details about these pragmas, see the GNU Compiler Collection online documentation at <http://gcc.gnu.org/onlinedocs/>.

- #pragma GCC dependency
- #pragma GCC diagnostic *kind option*
- #pragma GCC diagnostic push
- #pragma GCC diagnostic pop

- `#pragma GCC error string`
- `#pragma GCC poison`
- `#pragma GCC system_header`
- `#pragma GCC visibility push(visibility)`
- `#pragma GCC visibility pop`
- `#pragma GCC warning string`
- `#pragma message string`
- `#pragma once`
- `#pragma pop_macro("macro_name")`
- `#pragma push_macro("macro_name")`
- `#pragma redefine_extname oldname newname`
- `#pragma unused`

Supported IBM pragmas

- `#pragma disjoint`
- `#pragma execution_frequency`
- `#pragma ibm independent_loop`
- `#pragma nosimd`
- `#pragma option_override`
- `#pragma pack`
- `#pragma reachable`
- `#pragma simd_level`
- `#pragma STDC CX_LIMITED_RANGE`
- `#pragma unroll, #pragma nounroll`
- `#pragma weak`

Related information:



Supported IBM pragmas



Supported GCC pragmas

Chapter 3. Setting up and customizing XL C/C++

For complete prerequisite and installation information for XL C/C++, see "Before installing XL C/C++" in the *XL C/C++ Installation Guide*.

Using custom compiler configuration files

You can customize compiler settings and options by modifying the default configuration file or creating your own configuration file.

You have the following options to customize compiler settings:

- The XL C/C++ compiler installation process creates a default compiler configuration file. You can directly modify this configuration file to add default options for specific needs. However, if you later apply updates to the compiler, you must reapply all of your modifications to the newly installed configuration file.
- You can create your own custom configuration file that either overrides or complements the default configuration file. The compiler can recognize and resolve compiler settings that you specify in your custom configuration files with compiler settings that are specified in the default configuration file. Compiler updates that might later affect settings in the default configuration file does not affect the settings in your custom configuration files.

For more information, see "Using custom compiler configuration files" in the *XL C/C++ Compiler Reference*.

Chapter 4. Developing applications with XL C/C++

C/C++ application development consists of repeating cycles of editing, compiling, linking, and running. By default, compiling and linking are combined into a single step.

Notes:

1. Before you can use the compiler, you must first ensure that XL C/C++ is properly installed and configured. For more information, see the *XL C/C++ Installation Guide*.
2. To learn about writing C/C++ programs, refer to the *XL C/C++ Language Reference*.

The compiler phases

A typical compiler invocation executes some or all of these activities in sequence. For link time optimizations, some activities are executed more than once during a compilation. As each compilation component runs, the results are sent to the next step in the sequence.

1. Preprocessing of source files
2. Compilation, which may consist of the following phases, depending on what compiler options are specified:
 - a. Front-end parsing and semantic analysis
 - b. High-level optimization
 - c. Low-level optimization
 - d. Register allocation
 - e. Final assembly
3. Assemble the assembly (.s) files, and the unpreprocessed assembler (.S) files after they are preprocessed
4. Object linking to create an executable application

To see the compiler step through these phases, specify the **-v** compiler option when you compile your application. To see the amount of time the compiler spends in each phase, specify **-qphsinfo**.

Editing C/C++ source files

To create C/C++ source programs, you can use any text editor available to your system.

Source programs must be saved using a recognized file name suffix. See “XL C/C++ input and output files” on page 21 for a list of suffixes recognized by XL C/C++.

For a C or C++ source program to be a valid program, it must conform to the language definitions specified in the *XL C/C++ Language Reference*.

Compiling with XL C/C++

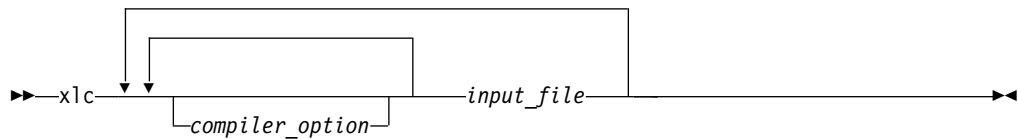
XL C/C++ is a command-line compiler. Invocation commands and options can be selected according to the needs of a particular C/C++ application.

Invoking the compiler

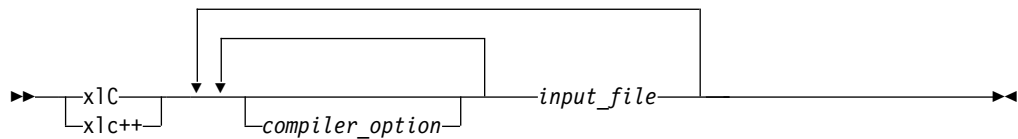
The compiler invocation commands perform all necessary steps to compile C or C++ source files, preprocessed files (.i or .ii), assemble any .s and .S files, and link the object files and libraries into an executable program.

