

IBM XL C/C++ Enterprise Edition for AIX



コンパイラー・リファレンス

バージョン 7.0

IBM XL C/C++ Enterprise Edition for AIX



コンパイラー・リファレンス

バージョン 7.0

ご注意

本書の情報およびそれによってサポートされる製品を使用する前に、503 ページの『特記事項』に記載する一般情報をお読みください。

本書は、IBM XL C/C++ Enterprise Edition for AIX バージョン 7 リリース 0 (プロダクト番号 5724-I11)、および新しい版で特に明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

<http://www.ibm.com/jp/manuals/main/mail.html>

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

<http://www.ibm.com/jp/manuals/> の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原 典： SC09-7887-00
IBM XL C/C++ Enterprise Edition for AIX
Compiler Reference
Version 7.0

発 行： 日本アイ・ビー・エム株式会社

担 当： ナショナル・ランゲージ・サポート

第1刷 2004.7

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財)日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

© Copyright International Business Machines Corporation 1995, 2004. All rights reserved.

© Copyright IBM Japan 2004

目次

構文図の読み方	vii
シンボル	vii
構文項目	vii
構文例	viii

第 1 部 概要

XL C/C++ Enterprise Edition	3
コンパイラー・モード	3
オブジェクト・モデル	6
コンパイラー・オプション	6
入力ファイルの種類	8
出力ファイルの種類	10
コンパイラー・メッセージおよびリスト情報	12
コンパイラー・メッセージ	12
コンパイラー・リスト	12

プログラムの並列化

IBM SMP ディレクティブ	15
OpenMP ディレクティブ	17
カウント可能ループ	17
並列化ループ内の縮約操作	19
並列環境での共用変数および Private 変数	19

他のプログラム言語での XL C/C++ Enterprise Edition の使用

第 2 部 構成と使用

コンパイル環境のセットアップ

環境変数の設定	27
bsh、ksh、または sh シェルにおける環境変数の設定	27
csh シェルにおける環境変数の設定	27
64 ビットまたは 32 ビット・モードへの環境変数の設定	28
並列処理ランタイム・オプションの設定	28
他の環境変数の設定	29

コンパイラーの起動

リンケージ・エディターの起動	31
----------------	----

コンパイラー・オプションの指定

コマンド行におけるコンパイラー・オプションの指定	33
-q オプション	33
フラグ・オプション	34
プログラム・ソース・ファイル内のコンパイラー・オプションの指定	35
構成ファイル内のコンパイラー・オプションの指定	36

構成ファイルの調整	37
構成ファイルの属性	37
アーキテクチャー固有の (32 ビットまたは 64 ビットの) コンパイル用コンパイラー・オプションの指定	39
矛盾するコンパイラー・オプションの解決	41

インクルード・ファイル用のパス名の指定

相対パス名を使用したインクルード・ファイルのディレクトリ検索順序	43
----------------------------------	----

プラグマを使用した並列処理の制御

他のプログラム言語での C および C++ の使用

言語間呼び出し規則	49
対応するデータ型	49
文字および集約データの特殊な取り扱い	50
言語間呼び出しにおけるサブルーチン・リンケージ規約の使用	52
言語間呼び出し - パラメーターの受け渡し	52
言語間呼び出し - 参照による呼び出しパラメーター	53
言語間呼び出し - 値パラメーターによる呼び出し	54
言語間呼び出し - 値によるパラメーターの受け渡し規則	54
言語間呼び出し - 関数を指すポインター	55
言語間呼び出し - 関数からの戻り値	56
言語間呼び出し - スタック・フロア	56
言語間呼び出し - スタック・オーバフロー	56
言語間呼び出し - トレースバック・テーブル	56
言語間呼び出し - 型のエンコードと検査	57
サンプル・プログラム: Fortran を呼び出す C/C++	57

第 3 部 解説

コンパイラー・オプション

コンパイラーのコマンド行オプション	61
コマンド行コンパイラー・オプションの要約	62
+ (正符号)	71
# (ポンド記号)	72
32、64	73
aggrcopy	75
alias	76
align	78
alloca	82
ansialias	83
arch	84
asm	89
asm_as	90
assert	91

attr	92
B	93
b	94
bitfields	95
bmaxdata	96
brtl	97
C	98
c	99
c_stdinc	100
cache	101
chars	104
check	105
cinc	107
compact	108
cplusemt	109
cpp_stdinc	114
D	115
dataimported	117
datalocal	118
dbxextra	120
digraph	121
directstorage	123
dollar	124
dpcl	125
E	126
e	128
eh	129
enum	130
expfile	134
extchk	135
F	136
f	137
fdpr	138
flag	139
float	141
flttrap	147
fold	149
format	150
fullpath	152
funcsect	153
G	154
g	155
genproto	157
halt	158
haltonmsg	160
heapdebug	161
hot	162
hsflt	164
hssngl	165
I	166
idirfirst	167
ignerrno	169
ignprag	170
info	171
initauto	175
inlglue	176

inline	177
ipa	181
isolated_call	191
keepinlines	193
keepparm	194
keyword	195
L	196
l	197
langlvl	198
largepage	216
ldbl128、longdouble	217
libansi	219
linedebug	220
list	221
listopt	222
longlit	223
longlong	225
M	226
ma	228
macpstr	229
maf	231
makedep	232
maxerr	234
maxmem	236
mbcs、dbcs	238
mkshrobj	239
namemangling	241
O、optimize	244
o	248
objmodel	250
oldpassbyvalue	251
P	252
p	254
pascal	255
path	256
pdf1、pdf2	257
pg	261
phsinfo	262
pic	263
prefetch	264
print	265
priority	266
proclcal、procimported、procunknown	267
proto	270
Q	271
r	274
report	275
rndflt	276
ro	278
roconst	279
roptr	281
rrm	282
rtti	283
S	285
s	287
saveopt	288

showinc	289	#pragma do_not_instantiate	362
showpdf	290	#pragma enum	363
smallstack	292	#pragma execution_frequency	367
smp	293	#pragma hashome	369
source	296	#pragma ibm_snapshot	371
sourcetype	297	#pragma implementation	372
spill	299	#pragma info	373
spnans	300	#pragma instantiate	376
srcmsg	301	#pragma ishome	377
staticinline	302	#pragma isolated_call	379
statsym	303	#pragma langlvl	381
stdinc	304	#pragma leaves	383
strict	305	#pragma loopid	384
strict_induction	307	#pragma map	385
suppress	308	#pragma mc_func	387
symtab	309	#pragma namemangling	389
syntaxonly	310	#pragma nameManglingRule	391
t	311	#pragma object_model	393
tabsize	313	#pragma options	394
tbtable	314	#pragma option_override	399
tempinc	316	#pragma pack	401
templaterecompile	317	#pragma pass_by_value	404
templateregistry	318	#pragma priority	406
tempmax	320	#pragma reachable	407
threaded	321	#pragma reg_killed_by	408
tmplparse	322	#pragma report	410
tocdata	323	#pragma stream_unroll	412
tocmerge	324	#pragma strings	414
trigraph	325	#pragma unroll	415
tune	327	#pragma unrollandfuse	417
twolink	329	#pragma weak	419
U	331	並列処理を制御するプラグマ	422
unique	332	#pragma ibm_critical	424
unroll	333	#pragma ibm_independent_calls	425
unwind	335	#pragma ibm_independent_loop	426
upconv	336	#pragma ibm_iterations	427
utf	337	#pragma ibm_parallel_loop	428
V	338	#pragma ibm_permutation	429
v	339	#pragma ibm_schedule	430
vftable	340	#pragma ibm_sequential_loop	432
W	341	#pragma omp_atomic	433
w	342	#pragma omp_parallel	434
warn64	343	#pragma omp_for	436
weaksymbol	344	#pragma omp_ordered	440
xcall	345	#pragma omp_parallel_for	441
xref	346	#pragma omp_section、#pragma omp_sections	442
y	347	#pragma omp_parallel_sections	444
Z	348	#pragma omp_single	445
汎用プラグマ	349	#pragma omp_master	447
#pragma align	352	#pragma omp_critical	448
#pragma alloca	353	#pragma omp_barrier	449
#pragma block_loop	354	#pragma omp_flush	450
#pragma chars	356	#pragma omp_threadprivate	451
#pragma comment	357	コンパイラ・モードおよびプロセッサのアーキ テクチャーの有効な組み合わせ	452
#pragma define	359		
#pragma disjoint	360		

コンパイラー・メッセージ	457
メッセージ重大度レベルとコンパイラー応答	457
コンパイラー戻りコード	458
コンパイラー・メッセージ・フォーマット	459

並列処理のサポート 461

IBM SMP 並列処理のためのランタイム・オプション	461
スケジューリング・アルゴリズム・オプション	462
並列環境オプション	462
パフォーマンス調整オプション	463
動的プロファイル・オプション	464
並列処理のための OpenMP ランタイム・オプション	465
スケジューリング・アルゴリズム環境変数	465
並列環境環境変数	466
動的プロファイル環境変数	467
並列処理に使用する組み込み関数	467

第 4 部 付録 471

付録 A. 事前定義マクロ 473

プラットフォームに関連するマクロ	473
----------------------------	-----

付録 B. 組み込み関数 475

付録 C. XL C/C++ Enterprise Edition における各国語サポート 493

マルチバイト・データを含むファイルの新規コード・ページへの変換	493
マルチバイト文字サポート	493
ストリング・リテラルと文字定数	494
プリプロセッサ・ディレクティブ	494
マクロ定義	494
コンパイラー・オプション	494
ファイル名とコメント	495
制約事項	495

付録 D. 問題解決 497

メッセージ・カタログ・エラー	497
コンパイル中のページ・スペース・エラーの訂正	498

付録 E. ASCII 文字セット 499

特記事項 503

プログラミング・インターフェース情報	505
商標	505
業界標準	505

構文図の読み方

本節では、構文図の読み方について説明します。構文図のシンボル、図に含まれる項目（キーワード、変数、区切り文字、演算子、フラグメント参照、オペランド）を定義し、これらの項目が含まれている構文例を示します。

