

Index

toggle Demote_sizes — allow	== operator to compare integer types other than int .	62
toggle Long_double_16 — provide support for	16-byte long double types	71
option -Hppc602 — enable PowerPC	602 mode	44

A

option	-A- — recognize only predefined assertions and macros that begin with two underscores	27
System V Application Binary Interface, PowerPC Processor Supplement	(ABI Supplement)	xvi, 193
Application Binary Interface	(ABI)	201
System V Release 4 Application Binary Interface	(ABI)	xvi
toggle Enforce_access_control — allow compiler-enforced	access control	64
toggle Fast_virtual_bases — speed up	access to members of virtual bases	65
	access-related type qualifiers	143
pragma	Alias — assign an external name to an internal variable or function identifier	94, 108
type qualifier	_Alias — assume variable can be accessed or modified indirectly	144
use pragma	Alias to define how a public symbol appears in the symbol table	216
pragma Global_aliasing_convention —	aliasing convention; construct external object name for internal variable or function identifier	36, 98, 108
case shifting external names in	aliasing conventions	110
toggle Noalias — no	aliasing; assume all variables are non-aliasable	73, 135
type qualifier _Noalias — no	aliasing; assume variable cannot be accessed or modified indirectly	144
using pragmas to	align data	180
padding to	align members	175
pragma	Align_members — set the maximum boundary for structure-member alignment ...	94, 123, 175, 181
bit field	alignment	178
pragma Align_members — set the maximum boundary for structure-member	alignment	94, 123, 175, 181
pragma Push_align_members — set maximum boundary for structure-member	alignment and save state	102, 182
pragma Pack — control default	alignment of data structures	101

sizes and byte	alignment of data types	173
size and	alignment of enumeration types	175
size and	alignment of struct and union types	175
	alignment of structure members	181
reserved words <code>_Packed</code> and <code>_Unpacked</code> -		
control member	alignment on individual struct or union	175
pragma <code>Pop_align_members</code> — restore	alignment prior to earlier pragma <code>Push_align_</code>	
	members	101, 184
pragma	<code>Alloc_text</code> — group functions in specified text	
	segment	95
intrinsic function	<code>_Alloca()</code>	245
pragma <code>Data</code> —	allocate named blocks for data storage	96, 110
dynamically	allocated data structures	273
	allocating a pointer to a virtual-function table	189
	allocating non-static data members	189
	allocation	137
optimization — small-data sections		
toggle <code>High_level_scheduling</code> — perform	allocation	68, 132
instruction scheduling before register		
toggle <code>Low_level_scheduling</code> — perform	allocation	72, 133
instruction scheduling after register	allocation	84, 136
toggle <code>Reg_alloc_enhance</code> — enhance register	allocation (enhanced)	136
optimization — register	allocation (global)	137
optimization — register	allocation to override explicit register declarations	137
using global register	annotate assembly file listing with lines from the	
option <code>-Hanno</code> —	source file	29
	anonymous structure-member selection	76
AT&T UNIX System V Release 4 Programmer's		
Guide:	ANSI C and Programming Support Tools	201
	ANSI C Standard conformance and portability	3
toggle <code>Strict_ANSI_math</code> — generate strict	ANSI C Standard conforming code for standard	
	math function calls	86
option <code>-Hansi</code> — accept only	ANSI C Standard conforming programs	3, 29
option <code>-Hppc</code> — place the compiler in PCC mode		
to relax some	ANSI C Standard restrictions	43
toggle <code>PCC</code> — relax	ANSI C Standard restrictions to compile PCC	
	programs	76
High C/C++ tracks	ANSI C++ Standard working draft	3
trigraphs required for	ANSI Standard C compatibility	89
toggle <code>Prototype_override_warnings</code> — warn		
when an	ANSI Standard C declaration overrides an old-	
	style function definition	82
optimization — inlining	ANSI standard C functions	129

toggle Recognize_library — inline code for certain functions in the	ANSI Standard C library	83, 129
toggle Single_math_lib — do	ANSI Standard C math function calls in single precision if possible	86
option -Hmwlib — use the MetaWare	(ANSI) C header files and library	41, 47
pipelined	Application Binary Interface (ABI)	201
storing start-up module crt1.o in an environment variable	architecture	279
PowerPC variable	archive library (embedded applications)	266
driver	ARGS — specify command line arguments ...	36, 236
toggle FP_varargs — always save floating-point	argument lists	200
toggle Non_FP_varargs — do not save floating-point	argument or response file	10
toggle Print_var_info — display mapping of auto-, register-, and	argument registers	67
environment variable ARGS — specify command line	argument registers	74
when not to preload	argument-class variables	80
optimization — preloading	arguments	36, 236
toggle Preload_args_from_memory — load function	arguments	136
toggle Parm_warnings — issue warnings when	arguments from memory	136
toggle Template_trivial_conversions — allow trivial conversions for matching	arguments into machine registers in the function prolog	78, 136
toggle Borland — relax some	arguments to a non-prototype function do not match declared parameters	75
free-chain pointer	arguments to function templates	88, 168
dynamic	ARM C++ standards to conform to the Borland class library	58
storage of	array	276
how	array out of memory error message	284
configuration file variable	arrays	176
profile—	arrays are stored	176
generating inline assembly code with inline assembler directive	AS — specify assembler	236
	ASCII file treated as a prefix to the source file	22
	_ASM	147
	.asm — default assembly file-name extension	9
	_Asm macro syntax	151
components of	_Asm macros	152
keyword	asm renamed _Asm for enhanced ASM facility	151
machine registers that can be modified in an configuration file variable AS — specify	_ASM statement	148
	assembler	236

environment variable <code>TOOLS DIR</code> — specify path to	assembler and linker	238
toggle <code>Use_power_pc_mnemonics</code> — use PowerPC assembler mnemonics		89
passing	assembler options on the driver command line	8
option <code>-Hasopt</code> — pass	assembler options to the assembler	30
data communication — sharing variables across C and	assembly	219
generating inline	assembly code with inline assembler directive <code>_ASM</code>	147
option <code>-Hanno</code> — annotate	assembly file listing with lines from the source file ..	29
option <code>-o</code> — specify executable, object, or	assembly file name	51
.asm — default	assembly file-name extension	9
debugging programs at the	assembly language level	119
generating source and	assembly listings	13
calling C functions from	assembly routines	218
calling	assembly routines from C or C++	216, 217
option <code>-Hasmcpp</code> — process C-style preprocessing directives in	assembly source file	30
option <code>-S</code> — produce an	assembly source file instead of an object file	53
.s — default	assembly-language file-name extension	9
optimization — single static	assembly-language files	6
toggle <code>Single_static_assignment</code> — place intermediate representation of code in single static	assignment	137
PCC mode emulates	assignment form	85
option <code>-Hunixerr</code> — use	AT&T Portable C Compiler	43, 76
global	AT&T Portable C Compiler and unsignedness preserving rules	90
pragma <code>Called_infrequently</code> — apply calling-convention	AT&T Portable C Compiler diagnostic level	76
undefined	AT&T Portable C Compiler error message format	49, 288
how the compiler maps	AT&T UNIX System V Release 4 Programmer's Guide: ANSI C and Programming Support Tools ..	201
toggle <code>Print_var_info</code> — display mapping of	atexit functions (embedded applications)	258
toggle	attribute	221
	attribute <code>CALLED_INFREQUENTLY</code> to functions containing a specified string in their name	95
	attributes	185
	auto- or register-class variables	172
	auto-, register-, and argument-class variables	80
	Auto_class_member_instantiation — enable automatic instantiation of class templates ...	56, 266

toggle	Auto_func_instantiation — enable automatic instantiation of function templates	56, 264
toggle	Autodebug — place named SDI for a named struct in the struct's debug repository	56, 226
minimizing symbolic debug information (SDI) with toggles	Autodebug, Forcedebug, and Nodebug	225
	autoexec.bat files	278
manual versus activating	automatic function inlining	130
toggle Auto_class_member_instantiation - enable	automatic inlining with -Hi options	244
toggle Auto_func_instantiation — enable	automatic instantiation of class templates	56, 266
turning off	automatic instantiation of function templates .	56, 264
	automatic template instantiations	264

B

	back-chain word	194
toggle	Back_substitute_epilog — replace unconditional jump to a function epilog with epilog sequence ...	57
toggle	Back_substitute_nodes — reduce jumping due to conditional blocks within a loop	57
direct non-virtual storage for virtual	bases	189
toggle Fast_virtual_bases — speed up access to members of virtual	bases	189
toggle Print_lattices — generate a graphical view of any class with one or more direct and indirect option -Hbatch — create a predefined preprocessor macro symbol	bases	65
toggle	bases	79, 191
option -HB — compile in twos-complement	bases of a class	188
System V Release 4 Application	batch file to call the compiler and linker	16, 31
PowerPC Embedded Application	_BE_PPC	22
System V Application	Behaved — specify that a program handles pointer-based variables in a "well-behaved" manner	57, 124
relocatable	big-endian mode	30
long long	binary form	170
storage unit boundary for a	Binary Interface (ABI)	xvi
	Binary Interface (EABI)	xvi
	Binary Interface, PowerPC Processor Supplement (ABI Supplement)	xvi, 193
	binary object files	12
	bit field	177
	bit field	177
	bit field alignment	178
	bit field widths	176
High C/C++ support of signed and unsigned	bit fields	176

structures, unions, and	bit fields	172
width and range of values of	bit fields affecting structs	178
result of	bit-field types	176
pragma Skip — insert	bitwise operation on signed integer	171
COMMON type defines a common	blank lines in a source-code listing	104, 232
mapping static or exported variables into a	block	111
named data section or common	block with pragma Data	110
uninitialized global variables without extern	blocks	219
qualifier defined as individual common	blocks for data storage	96, 110
pragma Data — allocate named	blocks within a loop	57
toggle Back_substitute_nodes — reduce jumping	Borland — relax some ARM C++ standards to	
due to conditional	conform to the Borland class library	58
toggle	Borland Turbo C/C++ error message format ..	48, 288
option -Hturboerr — use	boundary for a bit field	177
storage unit	boundary for structure-member	
pragma Align_members — set the maximum	alignment	94, 123, 175, 181
pragma Push_align_members — set maximum	boundary for structure-member alignment and	
specifying maximum	save state	102, 182
	boundary on which an unpacked structure member	
	must be aligned	101
	.bss — default section for uninitialized variables	
	(not const qualified)	184
inliner option -Hib — specify size of	buffer	246
temporary file	built-in inliner versus the stand-alone inliner	156, 244
High C/C++	byte alignment of data types	173
sizes and		

C

function naming conventions in	C	216
structured name space unavailable in	C	107
	.c — default C source file-name extension	9
option	-c — generate object file but do not link	27
data communication — sharing variables across	C and assembly	219
compiling and linking	C and C++ modules	155
mixing	C and C++ modules	64
interfacing	C and C++ with other languages	173
AT&T UNIX System V Release 4 Programmer's	C and Programming Support Tools	201
Guide: ANSI	C compatibility	89
trigraphs required for ANSI Standard	C extensions to C language	1
High		

High C/C++ tracks ANSI	C++ Standard working draft	3
toggle Borland — relax some ARM	C++ standards to conform to the Borland class library	58
	C++ static objects constructed before main() and destroyed after main()	256
	C++ templates	165
problems with	C++ templates	167
interfacing C and	C++ with other languages	173
option -Hc_wrap — interpret files included in C++ modules as C, not	C++, header files	33
	.C, .cc, and .cpp — default C++ source file-name extension	9
option -Hc_wrap — interpret files included in C++ modules as	C, not C++, header files	33
option -Hasmcpp — process	C-style preprocessing directives in assembly source file	30
wrapper files	c_wrap1.h and c_wrapr.h	33
toggle	Call_trace — generate information about a called function	58, 224
pragma	Called_infrequently — apply calling-convention attribute CALLED_INFREQUENTLY to functions containing a specified string in their name	95
disabling compiler intrinsics and	calling equivalent library functions	126
PowerPC registers used in the standard	calling sequence	198
pragma Called_infrequently — apply	calling-convention attribute CALLED_INFREQUENTLY to functions containing a specified string in their name	95
pragma	Calling_convention — allow High C/C++ modules to call functions not compiled with High C/C++	96
	casting a type qualifier	143
toggle Try_static_cast — replace old-style type	casts with static_cast	89, 164
source file-name extensions	.cc and .cpp	9
.C,	.cc, and .cpp — default C++ source file-name extension	9
environment variable	CEXT — default C source file-name extension	9, 236
free	chain	276
heap	chain	274
toggle Use_UP_rules — widen unsigned	char and unsigned short to unsigned int	76, 90
toggle Char_default_unsigned — make	char default type unsigned char	58
toggle Char_is_rep — make default representation of	char identical to signed char or unsigned char	59
toggle	Char_default_unsigned — make char default type unsigned char	58

toggle	Char_is_rep — make default representation of char identical to signed char or unsigned char	59
toggle	direct and indirect bases of a class	188
toggle Run_time_type_info — generate run-time type information (RTTI) for a class		85, 163
toggle Define_static_members — supply implicit definition of static class data members		62
problems with inlining class destructors		187
when to make a class exception-aware		162
toggle Make_externs_global — make objects class extern global		72
with storage class functions		157
inline class layout examples		190
toggle Borland — relax some ARM C++ standards class library		58
to conform to the Borland class members in the class lattice display		78
toggle Print_lattice_members — list class objects		186
storage mapping for C++ class template		266
instantiating a class templates		56, 266
toggle Auto_class_member_instantiation - enable automatic instantiation of class templates and duplicate definitions		266
class templates defined		165
toggle Nested_types — make nested types class where they are declared		73
invisible outside the class with one or more bases		79, 191
toggle Print_lattices — generate a graphical view of any classes		156
functions in template classes		69
inline functions within classes		36, 42, 65, 160, 162
toggle Exception_aware_class — compile exception-aware classes		87
toggle Template_common — solve the single-copy problem for template functions and classes defined		165
template classes with inlined, pure virtual member functions		165
toggle Import_vtabs — assume virtual function tables to be external for classes with templates		161
defining generic functions or clean-up		138
effect of non-exception-aware classes on clean-up of completely constructed objects		160
optimization — spill-code clean-up of partially constructed objects		161
clean-up of partially constructed or destructed heap objects		160
toggle Cleanup_spills — clean up register spill code		60, 138

.data — default section for uninitialized static variables and writable strings referenced from	code	184
.text — default section for executable	code	185
optimization — constants in	code	127
option -KPIC — generate position-independent	code	50
option -kplic — generate position-independent	code	50
option -Os — optimize for smaller	code	120
option -PIC — generate position-independent	code	52
option -pic — generate position-independent	code	52
toggle Command_line_ident — put information in a comment section in generated	code	60, 115
pragma Code — specify the code section name	Code	96, 113
optimization — dead (unreachable)	code elimination	128
toggle Source_code_order — produce	code in source order where possible	86, 125
optimization —	code in source-code order	125
dynamic	code loading	127
	code portability	3
CODE type denotes an executable	code section	111
pragma Code — specify the	code section name	96, 113
naming	code sections	112
effect of inlining on compile time and	code size	131
execution speed versus	code size	119
loop unrolling usually increases	code size	135
operator strength reductions that result in		
larger	code size	136
program execution speed versus	code size	119
inlining functions can increase	code size while decreasing execution time ...	129, 243
literals in data versus	code space	83
toggle Literals_in_code — put lengthy		
literals in	code space instead of data space	71, 127
optimization —	code space minimizing (optimizing for space)	125
	CODE type denotes an executable code section	111
generating inline assembly	code with inline assembler directive _ASM	147
limitations to	COMDAT	167
option -Hnocomdat — disable	COMDAT solution for template instantiation	41
prof	command — generate execution profile and read	
	mon.out	52
file names on the	command line	8, 17
option -Hpragma — invoke a pragma on the	command line	44, 94, 291
passing assembler options on the driver	command line	8
passing linker options on the driver	command line	8
specifying files on the driver	command line	8

- environment variable ARGS — specify
 - driver configuration file versus toggle
 - pragma Ident — insert a toggle Command_line_ident — put information in a
 - limitations of OMF mapping static or exported variables into a named data section or uninitialized global variables without extern qualifier defined as individual DATA, LIT, TEXT, or toggle Common_can_export — allow public variables in toggle Local_CSE_iterate — perform multiple iterations of local optimization — optimization — optimization — toggle Global_CSE — eliminate global toggle
 - data inter-module toggle Demote_sizes — allow == operator to toggle Induction_analysis — readjust loop counter to eliminate trigraphs required for ANSI Standard C toggle Pointers_compatible_with_ints — make any type pointer effect of source file-name extensions on environment variable CPLUS — specify C++ option -Hcplvl — specify the level of C++ using option -Hnoobject to check for customizing the increasing toggle Forcedebug — place named SDI for any in-scope struct in current
 - command line arguments 36, 236
 - command line character limit 17
 - command line options 18
 - Command_line_ident — put information in a
 - comment section in generated code 60, 115
 - comment in the generated object file 98, 115
 - comment section in generated code 60, 115
 - Common 167
 - common block with pragma Data 110
 - common blocks 219
 - COMMON data section type 96
 - common segments 61
 - common subexpression elimination 71, 126
 - common subexpression elimination (global) 125
 - common subexpression elimination (local) 125
 - common subexpression elimination (local, iterative) 126
 - common subexpressions 68, 125
 - COMMON type defines a common block 111
 - Common_can_export — allow public variables in common segments 61
 - communication — sharing variables across C and assembly 219
 - communication and public names 248
 - compare integer types other than int 62
 - compares 69, 134
 - compatibility 89
 - compatible with integers 76, 77
 - compilation 155
 - compilation 155, 236
 - compilation 9, 32
 - compilation errors 42
 - compilation process 16
 - compilation speed 43, 119
 - compilation unit 228

toggle Forcedebug — place named SDI for any in-scope struct in current functions inlined across	compilation unit	67, 228
option -Hwide — specify multi-module inlining (wide	compilation units	157
inliner option -Hiw — invoke inliner in multi-module (wide using the driver to	compilation)	49, 131, 248
toggle Exception_aware_class —	compilation) mode	131, 248
toggle PCC — relax ANSI C Standard restrictions to using the driver to setting	compile and link in one step	12
option -P — preprocess source file but do not	compile exception-aware classes	36, 42, 65, 160, 162
option -E — preprocess source file but do not	compile PCC programs	76
PCC mode emulates AT&T Portable C	compile without linking	13
option -Hbatch — create a batch file to call the AT&T Portable C overview of precedence of scope of changing default values of setting	compile-time environment variables	19
environment variable HCDEBUG — set	compile; write output to a file	10, 20, 52
toggle PCC_msgs — reduce AT&T Portable C	compile; write output to standard output	28
option -w — set the	Compiler	43, 76
pragma Warning_level — set	compiler and linker	16, 31
pragma Onwarn_error — enable specified warning messages as optimization — how the how the specifying setting how the how the customizing the	Compiler and unsignedness preserving rules	90
	compiler controls	17
	compiler controls	17, 18
	compiler controls	18
	compiler controls in configuration file	235
	compiler controls in source code	18
	compiler debug controls	237
	compiler diagnostic capabilities to the PCC level	76
	Compiler diagnostic level	76
	compiler diagnostic warning level	53, 289
	compiler diagnostic warning level	104, 289
	compiler error messages	100, 285, 289, 290
	compiler intrinsics	126
	compiler maps auto- or register-class variables	172
	compiler maps variables to storage	184
	compiler option reference	27
	compiler options	8, 27
	compiler pragmas	93
	compiler pragmas, defined	93
	compiler reads profiles	23
	compiler selects routines for inlining	245
	compiler storage mapping	173
	compiler to a local environment	93