The compiler invocation commands produce threadsafe code.

To compile a C source program, use the following basic invocation syntax:



To compile a C++ source program, use the following basic invocation syntax:



For most applications, compile with **xlc** or **xlc++**. You can use **xlc++** to compile either C or C++ program source, but compiling C++ files with **xlc** might result in link or runtime errors because libraries required for C++ code are not specified when the linker is called by the C compiler.

More invocation commands are available to meet specialized compilation needs, primarily to provide explicit compilation support for different levels and extensions of the C or C++ language. See "Invoking the compiler" in the *XL C/C++ Compiler Reference* for more information about compiler invocation commands available to you, including special invocations that are intended to assist developers in migrating from a GNU compilation environment to XL C/C++.

Specifying compiler options

Compiler options perform a variety of functions, such as setting compiler characteristics, describing the object code to be produced, controlling the diagnostic messages emitted, and performing some preprocessor functions.

You can specify compiler options in one or any combination of the following ways:

- On the command-line with command-line compiler options
- In your source code using directive statements
- In a makefile
- In the stanzas found in a compiler configuration file

You can also pass compiler options to the linker, assembler, and preprocessor.

For more information about compiler options and their usage, see "Compiler options reference" in the *XL C/C++ Compiler Reference*.

Priority sequence of compiler options

It is possible for option conflicts and incompatibilities to occur when multiple compiler options are specified. To resolve these conflicts in a consistent fashion, the compiler usually applies the following general priority sequence to most options:

1. Directive statements in your source file *override* command-line settings
2. Command-line compiler option settings *override* configuration file settings
3. Configuration file settings *override* default settings

Generally, if the same compiler option is specified more than once on a command-line when invoking the compiler, the last option specified prevails.

Note: Some compiler options, such as the **-I** option, do not follow the priority sequence described above. The compiler searches any directories specified with **-I** in the `xl.cfg` file before it searches the directories specified with **-I** on the command-line. The **-I** option is cumulative rather than preemptive. Other options with cumulative behavior are **-R** and **-l** (lowercase L).

XL C/C++ input and output files

These file types are recognized by XL C/C++.

For detailed information about these and additional file types used by the compiler, see "Types of input files" in the *XL C/C++ Compiler Reference* and "Types of output files" in the *XL C/C++ Compiler Reference*.

Table 4. Input file types

Filename extension	Description
.c	C source files
.C, .cc, .cp, .cpp, .cxx, .c++	C++ source files
.i	Preprocessed source files
.ii	Preprocessed C++ source files
.o	Object files
.s	Assembler files
.S	Unpreprocessed assembler files
.so	Shared object or library files

Table 5. Output file types

Filename extension	Description
a.out	Default name for executable file created by the compiler
.d	Make dependency file
.i	Preprocessed source files
.lst	Listing files
.o	Object files
.s	Assembler files
.so	Shared object or library files

Linking your compiled applications with XL C/C++

By default, you do not need to do anything special to link an XL C/C++ program. The compiler invocation commands automatically call the linker to produce an executable output file.

For example, you can use the following command to compile `file1.C` and `file3.C` to produce the object files `file1.o` and `file3.o`. All object files, including `file2.o`, are submitted to the linker to produce one executable.

```
xlc++ file1.C file2.o file3.C
```

Compiling and linking in separate steps

To produce object files that can be linked later, use the `-c` option.

```
xlc++ -c file1.C           # Produce one object file (file1.o)
xlc++ -c file2.C file3.C   # Or multiple object files (file1.o, file3.o)
xlc++ file1.o file2.o file3.o # Link object files with default libraries
```

For more information about compiling and linking your programs, see:

- "Linking" in the *XL C/C++ Compiler Reference*
- "Constructing a library" in the *XL C/C++ Optimization and Programming Guide*

Dynamic and static linking

XL C/C++ allows your programs to take advantage of the operating system facilities for both dynamic and static linking.

Dynamic linking means that the code for some external routines is located and loaded when the program is first run. When you compile a program that uses shared libraries, the shared libraries are dynamically linked to your program by default. Dynamically linked programs take up less disk space and less virtual memory if more than one program uses the routines in the shared libraries. During linking, they do not require any special precautions to avoid naming conflicts with library routines. They are designed to perform better than statically linked programs if several programs use the same shared routines at the same time. By using dynamic linking, you can upgrade the routines in the shared libraries without relinking.

Because this form of linking is the default, you need no additional options to turn it on.

Static linking means that the code for all routines called by your program becomes part of the executable file.

Statically linked programs can be moved to run on systems without the XL C/C++ runtime libraries. They might perform better than dynamically linked programs if they make many calls to library routines or call many small routines. They do require some precautions in choosing names for data objects and routines in the program if you want to avoid naming conflicts with library routines. They also might not work if you compile them on one level of the operating system and run them on a different level of the operating system.

Running your compiled application

After a program is compiled and linked, you can run the generated executable file on the command line.