構文図は、コマンド・ステートメントを構成する順序および部分（オプションおよび引き数）を図で表します。水平線上のメインパスに沿って、右から左へ、また、上から下へ読んでいきます。

シンボル

以下は、構文図に出現するシンボルです。

シンボル	定義
▶—	構文図の先頭を表します。
—→	構文図が次の行に続くことを表します。
▶—	構文が直前の行から続いていることを表します。
—▶	構文図の終わりを表します。

構文項目

構文図には、多くのさまざまな項目が含まれています。構文項目には、以下のものがあります。

- キーワード - コマンド名またはその他のリテラル情報です。
- 変数 - 変数は、小文字のイタリックで表され、ユーザーが指定可能な値の名前を表します。
- 区切り文字 - 区切り文字は、キーワード、変数、または演算子の始まりまたは終わりを表します。例えば、左括弧は区切り文字です。
- 演算子 - 演算子には、加算 (+)、減算 (-)、乗算 (*)、除法 (/)、等号 (=)、および実行する必要があるその他の数学演算子が含まれます。
- フラグメント参照 - 構文図の一部で、より詳しく示すために図から分離したものです。
- 区切り文字 - 区切り文字は、キーワード、変数、または演算子を区切ります。例えば、コンマ (,) は区切り文字です。

キーワード、変数、および演算子は、必須、オプション、またはデフォルトとして表示されます。フラグメント、区切り文字、および区切り文字は、必須またはオプションとして表示されます。

項目タイプ	定義
必須	必須項目は、水平線のメインパス上に表示されます。
オプション	オプション項目は、水平線のメインパスの下に表示されます。

デフォルト デフォルト項目は、水平線のメインパスの上に表示されます。

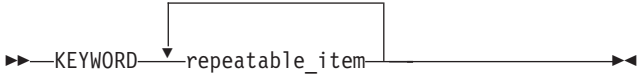
構文例

次の表では、構文例を示します。

表 1. 構文例

項目	構文例
必須項目	►►—KEYWORD—required_item—◄◄
必須項目は、水平線のメインパス上に表示されます。下記の項目を指定しなければなりません。	
必須の選択項目	►►—KEYWORD— <div>required_choice1 required_choice2</div> —◄◄
必須の選択項目 (2 つ以上の項目) は、水平線のメインパス上に縦に重ねて表示されます。重なった項目からいずれか 1 つを選択しなければなりません。	
オプション項目	►►—KEYWORD— <div>optional_item</div> —◄◄
オプション項目は、水平線のメインパスの下に表示されます。	
オプションの選択項目	►►—KEYWORD— <div>optional_choice1 optional_choice2</div> —◄◄
オプションの選択項目 (2 つ以上の項目) は、水平線のメインパスの下に縦に重ねて表示されます。重なった項目からいずれか 1 つを選択できます。	
デフォルト	►►—KEYWORD— <div>default_choice1 optional_choice2 optional_choice3</div> —◄◄
デフォルト項目は、水平線のメインパスの上に表示されます。これ以外の項目 (必須またはオプション) は、水平線のメインパス上 (必須)、または水平線のメインパスの下 (オプション) に表示されます。下記の例は、オプション項目のあるデフォルトを表したものです。	
変数	►►—KEYWORD— <i>variable</i> —◄◄
変数は、小文字のイタリックで表されます。これらは、名前または値を示します。	

表 1. 構文例 (続き)

項目	構文例
反復可能な項目	
水平線のメインパスの上を左に戻る矢印は、繰り返しの可能な項目を示しています。	
反復可能な項目のグループの上を左に戻る矢印は、そのいずれか 1 つの項目を選択できるか、または単一項目を繰り返すことができることを示しています。	

第 1 部 概要

XL C/C++ Enterprise Edition

IBM XL C/C++ Enterprise Edition for AIX を使用して、C および C++ プログラム・ソース・ファイルを実行可能オブジェクト・モジュールにコンパイルすることができます。

注: このトピック全体において、**xlc** および **xlC** コマンド呼び出しは、コンパイラーの動作を説明するために使用します。ただし、ユーザーの特定の環境が必要とする場合は、他の形式のコンパイラー呼び出しコマンドで代用でき、特に指定しない限り、コンパイラー・オプションの使用法はそのままです。

XL C/C++ Enterprise Edition コンパイラーについて詳しくは、本節の下記のトピックを参照してください。

- 『コンパイラー・モード』
- 6 ページの『オブジェクト・モデル』
- 6 ページの『コンパイラー・オプション』
- 8 ページの『入力ファイルの種類』
- 10 ページの『出力ファイルの種類』
- 12 ページの『コンパイラー・メッセージおよびリスト情報』

コンパイラー・モード

XL C/C++ Enterprise Edition コンパイラー呼び出しコマンドのさまざまな形式が、さまざまなレベルの C および C++ 言語をサポートしています。

たいていの場合、C++ ソース・ファイルをコンパイルするには **xlC** コマンドを、C ソース・ファイルをコンパイルするには **xlc** コマンドを使用する必要があります。C および C++ オブジェクト・ファイルが両方ともある場合は、**xlC** を使用してリンクしてください。

ユーザーの環境で必要であれば、コマンドの別の形式を使用することができます。各種コンパイラー呼び出しコマンドは、以下のとおりです。

コンパイラー呼び出し	
基本	特殊
xlC	xlC_r xlC_r4 xlC_r7 xlC128 xlC128_r xlC128_r4 xlC128_r7
xlc++	xlc++_r xlc++_r4 xlc++_r7 xlc++128 xlc++128_r xlc++128_r4 xlc++128_r7
xlc	xlc_r xlc128_r4 xlc128_r7 xlc128 xlc128_r xlc128_r4 xlc128_r7
cc	cc_r cc_r4 cc_r7 cc128 cc128_r cc128_r4 cc128_r7
c99	c99_r c99_r4 c99_r7 c99_128 c99_128_r c99_128_r4 c99_128_r7
c89	c89_r c89_r4 c89_r7 c89_128 c89_128_r c89_128_r4 c89_128_r7
xlCcore	xlCcore_r xlCcore_r4 xlCcore_r7 xlCcore128 xlCcore128_r xlCcore128_r4 xlCcore128_r7

基本コンパイラー呼び出しコマンドは、上記の各行の最初の項目として現れます。
以下の基準で、基本呼び出しを選択してください。

xlC xlC++	<p>両方ともソース・ファイルが C++ 言語ソース・コードとしてコンパイルされるように、コンパイラーを呼び出します。いずれかのソース・ファイルが C++ である場合、適正なランタイム・ライブラリーとリンクさせるには、この呼び出しを使用しなければなりません。ソース・ファイルは、-qalias=ansi セットを使用してコンパイルされます。</p> <p>サフィックス .c を持つファイルは、-+ コンパイラー・オプションを使用していないと見なし、-qlanglvl=ansi が有効なとき、C 言語ソース・コードとしてコンパイルされます。</p>
xlC	<p>C ソース・ファイル用のコンパイラーを呼び出します。この呼び出しでは、以下のコンパイラー・オプションが暗黙指定されます。</p> <ul style="list-style-type: none"> • -qlanglvl=extc89 • -qalias=ansi • -qcpluscmt • -qkeyword=inline
cc	<p>C ソース・ファイル用のコンパイラーを呼び出します。この呼び出しでは、以下のコンパイラー・オプションが暗黙指定されます。</p> <ul style="list-style-type: none"> • -qlanglvl=extended • -qnoro • -qnoroconst
c99	<p>ISO C99 言語フィーチャーをサポートする、C ソース・ファイルのコンパイラーを呼び出します。完全な ISO C99 規格では、C99 準拠のヘッダー・ファイルとランタイム・ライブラリーが必須です。この呼び出しでは、以下のオプションが暗黙指定されます。</p> <ul style="list-style-type: none"> • -qlanglvl=stdc99 • -qalias=ansi • -qstrict_induction • -D_ANSI_C_SOURCE • -D_ISOC99_SOURCE
c89	<p>ISO C89 言語フィーチャーがサポートされる、C ソース・ファイルのコンパイラーを呼び出します。この呼び出しでは、以下のオプションが暗黙指定されます。</p> <ul style="list-style-type: none"> • -qlanglvl=stdc89 • -qalias=ansi • -qstrict_induction • -qnolonglong • -D_ANSI_C_SOURCE <p>ANSI 規格 (ISO/IEC 9899:1990) に厳密に準拠させる場合は、この呼び出しを使用してください。</p>

xlCcore xlc++core	両方とも xlC および xlc++ のために上記で説明したコンパイラーを呼び出しますが、このコンパイラーはランタイム・ライブラリーのコアにのみリンクします。XL C/C++ Enterprise Edition によって提供されるランタイム・ライブラリー以外のランタイム・ライブラリーにアプリケーションをリンクしたい場合は、この呼び出しを使用します。
------------------------------	--

IBM XL C/C++ Enterprise Edition for AIX は、基本コンパイラー呼び出しで、バリエーションを提供します。これらのバリエーションについては以下で説明します。

- 128- サフィックス呼び出し** すべての **128-** サフィックス呼び出しコマンドは、機能的には対応する基本コンパイラー呼び出しに類似しています。それらは、**-qldbl128** オプションを指定します。これによってプログラム内での **long double** 型の長さは 64 から 128 ビットに増加します。これらは、C および C++ ランタイムの 128 バージョンともリンクします。
- _r- サフィックス呼び出し** すべての **_r** サフィックス呼び出しは、追加のマクロ名 **_THREAD_SAFE** および **_VACPP_MULTI__** を定義し、ライブラリー **-lc** および **-lpthreads** を追加します。コンパイラー・オプション **-qthreaded** も追加されます。Posix スレッド化アプリケーションを作成したい場合は、これらのコマンドを使用してください。

Posix Draft 7 に基づいたプログラムを Posix Draft 10 にマイグレーションするために、**_r7** 呼び出しが提供されています。追加情報については、**-qthreaded** を参照してください。**_r4** 呼び出しは、DCE スレッド化アプリケーションに使用してください。