specifying	compiler toggles	55
	compiler toggles defined	55
overriding default	compiler values	18, 93
	compiler warning message numbers	289
enabling and disabling	compiler warning messages by number	100
eliminating	compiler warnings	75
option -Hretwc — add the warning count to the	compiler's return code	45
toggle Enforce_access_control — allow	compiler-enforced access control	64
option -V — display controls passed to the	compiler; also display subprocesses	16, 53
	compiling and linking C and C++ modules	155
wrapping C header files when	compiling with C++	64
clean-up of	completely constructed objects	160
	condition register (PowerPC)	197
toggle Back_substitute_nodes — reduce jumping	conditional blocks within a loop	57
due to	conditional preprocessor directives	172
unsigned expressions in	configuration file	24, 36, 236
driver	configuration file naming conventions	24
driver	configuration file variable AS — specify assembler	236
	configuration file variable CPP — C preprocessor	
	variable	237
	configuration file variable LINKER — specify	
	linker	237
driver	configuration file versus command line options	18
driver	configuration language	238
option -Hansi — accept only ANSI C Standard	conforming programs	29
type qualifier	const	99, 114
matching	const parameters	167
.bss — default section for uninitialized	const qualified)	184
variables (not	const static variables map in storage	61, 127
toggle Const_in_code — determine how	Const_in_code — determine how const static	
toggle	variables map in storage	61, 127
	constant	170
two-character	constant defined	169
character	constant expression folding	127
optimization —	constant propagation	127
optimization —		
read-only static variables and read-only	constants	114
string	constants and read-only static variables	99, 185
.rdata — default section for read-only string	constants in code	127
optimization —	constants in the read-only data section	83, 127
toggle Read_only_strings — store all string	constraint error and warning messages	286

pragma Global_aliasing_convention — aliasing convention;	construct external object name for internal variable or function identifier	36, 98, 108
data structure	constructor_item	256
priority queues of	constructor_items and destructor_items	257
how the inliner	constructs unique names	249
	continuation of line	238
changing default values of compiler	controls in configuration file	235
option -V — display	controls passed to the compiler; also display subprocesses	16, 53
pragma Global_aliasing_convention — aliasing convention; construct external object name for internal variable or function identifier	36, 98, 108	
case shifting external names in aliasing	conventions	110
driver configuration file naming	conventions	24
notational and typographic	conventions	xiii
terminology	conventions	xiii
truncation in aliasing	conventions	110
function naming	conventions in C	216
function naming	conventions in C++	217
integer	conversion	171
option -fsoft — use emulation routines to perform floating-point arithmetic and	conversions	28
toggle Template_trivial_conversions — allow trivial	conversions for matching arguments to function templates	88, 168
toggle Prototype_conversion_warn — warn when a function argument is	converted due to a prototype declaration	81
environment variable	core PowerPC instruction set	251
environment variable	CPLUS — specify C++ compilation	155
predefined preprocessor macro symbols	CPLUS — specify C++ compilation	236
source file-name extensions .cc and	__CPLUSPLUS__ and __cplusplus	22
configuration file variable	.cpp	9
.C, .cc, and	CPP — C preprocessor variable	237
UNIX	.cpp — default C++ source file-name extension	9
environment variable	cpp preprocessor	237
	CPPEXT — default C++ source file-name extension	10, 237
optimization —	cross jumping (tail merging)	128
toggle	Cross_jump — allow cross-jumping (tail-merging) optimization	61, 128, 237
option -nocrt0 — do not load	crt0.o during the link	50
linking modules crt0.o and crtn.o with	crt1.o (embedded applications)	255
storing start-up module	crt1.o in an archive library (embedded applications)	260

example of source for start-up module	crt1.s (embedded applications)	254
link order of	crti.o and crtn.o (embedded applications)	255
linking modules	crti.o and crtn.o with crt1.o (embedded applications)	255
start-up module	crt1.o for embedded applications	253
	customizing the compilation process	16
	customizing the compiler to a local environment	93
	customizing your program with pragmas	105
D		
option	-D — define a name for preprocessor	28
.data1 — default section for writable strings whose addresses are referenced from within	.data	185
32-bit data segment	__DATA	96
mapping static or exported variables into a named data section or common block with		
pragma	Data	110
using pragmas to align	data	180
pragma	Data — allocate named blocks for data storage	96, 110
	.data — default section for uninitialized static variables and writable strings referenced from	
	code	184
small	data area pointer (PowerPC)	197
	data communication — sharing variables across C and assembly	219
sharing an instantiation of a member function	data member	266
or	data members	189
allocating non-static		
toggle Define_static_members — supply	data members	62
implicit definition of static class	data section	112
DATA type denotes a writable		
toggle Read_only_strings — store all string constants in the read-only	data section	83, 127
mapping static or exported variables into a named		
DATA, LIT, TEXT, or COMMON	data section or common block with pragma Data ..	110
small	data section type	96
global variables declared in the	data sections	203
pragma Data_seg — put static and global variables in specified	data segment	61
32-bit		
toggle Literals_in_code — put lengthy literals in code space instead of	data segment	97
pragma Data — allocate named blocks for	data segment __DATA	96
	data space	71, 127
	data storage	96, 110

	data structure constructor_item	256
	data structure destructor_item	256
dynamically allocated	data structures	273
initialization of	data structures	106
pragma Pack — control default alignment of	data structures	101
toggle Print_layout — print layout of C++	data structures	80
pragma Push_small_data — Assign	data to small-data section	103
	DATA type denotes a writable data section	112
implementation of	data types	169
pointer	data types	172
representation of integer	data types	170
sizes and byte alignment of	data types	173
literals in	data types described	169
	data versus code space	83
pragma	DATA, LIT, TEXT, or COMMON data section type	96
	Data_seg — put static and global variables in specified data segment	97
	.data1 — default section for writable strings whose addresses are referenced from within .data	185
toggle	Dbx — generate debug information in SUN dbx stab format	61
optimization—	dead (unreachable) code elimination	128
eliminating	dead stores	133
environment variable HCDEBUG — set compiler	debug controls	237
minimizing symbolic	debug information (SDI) with toggles Autodebug, Forceddebug, and Nodebug	225
toggle Dwarf — generate	debug information in DWARF format	63
toggle Dbx — generate	debug information in SUN dbx stab format	61
toggle Autodebug — place named SDI for a	debug repository	56, 226
named struct in the struct's	debug repository of a struct	226
	debugger support	223
heap-management specifics and	debugging	274
preparing for	debugging	14
scheduling code impairs symbolic	debugging	132
using unmangle in	debugging	158
	debugging and diagnostic tools	223
	debugging fully scheduled code	132
DWARF	debugging information format	xvi
option -g — place	debugging information in compiled code	29, 132, 223, 225
including	debugging information in the object file	14

	debugging inlined routines	245
	debugging optimized code	122
	debugging programs at the assembly language level	119
toggle Prototype_conversion_warn — warn when a function argument is converted due to a prototype	declaration	81
toggle Prototype_override_warnings — warn when an ANSI Standard C	declaration overrides an old-style function definition	82
unmangle — name unmangler; converts mangled function and variable names into a form similar to C++	declaration syntax	158
external	declarations	126
using global register allocation to override explicit register	declarations	137
example	declarations of reversed-endian variables	180
toggle Empty_is_void — force C++ (void) interpretation of function	declarations with no arguments	34, 64
global variables	declared in the data segment	61
toggle Parm_warnings — issue warnings when arguments to a non-prototype function do not match	declared parameters	75
module-definition file extension	.def	34
.text —	default section for executable code	185
.rdata —	default section for read-only string constants and read-only static variables	99, 185
.data —	default section for uninitialized static variables and writable strings referenced from code	184
.bss —	default section for uninitialized variables (not const qualified)	184
.data1 —	default section for writable strings whose addresses are referenced from within .data	185
toggle Char_default_unsigned — make char	default type unsigned char	58
toggle	Define_static_members — supply implicit definition of static class data members	62
toggle Prototype_override_warnings — warn when an ANSI Standard C declaration overrides an old-style function	definition	82
duplicate	definition error message from linker	56, 264
toggle Define_static_members — supply implicit	definition of static class data members	62
class templates and duplicate	definitions	266
toggle	Demote_sizes — allow == operator to compare integer types other than int	62
clean-up of partially constructed or	destructured heap objects	160

data structure	destructor_item	256
priority queues of constructor_items and	destructor_items	257
problems with inlining class	destructors	187
toggle PCC_msgs — reduce compiler	diagnostic capabilities to the PCC level	76
	diagnostic messages	283
debugging and	diagnostic tools	223
pragma Warning_level — set compiler	diagnostic warning level	104, 289
option -w — set the compiler	diagnostic warning level	53, 289
type ptrdiff_t —	difference of two pointers	172
	direct and indirect bases	188
	direct and indirect bases of a class	188
	direct non-virtual bases	189
generating inline assembly code with inline	directive _ASM	147
assembler		
unsigned expressions in conditional	directives	172
preprocessor	directories for locating libraries	53
option -YP — specify default	directory	43
option -Hobjdir — specify an object-file	directory	43
option -Hobjprefix — specify an object-file	directory	237
specifying installation	directory	6
working	directory for temporary files	237
environment variable TMPPREFIX- specify		
alternate	directory for temporary files	237
option -Htmpprefix — set	directory for temporary files	48
environment variable HCDIR — specify High		
C/C++	directory name	237
toggle Relative_includes — look up included	directory of the including file	84
files relative to the	directory to search for include files	50, 156
option -I — specify an alternate	directory to search for libraries specified with	
option -L<name> — specify a	option -l	15, 50
	disable (turn off) a toggle	43, 55
option -Hoff —	disable (turn off) a toggle	55, 99
pragma Off —	disable (turn off) inlining for specific	
inliner pragma Off_inline —	functions	100, 249
	disable (turn off) specified warning	
pragma Offwarn —	messages	100, 187, 289, 290
option -Htrap —	disable default OS/2 exception handling	48
optimization — in-order execution of I/O	(disable)	131
	disabling compiler intrinsics and calling equivalent	
	library functions	126
	displaying include files in listings	233

- toggle Use_reciprocal_for_divide — convert a floating-point divide-by-constant to a multiply-by-reciprocal 90, 128
- toggle Trap_div_by_zero — trap division by zero 88
- converting
 - division or modulo by an integer power of 2 to a shift or masking operation 136
 - division to multiplication by a reciprocal 128
- converting floating-point
 - DLL 34
 - listing symbols exported from a DLL 34
 - using a module-definition file to create a DLL 34
 - using option -Hdef to link a DLL entry point 35
 - defining a DLL entry point 35
 - function _DLL_InitTerm() — define a DLL version of the High C/C++ run-time library 35
 - option -Hdlllib — link with the (DLL) 34
 - option -Hdll — create a dynamic link library _DLL_InitTerm() — define a DLL entry point 35
 - function DONE statement (embedded applications) 254
- toggle Widen_float_args — widen float arguments to
 - double 91
 - option -fdouble — specify not less than double precision floating-point arithmetic 28
 - toggle Double_math_only — perform double precision floating-point arithmetic 62
- toggle Long_double_16 — provide support for
 - 16-byte long double types 71
 - toggle Double_math_only — perform double precision floating-point arithmetic 62
 - toggle Double_return — make non-prototype functions return type double, not type float 63
 - toggle Downshift_file_names — convert include file names to all lowercase 63
- passing assembler options on the
 - driver argument or response file 10
 - passing linker options on the
 - driver command line 8
 - driver command line 8
 - driver command line 8
 - driver configuration file 24, 36, 236
 - driver configuration file naming conventions 24
 - driver configuration file versus command line options 18
 - specifying files on the
 - driver configuration language 238
 - driver in and out of memory 43
 - driver program 238
 - driver to compile and link a program in one step 7
 - driver to compile and link in one step 12
 - driver to compile without linking 13
 - duplicate definition error message from linker 56, 264
 - duplicate definitions 266
- option -Hnoswap — do not swap the
 - using the
 - using the
 - using the
 - class templates and

toggle Dwarf — generate debug information in DWARF format	63
DWARF debugging information format	xvi
dynamic array out of memory error message	284
dynamic code loading	127
option -Hdll — create a dynamic link library (DLL)	34
dynamically allocated data structures	273

E

option -E — preprocess source file but do not compile; write output to standard output	28
PowerPC Embedded Application Binary Interface (EABI)	xvi
implementation of EasyThread	88
toggle Long_long_8 — provide support for eight-byte long long types	72
PowerPC Embedded Application Binary Interface (EABI) ...	xvi
building a shared library for embedded applications	260
floating-point emulation in embedded applications	251
start-up module crt1.o for embedded applications	253
atexit functions (embedded applications)	258
calling functions main() and _exit() from a start-up module (embedded applications)	254
DONE statement (embedded applications)	254
entry point of a start-up module (embedded applications)	253
example of source for start-up module crt1.s (embedded applications)	254
function __init() (embedded applications)	255
function _fini() (embedded applications)	255
function libraries (embedded applications)	252
host platform capabilities (embedded applications)	252
link order of crt1.o and crtn.o (embedded applications)	255
linking modules crt1.o and crtn.o with crt1.o (embedded applications)	255
mapping exception-handling code in memory (embedded applications)	262
shared library entry point (embedded applications)	260
special uses for start-up code (embedded applications)	259
stack set-up in a start-up module (embedded applications)	253
start-up code for shared libraries (embedded applications)	260
start-up code operations (embedded applications)	258
start-up code operations for shared libraries (embedded applications)	261
start-up file entry point (embedded applications)	260
start-up module for a shared library (embedded applications)	260
storing start-up module crt1.o in an archive library (embedded applications)	260

system-specific initialization in a start-up module	(embedded applications)	253
toggle	Empty_is_void — force C++ (void) interpretation of function declarations with no arguments ..	34, 64
PCC mode	emulates AT&T Portable C Compiler	43, 76
option -fsoft — use	emulation routines to perform floating-point arithmetic and conversions	28
option -Hon —	enable (turn on) a toggle	43, 55
toggle Trigraphs —	enable trigraphs	88
type qualifier _Reversed_Endian — set	endianness of variable opposite to that of the machine the program runs on	144, 180
	endianness-related type qualifiers	144
toggle Use_eieio —	enforce in-order execution of input/output	89
toggle	Enforce_access_control — allow compiler-enforced access control	64
pragma	Ensure_instantiation — ensure instantiation of specified template types	97, 268
toggle	Ensure_instantiation — force instantiation of all template types	64, 269
defining a DLL	entry point	35
start-up file	entry point (embedded applications)	260
	entry point of a start-up module (embedded applications)	253
toggle Long_enums — map type	enum variables to an int-sized area of memory	44, 71, 175
size and alignment of	enumeration types	175
customizing the compiler to a local	environment	93
	environment pointers (PowerPC)	197
	environment variable ARGS — specify command line arguments	36, 236
	environment variable CEXT — default C source file-name extension	9, 236
option -Hipname — set the name of the	environment variable containing the include file search path	38
	environment variable CPLUS — specify C++ compilation	155, 236
	environment variable CPPEXT — default C++ source file-name extension	10, 237
	environment variable HC_DUMPREG	48
	environment variable HCDEBUG — set compiler debug controls	237
	environment variable HCDIR — specify High C/C++ directory name	237
	environment variable MALLOC_LEVEL- control level of heap integrity checking	278

environment variable TMPPREFIX- specify alternate directory for temporary files	237
environment variable TOOLSDIR — specify path to assembler and linker	238
setting compile-time environment variables	19
PowerPC prolog and epilog code	211
toggle Back_substitute_epilog — replace unconditional jump to a function toggle	epilog with epilog sequence 57 Epilog_trace — generate information about a function being exited 65, 224
errno values	87
error and warning message syntax	285
error and warning messages	286
error message	284
error message format	287
error message format	287
error message format	41, 287, 288
error message format	48, 288
error message format	49, 288
error message formats	287
error message from linker	56, 264
error message numbers	73, 288, 290
error messages	100, 285, 289, 290
error messages	285
error messages	36, 287
error messages	286
errors	283
errors	284
errors	42
errors and warnings	285
exc_off.h and exc_pop.h	42
exception (PowerPC)	198
exception handling	65, 160
exception handling	42, 162
exception handling	48
exception-aware	162
exception-aware code	36, 162
exception-handling code in memory (embedded applications)	262
option -Hmscerr — use Microsoft C/C++	
option -Hturboerr — use Borland Turbo C/C++	
option -Hunixerr — use AT&T Portable C Compiler	
High C/C++	
duplicate definition	
toggle Msgno — output	
pragma Onwarn_error — enable specified warning messages as compiler	
lexical	
option -Hefmat — set format of syntactic	
file I/O	
system	
using option -Hnoobject to check for compilation	
user	
wrapper files	
floating-point invalid	
option -Hnonexcept_wrap — wrap include files to compile without	
option -Htrap — disable default OS/2 when to make a class	
option -Hexcept — compile and link mapping	