The default file name for the program executable file produced by the XL C/C++ compiler is **a.out**. You can select a different name with the **-o** compiler option.

You should avoid giving your program executable file the same name as system or shell commands, such as `test` or `cp`, as you could accidentally execute the wrong command. If you do decide to name your program executable file with the same name as a system or shell command, you should execute your program by specifying the path name to the directory in which your executable file resides, such as `./test`.

To run a program, enter the name of the program executable file together with any run time arguments on the command line.

Canceling execution

To suspend a running program, press the **Ctrl+Z** key while the program is in the foreground. Use the **fg** command to resume running.

To cancel a running program, press the **Ctrl+C** key while the program is in the foreground.

Setting runtime options

You can use environment variable settings to control certain runtime options and behaviors of applications created with the XL C/C++ compiler. Some environment variables do not control actual runtime behavior, but they can have an impact on how your applications run.

For more information on environment variables and how they can affect your applications at run time, see the *XL C/C++ Installation Guide*.

Running compiled applications on other systems

If you want to run an application developed with the XL C/C++ compiler on another system that does not have the compiler installed, you need to install a runtime environment on that system or link your application statically.

You can obtain the latest XL C/C++ Runtime Environment PTF images, together with licensing and usage information, from the [XL C/C++ for Linux support page](#).

XL C/C++ compiler diagnostic aids

XL C/C++ issues diagnostic messages when it encounters problems compiling your application. You can use these messages and other information provided in compiler output listings to help identify and correct such problems.

For more information about listing, diagnostics, and related compiler options that can help you resolve problems with your application, see the following topics in the *XL C/C++ Compiler Reference*:

- "Compiler messages and listings"
- "Error checking and debugging options"
- "Listings, messages, and compiler information options"

Debugging compiled applications

You can use a symbolic debugger to debug applications compiled with XL C/C++.

At compile time, you can use the **-g** or **-q1inedebug** option to instruct the XL C/C++ compiler to include debugging information in compiled output. For **-g**, you can also use different levels to balance between debug capability and compiler optimization. For more information about the debugging options, see "Error checking and debugging" in the *XL C/C++ Compiler Reference*.

You can then use **gdb** or any other symbolic debugger to step through and inspect the behavior of your compiled application.

Optimized applications pose special challenges when debugging. For more information about optimizing your code, see "Optimizing your applications" in the *XL C/C++ Optimization and Programming Guide*.

Determining which level of XL C/C++ is being used

To display the version and release level of XL C/C++ that you are using, invoke the compiler with the **--version** (**-qversion**) compiler option.

For example, to obtain detailed version information, enter the following command:

```
xlc++ --version
```

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd. Laboratory
8200 Warden Avenue
Markham, Ontario L6G 1C7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1998, 2014.

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>.

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Index

Special characters

- .a files 21
- .c and .C files 21
- .i files 21
- .ii files 21
- .lst files 21
- .mod files 21
- .o files 21
- .s files 21
- .S files 21
- .so files 21

A

- archive files 21
- assembler
 - source (.s) files 21
 - source (.S) files 21

B

- basic example, described viii

C

- C++ templates 15
- code optimization 6
- compilation
 - sequence of activities 19
- compiler
 - controlling behavior of 20
 - invoking 20
 - running 20
- compiler options
 - conflicts and incompatibilities 21
 - specification methods 20
- customization
 - for compatibility with GNU 3

D

- debugger support 24
 - output listings 23
 - symbolic 7
- debugging 24
- debugging compiled applications 23
- debugging information, generating 23
- dynamic linking 22

E

- editing source files 19
- executable files 21
- executing a program 23
- executing the linker 22

F

- files
 - editing source 19
 - input 21
 - output 21

G

- GCC options 9
- GCC pragmas 15
- GNU
 - compatibility with 3, 9

I

- input files 21
- invocation commands 20
- invoking a program 23
- invoking the compiler 20

L

- language standards 3
- language support 3
- level of XL C/C++, determining 24
- libraries 21
- linking
 - dynamic 22
 - static 22
- linking process 22
- listings 21

M

- macros 15
- migration
 - source code 20

O

- object files 21
 - creating 22
 - linking 22
- optimization
 - programs 6
- output files 21

P

- performance
 - optimizing transformations 6
- problem determination 23
- programs
 - running 23

R

- running the compiler 20
- runtime
 - libraries 21
- runtime environment 23
- runtime options 23

S

- shared object files 21
- source files 21
- source-level debugging support 7
- static linking 22
- symbolic debugger support 7

T

- tools 5
 - cleanpdf utility 5
 - install configuration utility 5
 - install utility 5
 - mergepdf utility 5
 - resetpdf utility 5
 - showpdf utility 5

U

- utilities 5
 - cleanpdf 5
 - install 5
 - mergepdf 5
 - resetpdf 5
 - showpdf 5

X

- xl.ccfg file 20



Product Number: 5765-J08; 5725-C73

Printed in USA

GI13-2875-00