AIX® バージョン 3.2.5 DCE アプリケーションから AIX バージョン 4.3.3 以降へのマイグレーション

メインの呼び出しコマンド (**c89** を除く) には、追加の **_r4-** サフィックス形式があります。これらの形式は、AIX バージョン 3.2.5 と AIX バージョン 4 で書かれた DCE アプリケーション間での互換性を提供します。それらはユーザー・アプリケーションを、正しい AIX バージョン 4 DCE ライブラリーへリンクし、**pthread** ライブラリーの最新バージョンと、AIX バージョン 3.2.5 でサポートされる初期バージョン間での互換性を提供します。

関連タスク

31 ページの『コンパイラーの起動』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

349 ページの『汎用プラグマ』

422 ページの『並列処理を制御するプラグマ』

321 ページの『threaded』

オブジェクト・モデル

XL C/C++ Enterprise Edition により、プログラムを 2 つのオブジェクト・モデルのいずれかにコンパイルできます。2 つのオブジェクト・モデルは、以下のとおりです。

- classic
- ibm

2 つのオブジェクト・モデルは、以下の領域で異なります。

- 仮想関数テーブルでのレイアウト
- 仮想基底クラスのサポート
- ネーム・マングリング体系

ランタイム・モジュールに、classic オブジェクト・モデルまたは以前のバージョンの IBM C++ コンパイラーを使用してコンパイルされた任意のランタイム・モジュールとの間の互換性を持たせる必要がある場合は、**classic** を選択してください。

パフォーマンスを改善したい場合は、**ibm** を選択します。とりわけ、多くの仮想基底クラスを持ったクラス階層の場合に役立ちます。派生クラスのサイズがかなり小さくなり、また仮想関数テーブルへのアクセスがより速くなります。

同じ継承の階層におけるすべてのクラスには、同じオブジェクト・モデルがなければなりません。

ターゲット・オブジェクト・モデルを指定するには、**-qobjmodel** コンパイラー・オプションまたは **object_model** プラグマを使用します。

関連資料

250 ページの『objmodel』

393 ページの『#pragma object_model』

コンパイラー・オプション

コンパイラー・オプションは、広範囲の種類の関数を実行します。例えば、コンパイラー特性の設定、オブジェクト・コードおよび作成されるコンパイラー出力の記述、いくつかのプリプロセッサ関数の実行などです。以下の 3 つの方法のいずれかを使用してコンパイラー・オプションを指定することができます。

- コマンド行で
- 構成ファイル (.cfg) で
- ソース・プログラム内で

上記の方法で明示的に設定されていないコンパイラー・オプションのほとんどについて、コンパイラーはデフォルト設定されているものと見なします。

コンパイラー・オプションを指定した場合、オプションの矛盾や非互換が起きる可能性があります。XL C/C++ Enterprise Edition は、これらの矛盾や非互換のほとんどを、一貫したやり方で以下のとおり解決します。

多くの場合、コンパイラーは、以下の順序で、対立するオプションまたは矛盾するオプションを解決します。

1. ソース・コード内のプラグマ・ステートメントは、コマンド行で指定されたコンパイラー・オプションをオーバーライドします。
2. コマンド行に指定されるコンパイラー・オプションは、構成ファイルに指定されたコンパイラー・オプションをオーバーライドします。対立または競合するコンパイラー・オプションが、同じコマンド行コンパイラー呼び出しに指定されている場合、その呼び出しの後のほうに指定されているオプションが優先されます。
3. 構成ファイルに指定されるコンパイラー・オプションは、コンパイラーのデフォルト設定をオーバーライドします。

この優先順位に従わないオプション競合については、41 ページの『矛盾するコンパイラー・オプションの解決』で説明します。

関連タスク

- 31 ページの『コンパイラーの起動』
- 33 ページの『コンパイラー・オプションの指定』
- 41 ページの『矛盾するコンパイラー・オプションの解決』

関連資料

- 61 ページの『コンパイラーのコマンド行オプション』
- 349 ページの『汎用プラグマ』
- 422 ページの『並列処理を制御するプラグマ』

入カファイルの種類

以下の種類のファイルを XL C/C++ Enterprise Edition コンパイラーに入力することができます。

C および C++ ソース・ファイル これらは C または C++ ソース・コードを含むファイルです。

C コンパイラーとして使用して C 言語ソース・ファイルをコンパイルするには、ソース・ファイルに、例えば `mysource.c` のように **.c** (小文字の c) サフィックスを付けなければなりません。

C++ コンパイラーを使用するには、ソース・ファイルに、**.C** (大文字 C)、**.cc**、**.cp**、**.cpp**、**.cxx**、または **.c++** サフィックスを付けなければなりません。他のファイルを C++ ソース・ファイルとしてコンパイルするには、**-+** コンパイラー・オプションを使用します。サフィックスが **.a**、**.o**、**.s**、または **.so** 以外で、このオプションで指定されているすべてのファイルは、C++ ソース・ファイルとしてコンパイルされます。

コンパイラーは、**.i** サフィックスを持つソース・ファイルも受け入れます。この拡張子は、プリプロセスされたソース・ファイルを示します。

コンパイラーは、現れた順にソース・ファイルを処理します。コンパイラーが指定されたソース・ファイルを見つけることができない場合は、エラー・メッセージを出し、次に指定されたファイルへ進みます。しかし、リンケージ・エディターは実行せず、一時オブジェクト・ファイルは除去されます。

ユーザー・プログラムは、いくつかのソース・ファイルから構成することができます。これらのソース・ファイルはすべて、コンパイラーを一回だけ呼び出すことで、一度にコンパイルすることができます。複数ソース・ファイルを、コンパイラーの一回の呼び出しを使用してコンパイルすることは可能ですが、呼び出しごとにコマンド行で指定できるコンパイラー・オプションは、1 セットのみになります。異なるコマンド行コンパイラー・オプションをそれぞれ指定したい場合は、個別に呼び出す必要があります。

デフォルトでは、コンパイラーは、すべての指定されたソース・ファイルをプリプロセスして、コンパイルします。通常このデフォルトを使用しますが、コンパイラーを使用して、コンパイルせずにソース・ファイルをプリプロセスすることができます。その場合、**-E** または **-P** オプションのいずれかを指定します。**-P** オプションを指定した場合は、プリプロセスされたソース・ファイル、`file_name.i` が作成され、処理は終了します。

-E オプションは、ソース・ファイルをプリプロセスし、標準出力へ書き込み、出力ファイルを生成せずに処理を停止します。

プリプロセスされたソース・ファイル	プリプロセスされたソース・ファイルは、例えば <i>file_name.i</i> のように、 .i サフィックスが付いています。コンパイラーは、プリプロセスされたソース・ファイル、 <i>file_name.i</i> をコンパイラーへ送ります。そこで、そのファイルは .c または .C ファイルと同じ方法で再びプリプロセスされます。プリプロセスされたファイルは、マクロやプリプロセッサ・ディレクティブのチェックに役立ちます。
オブジェクト・ファイル	オブジェクト・ファイルには、サフィックス .o が必要です (例えば、 <i>file_name.o</i>)。オブジェクト・ファイル、ライブラリー・ファイル、および非ストリップ実行可能ファイルは、リンケージ・エディターの入力として使用できます。コンパイル後、リンケージ・エディターは、実行可能なファイルを作成するため、指定されたすべてのオブジェクト・ファイルをリンクします。
アセンブラー・ファイル	アセンブラー・ファイルには、サフィックス .s が必要です (例えば、 <i>file_name.s</i>)。アセンブラー・ファイルは、オブジェクト・ファイルを作成するために、アセンブルされます。
共用ライブラリー・ファイル	共用ライブラリー・ファイルには、 .so サフィックスが必要です (例えば、 <i>file_name.so</i>)。
非ストリップ実行可能ファイル	拡張共通オブジェクト・ファイル形式 (XCOFF) ファイルは、AIX strip コマンドを使用してストリップされていないファイルですが、コンパイラーへの入力として使用できます。詳細については、「AIX コマンド・リファレンス」の strip コマンド、および「AIX Files Reference」の a.out ファイル形式の説明を参照してください。

関連概念

10 ページの『出力ファイルの種類』

関連資料

126 ページの『E』

252 ページの『P』

参照先

「コマンド・リファレンス 第 5 巻」の中の strip コマンド
Files Reference

出力ファイルの種類

XL C/C++ Enterprise Edition コンパイラーを呼び出すときに、以下の種類の出力ファイルを指定することができます。

実行可能ファイル デフォルトで、実行可能ファイルは **a.out** という名前になります。実行可能ファイルに別の名前を付けたい場合は、**-o***file_name* オプションを呼び出しコマンドで使います。このオプションは、*file_name* に指定した名前で作成可能ファイルを作成します。指定した名前は、実行可能ファイルの相対、または絶対パス名になります。

a.out ファイルの形式については、「*AIX Files Reference*」に説明があります。

オブジェクト・ファイル コンパイラーはオブジェクト・ファイルのサフィックスを **.o** とします (例えば、異なるサフィックスを指定して、またはサフィックスを指定しないで **-o***file_name* オプションを指定しない限り、*file_name.o* となります)。

-c オプションを指定する場合、出力オブジェクト・ファイル *file_name.o* が、それぞれの入力ソース・ファイル *file_name.x* に対して作成されます。ここで、*x* は認識された C または C++ ファイル名拡張子です。リンク・エディターは呼び出されず、オブジェクト・ファイルは現行ディレクトリーに配置されます。すべての処理は、コンパイルの完了時に停止します。

コンパイラーを呼び出すことによって、オブジェクト・ファイルを後で単一の実行可能ファイルに、リンク・エディットすることができます。

アセンブラー・ファイル アセンブラー・ファイルには、サフィックス **.s** が必要です (例えば、*file_name.s*)。

アセンブラー・ファイルは、**-S** オプションを指定して作成します。アセンブラー・ファイルは、オブジェクト・ファイルを作成するために、アセンブルされます。

プリプロセスされたソース・ファイル プリプロセスされたソース・ファイルは、例えば *file_name.i* のように、**.i** サフィックスが付いています。