- toggle
 - Exception_aware_class — compile exception-aware classes 36, 42, 65, 160, 162
- predefined preprocessor macro symbol
 - `__EXCEPTIONS__` 22
- wrapper files
 - `excl.h` and `excr.h` 163
- pragma
 - Exclude_instantiation — prevent instantiation of specified template types 97, 268
- `.text` — default section for
 - executable code 185
- CODE type denotes an
 - executable code section 111
 - executable file names 11
 - executable programs 5
- creating
 - executable, object, or assembly file name 51
- option `-o` — specify
 - execution character sets 169
- source and
 - execution of I/O (disable) 131
- optimization — in-order
 - execution of input/output 89
 - execution process traces 53
- toggle Use_eieio — enforce in-order
 - execution profile and read `gmon.out` 52
 - execution profile and read `mon.out` 52
 - execution speed 119, 128
 - execution speed versus code size 119
 - execution speed versus code size 119
- gprof command — generate
 - execution time 129, 243
- prof command — generate
 - `_exit()` from a start-up module (embedded applications) 254
- increasing
 - Exit_expr — evaluate an expression at program termination 97, 107
- program
 - explicit register declarations 137
- inlining functions can increase code size
 - Export_vtabs — make virtual function tables public and defined 65
 - while decreasing
 - exported from a DLL 34
- calling functions `main()` and
 - exported functions 248
- pragma
 - exported variables into a named data section or common block with `pragma Data` 110
- using global register allocation to override
 - expression at program start-up 104, 106
- toggle
 - expression at program termination 97, 107
 - explicit register declarations 137
 - Export_vtabs — make virtual function tables public and defined 65
 - exported from a DLL 34
 - exported functions 248
 - exported variables into a named data section or common block with `pragma Data` 110
 - expression at program start-up 104, 106
 - expression at program termination 97, 107
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70
- listing symbols
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70
- inliner option `-Hix` — do not inline
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70
- mapping static or
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70
- pragma Startup_expr — evaluate an
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70
- pragma Exit_expr — evaluate an
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70
- optimization — constant
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70
- optimization —
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70
- simplifying
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70
- unsigned
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70
- toggle Keep_global_refs_in_loops — do not
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70
- move
 - expression folding 127
 - expression simplification 128
 - expressions 128
 - expressions in conditional preprocessor directives 172
 - expressions with global or external variables out of loops 70

toggle VP_UP_warn — warn about	expressions with more than one value, depending on whether UP or VP rules are used	91
.asm — default assembly file-name	extension	9
.c — default C source file-name	extension	9
.C, .cc, and .cpp — default C++ source file-name	extension	9
.i and .ii — default preprocessed file-name	extension	9
.s — default assembly-language file-name	extension	9
environment variable CEXT — default C source file-name	extension	9, 236
environment variable CPPEXT — default C++ source file-name	extension	10, 237
option -Hsuffix — specify the object file module-definition file	extension	47
option -Hcppext — specify the C++ source file-name	extension .def	34
source file-name	extensions	10, 32
effect of source file-name	extensions .cc and .cpp	9
High C	extensions on compilation	155
toggle Make_externs_global — make objects with storage class	extensions to C language	1
global functions declared	extern global	72
uninitialized global variables without	extern inline	69
	extern qualifier defined as individual common blocks	219
	external declarations	126
	external name clashes and linker limitations	107
	external name to an internal variable or function identifier	94, 108
pragma Alias — assign an	external names in aliasing conventions	110
case shifting		
pragma Global_aliasing_convention — aliasing convention; construct	external object name for internal variable or function identifier	36, 98, 108
linker's	external symbol dictionary	107
	external symbol table	216
toggle Keep_global_refs_in_loops — do not move expressions with global or	external variables out of loops	70
option -Hnalib — allow	extra-ANSI function calls without leading underscore	41

F

toggle	Fast_virtual_bases — speed up access to members of virtual bases	65
option	-fdouble — specify not less than double precision floating-point arithmetic	28

driver argument or response	file	10
driver configuration	file	24, 36, 236
including debugging information in the object	file	14
option -Hefile — redirect output sent to stderr and send it to standard output or to a	file	35
option -Hlist — write a source-code listing to standard output or to a	file	13, 39, 231
option -Hmakeof — write a list of makefile dependencies to the named	file	25, 40
option -Hmake — write a list of makefile dependencies to a .u	file	25, 48
pragma Ident — insert a comment in the generated object	file	98, 115
specifying the optional linker output load	file	237
specifying the start-up	file	237
readme	file — identifies last minute changes and describes special files	xiv, 14
inliner option -Hib — specify size of temporary	file buffer	246
	file I/O errors	283
option -o — specify executable, object, or assembly	file name	51
executable	file names	11
	file names on the command line	8, 17
toggle Downshift_file_names — convert include	file names to all lowercase	63
using a module-definition	file to create a DLL	34
profile — ASCII	file treated as a prefix to the source file	22
profile — ASCII	file treated as a prefix to the source file	22
.asm — default assembly	file-name extension	9
.c — default C source	file-name extension	9
.C, .cc, and .cpp — default C++ source	file-name extension	9
.i and .ii — default preprocessed	file-name extension	9
.s — default assembly-language	file-name extension	9
environment variable CEXT — default C source	file-name extension	9, 236
environment variable CPPEXT — default C++ source	file-name extension	10, 237
option -Hcppext — specify the C++ source	file-name extensions	10, 32
source	file-name extensions .cc and .cpp	9
effect of source	file-name extensions on compilation	155
assembly-language	files	6
autoexec.bat	files	278
directory for temporary	files	237

environment variable TMPPREFIX- specify alternate directory for temporary	files	237
MetaWare library and header	files	15
option -Htmpprefix — set directory for temporary	files	48
option -I — specify an alternate directory to search for include	files	50, 156
readme file — identifies last minute changes and describes special	files	xiv, 14
displaying include	files in listings	233
specifying	files on the driver command line	8
pragma	Fini — place a call to each listed function in the program's .fini section	98, 256
function	_fini() (embedded applications)	255
toggle Double_return — make non-prototype functions return type double, not type	float	63
toggle Widen_float_args — widen	float arguments to double	91
toggle FP_varargs — always save	floating-point argument registers	67
toggle Non_FP_varargs — do not save	floating-point argument registers	74
option -fdouble — specify not less than double precision	floating-point arithmetic	28
option -fsingle — specify single precision	floating-point arithmetic	28
toggle Double_math_only — perform double precision	floating-point arithmetic	62
option -fsoft — use emulation routines to perform	floating-point arithmetic and conversions	28
toggle Use_reciprocal_for_divide — convert a	floating-point divide-by-constant to a multiply-by-reciprocal	90, 128
converting	floating-point division to multiplication by a reciprocal	128
	floating-point emulation in embedded applications	251
	floating-point invalid exception (PowerPC)	198
negative	floating-point number	171
representation of	floating-point numbers	171
	floating-point status and control register (PowerPC)	197
optimization — constant expression	folding	127
toggle	Forcedebug — place named SDI for any in-scope struct in current compilation unit	67, 228
minimizing symbolic debug information (SDI)	Forcedebug, and Nodebug	225
with toggles Autodebug,	format	287
changing the error message	format	287
High C error message	format	232
listing		

option -Hmscerr — use Microsoft C/C++ error message	format	41, 287, 288
option -Hturboerr — use Borland Turbo C/C++ error message	format	48, 288
option -Hunixerr — use AT&T Portable C Compiler error message	format	49, 288
toggle Dbx — generate debug information in SUN dbx stab	format	61
toggle Dwarf — generate debug information in DWARF	format	63
option -Hefmat — set	format of error messages	36, 287
option -Hwformat — set	format of warning messages	49, 287
High C/C++ error message	format strings for output-file names	54
toggle	formats	287
library functions malloc and	FP_varargs — always save floating-point argument registers	67
heap	free	273
	free chain	276
	free list	276
	free-chain links	276
	free-chain pointer array	276
	free-list pointers	276
option	-fsingle — specify single precision floating-point arithmetic	28
option	-fsoft — use emulation routines to perform floating-point arithmetic and conversions	28
option -p — generate code to count calls to each	function	52
option -pg — generate code to count calls to each	function	52
toggle Call_trace — generate information about a called	function	58, 224
toggle Saves_incoming_regs — save all incoming registers of a	function	85, 209
toggle Tag_table — generate tag table in separate .tag section for each	function	87, 209
toggle Tag_table_in_text — generate tag table in .text section for each	function	87, 209
	function __init() (embedded applications)	255
intrinsic	function _Alloca()	245
	function _DLL_InitTerm() — define a DLL entry point	35
	function _fini() (embedded applications)	255
	function _main() called from C++ function main()	258

	function <code>_main()</code> not called from a C function <code>main()</code>	258
<code>unmangle</code> — name unmangler; converts mangled	function and variable names into a form similar to C++ declaration syntax	158
<code>toggle Prototype_conversion_warn</code> — warn when a	function argument is converted due to a prototype declaration	81
<code>toggle Preload_args_from_memory</code> — load	function arguments into machine registers in the function prolog	78, 136
<code>toggle Prolog_trace</code> — generate information about a	function being entered	81, 224
<code>toggle Epilog_trace</code> — generate information about a	function being exited	65, 224
overhead of	function calls	243
<code>toggle Strict_ANSI_math</code> — generate strict ANSI C Standard conforming code for standard math	function calls	86
<code>toggle Single_math_lib</code> — do ANSI Standard C math	function calls in single precision if possible	86
option <code>-Hnolib</code> — allow extra-ANSI	function calls without leading underscore	41
<code>toggle Overload_info</code> — show	function chosen during overload operation	75
<code>toggle Empty_is_void</code> — force C++ (void) interpretation of	function declarations with no arguments	34, 64
<code>toggle Prototype_override_warnings</code> — warn when an ANSI Standard C declaration overrides an old-style	function definition	82
<code>toggle Parm_warnings</code> — issue warnings when arguments to a non-prototype	function do not match declared parameters	75
<code>toggle Back_substitute_epilog</code> — replace unconditional jump to a	function epilog with epilog sequence	57
<code>toggle Print_protos</code> — write a prototype-style	function header for every function to standard output	80
<code>pragma Alias</code> — assign an external name to an internal variable or	function identifier	94, 108
<code>pragma Global_aliasing_convention</code> — aliasing convention; construct external object name for internal variable or	function identifier	36, 98, 108
<code>toggle Reference_all_functions</code> — create a reference to every non-inline	function in the program	84
<code>pragma Fini</code> — place a call to each listed	function in the program's <code>.fini</code> section	98, 256
<code>pragma Init</code> — place a call to each listed manual versus automatic	function in the program's <code>.init</code> section	98, 256
	function inlining	130
	function libraries (embedded applications)	252
	function naming conventions in C	216
	function naming conventions in C++	217

sharing an instantiation of a member	function or data member	266
PowerPC	function return values	210
toggle Vtable_common — solve the single-copy	function tables	91
problem for virtual	function tables	186
virtual	function tables public and defined	65
toggle Export_vtabs — make virtual	function tables to be external for classes with	
toggle Import_vtabs — assume virtual	inlined, pure virtual member functions	68
instantiating a	function template	167
toggle Template_trivial_conversions — allow	function templates	88, 168
trivial conversions for matching arguments to	function templates	56, 264
toggle Auto_func_instantiation — enable	function templates defined	165
automatic instantiation of	function type qualifiers	145
PowerPC	function-call tracing	224
calling versus inlining transcendental math	function-calling sequence	193
comparing virtual pointer-to-member	functions	86
disabling compiler intrinsics and calling	functions	164
equivalent library	functions	126
inline class	functions	157
inlined member	functions	187
inlining C++	functions	250
optimization — inlining ANSI standard C	functions	129
optimization — inlining	functions	129
pure virtual member	functions	187
signal	functions	70
toggle Import_vtabs — assume virtual function	functions	68
tables to be external for classes with	functions	291
inlined, pure virtual member	functions	186
undeclared	functions (embedded applications)	258
virtual	functions _mw_register_ctor() and _mw_	
atexit	register_dtor()	256
toggle Template_common — solve the	functions and classes	87
single-copy problem for template	functions called during signal handling	198
inlining	functions can increase code size while decreasing	
template	execution time	129, 243
calling C	functions defined	165
	functions from assembly routines	218

pragma Alloc_text — group	functions in specified text segment	95
	functions in template classes	156
toggle Recognize_library — inline code for certain	functions in the ANSI Standard C library	83, 129
	functions inlined across compilation units	157
	calling functions main() and _exit() from a start-up module (embedded applications)	254
	library functions malloc and free	273
pragma Calling_convention — allow High C/C++ modules to call	functions not compiled with High C/C++	96
defining generic	functions or classes with templates	165
toggle Double_return — make non-prototype	functions return type double, not type float	63
inliner option -Hih — show	functions that have been inlined and/or defined	247
overloading multiple	functions with the same name	158

G

suppressing the inliner with option	-g	245
option	-g — place debugging information in compiled code	29, 132, 223, 225
toggle Run_time_type_info —	generate run-time type information (RTTI) for a class	85, 163
toggle Make_externs_global — make objects with storage class extern	global	72
	global attribute	221
toggle Global_CSE — eliminate	global common subexpressions	68, 125
	global functions declared extern inline	69
	global identifiers	94
	global offset table (GOT)	201
toggle Keep_global_refs_in_loops — do not move expressions with	global or external variables out of loops	70
using	global register allocation to override explicit register declarations	137
	global subobject count	65, 160
pragma Weak x = y — designate a weak	global symbol and assign it a value	105
	global variables declared in the data segment	61
pragma Data_seg — put static and	global variables in specified data segment	97
	global variables in the symbol table	219
uninitialized	global variables without extern qualifier defined as individual common blocks	219
optimization — common subexpression elimination	(global)	125
optimization — register allocation	(global)	137
option -Hgac — set a value for pragma	Global_aliasing_convention	36, 109

pragma	Global_aliasing_convention — aliasing convention; construct external object name for internal variable or function identifier	36, 98, 108
toggle	Global_CSE — eliminate global common subexpressions	68, 125
symbol	_GLOBAL_OFFSET_TABLE_	202
toggle	Globals_volatile — see toggle	
	Keep_global_refs_in_loops	68
gprof command — generate execution profile and read	gmon.out	52
global offset table	(GOT)	201
	gprof command — generate execution profile and read gmon.out	52
toggle Print_lattices — generate a	graphical view of any class with one or more bases	79, 191

H

option	-H — display names of files included during compilation	29
option	-h — search help files	11, 29
option	-H+w — issue all relevant warning messages	49
option	-Hanno — annotate assembly file listing with lines from the source file	29
option	-Hansi — accept only ANSI C Standard conforming programs	3, 29
option	-Hasmcpp — process C-style preprocessing directives in assembly source file	30
option	-Hasopt — pass assembler options to the assembler	30
option	-HB — compile in big-endian mode	30
option	-Hbatch — create a batch file to call the compiler and linker	16, 31
environment variable	HC_DUMPREG	48
option	-Hc_wrap — interpret files included in C++ modules as C, not C++, header files	33
environment variable	HCDEBUG — set compiler debug controls	237
environment variable	HCDIR — specify High C/C++ directory name	237
option	-Hcplus — include C++ header files and link C++ run-time library	31, 156
option	-Hcpp — use the system's outboard C macro preprocessor	21, 31
using option	-Hcpp with a profile	23
option	-Hcppext — specify the C++ source file-name extensions	10, 32
option	-Hcpplvl — specify the level of C++ compilation	9, 32

option	-Hdef — use a module-definition file for linking	34
using option	-Hdef to link a DLL	34
option	-Hdll — create a dynamic link library (DLL)	34
option	-Hdlllib — link with the DLL version of the High C/C++ run-time library	35
option -Hc_wrap — interpret files included in C++ modules as C, not C++, MetaWare library and	header files	33
option -Hmwlib — use the MetaWare (ANSI) C	header files	15
option -Hsyslib — use the system C	header files and library	41, 47
option -Hcplus — include C++ including C++	header files and library	41, 47
option -Hnocplus — do not include C++ wrapping C	header files and link C++ run-time library	31, 156
toggle Print_protos — write a prototype-style function	header files and linking the C++ run-time library ..	236
	header files or link to C++ run-time library	42
	header files when compiling with C++	64
environment variable MALLOC_LEVEL- control level of	header for every function to standard output	80
	heap chain	274
	heap free list	276
run-time	heap integrity checking	278
clean-up of partially constructed or destroyed	heap integrity checking	278
	heap manager	273
option	heap objects	160
option	heap-management specifics and debugging	274
option -h — search	-Hefile — redirect output sent to stderr and send it to standard output or to a file	35
option	-Hefmat — set format of error messages	36, 287
option	help files	11, 29
option	-Hexcept — compile and link exception-aware code	36, 162
option	-Hgac — set a value for pragma Global_aliasing_convention	36, 109
inliner option	-Hi — invoke the optional stand-alone inliner	36, 129, 246
inliner option	-Hia — do not inline functions whose addresses are taken	246
inliner option	-Hib — specify size of temporary file buffer	246
inliner option	-Hic — inline functions called fewer than n times ..	246
inliner option	-HiC — inline functions smaller than s and called fewer than n times	246
	High C error message format	287
	High C extensions to C language	1
PowerPC models and features not supported by	High C/C++	251

	High C/C++ built-in inliner versus the stand-alone inliner	156, 244
environment variable HCDIR — specify	High C/C++ directory name	237
	High C/C++ error message formats	287
	High C/C++ Language Extensions Manual	xiv
	High C/C++ macro preprocessor	20
pragma Calling_convention — allow	High C/C++ modules to call functions not compiled with High C/C++	96
option -Hdlllib — link with the DLL version of the	High C/C++ run-time library	35
	High C/C++ support of signed and unsigned bit fields	176
	High C/C++ tracks ANSI C++ Standard working draft	3
option -Hsched — turn on	high-level and low-level scheduling	47
optimization — instruction scheduling	(high-level)	132
toggle	High_level_scheduling — perform instruction scheduling before register allocation	68, 132
predefined preprocessor macro symbol	__HIGHC__	21, 22
inline option	-Hih — show functions that have been inlined and/or defined	247
option	-Hinlsize — specify the maximum size of inlined functions	37
option	-Hipname — set the name of the environment variable containing the include file search path	38
inline option	-Hir — do not inline recursive functions	247
inline option	-His — inline functions with stack size less than n bytes	247
inline option	-Hit — inline functions with fewer than n tree nodes	247
inline option	-Hiw — invoke inliner in multi-module (wide compilation) mode	131, 248
inline option	-Hix — do not inline exported functions	248
option	-Hkanji — use Kanji scan tables	38
option	-Hkeep — do not delete object files at link time	38
option	-HL — compile in little-endian mode	38
option	-Hldopt — pass linker options to the linker	38
option	-Hlines — specify number of lines printed per page	39
option	-Hlist — write a source-code listing to standard output or to a file	13, 39, 231
option	-Hlm — enable left margin setting	39
option	-Hmake — write a list of makefile dependencies to standard output	25, 40

option	-Hmakeof — write a list of makefile dependencies to the named file	25, 40
option	-Hmscerr — use Microsoft C/C++ error message format	41, 287, 288
option	-Hmwlib — use the MetaWare (ANSI) C header files and library	41, 47
option	-Hnalib — allow extra-ANSI function calls without leading underscore	41
option	-Hno_sys_include_dir — exclude /usr/include from include search path(s)	43
option	-Hnocomdat — disable COMDAT solution for template instantiation	41
option	-Hnocplus — do not include C++ header files or link to C++ run-time library	42
option	-Hnocpp — use the inboard High C/C++ macro preprocessor	21, 42
option	-Hnonexcept_wrap — wrap include files to compile without exception handling	42, 162
option	-Hnoobject — do not generate object file and do not link	42
option	-Hnopro — do not read a profile	43
option	-Hnoswap — do not swap the driver in and out of memory	43
option	-Hobjdir — specify an object-file directory	43
option	-Hobjprefix — specify an object-file directory	43
option	-Hoff — disable (turn off) a toggle	43, 55
option	-Hon — enable (turn on) a toggle	43, 55
	host platform capabilities (embedded applications)	252
option	-Hpcc — place the compiler in PCC mode to relax some ANSI C Standard restrictions	43
option	-Hpm — create an OS/2 Presentation Manager application	44
option	-Hppc602 — enable PowerPC 602 mode	44
option	-Hpragma — invoke a pragma on the command line	44, 94, 291
option	-Hpro — specify a profile	22, 23, 44
option	-Hrel — omit absolute include files from dependency entries	25, 44
option	-Hretwc — add the warning count to the compiler's return code	45
option	-Hrm — enable right margin setting	46
option	-Hsched — turn on high-level and low-level scheduling	47
option	-Hsuffix — specify the object file extension	47