プリプロセスされたソース・ファイルを作成するには、**-P** オプションを指定します。ソース・ファイルはプリプロセスされますが、コンパイルはされません。**-E** オプションからの出力を宛先変更して、**#line** ディレクティブを含むプリプロセス・ファイルを生成することもできます。

プリプロセスされたソース・ファイル *file_name.i* は、それぞれのソース・ファイルに対して作成され、作成されたソース・ファイルと同じファイル名を持ちます (拡張子は **.i**)。

リスト・ファイル リスト・ファイルは、例えば `file_name.lst` のように、`.lst` サフィックスが付いています。

呼び出しコマンドへのリスト関連オプションのいずれか 1 つを指定して、コンパイラー・リストを作成します (**-qnoprint** オプションを指定していない場合)。このリストを含むファイルは、現行ディレクトリーに置かれ、作成元のソース・ファイルと同じファイル名 (**.lst** 拡張子) となります。

共用ライブラリー・ファイル 共用ライブラリー・ファイルには、**.so** サフィックスが付いています (例えば `my_shrlib.so`)。

ターゲット・ファイル **-M** または **-qmakedep** オプションに関連する出力ファイルは、例えば **conversion.u** のように、サフィックス **.u** となります。

ファイルは、**make** コマンド用記述ファイルに組み込むのに適したターゲットを含みます。それぞれの C または C++ 入力ファイルごとに **.u** ファイルが 1 つ作成され、特定の入力ファイルに変更が加えられた場合に、別の入力ファイルの再コンパイルが必要かどうかを判別するために、**make** コマンドによって使用されます。**.u** ファイルは、その他のファイルには作成されません (**-+** オプションを使用して、その他のファイル・サフィックスを **.C** ファイルとして取り扱わない限り)。

関連概念

8 ページの『入力ファイルの種類』

関連資料

99 ページの『**c**』
126 ページの『**E**』
226 ページの『**M**』
232 ページの『**makedep**』
248 ページの『**o**』
252 ページの『**P**』
265 ページの『**print**』
285 ページの『**S**』

関連資料

99 ページの『**c**』
126 ページの『**E**』
226 ページの『**M**』
232 ページの『**makedep**』
248 ページの『**o**』
252 ページの『**P**』
265 ページの『**print**』
285 ページの『**S**』

コンパイラー・メッセージおよびリスト情報

コンパイラーは、C または C++ ソース・プログラムのコンパイル中にプログラム・エラーを検出すると、診断メッセージを標準エラー装置に発行し、該当するオプションが選択されている場合には、診断メッセージをリスト・ファイルに発行します。

コンパイラー・メッセージ

コンパイラーは C または C++ 言語特定のメッセージを出します。

► **C** コンパイラー・オプション **-qsrcmsg** を指定し、エラーが特定のコード行にある場合は、再構成されたソース行、または一部のソース行が、エラー・メッセージとともに `stderr` ファイルに含まれます。再構成されたソース行とは、すべてのマクロが展開された、プリプロセス済みのソース行です。

-qsource コンパイラー・オプションを指定する場合、コンパイラーはソース・リストにメッセージを置きます。例えば、コマンド行呼び出し **xlc -qsource filename.c** を使用してファイルをコンパイルする場合、現行ディレクトリーには **filename.lst** という名のファイルが見えます。

-qflag オプション、または **-w** オプションを指定すると、重大度に応じて出される診断メッセージを制御することができます。プログラム内に潜在する問題についての追加の情報メッセージを得るには、**-qinfo** オプションを使用します。

コンパイラー・リスト

コンパイラーによって作成されるリストは、デバッグに役立ちます。正しいオプションを指定すれば、コンパイルのすべての局面での情報を要求することができます。リストは、以下のセクションの組み合わせから構成されます。

- ソース・ファイル名およびコンパイル日時の他に、コンパイラー名、バージョン、およびリリースをリストする、ヘッダー・セクション。
- 入力ソース・コードに行番号を付けてリストする、ソース・セクション。行にエラーがある場合、関連付けられたエラー・メッセージが、ソース行の後に表示されます。
- コンパイル中に効力のあるオプションをリストする、オプション・セクション。
- コンパイル単位で使用される変数についての情報を提供する、属性および相互参照リスト・セクション。
- 各メイン・ソース・ファイルおよびインクルード・ファイルのファイル番号およびファイル名を示す、ファイル・テーブル・セクション。
- 診断メッセージを要約し、読み取られたソース行数をリストし、コンパイルが正常終了したかどうかを示す、コンパイル・エピローグ・セクション。
- オブジェクト・コードをリストする、オブジェクト・セクション。

ヘッダー・セクションを除く各セクションには、それを識別するセクション見出しがあります。セクション見出しは、二重かぎ括弧で囲まれています。

関連資料

457 ページの『メッセージ重大度レベルとコンパイラー応答』

459 ページの『コンパイラー・メッセージ・フォーマット』

139 ページの『flag』

171 ページの『info』

296 ページの『source』

301 ページの『srcmsg』

342 ページの『w』

プログラムの並列化

コンパイラーは、共用メモリー・プログラムの並列化をインプリメントするメソッドを 3 つ提供しています。それらは、以下のとおりです。

- プログラム・ループの自動並列化。
- IBM® SMP プラグマ・ディレクティブの使用による C プログラム・コードの明示的並列化。
- **OpenMP アプリケーション・プログラム・インターフェース**の仕様に準拠するプラグマ・ディレクティブを使用した C および C++ プログラム・コードの明示的並列化。

プログラムの並列化のすべてのメソッドは、**-qsmp** コンパイラー・オプションが **omp** サブオプションなしで有効な場合に使用可能です。**-qsmp=omp** コンパイラー・オプションを使用して、厳密な OpenMP 準拠を使用可能にすることができますが、これを行うと、自動並列化が使用不可になります。

プログラム・コードの並列領域は複数のスレッドによって実行され、複数のプロセッサ上で動作している可能性があります。作成されるスレッドの数は、ランタイム・オプションおよびライブラリー関数への呼び出しによって決まります。作業は、指定されたスケジューリング・アルゴリズムに従って使用可能なスレッドに分散されます。スレッドの作成および使用方法についての詳細は、「OpenMP Specification」のセクション 2.3『Parallel Construct』を参照してください。

注: **-qsmp** オプションは、必ず、スレッド・セーフ・コンパイラー呼び出しモードとともに使用しなければなりません。

XL C/C++ Enterprise Edition コンパイラーが提供する並列プログラミング・サポートについて詳しくは、本節の下記のトピックを参照してください。

- 『IBM SMP ディレクティブ』
- 17 ページの『OpenMP ディレクティブ』
- 17 ページの『カウント可能ループ』
- 19 ページの『並列化ループ内の縮約操作』
- 19 ページの『並列環境での共用変数および Private 変数』

OpenMP 仕様の完全な情報については、以下を参照してください。

- OpenMP Web サイト (www.openmp.org)
- OpenMP Specification (www.openmp.org/specs)

IBM SMP ディレクティブ

C IBM SMP ディレクティブは、カウント可能ループの並列化によって、共用メモリーの並列性を活用します。ループの形式が カウント可能ループ に記載されたいずれかのものであれば、そのループは カウント可能 と考えられます。

コンパイラーは自動的に、プログラム・コード内のすべてのカウント可能ループを見つけ、可能な場合はそれらをすべて並列化することができます。一般に、以下の条件がすべて満たされている場合にのみ、カウント可能ループが自動的に並列化されます。

- ループ反復の開始順序または終了順序が、プログラムの結果に影響しない。
- ループが入出力操作を含んでいない。
- **-qnostrict** オプションが有効でない場合、ループ内部の浮動小数点縮約が丸めエラーによって影響されない。
- **-qnostrict_induction** コンパイラー・オプションが有効である。
- **-qsmp** コンパイラー・オプションが **omp** サブオプションなしで有効である。コンパイラーは、スレッド・セーフ・コンパイラー・モードで呼び出さなければなりません。

また、選択したカウント可能ループの並列化をコンパイラーに明示的に指示することもできます。

XL C/C++ Enterprise Edition コンパイラーでは、プラグマ・ディレクティブが提供されています。これを使用して、コンパイラーが実行する自動並列化を改善することができます。プラグマは、2 つの一般カテゴリーに分かれます。

1. 1 つ目のカテゴリーのプラグマは、特定カウント可能ループの特性の情報をコンパイラーに提供できるようにするものです。コンパイラーはこの情報を使用して、ループの自動並列化をより効率的に実行します。
2. 2 つ目のカテゴリーは、並列化に対する明示制御をユーザーに与えるものです。これらのプラグマを使用して、ループ並列化を強制または抑止したり、特定の並列化アルゴリズムをループに適用したり、クリティカル・セクションを用いて共用変数へのアクセスを同期化したりします。

関連概念

- 17 ページの『OpenMP ディレクティブ』
- 17 ページの『カウント可能ループ』
- 19 ページの『並列化ループ内の縮約操作』
- 19 ページの『並列環境での共用変数および Private 変数』

関連タスク

- 28 ページの『並列処理ランタイム・オプションの設定』
- 47 ページの『プラグマを使用した並列処理の制御』

関連資料

- 293 ページの『smp』
- 422 ページの『並列処理を制御するプラグマ』
- 461 ページの『IBM SMP 並列処理のためのランタイム・オプション』
- 465 ページの『並列処理のための OpenMP ランタイム・オプション』
- 467 ページの『並列処理に使用する組み込み関数』

OpenMP 仕様の完全な情報については、以下を参照してください。

- OpenMP Web サイト (www.openmp.org)
- OpenMP Specification (www.openmp.org/specs)

OpenMP ディレクティブ

▶ C ▶ C++ OpenMP ディレクティブは、さまざまな型の並列領域を定義することによって、共用メモリーの並列性を活用します。並列領域は、プログラム・コードの反復セグメントおよび非反復セグメントの両方を含むことができます。

プラグマは、以下の 4 つの一般的なカテゴリーに分かれています。

1. 1 つ目のカテゴリーのプラグマは、作業がスレッドによって並列で行われる並列領域を定義できるようにするものです。ほとんどの OpenMP ディレクティブは、囲んでいる並列領域に、静的にバインドするか、または動的にバインドするかのいずれかです。
2. 2 つ目のカテゴリーは、作業がどのように並列領域のスレッド全体に分散および共用されるかを定義できるようにするものです。
3. 3 つ目のカテゴリーは、スレッド間の同期を制御できるようにするものです。
4. 4 つ目のカテゴリーは、スレッド全体のデータの可視性のスコープを定義できるようにするものです。

関連概念

15 ページの『IBM SMP ディレクティブ』

『カウント可能ループ』

19 ページの『並列化ループ内の縮約操作』

19 ページの『並列環境での共用変数および Private 変数』

関連タスク

28 ページの『並列処理ランタイム・オプションの設定』

47 ページの『プラグマを使用した並列処理の制御』

関連資料

293 ページの『smp』

422 ページの『並列処理を制御するプラグマ』

461 ページの『IBM SMP 並列処理のためのランタイム・オプション』

465 ページの『並列処理のための OpenMP ランタイム・オプション』

467 ページの『並列処理に使用する組み込み関数』

「General Programming Concepts: Writing and Debugging Programs」の『Parallel Programming』も参照してください。

OpenMP 仕様の完全な情報については、以下を参照してください。

- OpenMP Web サイト (www.openmp.org)
- OpenMP Specification (www.openmp.org/specs)

カウント可能ループ

ループが以下に示したいずれかの形式のものであれば、そのループはカウント可能と考えられます。

- ループ内への分岐もループ外側への分岐もない。
- *incr_expr* 式がクリティカル・セクション内にない。

以下は、カウント可能ループの例です。

```
for ([iv]; exit_cond; incr_expr)
    statement

for ([iv]; exit_cond; [expr]) {
    [declaration_list]
    [statement_list]
    incr_expr;
    [statement_list]
}

while (exit_cond) {
    [declaration_list]
    [statement_list]
    incr_expr;
    [statement_list]
}

do {
    [declaration_list]
    [statement_list]
    incr_expr;
    [statement_list]
} while (exit_cond)
```

以下の定義は、上記の例に適用されます。

<i>exit_cond</i>	は、次の形式を取ります。	<i>iv</i> <= <i>ub</i> <i>iv</i> < <i>ub</i> <i>iv</i> >= <i>ub</i> <i>iv</i> > <i>ub</i>
<i>incr_expr</i>	は、次の形式を取ります。	<i>++iv</i> <i>iv++</i> <i>--iv</i> <i>iv--</i> <i>iv += incr</i> <i>iv -= incr</i> <i>iv = iv + incr</i> <i>iv = incr + iv</i> <i>iv = iv - incr</i>

<i>iv</i>	反復変数。反復変数は、自動またはレジスター・ストレージ・クラスのいずれかを 持つ符号付き整数であり、そのアドレスを取り出されることなく、 <i>incr_expr</i> 内以 外のループではどの場所でも変更されません。
<i>incr</i>	ループ・インバリエント符号付き整数式。この式の値は実行時に認識され、0 では ありません。 <i>incr</i> は、extern 変数や static 変数、ポインターやポインター変数、 関数呼び出し、またはアドレスを取り出された変数を参照することはできません。
<i>ub</i>	ループ・インバリエント符号付き整数式。 <i>ub</i> は、extern 変数や static 変数、ポイ ンターやポインター変数、関数呼び出し、またはアドレスを取り出された変数を参 照することはできません。

関連概念

- 15 ページの『IBM SMP ディレクティブ』
- 17 ページの『OpenMP ディレクティブ』
- 19 ページの『並列化ループ内の縮約操作』
- 19 ページの『並列環境での共用変数および Private 変数』

関連タスク

28 ページの『並列処理ランタイム・オプションの設定』
47 ページの『プラグマを使用した並列処理の制御』

関連資料

293 ページの『smp』
422 ページの『並列処理を制御するプラグマ』
461 ページの『IBM SMP 並列処理のためのランタイム・オプション』
465 ページの『並列処理のための OpenMP ランタイム・オプション』
467 ページの『並列処理に使用する組み込み関数』

並列化ループ内の縮約操作

自動並列化と明示並列化のいずれの際にも、コンパイラーはループ内のほとんどの縮約操作を認識して、適切に処理することができます。具体的には、コンパイラーは、以下のいずれかの形式の縮約ステートメントを処理することができます。

```
var = var op expr;  
var assign_op expr;
```

ここで、

<i>var</i>	は、アドレスを取り出されず、ループ内の他のどの場所でも（ネストされたすべてのループを含む）参照されない自動変数またはレジスター変数を指定する ID です。例えば、以下のコードでは、ネストされたループ内の S だけが縮約として認識されます。 <pre>int i,j, S=0; #pragma ibm parallel_loop for (i= 0 ;i < N; i++) { S = S+ i; #pragma ibm parallel_loop for (j=0;j< M; j++) { S = S + j; } }</pre>
<i>op</i>	は、以下の演算子のいずれかです。 <pre>+ - * ^ &</pre>
<i>assign_op</i>	は、以下の演算子のいずれかです。 <pre>+= -= *= ^= = &=</pre>
<i>expr</i>	は、任意の有効な式です。

認識された縮約は、**-qinfo=reduction** オプションによってリストされます。IBM ディレクティブを使用する場合は、クリティカル・セクションを使用して、コンパイラーに認識されないすべての縮約変数へのアクセスを同期化してください。OpenMP ディレクティブは、縮約変数を明示的に指定する機構を提供します。

並列環境での共用変数および Private 変数

変数は、並列環境において共用コンテキストか **private** コンテキストのいずれかを持つことができます。

- 共用コンテキストでの変数は、関連する並列ループ内で実行されているすべてのスレッドに対して可視になります。
- **private** コンテキストでの変数は、他のスレッドから見えません。スレッドはそれぞれ独自に変数の **private** コピーを持っており、スレッドによってそのコピーに行われた変更内容は、他のスレッドには不可視です。

変数のデフォルトのコンテキストは、以下の規則によって決まります。

- **静的**ストレージの存在期間を持つ変数は共用である。
- 動的に割り振られたオブジェクトは共用である。
- 自動ストレージの存在期間を持つ変数は **private** である。
- ヒープが割り振られたメモリー内の変数は共用である。共用ヒープは 1 つしか存在できません。
- 並列構成の外部で定義されたすべての変数は、並列ループが現れると共用になる。
- ループ反復変数は、各自のループ内では **private** である。ループの後の反復変数の値は、ループが連続的に実行される場合と同じになる。
- **alloca** 関数によって並列ループ内に割り振られたメモリーは、そのループの反復 1 つ分の期間しか存在せず、各スレッドに対して **private** である。

以下のコード・セグメントは、これらのデフォルト・ルールの例を示したものです。

```
int E1;                                /* shared static */
void main (argc,...) {                 /* argc is shared */
    int i;                             /* shared automatic */
    void *p = malloc(...);             /* memory allocated by malloc */
                                        /* is accessible by all threads */
                                        /* and cannot be privatized */

    #pragma omp parallel firstprivate (p)
    {
        int b;                         /* private automatic */
        static int s;                  /* shared static */

        #pragma omp for
        for (i =0;...) {
            b = 1;                     /* b is still private here ! */
            foo (i);                   /* i is private here because it */
                                        /* is an iteration variable */
        }

        #pragma omp parallel
        {
            b = 1;                     /* b is shared here because it */
                                        /* is another parallel region */
        }
    }
}

int E2;                                /*shared static */
void foo (int x) {                     /* x is private for the parallel */
                                        /* region it was called from */

    int c;                             /* the same */
    ... }

```


コンパイラーは、プログラムのセマンティクスの変更が必要でなければ、いくつかの共用変数を `private` にすることができます。例えば、それぞれのループ反復が共用変数の固有の値を使用する場合は、その変数を `private` にすることができます。`private` になった共用変数は、**-qinfo=private** オプションによって報告されます。クリティカル・セクションを使用して、このレポートにリストされないすべての共用変数へのアクセスを同期化してください。

OpenMP プリプロセッサ・ディレクティブには、選択したデータ変数の可視性コンテキストを指定できるものがあります。データ・スコープ属性文節の概要を以下にリストします。

データ・スコープ 属性文節	説明
<code>private</code>	private 文節は、リスト内の変数がチーム内のそれぞれのスレッドに対して <code>private</code> であることを宣言します。
<code>firstprivate</code>	firstprivate 文節は、 <code>private</code> 文節によって提供される機能のスーパーセットを提供します。
<code>lastprivate</code>	lastprivate 文節は、 <code>private</code> 文節によって提供される機能のスーパーセットを提供します。
<code>shared</code>	shared 文節は、チーム内のすべてのスレッド間でリストに表示される変数を共用します。チーム内のすべてのスレッドが、共用変数用の同じストレージ域にアクセスします。
<code>reduction</code>	reduction 文節は、指定された演算子を使用して、リスト内に表示されたスカラー変数の縮小を実行します。
<code>default</code>	default 文節は、変数のデータ・スコープ属性をユーザーが操作できるようにします。

詳しくは、OpenMP ディレクティブの説明、または OpenMP C および C++ アプリケーション・プログラム・インターフェースの仕様を参照してください。

関連概念

- 15 ページの『IBM SMP ディレクティブ』
- 17 ページの『OpenMP ディレクティブ』
- 17 ページの『カウント可能ループ』
- 19 ページの『並列化ループ内の縮約操作』

関連タスク

- 28 ページの『並列処理ランタイム・オプションの設定』
- 47 ページの『プラグマを使用した並列処理の制御』

関連資料

- 293 ページの『smp』
- 422 ページの『並列処理を制御するプラグマ』
- 461 ページの『IBM SMP 並列処理のためのランタイム・オプション』
- 465 ページの『並列処理のための OpenMP ランタイム・オプション』
- 467 ページの『並列処理に使用する組み込み関数』