- option -Hsyslib — use the system C header files and library 41, 47
 - option -Hthread — compile and link thread-safe programs 47
 - option -Htmpprefix — set directory for temporary files 48
 - option -Htrap — disable default OS/2 exception handling .. 48
 - option -Hturboerr — use Borland Turbo C/C++ error message format 48, 288
 - option -Humake — write a list of makefile dependencies to a .u file 25, 48
 - option -Hunixerr — use AT&T Portable C Compiler error message format 49, 288
 - option -Hunroll — replace loops with loop-body code (unroll or unwind loops) 49
 - option -Hwformat — set format of warning messages . 49, 287
 - option -Hwide — specify multi-module inlining (wide compilation) 49, 131, 248
- I**
- option -I — specify an alternate directory to search for include files 50, 156
 - .i and .ii — default preprocessed file-name extension 9
 - optimization — in-order execution of I/O (disable) 131
 - file I/O errors 283
 - pragma Ident — insert a comment in the generated object file 98, 115
 - pragma Alias — assign an external name to an internal variable or function identifier 94, 108
 - pragma Global_aliasing_convention — aliasing convention; construct external object name for internal variable or function identifier 36, 98, 108
 - global identifiers 94
 - re-ordering machine instructions to avoid idle time 132
 - processor IEEE Standard 754 171
 - .i and .ii — default preprocessed file-name extension 9
 - toggle Import_vtabs — assume virtual function tables to be external for classes with inlined, pure virtual member functions 68
 - optimization — spill analysis (improved) 138
 - optimization — in-order execution of I/O (disable) 131
 - toggle Use_eieio — enforce in-order execution of input/output 89
 - toggle Forcedebug — place named SDI for any in-scope struct in current compilation unit 228
 - toggle Forcedebug — place named SDI for any in-scope struct in current compilation unit 67, 228
 - option -Hnocpp — use the inboard High C/C++ macro preprocessor 21, 42

option -l — specify a library to	include at link time	15, 50
option -Hcplus —	include C++ header files and link C++ run-time	
	library	31, 156
option -Hnocplus — do not	include C++ header files or link to C++ run-time	
	library	42
profile listed as an	include file	233
toggle Downshift_file_names — convert	include file names to all lowercase	63
option -Hipname — set the name of the		
environment variable containing the	include file search path	38
option -I — specify an alternate directory to		
search for	include files	50, 156
option -Hrel — omit absolute	include files from dependency entries	25, 44
displaying	include files in listings	233
option -Hnonexcept_wrap — wrap	include files to compile without exception	
	handling	42, 162
option -Hno_sys_include_dir — exclude		
/usr/include from	include search path(s)	43
option -H — display names of files	included during compilation	29
toggle Relative_includes — look up	included files relative to the directory of the	
	including file	84
option -Hc_wrap — interpret files	included in C++ modules as C, not C++, header files	33
direct and	indirect bases of a class	188
optimization — loop	induction variable elimination	134
toggle	Induction_analysis — readjust loop counter to	
	eliminate compares	69, 134
pragma	Init — place a call to each listed function in the	
	program's .init section	98, 256
function	__init() (embedded applications)	255
system-specific	initialization in a start-up module (embedded	
	applications)	253
structure	initialization nesting level	233
	initialization of data structures	106
pragma	Initialization_level — set initialization priority of	
	modules	99, 105
pragma Static_segment — name a section for		
storing	initialized static variables	104, 114
type qualifier	_Inline	130, 244
global functions declared extern	inline	69
generating inline assembly code with	inline assembler directive _ASM	147
generating	inline assembly code with inline assembler	
	directive _ASM	147
	inline class functions	157
toggle Recognize_library —	inline code for certain functions in the ANSI	
	Standard C library	83, 129

inliner option -Hix — do not toggle Inline_common — solve the single-copy problem for	inline exported functions	248
inliner option -Hic —	inline functions	69
inliner option -HiC —	inline functions called fewer than n times	246
	inline functions smaller than s and called fewer than n times	246
inliner option -Hia — do not	inline functions whose addresses are taken	246
inliner option -Hit —	inline functions with fewer than n tree nodes	247
inliner option -His —	inline functions with stack size less than n bytes ...	247
	inline functions within classes	69
inliner option -Hir — do not toggle	inline recursive functions	247
functions	Inline_common — solve the single-copy problem for inline functions	69
option -Hinlsize — specify the maximum size of	inlined across compilation units	157
debugging	inlined functions	37
toggle Import_vtabs — assume virtual function tables to be external for classes with	inlined member functions	187
invoking the	inlined routines	245
when to use the stand-alone	inlined, pure virtual member functions	68
how the	inliner	243
	inliner	244
	inliner	157
	inliner constructs unique names	249
	inliner option -Hi — invoke the optional stand-alone inliner	36, 129, 246
	inliner option -Hia — do not inline functions whose addresses are taken	246
	inliner option -Hib — specify size of temporary file buffer	246
	inliner option -Hic — inline functions called fewer than n times	246
	inliner option -HiC — inline functions smaller than s and called fewer than n times	246
	inliner option -Hih — show functions that have been inlined and/or defined	247
	inliner option -Hir — do not inline recursive functions	247
	inliner option -His — inline functions with stack size less than n bytes	247
	inliner option -Hit — inline functions with fewer than n tree nodes	247
	inliner option -Hiw — invoke inliner in multi-module (wide compilation) mode	131, 248

	inliner option -Hix — do not inline exported functions	248
overriding	inliner options that suppress inlining	250
	inliner pragma Off_inline — disable (turn off) inlining for specific functions	100, 249
	inliner pragma On_inline — enable (turn on) inlining for specific functions	100, 250
	inliner pragma Pop_inline — pop topmost stack entry for each specified function	102, 250
High C/C++ built-in	inliner versus the stand-alone inliner	156, 244
suppressing the	inliner with option -g	245
how the compiler selects routines for	inlining	245
manual versus automatic function	inlining	130
overriding inliner options that suppress	inlining	250
option -Hwide — specify multi-module	inlining (wide compilation)	49, 131, 248
optimization —	inlining ANSI standard C functions	129
	inlining C++ functions	250
problems with	inlining class destructors	187
inliner pragma Off_inline — disable (turn off)	inlining for specific functions	100, 249
inliner pragma On_inline — enable (turn on)	inlining for specific functions	100, 250
optimization —	inlining functions	129
	inlining functions can increase code size while decreasing execution time	129, 243
effect of	inlining on compile time and code size	131
rules for	inlining routines	245
calling versus	inlining transcendental math functions	86
activating automatic	inlining with -Hi options	244
controlling	inlining with pragmas	245
toggle Use_eieio — enforce in-order execution of	input/output	89
pragma Page —	insert page ejects in a source-code listing	101, 231
specifying	installation directory	237
	instantiating a class template	266
	instantiating a function template	167
sharing an	instantiation of a member function or data member	266
toggle Ensure_instantiation — force	instantiation of all template types	64, 269
toggle Auto_class_member_instantiation - enable automatic	instantiation of class templates	56, 266
toggle Auto_func_instantiation — enable automatic	instantiation of function templates	56, 264
pragma Ensure_instantiation — ensure	instantiation of specified template types	97, 268
pragma Exclude_instantiation — prevent	instantiation of specified template types	97, 268
turning off automatic template	instantiations	264

- full type checking for toggle
 - representation of sign of the remainder on toggle
 - Demote_sizes — allow == operator to compare
 - right shift of a signed heap
 - System V Application Binary
 - toggle Single_static_assignment — place
 - pragma Alias — assign an external name to an
 - pragma Global_aliasing_convention — aliasing convention; construct external object name for
 - optimization — compiler
 - toggle Live_dead_iterate — perform multiple
 - optimization — live/dead analysis
 - optimization — common subexpression elimination (local,
 - instantiations of templates 165
 - Int_function_warnings — warn about missing and mistyped returns 69
 - integer conversion 171
 - integer data types 170
 - integer division 171
 - integer types other than int 62
 - integral type 171
 - integrity checking 278
 - inter-module communication and public names 248
 - Interface, PowerPC Processor Supplement (ABI Supplement) xvi, 193
 - interfacing C and C++ with other languages 173
 - intermediate representation of code in single static assignment form 85
 - internal variable or function identifier 94, 108
 - internal variable or function identifier 36, 98, 108
 - intrinsic function _Alloca() 245
 - intrinsics 126
 - iterations of live/dead analysis 71
 - (iterative) 133
 - iterative) 126

J

- toggle Back_substitute_epilog — replace unconditional
- optimization — cross
- toggle Back_substitute_nodes — reduce
- jump to a function epilog with epilog sequence 57
- jumping (tail merging) 128
- jumping due to conditional blocks within a loop 57

K

- option -Hkanji — use
- toggle
- Kanji scan tables 38
- Keep_global_refs_in_loops — do not move expressions with global or external variables out of loops 70
- keyword asm renamed _Asm for enhanced ASM facility 151
- keyword typeid 85
- C++ keywords not reserved in Level 1 C++ 33
- option -kpic — generate position-independent code 50

option	-KPIC — generate position-independent code	50
L		
option	-l — specify a library to include at link time	15, 50
option	-L<name> — specify a directory to search for libraries specified with option -l	15, 50
High C extensions to C	language	1
High C/C++	Language Extensions Manual	xiv
interfacing C and C++ with other	languages	173
toggle Print_lattice_members — list class members in the class	lattice display	78
C++ keywords not reserved in	Level 1 C++	33
	level numbers in source-code listings	232
	lexical error messages	285
including run-time	libraries	15
linker search strategy for locating	libraries	15
linking run-time	libraries	14
locating MetaWare run-time	libraries	16
MetaWare and system	libraries	15
option -YP — specify default directories for locating	libraries	53
specifying	libraries	237
function	libraries (embedded applications)	252
start-up code for shared	libraries (embedded applications)	260
start-up code operations for shared	libraries (embedded applications)	261
option -L<name> — specify a directory to search for	libraries specified with option -l	15, 50
including C++ header files and linking the C++ run-time	library	236
option -Hcplus — include C++ header files and link C++ run-time	library	31, 156
option -Hdlllib — link with the DLL version of the High C/C++ run-time	library	35
option -Hmwlib — use the MetaWare (ANSI) C header files and	library	41, 47
option -Hnocplus — do not include C++ header files or link to C++ run-time	library	42
option -Hsyslib — use the system C header files and	library	41, 47
toggle Borland — relax some ARM C++ standards to conform to the Borland class	library	58
toggle Recognize_library — inline code for certain functions in the ANSI Standard C	library	83, 129

option -Hdll — create a dynamic link start-up module for a shared storing start-up module crt1.o in an archive	library (DLL)	34
MetaWare	library (embedded applications)	260
shared	library (embedded applications)	260
building a shared	library and header files	15
disabling compiler intrinsics and calling equivalent	library entry point (embedded applications)	260
	library for embedded applications	260
option -l — specify a	library functions	126
optimization — register	library functions malloc and free	273
toggle Loop_reg_rename — perform register	library to include at link time	15, 50
toggle Reduce_reg_contention — limit the	lifetime analysis	137
continuation of	lifetime analysis	72, 137
pragma Skip — insert blank	lifetime of registers to reduce spill code	84, 136, 138
option -Hlines — specify number of	line	238
option -c — generate object file but do not	lines in a source-code listing	104, 232
option -Hnoobject — do not generate object	lines printed per page	39
file and do not	link	27
using option -Hdef to	link	42
using the driver to compile and	link a DLL	34
option -Hcplus — include C++ header files and	link a program in one step	7
using the driver to compile and	link C++ run-time library	31, 156
option -Hdll — create a dynamic	link in one step	12
	link library (DLL)	34
	link order of crt1.o and crtn.o (embedded applications)	255
	link register	198
	link thread-safe programs	47
	link time	15
option -Hthread — compile and	link to C++ run-time library	42
option -l — specify a library to include at	link with the DLL version of the High C/C++ run-time library	35
option -Hnocplus — do not include C++ header	linkage	158
files or	linkage table (PLT)	202
option -Hdlllib —	linker	238
type-safe	linker	16, 31
procedure	LINKER — specify linker	237
environment variable TOOLSDIR — specify path	linker limitations	107
to assembler and	linker options on the driver command line	8
option -Hbatch — create a batch file to call	linker options to the linker	38
the compiler and		
configuration file variable		
external name clashes and		
passing		
option -Hldopt — pass		

specifying the optional	linker output load file	237
	linker search strategy for locating libraries	15
	linker's external symbol dictionary	107
option -Hdef — use a module-definition file		
for	linking	34
compiling and	linking C and C++ modules	155
	linking modules crt1.o and crtn.o with crt1.o	
	(embedded applications)	255
	linking run-time libraries	14
including C++ header files and	linking the C++ run-time library	236
free-chain	links	276
pragma Page — insert page ejects in a		
source-code	listing	101, 231
pragma Skip — insert blank lines in a		
source-code	listing	104, 232
toggle List — generate a source-code	listing	70, 231
	listing format	232
pragma Title — put a title at the top of each		
source-code	listing page	104, 232
	listing symbols exported from a DLL	34
option -Hlist — write a source-code	listing to standard output or to a file	13, 39, 231
option -Hanno — annotate assembly file	listing with lines from the source file	29
displaying include files in	listings	233
generating source and assembly	listings	13
level numbers in source-code	listings	232
	LIT type designates a read-only section	112
DATA,	LIT, TEXT, or COMMON data section type	96
pragma	Literals — name a section for storing literals and	
	static variables	99, 114
toggle Literals_in_code — put lengthy	literals in code space instead of data space	71, 127
	literals in data versus code space	83
toggle	Literals_in_code — put lengthy literals in code	
	space instead of data space	71, 127
option -HL — compile in	little-endian mode	38
optimization—	live/dead analysis	133
optimization—	live/dead analysis (iterative)	133
toggle	Live_dead_iterate — perform multiple iterations	
	of live/dead analysis	71
option -nocrt0 — do not	load crt0.o during the link	50
specifying the optional linker output	load file	237
	loader capabilities(embedded applications)	252
dynamic code	loading	127
customizing the compiler to a	local environment	93

	local static variable storage	185
optimization — common subexpression elimination	(local)	125
optimization — common subexpression elimination	(local, iterative)	126
toggle	Local_CSE_iterate — perform multiple iterations of local common subexpression elimination	71, 126
toggle Long_double_16 — provide support for 16-byte	long double types	71
	long long bit field	177
toggle Long_long_8 — provide support for eight-byte	long long types	72
toggle	Long_double_16 — provide support for 16-byte long double types	71
toggle	Long_enums — map type enum variables to an int-sized area of memory	44, 71, 175
toggle	Long_long_8 — provide support for eight-byte long long types	72
toggle Back_substitute_nodes — reduce jumping due to conditional blocks within a	loop	57
toggle Induction_analysis — readjust	loop counter to eliminate compares	69, 134
	loop factors	280
optimization —	loop induction variable elimination	134
optimization —	loop memory reference elimination	134
toggle Strength_reduce — perform	loop strength reduction	86, 134
optimization —	loop strength reduction	134
	loop unrolling usually increases code size	135
pragma	Loop_factor — unrolling or unwinding loops	99, 135, 279, 280
toggle	Loop_reg_rename — perform register lifetime analysis	72, 137
toggle Keep_global_refs_in_loops — do not move expressions with global or external variables out of	loops	70
option -Hunroll — replace	loops with loop-body code (unroll or unwind loops)	49
toggle Mem_refs_from_loop — replace variable memory references in	loops with register references	72, 134
option -Hsched — turn on high-level and	low-level scheduling	47
optimization — instruction scheduling	(low-level)	133
toggle	Low_level_scheduling — perform instruction scheduling after register allocation	72, 133
toggle Downshift_file_names — convert include file names to all	lowercase	63
	LR save word	194