OpenMP 仕様の完全な情報については、以下を参照してください。

- OpenMP Web サイト www.openmp.org
- OpenMP Specification www.openmp.org/specs

他のプログラム言語での XL C/C++ Enterprise Edition の使用

XL C/C++ Enterprise Edition を使用して、C および C++ プログラムからその他の XL 言語で作成された関数を呼び出すことができます。同様に、その他の XL 言語プログラムは、C および C++ で書かれた関数を呼び出すことができます。使用している言語の構文について、すでに熟知している必要があります。

詳しくは、49 ページの『他のプログラム言語での C および C++ の使用』を参照してください。

第 2 部 構成と使用

コンパイル環境のセットアップ

C または C++ プログラムをコンパイルする前に、ご使用のシステムに応じた環境変数と構成ファイルとを設定しておかなければなりません。

XL C/C++ Enterprise Edition コンパイラーが使用する環境変数について詳しくは、本節の以下のトピックを参照してください。

- 『環境変数の設定』
- 28 ページの『64 ビットまたは 32 ビット・モードへの環境変数の設定』
- 28 ページの『並列処理ランタイム・オプションの設定』
- 29 ページの『他の環境変数の設定』

環境変数の設定

Bourne シェル (**bsh** または **sh**)、Korn シェル (**ksh**)、または C シェル (**csh**) のいずれを使用しているかに応じて、環境変数を設定するために別々のコマンドを使用します。現行シェルを判別するには、次の **echo** コマンドを使用します。

```
echo $SHELL
```

Bourne シェルのパスは、**/bin/bsh** または **/bin/sh** です。Korn シェルのパスは、**/bin/ksh** です。C シェルのパスは、**/bin/csh** です。

NLSPATH および **LANG** 環境変数の詳細については、「システム・ユーザーズ・ガイド: オペレーティング・システムおよびデバイス」を参照してください。AIX 国際言語機能については、「*AIX General Programming Concepts*」に説明があります。

bsh、ksh、または sh シェルにおける環境変数の設定

以下のステートメントは、コマンド行に入力するか、またはコマンド・ファイル・スクリプトに挿入するもので、Bourne または Korn シェルで環境変数を設定する方法を示しています。表示されたパスは、コンパイラーをデフォルトのインストール・ロケーションにインストールしていることを前提としています。

```
LANG=en_US  
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/L/%N  
export LANG NLSPATH
```

全ユーザーがアクセスできるような変数を設定するには、ファイル **/etc/profile** にコマンドを追加します。特定ユーザー専用の変数を設定するには、ユーザーのホーム・ディレクトリーのファイル **.profile** にコマンドを追加します。環境変数は、ユーザーがログインするたびに設定されます。

csh シェルにおける環境変数の設定

次のステートメントは、環境変数を csh シェルで設定する方法を示したものです。表示されたパスは、コンパイラーをデフォルトのインストール・ロケーションにインストールしていることを前提としています。

```
setenv LANG en_US
setenv NLSPATH /usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/L/%N
```

csh シェルでは、全ユーザーがアクセスできるような環境変数を設定することはできません。特定ユーザー専用の変数を設定するには、ユーザーのホーム・ディレクトリーのファイル **.cshrc** にコマンドを追加します。環境変数は、ユーザーがログインするたびに設定されます。

関連タスク

『64 ビットまたは 32 ビット・モードへの環境変数の設定』

『並列処理ランタイム・オプションの設定』

29 ページの『他の環境変数の設定』

関連資料

以下も参照してください。

システム・ユーザーズ・ガイド: オペレーティング・システムおよびデバイス

「AIX 5L™ Version 5.1 General Programming Concepts: Writing and Debugging Programs」

64 ビットまたは 32 ビット・モードへの環境変数の設定

OBJECT_MODE 環境変数を設定して、デフォルトのコンパイル・モードを指定することができます。 OBJECT_MODE 環境変数に可能な値は、以下のとおりです。

(unset)	コンパイラ・プログラムが 32 ビット・オブジェクトを生成したり使用したりします。
32	コンパイラ・プログラムが 32 ビット・オブジェクトを生成したり使用したりします。
64	コンパイラ・プログラムが 64 ビット・オブジェクトを生成したり使用したりします。
32_64	32 ビットおよび 64 ビットの両方のオブジェクトを受け入れるようにコンパイラ・プログラムを設定します。ただし、この値を使用してはいけません。コンパイラは、このモードでは機能しません。コンパイル時に他のコンパイル・オプションが設定されている場合にこれを選択すると、エラー・メッセージが出される場合があります。

関連タスク

39 ページの『アーキテクチャ固有の (32 ビットまたは 64 ビットの) コンパイル用コンパイラ・オプションの指定』

27 ページの『環境変数の設定』

並列処理ランタイム・オプションの設定

XLSPMPOPTS 環境変数は、ループ並列化を使用してプログラムのオプションを設定します。例えば、プログラム実行時に 4 つのスレッドを作成させ、チャンク・サイズが 5 の動的スケジューリングを使用するには、以下に示すように、XLSPMPOPTS 環境変数を設定します。

```
XLSPMPOPTS=PARTHDS=4:SCHEDULE=DYNAMIC=5
```


追加の環境変数は、プログラム並列化のオプションを OpenMP 準拠のディレクティブを使用して設定します。

関連タスク

27 ページの『環境変数の設定』

関連資料

461 ページの『IBM SMP 並列処理のためのランタイム・オプション』

465 ページの『並列処理のための OpenMP ランタイム・オプション』

他の環境変数の設定

コンパイラーを使用する前に、以下の環境変数が設定されていることを確認してください。

PATH	コンパイラーの実行可能ファイルのディレクトリ検索パスを指定します。
MANPATH	オプションで、man ページを見つけるためのディレクトリ検索パスを指定します。MANPATH には、デフォルトの man パスの前に、が含まれていなければなりません。
OBJECT_MODE	オプションで、デフォルトのコンパイル・モードを 32 ビットまたは 64 ビットに指定します。
LANG	メッセージおよびヘルプ・ファイル用の各国語を指定します。

LANG 環境変数は、システムに用意されている任意のロケールに設定することができます。詳しくは、「*AIX General Programming Concepts*」のロケールについての説明を参照してください。

米国英語用の各国語コードは **en_US** です。適切なメッセージ・カタログがすでにシステムにインストール済みであれば、**en_US** を有効な任意のほかの各国語コードに置き換えることができます。

ご使用のシステムの各国語の現行設定値を判別するには、以下の両方の **echo** コマンドを使用してください。

```
echo $LANG
echo $NLSPATH
```

NLSPATH	メッセージおよびヘルプ・ファイルのパス名を指定します。
PDFDIR	オプションで、プロファイル・データ・ファイルが作成されるディレクトリを指定します。デフォルト値は設定されず、コンパイラーはプロファイル・データ・ファイルを現行作業ディレクトリに入れます。プロファイル指示フィードバックの場合は、この変数を絶対パスに設定することをお勧めします。
TMPDIR	オプションで、一時ファイルが作成されるディレクトリを指定します。高水準の最適化ではページング・ファイルと一時ファイルが非常に大きなディスク・スペースを必要とする場合があり、デフォルト・ロケーション /tmp は高水準の最適化には不十分である可能性があります。

注: **LANG** および **NLSPATH** 環境変数はオペレーティング・システムのインストール時に初期設定され、使用したいものとは異なっていることがあります。

関連タスク

27 ページの『環境変数の設定』

関連資料

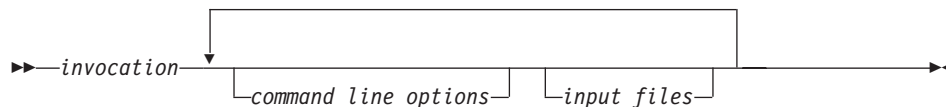
28 ページの『64 ビットまたは 32 ビット・モードへの環境変数の設定』

以下も参照してください。

「AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs」

コンパイラーの起動

IBM XL C/C++ Enterprise Edition コンパイラーは、下の構文を使用して起動します。ここで、*invocation* は、有効な XL C/C++ Enterprise Edition 呼び出しコマンドに置き換えることができます。



コンパイラー呼び出しコマンドのパラメーターは、入力ファイル名、コンパイラー・オプション名、およびリンケージ・エディター・オプション名にすることができます。C++ コンパイルでは、*munch* ユーティリティーに渡すための *munch* オプションもパラメーターに含むことができます。

コンパイラー・オプションは、広範囲の種類の関数を実行します。例えば、コンパイラー特性の設定、オブジェクト・コードおよび作成されるコンパイラー出力の記述、いくつかのプリプロセッサ関数の実行などです。

デフォルトでは、呼び出しコマンドはコンパイラーとリンケージ・エディターを両方とも呼び出します。このコマンドは、リンケージ・エディター・オプションをリンケージ・エディターに渡します。その結果、これらの呼び出しコマンドは、すべてのリンケージ・エディター・オプションの受け入れも行います。

リンク・エディットしないでコンパイルするには、**-c** コンパイラー・オプションを使用します。**-c** オプションによって、コンパイル完了後にコンパイラーが停止され、**-o** オプションが使用されて異なるオブジェクト・ファイル名が指定されていない限り、各 *file_name.c* 入力ソース・ファイルごとに出力としてオブジェクト・ファイル *file_name.o* が作成されます。リンケージ・エディターは起動されません。同じ呼び出しコマンドを使用し、**-c** オプションを付けずにオブジェクト・ファイルを指定することにより、そのオブジェクト・ファイルを後でリンク・エディットすることができます。

リンケージ・エディターの起動

リンケージ・エディターは、指定されたオブジェクト・ファイルをリンク・エディットして、1 つの実行可能ファイルを作成します。**-E**、**-P**、**-c**、**-S**、**-qsyntaxonly**、または **#** のコンパイラー・オプションの 1 つを指定しない限り、呼び出しコマンドの 1 つを指定してコンパイラーを起動すると、自動的にリンケージ・エディターが呼び出されます。

入力ファイル

オブジェクト・ファイル、非ストリップ実行可能ファイル、およびライブラリー・ファイルは、リンケージ・エディターの入力データとして使用できます。オブジェクト・ファイルには、サフィックス **.o** が必要です (例えば

year.o)。静的ライブラリー・ファイル名には、サフィックス **.a** が付きます (例えば **libold.a**)。動的ライブラリー・ファイル名には、サフィックス **.so** が付きます (例えば **libold.so**)。

ライブラリー・ファイルは、AIX **ar** コマンドを使用して、1 つ以上のファイルを 1 つのアーカイブ・ファイルに結合することにより作成されます。**ar** コマンドの説明については、「*AIX Commands Reference*」を参照してください。

出力ファイル

リンケージ・エディターは、実行可能ファイル を生成して、そのファイルを現行ディレクトリーに入れます。実行可能ファイルのデフォルト名は **a.out** です。実行可能ファイルに明示的に名前を付けるには、コンパイラ呼び出しコマンドで **-o file_name** オプションを使用します。*file_name* は、実行可能ファイルに指定する名前です。例えば、**myfile.c** をコンパイルし、**myfile** という名の実行可能ファイルを生成するには、次を入力します。

```
xlc myfile.c -o myfile
```

-qmksbobj オプションを使用して、共用ライブラリーを作成する場合、作成される共用オブジェクトのデフォルト名は **shr.o** です。

ld コマンドによって、明示的にリンケージ・エディターを起動することができます。コンパイラ呼び出しコマンドは、リンケージ・エディター・オプションをいくつか設定して、標準ファイルを実行可能出力にリンクします (デフォルト)。多くの場合、コンパイラ呼び出しコマンドの 1 つを使用して、オブジェクト・ファイルをリンク・エディットするようにしてください。

注: オブジェクト・ファイルのリンク・エディット時には、**ld** コマンドの **-e** オプションを使用しないでください。実行可能出力のデフォルト・エントリー・ポインタは、**__start** です。**-e** フラグを指定してこのラベルを変更すると、エラーの原因となる可能性があります。

関連概念

3 ページの『コンパイラー・モード』

関連タスク

33 ページの『コンパイラー・オプションの指定』

31 ページの『コンパイラーの起動』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

457 ページの『メッセージ重大度レベルとコンパイラー応答』

以下も参照してください。

「コマンド・リファレンス 第 5 巻」の **ld** コマンド

コンパイラー・オプションの指定

以下の方法のいずれかを使用してコンパイラー・オプションを指定することができます。

- コマンド行上に指定する (33 ページを参照してください)
- ソース・プログラム内に指定する (35 ページを参照してください)
- 構成 (.cfg) ファイル内に指定する (36 ページを参照してください)

上記の方法で明示的に設定されていないコンパイラー・オプションのほとんどについて、コンパイラーはデフォルト設定されているものと見なします。

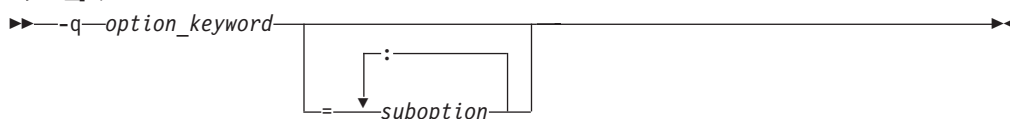
コマンド行におけるコンパイラー・オプションの指定

コマンド行で指定されたほとんどのオプションは、オプションのデフォルト設定と、構成ファイルのオプション・セットをオーバーライドします。同様に、コマンド行で指定されたほとんどのオプションは、プラグマ・ディレクティブによって逆にオーバーライドされ、ソース・ファイル中にコンパイラー・オプションを設定する手段を提供します。この方式に従わないオプションについては、『矛盾するコンパイラー・オプションの解決』にリストします。

コマンド行オプションには、次の 2 種類があります。

- **-qoption_keyword** (コンパイラー特定)
- フラグ・オプション

-q オプション



-qoption_keyword 書式のコマンド行オプションは、オン/オフ・スイッチと似ています。ほとんどの **-q** オプションの場合、特定のオプションが複数回指定されると、そのオプションの中でコマンド行に最後に出現したものがコンパイラーによって認識されるオプションです。例えば、**-qsource** により、コンパイラー・リストを作成するソース・オプションをオンにします。次に、**-qnosource** により、ソース・オプションをオフにすると、ソース・リストは作成されません。例を以下に示します。

```
xlc -qnosource MyFirstProg.c -qsource MyNewProg.c
```

このように指定すると、**MyNewProg.c** と **MyFirstProg.c** の両方のソース・リストが作成されます。これは、(**-qsource**) と指定された最後の **source** オプションが優先されるからです。

同一コマンド行で、複数の **-qoption_keyword** インスタンスを指定することができますが、これらのインスタンスはブランクで分けなければなりません。オプション・キーワードは、大文字または小文字のいずれかで表示されますが、**-q** は小文字で

指定しなければなりません。ファイル名の前または後に、任意の **-qoption_keyword** を指定することができます。例を以下に示します。

```
xlc -qLIST -qfloat=nomaf file.c
xlc file.c -qxref -qsource
```

多くのコンパイラー・オプションを省略することもできます。例えば、**-qopt** を指定するのは、コマンド行で **-qoptimize** を指定するのと同様です。

オプションの中には、サブオプションを持つものがあります。**-qoption** の次に等号を使用して、これらのサブオプションを指定します。オプションに複数のサブオプションを指定できる場合、コロン (:) で、各サブオプションを次のサブオプションから分けなければなりません。例を以下に示します。

```
xlc -qflag=w:e -qattr=full file.c
```

報告されるメッセージの重大度レベルを指定するために **-qflag** オプションを使用して、C ソース・ファイルの file.c をコンパイルします。**-qflag** サブオプション **w** (警告) はリストに関して報告される最低限の重大度レベルを設定し、サブオプション **e** (エラー) は端末に関して報告される最低限の重大度レベルを設定します。**full** サブオプションを指定した **-qflag** オプション **-qattr** は、プログラム内のすべての ID の属性リストを作成します。

フラグ・オプション

AIX システムで使用できるコンパイラーは、多くの共通標準フラグ・オプションを使用します。IBM XL C/C++ Enterprise Edition は、これらのフラグをサポートします。小文字のフラグは、対応する大文字フラグとは異なります。例えば、**-c** と **-C** は、別々のコンパイラー・オプションです。**-c** は、コンパイラーがプリプロセスとコンパイルのみを行い、リンケージ・エディターを起動しないことを指定します。一方、**-C** は、ユーザー・コメントの保存を指定するために **-P** または **-E** とともに使用することができます。

IBM XL C/C++ Enterprise Edition は、他の AIX プログラミング・ツールおよびユーティリティー (例えば、AIX **ld** コマンド) に対するフラグもサポートします。コンパイラーは、リンク・エディット時に、**ld** に対するこれらのフラグの受け渡しを行います。

フラグ・オプションの中には、フラグの一部を形成する引き数を持つものがあります。例を以下に示します。

```
xlc stem.c -F/home/tools/test3/new.cfg:xlc
```

ここで、new.cfg はカスタム構成ファイルです。

1 つのストリングで引き数を指定しないフラグを指定することができます。例を以下に示します。

```
xlc -Ocv file.c
```

これは、以下の指定と同じ効果があります。

```
xlc -O -c -v file.c
```

そして、C ソース・ファイル `file.c` を最適化を指定して (**-O**) コンパイルし、コンパイラーの進行について報告します (**-v**) が、リンケージ・エディターは呼び出しません (**-c**)。

引き数を指定するフラグ・オプションは、単一ストリングの一部として指定することができます。しかし、引き数を指定する 1 フラグしか使用することができず、さらにこのフラグは指定される最後のオプションでなければなりません。例えば、(実行可能ファイルの名前を指定するため) ほかのフラグと一緒に **-o** フラグを使用することができるのは、**-o** オプションとその引き数が最後に指定されている場合のみです。例を以下に示します。

```
xlc -Ovo test test.c
```

これは、以下の指定と同じ効果があります。

```
xlc -O -v -otest test.c
```

ほとんどのフラグ・オプションは一文字ですが、中には二文字のものもあります。**-pg** (拡張プロファイル) を指定するのは **-p -g** (プロファイルの **-p**、およびデバッグ情報生成の **-g**) を指定するのと同じではないことに注意してください。該当する文字の組み合わせを使用する別のオプションがある場合、単一ストリングで複数のオプションを指定しないように注意してください。

関連概念

6 ページの『コンパイラー・オプション』

関連タスク

31 ページの『コンパイラーの起動』

『プログラム・ソース・ファイル内のコンパイラー・オプションの指定』

36 ページの『構成ファイル内のコンパイラー・オプションの指定』

39 ページの『アーキテクチャー固有の (32 ビットまたは 64 ビットの) コンパイル用コンパイラー・オプションの指定』

41 ページの『矛盾するコンパイラー・オプションの解決』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

プログラム・ソース・ファイル内のコンパイラー・オプションの指定

`#pragma` ディレクティブを使用すると、プログラム・ソース内でコンパイラー・オプションを指定できます。

プラグマは、コンパイラーに対してインプリメンテーションを定義した命令です。プラグマには、以下の一般的な書式があり、*character_sequence* は、特定のコンパイラーの命令および引き数 (ある場合) を提供する一連の文字です。

▶▶ `#pragma` *character_sequence* ▶▶

プラグマの *character_sequence* は、特に指定がない限り、マクロ置換されます。複数のプラグマの構成は、1 つの `#pragma` ディレクティブで指定できます。コンパイラーは、認識されないプラグマを無視し、無視したことを示す通知メッセージを発行します。

プログラム・ソース・ファイルでプラグマ・ディレクティブにより指定されたオプションは、ほかのプラグマ・ディレクティブを除く、ほかのすべてのオプション設定をオーバーライドします。2 回以上同じプラグマ・ディレクティブを指定する効果はさまざまです。特定の情報についてはそれぞれのプラグマに関する説明を参照してください。