M

option re-ordering	-M — specify the memory model	50
toggle Preload_args_from_memory — load function arguments into	machine instructions to avoid processor idle time ..	132
predefined	machine registers in the function prolog	78, 136
option -U — undefine a symbol	machine registers that can be modified in an _ASM statement	148
High C/C++	macro _OS2	22
option -Hcpp — use the system's outboard C	(macro name)	53
option -Hnocpp — use the inboard High C/C++	macro preprocessor	20
predefined preprocessor	macro preprocessor	21, 31
_Asm	macro preprocessor	21, 42
components of _Asm	macro symbols	21
option -A- — recognize only predefined assertions and	macro syntax	151
function _main() called from C++ function	macros	152
function _main() not called from a C function	macros that begin with two underscores	27
calling functions	main()	258
C++ static objects constructed before	main()	258
function	main() and _exit() from a start-up module (embedded applications)	254
function	main() and destroyed after main()	256
toggle Char_default_unsigned —	_main() called from C++ function main()	258
toggle	_main() not called from a C function main()	258
option -Hmake — write a list of	make char default type unsigned char	58
option -Hmake — write a list of	Make_externs_global — make objects with storage class extern global	72
option -Hmakeof — write a list of	makefile dependencies to a .u file	25, 48
library functions	makefile dependencies to standard output	25, 40
environment variable	makefile dependencies to the named file	25, 40
unmangle — name unmangler; converts	malloc and free	273
name	MALLOC_LEVEL- control level of heap integrity checking	278
toggle Const_in_code — determine how const static variables	mangled function and variable names into a form similar to C++ declaration syntax	158
toggle Long_enums —	mangling	157
compiler storage	map in storage	61, 127
	map type enum variables to an int-sized area of memory	44, 71, 175
	mapping	173
	mapping exception-handling code in memory (embedded applications)	262

storage	mapping for C++ class objects	186
toggle Print_var_info — display	mapping of auto-, register-, and argument-class variables	80
	mapping static or exported variables into a named data section or common block with pragma Data	110
how the compiler	maps auto- or register-class variables	172
how the compiler	maps variables to storage	184
option -Hlm — enable left	margin setting	39
option -Hrm — enable right	margin setting	46
converting division or modulo by an integer		
power of 2 to a shift or	masking operation	136
toggle Strict_ANSI_math — generate strict	math function calls	86
ANSI C Standard conforming code for standard	math function calls in single precision if possible	86
toggle Single_math_lib — do ANSI Standard C	math functions	86
calling versus inlining transcendental	Mem_refs_from_loop — replace variable	
toggle	memory references in loops with register	
	references	72, 134
reserved words _Packed and _Unpacked -	member alignment on individual struct or union	175
control	member function or data member	266
sharing an instantiation of a	member functions	187
inlined	member functions	187
pure virtual		
toggle Import_vtabs — assume virtual function	member functions	68
tables to be external for classes with		
inlined, pure virtual	member must be aligned	101
specifying maximum boundary on which an	members	181
unpacked structure	members	189
alignment of structure	members	175
allocating non-static data	members	62
padding to align	members in the class lattice display	78
toggle Define_static_members — supply		
implicit definition of static class data	members of virtual bases	65
toggle Print_lattice_members — list class	memory	136
toggle Fast_virtual_bases — speed up access	memory	43
to	memory	273
optimization — preloading arguments from	memory	44, 71, 175
option -Hnoswap — do not swap the driver in	memory error message	284
and out of	memory model	50
reclaiming		
toggle Long_enums — map type enum variables		
to an int-sized area of		
dynamic array out of		
option -M — specify the		

optimization — loop	memory reference elimination	134
toggle Mem_refs_from_loop — replace variable	memory references in loops with register	
	references	72, 134
optimization — cross jumping (tail	merging)	128
warning	message controls	289
changing the error	message format	287
option -Hmscerr — use Microsoft C/C++ error	message format	41, 287, 288
option -Hturboerr — use Borland Turbo C/C++		
error	message format	48, 288
option -Hunixerr — use AT&T Portable C		
Compiler error	message format	49, 288
High C/C++ error	message formats	287
compiler warning	message numbers	289
toggle Msgno — output error	message numbers	73, 288, 290
controlling warning	message output by level	289
controlling warning	messages	288
diagnostic	messages	283
lexical error	messages	285
option -Hefmat — set format of error	messages	36, 287
syntactic error	messages	286
controlling individual warning	messages by number	289
option -Hmscerr — use	Microsoft C/C++ error message format	41, 287, 288
toggle Optimize_for_space — produce code that	minimizes space	75, 125
optimization — code space	minimizing (optimizing for space)	125
	minimizing symbolic debug information (SDI)	
	with toggles Autodebug, Forcdebug, and	
	Nodebug	225
	mixing C and C++ modules	64
toggle Use_power_pc_mnemonics — use PowerPC		
assembler	mnemonics	89
option -M — specify the memory	model	50
relocatable object	module	6
storing start-up	module crt1.o in an archive library (embedded	
	applications)	260
	module-definition file extension .def	34
	module-definition file for linking	34
	module-definition file to create a DLL	34
option -Hdef — use a	modules	155
using a	modules	64
compiling and linking C and C++		
mixing C and C++		
pragma Initialization_level — set	modules	99, 105
initialization priority of	modules to call functions not compiled with High	
pragma Calling_convention — allow High C/C++	C/C++	96

converting division or modulo by an integer power of 2 to a shift or masking operation	136
prof command — generate execution profile and read	
toggle	mon.out
inliner option -Hiw — invoke inliner in	52
option -Hwide — specify	Msgno — output error message numbers 73, 288, 290
overloading	multi-module (wide compilation) mode
toggle Live_dead_iterate — perform	131, 248
toggle Local_CSE_iterate — perform	multi-module inlining (wide compilation)
converting	49, 131, 248
converting floating-point division to	multiple functions with the same name
toggle Use_reciprocal_for_divide — convert a	158
floating-point divide-by-constant to a	multiple iterations of live/dead analysis
functions	71
	multiple iterations of local common subexpression elimination
	71, 126
	multiplication by a constant to additions and shifts 136
	multiplication by a reciprocal
	128
	multiply-by-reciprocal
	90, 128
	_mw_register_ctor() and _mw_register_dtor()
	256
N	
external	name clashes and linker limitations
structured	107
unmangle —	name space unavailable in C
	107
	name unmangler; converts mangled function and variable names into a form similar to C++ declaration syntax
	158
mapping static or exported variables into a	named data section or common block with pragma Data
	110
toggle Autodebug — place	named SDI for a named struct in the struct's debug repository
	56, 226
toggle Forcedebug — place	named SDI for any in-scope struct in current compilation unit
	228
toggle Forcedebug — place	named SDI for any in-scope struct in current compilation unit
	67, 228
toggle Nodebug — do not generate SDI for any	named struct within toggle's scope
	74, 227
	naming code sections
	112
function	naming conventions in C
	216
function	naming conventions in C++
	217
	negative floating-point number
	171
toggle	Nested_types — make nested types invisible outside the class where they are declared
	73
statement	nesting level
	233
structure initialization	nesting level
	233
toggle	Noalias — no aliasing; assume all variables are non-aliasable
	73, 135

type qualifier	_Noalias — no aliasing; assume variable cannot be accessed or modified indirectly	144
option	-nocrt0 — do not load crt0.o during the link	50
minimizing symbolic debug information (SDI) with toggles Autodebug, Forcedebug, and toggle	Nodebug	225
effect of toggle Reference_all_functions — create a reference to every	Nodebug — do not generate SDI for any named struct within toggle's scope	74, 227
toggle Parm_warnings — issue warnings when arguments to a	non-exception-aware classes on clean-up	161
toggle Double_return — make	non-inline function in the program	84
allocating	non-prototype function do not match declared parameters	75
direct	non-prototype functions return type double, not type float	63
toggle	non-static data members	189
ignoring sequence	non-virtual bases	189
	non-volatile registers	197
	Non_FP_varargs — do not save floating-point argument registers	74
	notational and typographic conventions	xiii
	numbers in files with option -Hrm	46

O

option	-O — set optimization level	51
option	-o — specify executable, object, or assembly file name	51
option -S — produce an assembly source file instead of an	object file	53
including debugging information in the	object file	14
pragma Ident — insert a comment in the generated	object file	98, 115
option -Hnoobject — do not generate	object file and do not link	42
option -c — generate	object file but do not link	27
option -Hsuffix — specify the	object file extension	47
redundant SDI in	object files	225
relocatable binary	object files	12
option -Hkeep — do not delete	object files at link time	38
relocatable	object module	6
pragma Global_aliasing_convention — aliasing convention; construct external	object name for internal variable or function identifier	36, 98, 108
option -o — specify executable,	object, or assembly file name	51
option -Hobjdir — specify an	object-file directory	43

option -Hobjprefix — specify an	object-file directory	43
clean-up of completely constructed	objects	160
clean-up of partially constructed	objects	161
storage mapping for C++ class	objects	186
layout of	objects in storage	186
toggle Make_externs_global — make	objects with storage class extern global	72
option -H — display names	of files included during compilation	29
inliner pragma	Off_inline — disable (turn off) inlining for specific	
	functions	100, 249
global	offset table (GOT)	201
pragma	Offwarn — disable (turn off) specified warning	
	messages	100, 187, 289, 290
pragma Popwarn — cancel previous	Offwarn or Onwarn	102, 289
limitations of	OMF Common	167
inliner pragma	On_inline — enable (turn on) inlining for specific	
	functions	100, 250
pragma	Onwarn — enable (turn on) specified warning	
	messages	100, 289, 290
pragma	Onwarn_error — enable specified warning	
	messages as compiler error	
	messages	100, 285, 289, 290
widening an	operand	90
pragma OS_id — generate .osinfo section to	operating system	100
identify	operator strength reduction	136
optimization —	operator strength reductions that result in larger	
	code size	136
toggle Demote_sizes — allow ==	operator to compare integer types other than int	62
toggle Postfix_takes_two_args — force	operators to require two arguments	77
overloaded postfix	optimization	61, 128, 237
toggle Cross_jump — allow cross-jumping	optimization — code in source-code order	125
(tail-merging)	optimization — code space minimizing (optimizing	
	for space)	125
	optimization — common subexpression elimination	
	(global)	125
	optimization — common subexpression elimination	
	(local)	125
	optimization — common subexpression elimination	
	(local, iterative)	126
	optimization — compiler intrinsics	126
	optimization — constant expression folding	127
	optimization — constant propagation	127

optimization — constants in code	127
optimization — cross jumping (tail merging)	128
optimization — dead (unreachable) code elimination	128
optimization — expression simplification	128
optimization — in-order execution of I/O (disable)	131
optimization — inlining ANSI standard C functions	129
optimization — inlining functions	129
optimization — instruction scheduling (high-level)	132
optimization — instruction scheduling (low-level)	133
optimization — live/dead analysis	133
optimization — live/dead analysis (iterative)	133
optimization — loop induction variable elimination	134
optimization — loop memory reference elimination	134
optimization — loop strength reduction	134
optimization — operator strength reduction	136
optimization — preloading arguments from memory	136
optimization — register allocation (enhanced)	136
optimization — register allocation (global)	137
optimization — register lifetime analysis	137
optimization — single static assignment	137
optimization — small-data sections allocation	137
optimization — spill analysis (improved)	138
optimization — spill-code clean-up	138
optimization — spill-code reduction	138
optimization — tail recursion	139
option -O — set optimization level	51
specifying optimization levels	118
default optimizations	119
space-saving optimizations	128
performing additional optimizations not usually available	137
option -Os — optimize for smaller code	120
toggle Tail_recursion — optimize recursive functions where possible ..	87, 139
toggle Optimize_for_space — produce code that minimizes space	75, 125
debugging optimized code	122
optimization — code space minimizing (optimizing for space)	125
optimizing program performance	117

	option -A- — recognize only predefined assertions and macros that begin with two underscores	27
	option -c — generate object file but do not link	27
	option -D — define a name for preprocessor	28
	option -E — preprocess source file but do not compile; write output to standard output	28
	option -fdouble — specify not less than double precision floating-point arithmetic	28
	option -fsingle — specify single precision floating-point arithmetic	28
	option -fsoft — use emulation routines to perform floating-point arithmetic and conversions	28
suppressing the inliner with	option -g	245
	option -g — place debugging information in compiled code	29, 132, 223, 225
	option -H — display names of files included during compilation	29
	option -h — search help files	11, 29
	option -H+w — issue all relevant warning messages	49
	option -Hanno — annotate assembly file listing with lines from the source file	29
	option -Hansi — accept only ANSI C Standard conforming programs	29
	option -Hansi — accept only ANSI C Standard conforming programs	3
	option -Hasmcpp — process C-style preprocessing directives in assembly source file	30
	option -Hasopt — pass assembler options to the assembler	30
	option -HB — compile in big-endian mode	30
	option -Hbatch — create a batch file to call the compiler and linker	16, 31
	option -Hc_wrap — interpret files included in C++ modules as C, not C++, header files	33
	option -Hcplus — include C++ header files and link C++ run-time library	31, 156
	option -Hcpp — use the system's outboard C macro preprocessor	21, 31
using	option -Hcpp with a profile	23
	option -Hcppext — specify the C++ source file-name extensions	10, 32
	option -Hcpplvl — specify the level of C++ compilation	9, 32

	option -Hdef — use a module-definition file for linking	34
using	option -Hdef to link a DLL	34
	option -Hdll — create a dynamic link library (DLL)	34
	option -Hdllib — link with the DLL version of the High C/C++ run-time library	35
	option -Hefile — redirect output sent to stderr and send it to standard output or to a file	35
	option -Hefmat — set format of error messages	36, 287
	option -Hexcept — compile and link exception-aware code	36, 162
	option -Hgac — set a value for pragma Global_aliasing_convention	36, 109
inliner	option -Hi — invoke the optional stand-alone inliner	36, 129, 246
inliner	option -Hia — do not inline functions whose addresses are taken	246
inliner	option -Hib — specify size of temporary file buffer	246
inliner	option -Hic — inline functions called fewer than n times	246
inliner	option -HiC — inline functions smaller than s and called fewer than n times	246
inliner	option -Hih — show functions that have been inlined and/or defined	247
	option -Hinlsize — specify the maximum size of inlined functions	37
	option -Hipname — set the name of the environment variable containing the include file search path	38
inliner	option -Hir — do not inline recursive functions	247
inliner	option -His — inline functions with stack size less than n bytes	247
inliner	option -Hit — inline functions with fewer than n tree nodes	247
inliner	option -Hiw — invoke inliner in multi-module (wide compilation) mode	131, 248
inliner	option -Hix — do not inline exported functions	248
	option -Hkanji — use Kanji scan tables	38
	option -Hkeep — do not delete object files at link time	38
	option -HL — compile in little-endian mode	38
	option -Hldopt — pass linker options to the linker ..	38
	option -Hlines — specify number of lines printed per page	39

- option -Hlist — write a source-code listing to
standard output or to a file 13, 39, 231
- option -Hlm — enable left margin setting 39
- option -Hmake — write a list of makefile
dependencies to standard output 25, 40
- option -Hmakeof — write a list of makefile
dependencies to the named file 25, 40
- option -Hmscerr — use Microsoft C/C++ error
message format 41, 287, 288
- option -Hmwlib — use the MetaWare (ANSI) C
header files and library 41, 47
- option -Hnalib — allow extra-ANSI function calls
without leading underscore 41
- option -Hno_sys_include_dir — exclude
/usr/include from include search path(s) 43
- option -Hnocomdat — disable COMDAT solution
for template instantiation 41
- option -Hnocplus — do not include C++ header
files or link to C++ run-time library 42
- option -Hnocpp — use the inboard High C/C++
macro preprocessor 21, 42
- option -Hnonexcept_wrap — wrap include files to
compile without exception handling 42, 162
- option -Hnoobject — do not generate object file
and do not link 42
- option -Hnopro — do not read a profile 43
- option -Hnoswap — do not swap the driver in and
out of memory 43
- option -Hobjdir — specify an object-file directory .. 43
- option -Hobjprefix — specify an object-file
directory 43
- option -Hoff — disable (turn off) a toggle 43, 55
- option -Hon — enable (turn on) a toggle 43, 55
- option -Hpcc — place the compiler in PCC mode to
relax some ANSI C Standard restrictions 43
- option -Hpm — create an OS/2 Presentation
Manager application 44
- option -Hppc602 — enable PowerPC 602 mode 44
- option -Hpragma — invoke a pragma on the
command line 44, 94, 291
- option -Hpro — specify a profile 22, 23, 44
- option -Hrel — omit absolute include files from
dependency entries 25, 44
- option -Hretwc — add the warning count to the
compiler's return code 45

option -Hrm — enable right margin setting	46
option -Hsched — turn on high-level and low-level scheduling	47
option -Hsuffix — specify the object file extension .	47
option -Hsyslib — use the system C header files and library	41, 47
option -Hthread — compile and link thread-safe programs	47
option -Htmpprefix — set directory for temporary files	48
option -Htrap — disable default OS/2 exception handling	48
option -Hturboerr — use Borland Turbo C/C++ error message format	48, 288
option -Humake — write a list of makefile dependencies to a .u file	25, 48
option -Hunixerr — use AT&T Portable C Compiler error message format	49, 288
option -Hunroll — replace loops with loop-body code (unroll or unwind loops)	49
option -Hwformat — set format of warning messages	49, 287
option -Hwide — specify multi-module inlining (wide compilation)	49, 131, 248
option -I — specify an alternate directory to search for include files	50, 156
option -kpic — generate position-independent code	50
option -KPIC — generate position-independent code	50
option -l — specify a library to include at link time .	15
option -l — specify a library to include at link time .	50
option -L<name> — specify a directory to search for libraries specified with option -l	15, 50
option -M — specify the memory model	50
option -nocrt0 — do not load crt0.o during the link .	50
option -O — set optimization level	51
option -o — specify executable, object, or assembly file name	51
option -Os — optimize for smaller code	120
option -p — generate code to count calls to each function	52
option -P — preprocess source file but do not compile; write output to a file	10, 20, 52
option -pg — generate code to count calls to each function	52

option -pic — generate position-independent code ..	52
option -PIC — generate position-independent code .	52
option -S — produce an assembly source file instead of an object file	53
option -U — undefine a symbol (macro name)	53
option -V — display controls passed to the compiler; also display subprocesses	16, 53
option -w — set the compiler diagnostic warning level	53, 289
option -YP — specify default directories for locating libraries	53
compiler option reference	27
specifying the optional linker output load file	237
inline option -Hi — invoke the optional stand-alone inliner	36, 129, 246
driver configuration file versus command line options	18
specifying compiler options	8, 27
specifying values of options and toggles	236
passing assembler options on the driver command line	8
passing linker options on the driver command line	8
overriding inliner options that suppress inlining	250
option -Hasopt — pass assembler options to the assembler	30
option -Hldopt — pass linker options to the linker	38
option -Os — optimize for smaller code	120
option -Htrap — disable default OS/2 exception handling	48
option -Hpm — create an OS/2 Presentation Manager application	44
pragma OS_id — generate .osinfo section to identify operating system	100
predefined macro _OS2	22
pragma OS_id — generate .osinfo section to identify operating system	100
toggle Print_protos — write a prototype-style function header for every function to standard output	80
specifying the optional linker output load file	237
option -Hefile — redirect output sent to stderr and send it to standard output or to a file	35
option -P — preprocess source file but do not compile; write output to a file	10, 20, 52
option -E — preprocess source file but do not compile; write output to standard output	28
format strings for output-file names	54
format strings for output-file names	54
toggle overhead of function calls	243
Overload_info — show function chosen during overload operation	75

toggle Postfix_takes_two_args — force	overloaded postfix operators to require two arguments	77
	overloading multiple functions with the same name	158
using global register allocation to	override explicit register declarations	137
	overriding default compiler values	18, 93
	overriding inliner options that suppress inlining	250
	overriding pragmas and toggles in a profile	18
P		
option	-p — generate code to count calls to each function ..	52
option	-P — preprocess source file but do not compile; write output to a file	10, 20, 52
pragma	Pack — control default alignment of data structures	101
reserved words	_Packed and _Unpacked — control member alignment on individual struct or union	175
	padding to align members	175
pragma	Page — insert page ejects in a source-code listing	101, 231
PowerPC	parameter passing	199
matching const	parameters	167
toggle	Parm_warnings — issue warnings when arguments to a non-prototype function do not match declared parameters	75
clean-up of	partially constructed objects	161
clean-up of	partially constructed or destructed heap objects	160
	passing assembler options on the driver command line	8
	passing linker options on the driver command line	8
option -Hipname — set the name of the environment variable containing the include file search	path	38
environment variable TOOLSDIR — specify	path to assembler and linker	238
option -Hno_sys_include_dir — exclude /usr/include from include search	path(s)	43
absolute	pathname defined	45
toggle	PCC — relax ANSI C Standard restrictions to compile PCC programs	76
	PCC mode emulates AT&T Portable C Compiler	43, 76
option -Hpcc — place the compiler in	PCC mode to relax some ANSI C Standard restrictions	43
toggle	PCC_msgs — reduce compiler diagnostic capabilities to the PCC level	76

option	-pg — generate code to count calls to each function	52
option	-pic — generate position-independent code	52
option	-PIC — generate position-independent code	52
	pipelined architecture	279
procedure linkage table	(PLT)	202
small data area	pointer (PowerPC)	197
stack	pointer (PowerPC)	198
free-chain	pointer array	276
	pointer data types	172
allocating a	pointer to a virtual-function table	189
toggle <code>Pointers_compatible</code> — assign	pointer values to incompatible pointer variables	76, 77
referencing	pointer-based variables	57
toggle <code>Behaved</code> — specify that a program	pointer-based variables in a "well-behaved"	
handles	manner	57, 124
limitations in	pointer-to-member comparison	164
comparing virtual	pointer-to-member functions	164
free-list	pointers	276
type <code>ptrdiff_t</code> — difference of two	pointers	172
virtual-base	pointers	191
environment	pointers (PowerPC)	197
toggle	<code>Pointers_compatible</code> — assign pointer values to	
	incompatible pointer variables	76, 77
toggle	<code>Pointers_compatible_with_ints</code> — make any type	
	pointer compatible with integers	76, 77
pragma	<code>Pop</code> — reinstate prior status of a toggle	55, 101
inliner pragma <code>Pop_inline</code> —	pop topmost stack entry for each specified	
	function	102, 250
pragma	<code>Pop_align_members</code> — restore alignment prior to	
	earlier pragma <code>Push_align_members</code>	101, 184
pragma	<code>Pop_ignore</code> — reinstate prior ignore status of a	
	pragma	101
inliner pragma	<code>Pop_inline</code> — pop topmost stack entry for each	
	specified function	102, 250
pragma	<code>Pop_small_data</code> — reinstate prior small-data	
	specification	102
pragma	<code>Popwarn</code> — cancel previous <code>Offwarn</code> or	
	<code>Onwarn</code>	102, 289
ANSI C Standard conformance and	portability	3
PCC mode emulates AT&T	Portable C Compiler	43, 76
AT&T	Portable C Compiler and unsignedness preserving	
	rules	90
AT&T	Portable C Compiler diagnostic level	76
option <code>-Hunixerr</code> — use AT&T	Portable C Compiler error message format	49, 288

option -kpic — generate	position-independent code	50
option -KPIC — generate	position-independent code	50
option -PIC — generate	position-independent code	52
option -pic — generate	position-independent code	52
toggle	Postfix_takes_two_args — force overloaded postfix operators to require two arguments	77
option -Hppc602 — enable	PowerPC 602 mode	44
toggle Use_power_pc_mnemonics — use	PowerPC assembler mnemonics	89
	PowerPC Embedded Application Binary Interface (EABI)	xvi
	PowerPC function return values	210
	PowerPC function-calling sequence	193
core	PowerPC instruction set	251
	PowerPC models and features not supported by High C/C++	251
	PowerPC parameter passing	199
System V Application Binary Interface,	PowerPC Processor Supplement (ABI Supplement)	xvi, 193
	PowerPC prolog and epilog code	211
	PowerPC register save areas	199
	PowerPC registers	196
	PowerPC registers used in the standard calling sequence	198
	PowerPC run-time organization	193
	PowerPC stack-frame layout	193
	PowerPC variable argument lists	200
condition register	(PowerPC)	197
environment pointers	(PowerPC)	197
floating-point invalid exception	(PowerPC)	198
floating-point status and control register	(PowerPC)	197
small data area pointer	(PowerPC)	197
stack pointer	(PowerPC)	198
volatile registers	(PowerPC)	197, 198
small-data section	.PPC.EMB.sbss0	103, 204
small-data section	.PPC.EMB.sdata0	103, 204
	pragma Alias — assign an external name to an internal variable or function identifier	94, 108
use	pragma Alias to define how a public symbol appears in the symbol table	216
	pragma Align_members — set the maximum boundary for structure-member alignment	94, 123, 175, 181

- pragma Alloc_text — group functions in specified text segment 95
- pragma Called_infrequently — apply calling-convention attribute CALLED_INFREQUENTLY to functions containing a specified string in their name 95
- pragma Calling_convention — allow High C/C++ modules to call functions not compiled with High C/C++ 96
- pragma Code — specify the code section name 96, 113
- mapping static or exported variables into a named data section or common block with
 - pragma Data 110
 - pragma Data — allocate named blocks for data storage 96, 110
 - pragma Data_seg — put static and global variables in specified data segment 97
 - pragma Ensure_instantiation — ensure instantiation of specified template types 97, 268
 - pragma Exclude_instantiation — prevent instantiation of specified template types 97, 268
 - pragma Exit_expr — evaluate an expression at program termination 97, 107
 - pragma Fini — place a call to each listed function in the program's .fini section 98, 256
 - option -Hgac — set a value for
 - pragma Global_aliasing_convention 36, 109
 - pragma Global_aliasing_convention — aliasing convention; construct external object name for internal variable or function identifier 36, 98, 108
 - pragma Ident — insert a comment in the generated object file 98, 115
 - pragma Init — place a call to each listed function in the program's .init section 98, 256
 - pragma Initialization_level — set initialization priority of modules 99, 105
 - pragma Literals — name a section for storing literals and static variables 99, 114
 - pragma Loop_factor — unrolling or unwinding loops 99, 135, 279, 280
 - pragma Off — disable (turn off) a toggle 55, 99
 - inliner
 - pragma Off_inline — disable (turn off) inlining for specific functions 100, 249
 - pragma Offwarn — disable (turn off) specified warning messages 100, 187, 289, 290
 - pragma On — enable (turn on) a toggle 55, 100
 - option -Hpragma — invoke a
 - pragma on the command line 44, 94, 291

inliner	pragma On_inline — enable (turn on) inlining for specific functions	100, 250
	pragma Onwarn — enable (turn on) specified warning messages	100, 289, 290
	pragma Onwarn_error — enable specified warning messages as compiler error messages	100, 285, 289, 290
	pragma OS_id — generate .osinfo section to identify operating system	100
	pragma Pack — control default alignment of data structures	101
	pragma Page — insert page ejects in a source-code listing	101, 231
	pragma Pop — reinstate prior status of a toggle	55, 101
	pragma Pop_align_members — restore alignment prior to earlier pragma Push_align_members	101, 184
	pragma Pop_ignore — reinstate prior ignore status of a pragma	101
inliner	pragma Pop_inline — pop topmost stack entry for each specified function	102, 250
	pragma Pop_small_data — reinstate prior small-data specification	102
	pragma Popwarn — cancel previous Offwarn or Onwarn	102, 289
	pragma Push_align_members — set maximum boundary for structure-member alignment and save state	102, 182
	pragma Push_ignore — ignore pragmas not supported	102
	pragma Push_small_data — Assign data to small-data section	103
	pragma Skip — insert blank lines in a source-code listing	104, 232
	pragma Startup_expr — evaluate an expression at program start-up	104, 106
	pragma Static_segment — name a section for storing initialized static variables	104, 114
	pragma Title — put a title at the top of each source-code listing page	104, 232
	pragma Warning_level — set compiler diagnostic warning level	104, 289
	pragma Weak x = y — designate a weak global symbol and assign it a value	105
controlling inlining with	pragmas	245

customizing your program with	pragmas	105
setting compiler	pragmas	93
overriding	pragmas and toggles in a profile	18
compiler	pragmas defined	93
pragma Push_ignore — ignore	pragmas not supported	102
using	pragmas to align data	180
using	pragmas to turn toggles On and Off	55
	precedence of compiler controls	17, 18
option -fdouble — specify not less than	precision floating-point arithmetic	28
double	precision floating-point arithmetic	28
option -fsingle — specify single	precision floating-point arithmetic	62
toggle Double_math_only — perform double		
toggle Single_math_lib — do ANSI Standard C	precision if possible	86
math function calls in single	predefined assertions and macros that begin with	
option -A- — recognize only	two underscores	27
	predefined macro _OS2	22
	predefined preprocessor macro symbol	
	__EXCEPTIONS__	22
	predefined preprocessor macro symbol	
	__HIGHC__	21, 22
	predefined preprocessor macro symbol	
	__STDC__	21, 22
	predefined preprocessor macro symbol _BE_PPC ...	22
	predefined preprocessor macro symbols	21
	predefined preprocessor macro symbols	
	__CPLUSPLUS__ and __cplusplus	22
profile — ASCII file treated as a	prefix to the source file	22
when not to	preload arguments	136
toggle	Preload_args_from_memory — load function	
	arguments into machine registers in the	
	function prolog	78, 136
option -P —	preprocess source file but do not compile; write	
	output to a file	10, 20, 52
option -E —	preprocess source file but do not compile; write	
	output to standard output	28
.i and .ii — default	preprocessed file-name extension	9
option -Hasmcpp — process C-style	preprocessing directives in assembly source file	30
High C/C++ macro	preprocessor	20
option -D — define a name for	preprocessor	28
option -Hcpp — use the system's outboard C		
macro	preprocessor	21, 31
option -Hnocpp — use the inboard High C/C++		
macro	preprocessor	21, 42

UNIXcpp	preprocessor	237
unsigned expressions in conditional	preprocessor directives	172
predefined	preprocessor macro symbols	21
configuration file variable CPP — C	preprocessor variable	237
option -Hpm — create an OS/2	Presentation Manager application	44
toggle	Print_lattice_members — list class members in the class lattice display	78
toggle	Print_lattices — generate a graphical view of any class with one or more bases	79, 191
toggle	Print_layout — print layout of C++ data structures .	80
toggle	Print_protos — write a prototype-style function header for every function to standard output	80
toggle	Print_template_usage — display templates used in a C++ compilation	80, 270
toggle	Print_var_info — display mapping of auto-, register-, and argument-class variables	80
pragma Initialization_level — set initialization	priority of modules	99, 105
	priority queues of constructor_items and destructor_items	257
	procedure linkage table (PLT)	202
re-ordering machine instructions to avoid	processor idle time	132
	prof command — generate execution profile and read mon.out	52
option -Hnopro — do not read a	profile	43
option -Hpro — specify a	profile	22, 23, 44
overriding pragmas and toggles in a	profile	18
using option -Hcpp with a	profile	23
	profile — ASCII file treated as a prefix to the source file	22
gprof command — generate execution	profile and read gmon.out	52
prof command — generate execution	profile and read mon.out	52
	profile listed as an include file	233
how the compiler reads	profiles	23
toggle Behaved — specify that a	program handles pointer-based variables in a "well-behaved" manner	57, 124
type qualifier _Reversed_Endian — set endianness of variable opposite to that of the machine the	program runs on	144, 180
pragma Exit_expr — evaluate an expression at	program termination	97, 107
AT&T UNIX System V Release 4	Programmer's Guide: ANSI C and Programming Support Tools	201

toggle Preload_args_from_memory — load function arguments into machine registers in the function	prolog	78, 136
PowerPC	prolog and epilog code	211
toggle	Prolog_trace — generate information about a function being entered	81, 224
optimization — constant	propagation	127
toggle Print_protos — write a	prototype-style function header for every function to standard output	80
toggle	Prototype_conversion_warn — warn when a function argument is converted due to a prototype declaration	81
toggle	Prototype_override_warnings — warn when an ANSI Standard C declaration overrides an old-style function definition	82
type	ptrdiff_t — difference of two pointers	172
toggle Export_vtabs — make virtual function tables	public and defined	65
inter-module communication and	public names	248
use pragma Alias to define how a	public symbol appears in the symbol table	216
toggle Common_can_export — allow	public variables in common segments	61
toggle Import_vtabs — assume virtual function tables to be external for classes with inlined,	pure virtual member functions	187
pragma Pop_align_members — restore alignment prior to earlier pragma	pure virtual member functions	68
pragma	Push_align_members	101, 184
pragma	Push_align_members — set maximum boundary for structure-member alignment and save state	102, 182
pragma	Push_ignore — ignore pragmas not supported	102
pragma	Push_small_data — Assign data to small-data section	103

Q

casting a type	qualifier	143
type	qualifier _Alias — assume variable can be accessed or modified indirectly	144
type	qualifier _Inline	130, 244
type	qualifier _Noalias — no aliasing; assume variable cannot be accessed or modified indirectly	144
type	qualifier _Reversed_Endian — set endianness of variable opposite to that of the machine the program runs on	144, 180
type	qualifier const	99, 114

uninitialized global variables without extern	qualifier defined as individual common blocks	219
access-related type	qualifiers	143
endianness-related type	qualifiers	144
function type	qualifiers	145
special type	qualifiers	142
priority	queues of constructor_items and destructor_items .	257

R

	.rdata — default section for read-only string constants and read-only static variables	99, 185
	re-ordering machine instructions to avoid processor idle time	132
toggle Read_only_strings — store all string constants in the LIT type designates a	read-only data section	83, 127
	read-only section	112
	read-only static variables and read-only string constants	114
.rdata — default section for	read-only string constants and read-only static variables	99, 185
toggle	Read_only_strings — store all string constants in the read-only data section	83, 127
	readme file — identifies last minute changes and describes special files	xiv, 14
converting floating-point division to multiplication by a	reciprocal	128
	reclaiming memory	273
toggle	Recognize_library — inline code for certain functions in the ANSI Standard C library ...	83, 129
optimization — tail	recursion	139
inliner option -Hir — do not inline	recursive functions	247
toggle Tail_recursion — optimize	recursive functions where possible	87, 139
option -Hefile —	redirect output sent to stderr and send it to standard output or to a file	35
toggle	Reduce_reg_contention — limit the lifetime of registers to reduce spill code	84, 136, 138
optimization — loop strength	reduction	134
optimization — operator strength	reduction	136
optimization — spill-code	reduction	138
toggle Strength_reduce — perform loop strength	reduction	86, 134
operator strength	reductions that result in larger code size	136
	redundant SDI in object files	225
toggle	Reference_all_functions — create a reference to every non-inline function in the program	84

	referencing pointer-based variables	57
toggle	Reg_alloc_enhance — enhance register allocation	84, 136
link	register	198
condition	register (PowerPC)	197
floating-point status and control	register (PowerPC)	197
toggle High_level_scheduling — perform instruction scheduling before	register allocation	68, 132
toggle Low_level_scheduling — perform instruction scheduling after	register allocation	72, 133
toggle Reg_alloc_enhance — enhance	register allocation	84, 136
optimization —	register allocation (enhanced)	136
optimization —	register allocation (global)	137
using global	register allocation to override explicit register declarations	137
optimization —	register lifetime analysis	137
toggle Loop_reg_rename — perform	register lifetime analysis	72, 137
toggle Mem_refs_from_loop — replace variable memory references in loops with	register references	72, 134
PowerPC	register save areas	199
toggle Cleanup_spills — clean up	register spill code	60, 138
toggle Print_var_info — display mapping of	register-, and argument-class variables	80
auto-,	register-class variables	172
how the compiler maps auto- or	registers	197
non-volatile	registers	196
PowerPC	registers	67
toggle FP_varargs — always save floating-point argument	registers	74
toggle Non_FP_varargs — do not save floating-point argument	registers (PowerPC)	197, 198
volatile	registers in the function prolog	78, 136
toggle Preload_args_from_memory — load function arguments into machine	registers of a function	85, 209
toggle Saves_incoming_regs — save all incoming	registers that can be modified in an <code>_ASM</code> statement	148
machine	registers to reduce spill code	84, 136, 138
toggle Reduce_reg_contention — limit the lifetime of	registers used in the standard calling sequence	198
PowerPC	Relative_includes — look up included files relative to the directory of the including file	84
toggle	Release 4 Programmer's Guide: ANSI C and Programming Support Tools	201
AT&T UNIX System V		

	relocatable binary object files	12
	relocatable object module	6
	remainder on integer division	171
sign of the		
toggle Autodebug — place named SDI for a	repository	56, 226
named struct in the struct's debug	repository of a struct	226
debug	reserved words <code>_Packed</code> and <code>_Unpacked</code> —	
	control member alignment on individual struct	
	or union	175
driver argument or	response file	10
option -Hretwc — add the warning count to the		
compiler's	return code	45
toggle Double_return — make non-prototype	return type double, not type float	63
functions	return values	210
PowerPC function		
toggle Int_function_warnings — warn about	returns	69
missing and mistyped	reversed-endian variables	180
example declarations of	<code>_Reversed_Endian</code> — set endianness of variable	
type qualifier	opposite to that of the machine the program runs	
	on	144, 180
option -Hrm — enable	right margin setting	46
	right shift of a signed integral type	171
default	rounding mode	171
toggle RTTI_common — solve the single-copy	RTTI structures	85
problem for	(RTTI) for a class	85, 163
toggle Run_time_type_info — generate run-time	RTTI_common — solve the single-copy problem	
type information	for RTTI structures	85
toggle	run-time heap manager	273
	run-time libraries	15
including	run-time libraries	14
linking	run-time libraries	16
locating MetaWare		
including C++ header files and linking the	run-time library	236
C++		
option -Hcplus — include C++ header files and	run-time library	31, 156
link C++		
option -Hdlllib — link with the DLL version	run-time library	35
of the High C/C++		
option -Hnocplus — do not include C++ header	run-time library	42
files or link to C++	run-time organization	193
PowerPC	Run_time_type_info	163
when not to use toggle		

toggle	Run_time_type_info — generate run-time type information (RTTI) for a class	85, 163
S		
	.s — default assembly-language file-name extension	9
option	-S — produce an assembly source file instead of an object file	53
PowerPC register	save areas	199
LR	save word	194
toggle	Saves_incoming_regs — save all incoming registers of a function	85, 209
small-data section	.sbss	103, 205
small-data section	.sbss2	103, 205
option -Hkanji — use Kanji debugging fully	scan tables	38
option -Hsched — turn on high-level and low-level optimization — instruction optimization — instruction	scheduled code	132
toggle Low_level_scheduling — perform instruction	scheduling	47
toggle High_level_scheduling — perform instruction	scheduling (high-level)	132
	scheduling (low-level)	133
	scheduling after register allocation	72, 133
	scheduling before register allocation	68, 132
	scheduling code impairs symbolic debugging	132
	scope of compiler controls	18
small-data section	.sdata	103, 205
small-data section	.sdata2	103, 205
	SDI — symbolic debug information	225
toggle Autodebug — place named	SDI for a named struct in the struct's debug repository	56, 226
toggle Forcedebug — place named	SDI for any in-scope struct in current compilation unit	67, 228
toggle Nodebug — do not generate redundant minimizing symbolic debug information	SDI for any named struct within toggle's scope	74, 227
	SDI in object files	225
	(SDI) with toggles Autodebug, Forcedebug, and Nodebug	225
option -I — specify an alternate directory to option -L<name> — specify a directory to option -h — option -Hipname — set the name of the environment variable containing the include file	search for include files	50, 156
	search for libraries specified with option -l	15, 50
	search help files	11, 29
	search path	38

option -Hno_sys_include_dir — exclude /usr/include from include	search path(s)	43
linker	search strategy for locating libraries	15
global variables declared in the data	segment	61
pragma Alloc_text — group functions in specified text	segment	95
pragma Data_seg — put static and global variables in specified data	segment	97
32-bit data	segment _DATA	96
toggle Common_can_export — allow public variables in common	segments	61
option -Hefile — redirect output	sent to stderr and send it to standard output or to a file	35
PowerPC function-calling	sequence	193
toggle Back_substitute_epilog — replace unconditional jump to a function epilog with epilog	sequence	57
ignoring	sequence numbers in files with option -Hrm	46
pragma Initialization_level —	set initialization priority of modules	99, 105
start-up code for	shared libraries (embedded applications)	260
start-up code operations for	shared libraries (embedded applications)	261
start-up module for a	shared library (embedded applications)	260
building a	shared library entry point (embedded applications)	260
right	shared library for embedded applications	260
converting division or modulo by an integer	shift of a signed integral type	171
power of 2 to a	shift or masking operation	136
case	shifting external names in aliasing conventions	110
converting multiplication by a constant to additions and	shifts	136
toggle Use_UP_rules — widen unsigned char and unsigned	short to unsigned int	76, 90
	sign bit	171
	sign of the remainder on integer division	171
	signal functions	70
functions called during	signal handling	198
High C/C++ support of	signed and unsigned bit fields	176
toggle Char_is_rep — make default representation of char identical to	signed char or unsigned char	59
result of bitwise operation on	signed integer	171
right shift of a	signed integral type	171
optimization — expression	simplification	128
template	single-copy heuristic	163
	single-copy problem	166, 268

toggle Inline_common — solve the	single-copy problem for inline functions	69
toggle RTTI_common — solve the	single-copy problem for RTTI structures	85
toggle Template_common — solve the	single-copy problem for template functions and classes	87
toggle Vtable_common — solve the	single-copy problem for virtual function tables	91
toggle	Single_math_lib — do ANSI Standard C math function calls in single precision if possible	86
toggle	Single_static_assignment — place intermediate representation of code in single static assignment form	85
execution speed versus code	size	119
loop unrolling usually increases code	size	135
operator strength reductions that result in larger code	size	136
	size and alignment of enumeration types	175
	size and alignment of struct and union types	175
option -Hinlsize — specify the maximum	size of inlined functions	37
inliner option -Hib — specify	size of temporary file buffer	246
inlining functions can increase code	size while decreasing execution time	129, 243
	sizes and byte alignment of data types	173
pragma	Skip — insert blank lines in a source-code listing	104, 232
pragma Push_small_data — Assign data to	small-data section	103
	small-data section .PPC.EMB.sbss0	103, 204
	small-data section .PPC.EMB.sdata0	103, 204
	small-data section .sbss	103, 205
	small-data section .sbss2	103, 205
	small-data section .sdata	103, 205
	small-data section .sdata2	103, 205
optimization —	small-data sections allocation	137
pragma Pop_small_data — reinstate prior	small-data specification	102
option -Os — optimize for	smaller code	120
setting compiler controls in	source code	18
option -Hanno — annotate assembly file	source file	29
listing with lines from the	source file but do not compile; write output to a file	10, 20, 52
option -P — preprocess	source file but do not compile; write output to standard output	28
option -E — preprocess	source file instead of an object file	53
option -S — produce an assembly	source-code listing	101, 231
pragma Page — insert page ejects in a	source-code listing	104, 232
pragma Skip — insert blank lines in a	source-code listing	70, 231
toggle List — generate a		

pragma Title — put a title at the top of each	source-code listing page	104, 232
option -Hlist — write a	source-code listing to standard output or to	
level numbers in	a file	13, 39, 231
optimization — code in	source-code listings	232
toggle	source-code order	125
optimization — code space minimizing	Source_code_order — produce code in source order	
(optimizing for	where possible	86, 125
inline pragma Off_inline — disable (turn	space)	125
off) inlining for	space-saving optimizations	128
inline pragma On_inline — enable (turn on)	specific functions	100, 249
inlining for	specific functions	100, 250
inline pragma Pop_inline — pop topmost stack	specified function	102, 250
entry for each	specify linker	237
configuration file variable LINKER —	speed	43, 119
increasing compilation	speed	119, 128
increasing execution	speed up access to members of virtual bases	65
toggle Fast_virtual_bases —	speed versus code size	119
execution	spill code	60, 138
toggle Cleanup_spills — clean up register	spill code	84, 136, 138
toggle Reduce_reg_contention — limit the	spill-code clean-up	138
lifetime of registers to reduce	spill-code reduction	138
optimization —	stab format	61
optimization —	stack entry for each specified function	102, 250
toggle Dbx — generate debug information in	stack pointer (PowerPC)	198
SUN dbx	stack set-up in a start-up module (embedded	
inline pragma Pop_inline — pop topmost	applications)	253
inline option -His — inline functions with	stack size less than n bytes	247
PowerPC	stack-frame layout	193
High C/C++ built-in inliner versus the	stand-alone inliner	156, 244
inline option -Hi — invoke the optional	stand-alone inliner	36, 129, 246
when to use the	stand-alone inliner	157
trigraphs required for ANSI	Standard C compatibility	89
optimization — inlining ANSI	standard C functions	129
toggle Recognize_library — inline code for	Standard C library	83, 129
certain functions in the ANSI	Standard C math function calls in single precision	
toggle Single_math_lib — do ANSI	if possible	86
PowerPC registers used in the	standard calling sequence	198

toggle Strict_ANSI_math — generate strict ANSI C Standard conforming code for option -Hpcc — place the compiler in PCC mode to relax some ANSI C	standard math function calls	86
toggle Borland — relax some ARM C++	Standard restrictions	43
pragma Startup_expr — evaluate an expression at program	standards to conform to the Borland class library	58
special uses for	start-up	104, 106
	start-up code (embedded applications)	259
	start-up code for shared libraries (embedded applications)	260
	start-up code operations (embedded applications) .	258
	start-up code operations for shared libraries (embedded applications)	261
specifying the	start-up file	237
	start-up file entry point (embedded applications) ...	260
calling functions main() and _exit() from a	start-up module (embedded applications)	254
entry point of a	start-up module (embedded applications)	253
stack set-up in a	start-up module (embedded applications)	253
system-specific initialization in a	start-up module (embedded applications)	253
storing	start-up module crt1.o in an archive library (embedded applications)	260
	start-up module crt1.s (embedded applications)	254
example of source for	start-up module ctrl.o for embedded applications .	253
	start-up module for a shared library (embedded applications)	260
pragma	Startup_expr — evaluate an expression at program start-up	104, 106
machine registers that can be modified in an	statement	148
_ASM	statement (embedded applications)	254
DONE	statement nesting level	233
pragma Data_seg — put	static and global variables in specified data segment	97
optimization — single	static assignment	137
toggle Single_static_assignment — place	static assignment form	85
intermediate representation of code in single	static class data members	62
toggle Define_static_members — supply	static objects constructed before main() and destroyed after main()	256
implicit definition of	static or exported variables into a named data section or common block with pragma Data	110
C++	static variable storage	185
mapping	static variables	99, 185
local		
.rdata — default section for read-only string constants and read-only		

pragma Literals — name a section for storing literals and read-only	static variables	99, 114
.data — default section for uninitialized	static variables and read-only string constants	114
toggle Const_in_code — determine how const	static variables and writable strings referenced from code	184
toggle Try_static_cast — replace old-style type casts with	static variables map in storage	61, 127
pragma	static_cast	89, 164
floating-point	Static_segment — name a section for storing initialized static variables	104, 114
pragma Pop_ignore — reinstate prior ignore	status and control register (PowerPC)	197
pragma Pop — reinstate prior	status of a pragma	101
predefined preprocessor macro symbol	status of a toggle	55, 101
option -Hefile — redirect output sent to	__STDC__	21, 22
how the compiler maps variables to	stderr and send it to standard output or to a file	35
layout of objects in	storage	184
local static variable	storage	186
pragma Data — allocate named blocks for data	storage	185
toggle Const_in_code — determine how const	storage	96, 110
static variables map in	storage	61, 127
toggle Make_externs_global — make objects with	storage class extern global	72
compiler	storage for virtual bases	189
how arrays are	storage mapping	173
eliminating dead	storage mapping for C++ class objects	186
pragma Literals — name a section for	storage of arrays	176
optimization — loop	storage unit boundary for a bit field	177
optimization — operator	stored	176
toggle	stores	133
toggle Strict_ANSI_math — generate	storing literals and static variables	99, 114
toggle	strength reduction	134
read-only static variables and read-only	strength reduction	136
.rdata — default section for read-only	Strength_reduce — perform loop strength reduction	86, 134
toggle Read_only_strings — store all	strict ANSI C Standard conforming code for standard math function calls	86
	Strict_ANSI_math — generate strict ANSI C Standard conforming code for standard math function calls	86
	string constants	114
	string constants and read-only static variables	99, 185
	string constants in the read-only data section	83, 127

format	strings for output-file names	54
.data — default section for uninitialized static variables and writable	strings referenced from code	184
.data1 — default section for writable	strings whose addresses are referenced from within .data	185
debug repository of a	struct	226
size and alignment of	struct and union types	175
toggle Forcedebug — place named SDI for any in-scope	struct in current compilation unit	67, 228
toggle Autodebug — place named SDI for a named	struct in the struct's debug repository	56, 226
reserved words <code>_Packed</code> and <code>_Unpacked</code> - control member alignment on individual	struct or union	175
toggle Nodebug — do not generate SDI for any named	struct within toggle's scope	74, 227
bit fields affecting	structs	178
alignment of	structure initialization nesting level	233
pragma <code>Align_members</code> — set the maximum boundary for	structure members	181
pragma <code>Push_align_members</code> — set maximum boundary for anonymous	structure-member alignment	94, 123, 175, 181
initialization of data	structure-member alignment and save state ..	102, 182
pragma <code>Pack</code> — control default alignment of data	structure-member selection	76
toggle <code>Print_layout</code> — print layout of C++ data	structured name space unavailable in C	107
toggle <code>Local_CSE_iterate</code> — perform multiple iterations of local common	structures	106
optimization — common	structures	101
optimization — common	structures	80
optimization — common	structures, unions, and bit fields	172
toggle <code>Global_CSE</code> — eliminate global common	subexpression elimination	71, 126
global	subexpression elimination (global)	125
option <code>-V</code> — display controls passed to the compiler; also display	subexpression elimination (local)	125
toggle <code>Dbx</code> — generate debug information in	subexpression elimination (local, iterative)	126
option <code>-Hnoswap</code> — do not	subexpressions	68, 125
option <code>-U</code> — undefine a	subobject count	65, 160
pragma <code>Weak x = y</code> — designate a weak global	subprocesses	16, 53
	SUN dbx stab format	61
	swap the driver in and out of memory	43
	symbol (macro name)	53
	symbol <code>_GLOBAL_OFFSET_TABLE_</code>	202
	symbol and assign it a value	105

use pragma Alias to define how a public	symbol appears in the symbol table	216
use pragma Alias to define how a public	symbol appears in the symbol table	216
linker's external	symbol dictionary	107
external	symbol table	216
global variables in the	symbol table	219
SDI—	symbolic debug information	225
minimizing	symbolic debug information (SDI) with toggles	
scheduling code impairs	Autodebug, Forcedebug, and Nodebug	225
predefined preprocessor macro	symbolic debugging	132
listing	symbols	21
__Asm macro	symbols exported from a DLL	34
error and warning message	syntactic error messages	286
unmangle — name unmangler; converts mangled	syntax	151
function and variable names into a form	syntax	285
similar to C++ declaration		
pragma OS_id — generate .osinfo section to	syntax	158
identify operating	system	100
option -Hsyslib — use the	system C header files and library	41, 47
MetaWare and	system errors	284
AT&T UNIX	system libraries	15
	System V Application Binary Interface, PowerPC	
	Processor Supplement (ABI Supplement) .	xvi, 193
	System V Release 4 Application Binary Interface	
	(ABI)	xvi
	System V Release 4 Programmer's Guide:	
	ANSI C and Programming Support Tools	201
	system-specific initialization in a start-up module	
	(embedded applications)	253

T

toggle Import_vtabs — assume virtual function	tables to be external for classes with inlined, pure	
	virtual member functions	68
toggle Tag_table — generate tag table in	.tag section for each function	87, 209
separate	tag table in .text section for each function	87, 209
toggle Tag_table_in_text — generate	tag table in separate .tag section for each	
toggle Tag_table — generate	function	87, 209
toggle	Tag_table — generate tag table in separate .tag	
toggle	section for each function	87, 209
toggle	Tag_table_in_text — generate tag table in .text	
toggle	section for each function	87, 209
optimization — cross jumping	(tail merging)	128

optimization—	tail recursion	139
toggle Cross_jump — allow cross-jumping	(tail-merging) optimization	61, 128, 237
toggle	Tail_recursion — optimize recursive functions where possible	87, 139
instantiating a class	template	266
instantiating a function	template	167
functions in	template classes	156
	template classes defined	165
toggle Template_common — solve the single-copy problem for	template functions and classes	87
	template functions defined	165
option -Hnocomdat — disable COMDAT solution for	template instantiation	41
turning off automatic	template instantiations	264
	template single-copy problem	166, 268
pragma Ensure_instantiation — ensure instantiation of specified	template types	97, 268
pragma Exclude_instantiation — prevent instantiation of specified	template types	97, 268
toggle Ensure_instantiation — force instantiation of all	template types	64, 269
toggle	Template_common — solve the single-copy problem for template functions and classes	87
toggle	Template_trivial_conversions — allow trivial conversions for matching arguments to function templates	88, 168
C++	templates	165
defining generic functions or classes with	templates	165
full type checking for instantiations of	templates	165
problems with C++	templates	167
toggle Auto_class_member_instantiation - enable automatic instantiation of class	templates	56, 266
toggle Auto_func_instantiation — enable automatic instantiation of function	templates	56, 264
class	templates and duplicate definitions	266
class	templates defined	165
function	templates defined	165
toggle Print_template_usage — display	templates used in a C++ compilation	80, 270
inliner option -Hib — specify size of	temporary file buffer	246
directory for	temporary files	237
environment variable TMPPREFIX- specify alternate directory for	temporary files	237
option -Htmpprefix — set directory for	temporary files	48
	.text — default section for executable code	185

toggle Tag_table_in_text — generate tag table in	.text section for each function	87, 209
pragma Alloc_text — group functions in specified	text segment	95
DATA, LIT,	TEXT, or COMMON data section type	96
option -Hthread — compile and link	thread-safe programs	47
pragma	Title — put a title at the top of each source-code listing page	104, 232
environment variable	TMPPREFIX- specify alternate directory for temporary files	237
option -Hoff — disable (turn off) a	toggle	43, 55
option -Hon — enable (turn on) a	toggle	43, 55
pragma Off — disable (turn off) a	toggle	55, 99
pragma On — enable (turn on) a	toggle	55, 100
pragma Pop — reinstate prior status of a	toggle	55, 101
	toggle Auto_class_member_instantiation — enable automatic instantiation of class templates	56, 266
	toggle Auto_func_instantiation — enable automatic instantiation of function templates	56, 264
	toggle Autodebug — place named SDI for a named struct in the struct's debug repository	56, 226
	toggle Back_substitute_epilog — replace unconditional jump to a function epilog with epilog sequence	57
	toggle Back_substitute_nodes — reduce jumping due to conditional blocks within a loop	57
	toggle Behaved — specify that a program handles pointer-based variables in a "well-behaved" manner	57, 124
	toggle Borland — relax some ARM C++ standards to conform to the Borland class library	58
	toggle Call_trace — generate information about a called function	58, 224
	toggle Char_default_unsigned — make char default type unsigned char	58
	toggle Char_is_rep — make default representation of char identical to signed char or unsigned char ..	59
	toggle Cleanup_spills — clean up register spill code	60, 138
	toggle Command_line_ident — put information in a comment section in generated code	60, 115
	toggle Common_can_export — allow public variables in common segments	61

- toggle `Const_in_code` — determine how const static variables map in storage 61, 127
- toggle `Cross_jump` — allow cross-jumping (tail-merging) optimization 61, 128, 237
- toggle `Dbx` — generate debug information in SUN dbx stab format 61
- toggle `Define_static_members` — supply implicit definition of static class data members 62
- toggle `Demote_sizes` — allow `==` operator to compare integer types other than `int` 62
- toggle `Double_math_only` — perform double precision floating-point arithmetic 62
- toggle `Double_return` — make non-prototype functions return type `double`, not type `float` 63
- toggle `Downshift_file_names` — convert include file names to all lowercase 63
- toggle `Dwarf` — generate debug information in DWARF format 63
- toggle `Empty_is_void` — force C++ (void) interpretation of function declarations with no arguments 34, 64
- toggle `Enforce_access_control` — allow compiler-enforced access control 64
- toggle `Ensure_instantiation` — force instantiation of all template types 64, 269
- toggle `Epilog_trace` — generate information about a function being exited 65, 224
- toggle `Exception_aware_class` — compile exception-aware classes 36, 42, 65, 160, 162
- toggle `Export_vtabs` — make virtual function tables public and defined 65
- toggle `Fast_virtual_bases` — speed up access to members of virtual bases 65
- toggle `Forceddebug` — place named SDI for any in-scope struct in current compilation unit . 67, 228
- toggle `FP_varargs` — always save floating-point argument registers 67
- toggle `Global_CSE` — eliminate global common subexpressions 68, 125
- toggle `Globals_volatile` — see toggle `Keep_global_refs_in_loops` 68
- toggle `High_level_scheduling` — perform instruction scheduling before register allocation 68, 132

toggle Import_vtabs — assume virtual function tables to be external for classes with inlined, pure virtual member functions	68
toggle Induction_analysis — readjust loop counter to eliminate compares	69, 134
toggle Inline_common — solve the single-copy problem for inline functions	69
toggle Int_function_warnings — warn about missing and mistyped returns	69
toggle Keep_global_refs_in_loops — do not move expressions with global or external variables out of loops	70
toggle List — generate a source-code listing ..	70, 231
toggle Literals_in_code — put lengthy literals in code space instead of data space	71, 127
toggle Live_dead_iterate — perform multiple iterations of live/dead analysis	71
toggle Local_CSE_iterate — perform multiple iterations of local common subexpression elimination	71, 126
toggle Long_double_16 — provide support for 16-byte long double types	71
toggle Long_enums — map type enum variables to an int-sized area of memory	44, 71, 175
toggle Long_long_8 — provide support for eight-byte long long types	72
toggle Loop_reg_rename — perform register lifetime analysis	72, 137
toggle Low_level_scheduling — perform instruction scheduling after register allocation	72, 133
toggle Make_externs_global — make objects with storage class extern global	72
toggle Mem_refs_from_loop — replace variable memory references in loops with register references	72, 134
toggle Msgno — output error message numbers	73, 288, 290
toggle Nested_types — make nested types invisible outside the class where they are declared	73
toggle Noalias — no aliasing; assume all variables are non-aliasable	73, 135
toggle Nodebug — do not generate SDI for any named struct within toggle's scope	74, 227
toggle Non_FP_varargs — do not save floating-point argument registers	74

- toggle Optimize_for_space — produce code that
minimizes space 75, 125
- toggle Overload_info — show function chosen
during overload operation 75
- toggle Parm_warnings — issue warnings when
arguments to a non-prototype function do not
match declared parameters 75
- toggle PCC — relax ANSI C Standard restrictions
to compile PCC programs 76
- toggle PCC_msgs — reduce compiler diagnostic
capabilities to the PCC level 76
- toggle Pointers_compatible — assign pointer
values to incompatible pointer variables 76, 77
- toggle Pointers_compatible_with_ints — make
any type pointer compatible with integers 76, 77
- toggle Postfix_takes_two_args — force overloaded
postfix operators to require two arguments 77
- toggle Preload_args_from_memory — load
function arguments into machine registers
in the function prolog 78, 136
- toggle Print_lattice_members — list class
members in the class lattice display 78
- toggle Print_lattices — generate a graphical view
of any class with one or more bases 79, 191
- toggle Print_layout — print layout of C++ data
structures 80
- toggle Print_protos — write a prototype-style
function header for every function to standard
output 80
- toggle Print_template_usage — display templates
used in a C++ compilation 80, 270
- toggle Print_var_info — display mapping of auto-,
register-, and argument-class variables 80
- toggle Prolog_trace — generate information about
a function being entered 81, 224
- toggle Prototype_conversion_warn — warn when a
function argument is converted due to a prototype
declaration 81
- toggle Prototype_override_warnings — warn
when an ANSI Standard C declaration overrides
an old-style function definition 82
- toggle Read_only_strings — store all string
constants in the read-only data section 83, 127
- toggle Recognize_library — inline code for certain
functions in the ANSI Standard C library ... 83, 129

	toggle Reduce_reg_contention — limit the lifetime of registers to reduce spill code	84, 136, 138
	toggle Reference_all_functions — create a reference to every non-inline function in the program	84
	toggle Reg_alloc_enhance — enhance register allocation	84, 136
	toggle Relative_includes — look up included files relative to the directory of the including file	84
	toggle RTTI_common — solve the single-copy problem for RTTI structures	85
when not to use	toggle Run_time_type_info	163
	toggle Run_time_type_info — generate run-time type information (RTTI) for a class	85, 163
	toggle Saves_incoming_regs — save all incoming registers of a function	85, 209
	toggle Single_math_lib — do ANSI Standard C math function calls in single precision if possible	86
	toggle Single_static_assignment — place intermediate representation of code in single static assignment form	85
	toggle Source_code_order — produce code in source order where possible	86, 125
	toggle Strength_reduce — perform loop strength reduction	86, 134
	toggle Strict_ANSI_math — generate strict ANSI C Standard conforming code for standard math function calls	86
	toggle Tag_table — generate tag table in separate .tag section for each function	87, 209
	toggle Tag_table_in_text — generate tag table in .text section for each function	87, 209
	toggle Tail_recursion — optimize recursive functions where possible	87, 139
	toggle Template_common — solve the single-copy problem for template functions and classes	87
	toggle Template_trivial_conversions — allow trivial conversions for matching arguments to function templates	88, 168
	toggle Trap_div_by_zero — trap division by zero ...	88
	toggle Trigraphs — enable trigraphs	88
	toggle Try_static_cast — replace old-style type casts with static_cast	89, 164
	toggle Use_eieio — enforce in-order execution of input/output	89
	toggle Use_power_pc_mnemonics — use PowerPC assembler mnemonics	89

	toggle Use_reciprocal_for_divide — convert a floating-point divide-by-constant to a multiply-by-reciprocal	90, 128
	toggle Use_UP_rules — widen unsigned char and unsigned short to unsigned int	76, 90
	toggle VP_UP_warn — warn about expressions with more than one value, depending on whether UP or VP rules are used	91
	toggle Vtable_common — solve the single-copy problem for virtual function tables	91
	toggle Widen_float_args — widen float arguments to double	91
specifying compiler	toggles	55
specifying values of options and minimizing symbolic debug information (SDI)	toggles	236
with compiler	toggles Autodebug, Forcdebug, and Nodebug	225
overriding pragmas and	toggles defined	55
using pragmas to turn	toggles in a profile	18
defining a variable as a string of	toggles On and Off	55
debugging and diagnostic	tokens	238
environment variable	tools	223
	TOOLS DIR — specify path to assembler and linker	238
environment variable	TOOLS DIR cify path to assembler and linker	238
execution process	traces	53
function-call	tracing	224
calling versus inlining	transcendental math functions	86
toggle Trap_div_by_zero —	trap division by zero	88
toggle	Trap_div_by_zero — trap division by zero	88
inliner option -Hit — inline functions with fewer than n	tree nodes	247
toggle	Trigraphs — enable trigraphs	88
	trigraphs required for ANSI Standard C compatibility	89
toggle Template_trivial_conversions — allow	trivial conversions for matching arguments to function templates	88, 168
	truncation in aliasing conventions	110
toggle	Try_static_cast — replace old-style type casts with static_cast	89, 164
pragma Off — disable	(turn off) a toggle	55, 99
inliner pragma Off_inline — disable	(turn off) inlining for specific functions	100, 249
pragma Offwarn — disable	(turn off) specified warning messages	100, 187, 289, 290
pragma On — enable	(turn on) a toggle	55, 100

inline pragma On_inline — enable	(turn on) inlining for specific functions	100, 250
pragma Onwarn — enable	(turn on) specified warning messages .	100, 289, 290
DATA, LIT, TEXT, or COMMON data section	twos-complement binary form	170
right shift of a signed integral	type	96
toggle Try_static_cast — replace old-style	type	171
full	type casts with static_cast	89, 164
COMMON	type checking for instantiations of templates	165
DATA	type defines a common block	111
CODE	type denotes a writable data section	112
LIT	type denotes an executable code section	111
toggle Double_return — make non-prototype	type designates a read-only section	112
functions return	type double, not type float	63
toggle Double_return — make non-prototype	type double, not type float	63
functions return	type double, not type float	63
toggle Long_enums — map	type enum variables to an int-sized area of	
	memory	44, 71, 175
toggle Run_time_type_info — generate run-time	type information (RTTI) for a class	85, 163
	type ptrdiff_t — difference of two pointers	172
casting a	type qualifier	143
	type qualifier _Alias — assume variable can be	
	accessed or modified indirectly	144
	type qualifier _Inline	130, 244
	type qualifier _Noalias — no aliasing; assume	
	variable cannot be accessed or modified	
	indirectly	144
	type qualifier _Reversed_Endian — set endianness	
	of variable opposite to that of the machine the	
	program runs on	144, 180
	type qualifier const	99, 114
access-related	type qualifiers	143
endianness-related	type qualifiers	144
function	type qualifiers	145
special	type qualifiers	142
	type-safe linkage	158
keyword	typeid	85
implementation of data	types	169
pointer data	types	172
pragma Ensure_instantiation — ensure	types	97, 268
instantiation of specified template		
pragma Exclude_instantiation — prevent	types	97, 268
instantiation of specified template		
representation of integer data	types	170

size and alignment of enumeration	types	175
size and alignment of struct and union	types	175
sizes and byte alignment of data	types	173
toggle <code>Ensure_instantiation</code> — force instantiation of all template	types	64, 269
toggle <code>Long_double_16</code> — provide support for 16-byte long double	types	71
toggle <code>Long_long_8</code> — provide support for eight-byte long long	types	72
width and range of values of bit-field data	types	176
toggle <code>Nested_types</code> — make nested	types described	169
toggle <code>Demote_sizes</code> — allow <code>==</code> operator to compare integer notational and	types invisible outside the class where they are declared	73
	types other than int	62
	typographic conventions	xiii

U

option	-U — undefine a symbol (macro name)	53
option <code>-Hmake</code> — write a list of makefile dependencies to a	.u file	25, 48
toggle <code>Back_substitute_epilog</code> — replace	unconditional jump to a function epilog with epilog sequence	57
	undeclared functions	291
option -U —	undefine a symbol (macro name)	53
	undefined attributes	185
option <code>-Hnolib</code> — allow extra-ANSI function calls without leading	underscore	41
option <code>-A-</code> — recognize only predefined assertions and macros that begin with two	underscores	27
.data — default section for	uninitialized global variables without extern qualifier defined as individual common blocks ..	219
.bss — default section for	uninitialized static variables and writable strings referenced from code	184
reserved words <code>_Packed</code> and <code>_Unpacked</code> - control member alignment on individual struct	uninitialized variables (not const qualified)	184
or	union	175
size and alignment of struct and structures,	union types	175
toggle <code>Forceddebug</code> — place named SDI for any in-scope struct in current compilation	unions, and bit fields	172
	unit	67, 228
	UNIX cpp preprocessor	237

AT&T	UNIX System V Release 4 Programmer's Guide: ANSI C and Programming Support Tools	201
invoking the C++ name	unmangler	157
invoking the name	unmangler	158
examples of name	unmangling	159
reserved words <code>_Packed</code> and	<code>_Unpacked</code> — control member alignment on individual struct or union	175
specifying maximum boundary on which an	unpacked structure member must be aligned	101
optimization — dead	(unreachable) code elimination	128
option <code>-Hunroll</code> — replace loops with	loop-body code (unroll or unwind loops)	49
pragma <code>Loop_factor</code> —	unrolling or unwinding loops 99, 135, 279, 280	
loop	unrolling usually increases code size	135
High C/C++ support of signed and	unsigned bit fields	176
toggle <code>Char_is_rep</code> — make default		
representation of char identical to signed		
char or	unsigned char	59
toggle <code>Use_UP_rules</code> — widen	unsigned char and unsigned short to unsigned int	76, 90
	unsigned expressions in conditional preprocessor directives	172
AT&T Portable C Compiler and	unsignedness preserving rules	90
UP—	unsignedness preserving rules	76, 90
option <code>-Hunroll</code> — replace loops with		
loop-body code (unroll or	unwind loops)	49
pragma <code>Loop_factor</code> — unrolling or	unwinding loops 99, 135, 279, 280	
	UP — unsignedness preserving rules	76, 90
toggle <code>VP_UP_warn</code> — warn about expressions		
with more than one value, depending on		
whether	UP or VP rules are used	91
toggle	<code>Use_eieio</code> — enforce in-order execution of input/ output	89
toggle	<code>Use_power_pc_mnemonics</code> — use PowerPC assembler mnemonics	89
toggle	<code>Use_reciprocal_for_divide</code> — convert a floating- point divide-by-constant to a multiply-by- reciprocal	90, 128
toggle	<code>Use_UP_rules</code> — widen unsigned char and unsigned short to unsigned int	76, 90
	user-modifiable driver configuration-file variables	235
special	uses for start-up code (embedded applications)	259
	using option <code>-Hnoobject</code> to check for compilation errors	42
option <code>-Hno_sys_include_dir</code> — exclude	<code>/usr/include</code> from include search path(s)	43

V

option	-V — display controls passed to the compiler; also display subprocesses	16, 53
PowerPC	variable argument lists	200
configuration file	variable AS — specify assembler	236
defining a	variable as a string of tokens	238
type qualifier <code>_Alias</code> — assume	variable can be accessed or modified indirectly	144
type qualifier <code>_Noalias</code> — no aliasing; assume	variable cannot be accessed or modified indirectly	144
option <code>-Hipname</code> — set the name of the		
environment	variable containing the include file search path	38
environment	variable <code>CPLUS</code> — specify C++ compilation	155
configuration file	variable <code>CPP</code> — C preprocessor variable	237
optimization — loop induction	variable elimination	134
environment	variable <code>HC_DUMPREG</code>	48
configuration file	variable <code>LINKER</code> — specify linker	237
toggle <code>Mem_refs_from_loop</code> — replace	variable memory references in loops with register references	72, 134
unmangle — name unmangler; converts mangled		
function and	variable names into a form similar to C++ declaration syntax	158
type qualifier <code>_Reversed_Endian</code> — set		
endianness of	variable opposite to that of the machine the program runs on	144, 180
pragma <code>Alias</code> — assign an external name to an		
internal	variable or function identifier	94, 108
pragma <code>Global_aliasing_convention</code> — aliasing		
convention; construct external object name	variable or function identifier	36, 98, 108
for internal	variable storage	185
local static		
<code>.rdata</code> — default section for read-only string	variables	99, 185
constants and read-only static	variables	180
example declarations of reversed-endian	variables	172
how the compiler maps auto- or register-class		
pragma <code>Literals</code> — name a section for storing		
literals and static	variables	99, 114
pragma <code>Static_segment</code> — name a section for		
storing initialized static	variables	104, 114
referencing pointer-based	variables	57
setting compile-time environment	variables	19
toggle <code>Pointers_compatible</code> — assign pointer		
values to incompatible pointer	variables	76, 77
toggle <code>Print_var_info</code> — display mapping of		
auto-, register-, and argument-class	variables	80

user-modifiable driver configuration-file	variables	235
.bss — default section for uninitialized data communication — sharing	variables (not const qualified)	184
read-only static	variables across C and assembly	219
.data — default section for uninitialized static	variables and read-only string constants	114
toggle Noalias — no aliasing; assume all global	variables and writable strings referenced from code	184
toggle Behaved — specify that a program handles pointer-based	variables are non-aliasable	73, 135
toggle Common_can_export — allow public	variables declared in the data segment	61
pragma Data_seg — put static and global	variables in a "well-behaved" manner	57, 124
global	variables in common segments	61
mapping static or exported	variables in specified data segment	97
toggle Const_in_code — determine how const static	variables in the symbol table	219
toggle Keep_global_refs_in_loops — do not move expressions with global or external	variables into a named data section or common block with pragma Data	110
toggle Long_enums — map type enum how the compiler maps uninitialized global	variables map in storage	61, 127
storage for	variables out of loops	70
toggle Fast_virtual_bases — speed up access to members of	variables to an int-sized area of memory 44, 71, 175	
toggle Vtable_common — solve the single-copy problem for	variables to storage	184
toggle Export_vtabs — make	variables without extern qualifier defined as individual common blocks	219
toggle Import_vtabs — assume	virtual bases	189
pure	virtual bases	65
comparing	virtual function tables	91
allocating a pointer to a	virtual function tables	186
toggle Empty_is_void — force C++	virtual function tables public and defined	65
	virtual function tables to be external for classes with inlined, pure virtual member functions	68
	virtual functions	186
	virtual member functions	187
	virtual pointer-to-member functions	164
	virtual-base pointers	191
	virtual-function table	189
	(void) interpretation of function declarations with no arguments	34, 64
	volatile registers (PowerPC)	197, 198
	VP — value preserving rules	90
toggle	VP_UP_warn — warn about expressions with more than one value, depending on whether UP or VP rules are used	91

toggle Vtable_common — solve the single-copy problem
for virtual function tables 91

W

option -w — set the compiler diagnostic warning
level 53, 289

toggle VP_UP_warn — warn about expressions with more than one value,
depending on whether UP or VP rules are used 91

toggle Int_function_warnings — warn about missing and mistyped returns 69

toggle Prototype_conversion_warn — warn when a function argument is converted due
to a prototype declaration 81

toggle Prototype_override_warnings — warn when an ANSI Standard C declaration
overrides an old-style function definition 82

option -Hretwc — add the warning count to the compiler's return code 45

option -w — set the compiler diagnostic warning level 53, 289

warning message controls 289

warning message numbers 289

warning message output by level 289

warning message syntax 285

warning messages 286

warning messages 49

warning messages 49, 287

warning messages 100, 187, 289, 290

warning messages 100, 289, 290

warning messages 288

warning messages as compiler error
messages 100, 285, 289, 290

warning messages by number 100

warning messages by number 289

Warning_level — set compiler diagnostic
warning level 104, 289

warnings 285

warnings when arguments to a non-prototype
function do not match declared parameters 75

weak global symbol and assign it a value 105

Weak x = y — designate a weak global symbol
and assign it a value 105

"well-behaved" manner 57, 124

(wide compilation) 49, 131, 248

(wide compilation) mode 131, 248

option -w — set the compiler diagnostic warning level 53, 289

toggle VP_UP_warn — warn about expressions with more than one value,
depending on whether UP or VP rules are used 91

toggle Int_function_warnings — warn about missing and mistyped returns 69

toggle Prototype_conversion_warn — warn when a function argument is converted due
to a prototype declaration 81

toggle Prototype_override_warnings — warn when an ANSI Standard C declaration
overrides an old-style function definition 82

option -Hretwc — add the warning count to the compiler's return code 45

option -w — set the compiler diagnostic warning level 53, 289

warning message controls 289

warning message numbers 289

warning message output by level 289

warning message syntax 285

warning messages 286

warning messages 49

warning messages 49, 287

warning messages 100, 187, 289, 290

warning messages 100, 289, 290

warning messages 288

warning messages as compiler error
messages 100, 285, 289, 290

warning messages by number 100

warning messages by number 289

Warning_level — set compiler diagnostic
warning level 104, 289

warnings 285

warnings when arguments to a non-prototype
function do not match declared parameters 75

weak global symbol and assign it a value 105

Weak x = y — designate a weak global symbol
and assign it a value 105

"well-behaved" manner 57, 124

(wide compilation) 49, 131, 248

(wide compilation) mode 131, 248

toggle Use_UP_rules —	widen unsigned char and unsigned short to unsigned int	76, 90
toggle	Widen_float_args — widen float arguments to double	91
	widening an operand	90
	width and range of values of bit-field types	176
bit field	widths	176
option -Hnonexcept_wrap —	wrap include files to compile without exception handling	42, 162
	wrapper files c_wrap1.h and c_wrapr.h	33
	wrapper files exc_off.h and exc_pop.h	42
	wrapper files excl.h and excr.h	163
	wrapping C header files when compiling with C++ .	64
DATA type denotes a	writable data section	112
.data — default section for uninitialized static variables and	writable strings referenced from code	184
.data1 — default section for	writable strings whose addresses are referenced from within .data	185
option -E — preprocess source file but do not compile;	write output to standard output	28

Y

option	-YP — specify default directories for locating libraries	53
--------	--	----

Z

toggle Trap_div_by_zero — trap division by	zero	88
--	------------	----