プラグマ設定は組み込まれたファイルに持ち越すことができます。プラグマ設定からの望ましくない副次作用の可能性を除くには、プログラム・ソースでプラグマ定義の振る舞いが必要なくなった時点でプラグマ設定のリセットを考慮する必要があります。一部のプラグマ・オプションは **reset** または **pop** サブオプションを提供し、これを実行するのを助けます。

これらの **#pragma** ディレクティブは、これらのディレクティブが適用されるオプションの詳細記述にリストされます。各種の **#pragma** プリプロセッサ・ディレクティブに関する詳細な説明については、*汎用プラグマ* を参照してください。

関連概念

6 ページの『コンパイラー・オプション』

関連タスク

31 ページの『コンパイラーの起動』

33 ページの『コマンド行におけるコンパイラー・オプションの指定』

『構成ファイル内のコンパイラー・オプションの指定』

39 ページの『アーキテクチャー固有の (32 ビットまたは 64 ビットの) コンパイル用コンパイラー・オプションの指定』

41 ページの『矛盾するコンパイラー・オプションの解決』

関連資料

349 ページの『汎用プラグマ』

422 ページの『並列処理を制御するプラグマ』

構成ファイル内のコンパイラー・オプションの指定

デフォルト構成ファイル (*/etc/vac.cfg*) は、コンパイラー用の値およびコンパイラー・オプションを定義します。コンパイラーは、C または C++ プログラムをコンパイルするときにこのファイルを参照します。構成ファイルはプレーン・テキスト・ファイルであり、特定のコンパイル要件をサポートするためか、または別の C または C++ コンパイル環境をサポートするために、このファイルに項目を作成することができます。

構成ファイルで指定したほとんどのオプションは、オプションのデフォルト設定をオーバーライドします。同様に、構成ファイルで指定されたほとんどのオプションは、逆に、ソース・ファイル内またはコマンド行上で設定されたオプションでオー

オーバーライドされます。この方式に従わないオプションについては、『矛盾するコンパイラー・オプションの解決』にリストします。

構成ファイルの調整

デフォルト構成ファイルは、`/etc/vac.cfg` にインストールされます。

このファイルをコピーして、そのコピーに変更を加え、特定のコンパイル要件をサポートさせるか、または別の C または C++ コンパイル環境をサポートさせることができます。構成ファイル作成に関して詳しくは、

既存のスタンザを変更するか、または新しいスタンザを構成ファイルに追加することができます。例えば、**-qnoro** を **xlC** コンパイラー呼び出しコマンドのデフォルトにするには、構成ファイルのコピー版の **xlC** スタンザに **-qnoro** を追加します。

さまざまな複数の名前に、コンパイラー呼び出しコマンドをリンクさせることができます。コンパイラーを起動するときに指定する名前により、コンパイラーが使用する構成ファイルのスタンザが決まります。構成ファイルのコピーにほかのスタンザを追加して、ユーザー固有のコンパイル環境をカスタマイズすることができます。コンパイラー呼び出しコマンドで **-F** オプションを使用して、追加のスタンザを選択するか、または別の構成ファイル内の特定のスタンザを指定するためのリンクを作成することができます。例を以下に示します。

```
xlC myfile.c -Fmyconfig:SPECIAL
```

これにより、ユーザーの作成した `myconfig.cfg` 構成ファイルの `SPECIAL` スタンザを使用して、`myfile.c` をコンパイルします。

構成ファイルの属性

構成ファイルには、いくつかのスタンザが含まれます。以下は、構成ファイル内のスタンザによって定義される項目の一部です。

as	アセンブラーで使用されるパス名です。デフォルトは <code>/bin/as</code> です。
asopt	アセンブラー用のオプション・リストで、コンパイラー用ではありません。これらのオプションは、コンパイラーによるすべての標準処理をオーバーライドして、 as スタンザで指定されたアセンブラーに送られます。対応するフラグにパラメーターを指定する場合、ストリングは、1 つの文字の後にコロン (:) が続く形のフラグ文字の連結として、AIX getopt() サブルーチン用にフォーマットされます。
ccomp	C フロントエンドです。デフォルトは <code>/usr/vac/exe/xlcentry</code> です。
codeopt	コンパイラーのコード生成フェーズ用のオプション・リストです。
cppcode	コンパイラーのコード生成フェーズで使用されるパス名です。デフォルトは <code>/usr/vac/exe/xlccode</code> です。
cppopt	コンパイラーの字句解析フェーズ用のオプション・リストです。
crt	リンケージ・エディターへの先頭パラメーターとして渡されるオブジェクト・ファイルのパス名です。 -p と -pg オプションのいずれも指定しない場合、 crt 値が使用されます。デフォルトは <code>/lib/crt0.o</code> です。
csuffix	ソース・プログラムのサフィックスです。デフォルトは c (小文字の c) です。
dis	逆アセンブラーのパス名です。デフォルトは <code>/usr/vacpp/exe/dis</code> です。
gcrt	リンケージ・エディターへの先頭パラメーターとして渡されるオブジェクト・ファイルのパス名です。 -pg オプションを指定すると、 gcrt 値が使用されます。デフォルトは <code>/lib/gcrt0.o</code> です。

ld	C または C++ プログラムのリンクに使用されるパス名。デフォルトは /bin/ld です。
ldopt	コンパイラーのリンケージ・エディター部分に送られるオプション・リストです。これらは、コンパイラーによる標準処理のすべてをオーバーライドして、リンケージ・エディターに送られます。対応するフラグにパラメーターを指定する場合、ストリングは、1 つの文字の後にコロン (:) が続く形のフラグ文字の連結として、 getopt() サブルーチン用にフォーマットされます。
libraries2	コンマで区切られたライブラリー・オプションです。コンパイラーは、最後のパラメーターとしてこのオプションをリンケージ・エディターに渡します。 libraries2 は、リンケージ・エディターがプロファイルおよび非プロファイルの両方に応じてリンク・エディット時に使用するライブラリーを指定します。デフォルトは空です。
mcr	-p オプションを指定した場合に、リンケージ・エディターへの先頭パラメーターとして渡されるオブジェクト・ファイルのパス名です。デフォルトは /lib/mcrt0.o です。
options	コンマで区切られたオプション・フラグのストリングです。このストリングは、コマンド行で入力されたかのように、コンパイラーにより処理されます。
osuffix	オブジェクト・ファイルのサフィックスです。デフォルトは .o です。
proflibs	コンマで区切られたライブラリー・オプションです。コンパイラーは、プロファイル・オプションの指定時に、これらのオプションをリンケージ・エディターに渡します。 proflibs は、リンク・エディット時にリンケージ・エディターが使用するプロファイル・ライブラリーを指定します。デフォルトは、 -L/lib/profiled および -L/usr/lib/profiled です。
ssuffix	アセンブラー・ファイルのサフィックスです。デフォルトは .s です。
use	属性の値は、名前付きのスタンザまたはローカル・スタンザから採用されます。単一値属性の場合、ローカルまたはデフォルトのスタンザに値が指定されていないと、 use スタンザの値が適用されます。コンマで区切られたリストの場合、 use スタンザの値が、ローカル・スタンザの値に追加されます。
xlc	xlc コンパイラー・コンポーネントのパス名。デフォルトは /usr/vac/bin/xlc です。
xlC	xlC コンパイラー・コンポーネントのパス名です。デフォルトは /usr/vac/bin/xlC です。

関連概念

6 ページの『コンパイラー・オプション』

関連タスク

31 ページの『コンパイラーの起動』

33 ページの『コマンド行におけるコンパイラー・オプションの指定』

35 ページの『プログラム・ソース・ファイル内のコンパイラー・オプションの指定』

39 ページの『アーキテクチャー固有の (32 ビットまたは 64 ビットの) コンパイル用コンパイラー・オプションの指定』

41 ページの『矛盾するコンパイラー・オプションの解決』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

「*Getting Started with XL C/C++ Enterprise Edition for AIX*」の『コンパイラーの構成』セクションも参照してください。

アーキテクチャー固有の (32 ビットまたは 64 ビットの) コンパイラー・オプションの指定

IBM XL C/C++ Enterprise Edition コンパイラー・オプションを使用して、特定のプロセッサ・アーキテクチャーで使用するためにコンパイラー出力を最適化することができます。32 ビットまたは 64 ビットのいずれかのモードでコンパイルするようにコンパイラーに指示することもできます。

コンパイラーは、コンパイラー・モードを判別する最後に検出された有効なコンパイラー・オプションによって、以下の順でコンパイラー・オプションを評価します。

1. 内部のデフォルト (32 ビット・モード)
2. 以下の OBJECT_MODE 環境変数の設定

OBJECT_MODE の設定値	ユーザーが選択したコンパイル・モードの振る舞い (構成ファイルまたはコマンド行オプションでオーバーライドされない場合に限る)
未設定	32 ビットのコンパイラー・モード
32	32 ビットのコンパイラー・モード
64	64 ビットのコンパイラー・モード
32_64	明示的構成ファイルまたはコマンド行コンパイラー・モード設定値が存在していない限り、致命的エラーであり、「1501-254 OBJECT_MODE=32_64 is not a valid setting for the compiler」というメッセージを出して停止します。
その他	明示的構成ファイルまたはコマンド行コンパイラー・モード設定値が存在していない限り、致命的エラーであり、「1501-255 OBJECT_MODE setting is not recognized and is not a valid setting for the compiler」というメッセージを出して停止します。

3. 構成ファイルの設定
4. コマンド行コンパイラー・オプション (**-q32**、**-q64**、**-qarch**、**-qtune**)
5. ソース・ファイルのステートメント (**#pragma options tune=suboption**)

コンパイラーが実際に使用するコンパイル・モードは、**-q32**、**-q64**、**-qarch**、および **-qtune** コンパイラー・オプションの設定値の組み合わせに応じて、以下の条件に従って決定されます。

- コンパイラー・モード は、**-q32** または **-q64** コンパイラー・オプションの最後に検出されたインスタンスに応じて設定されます。これらのコンパイラー・オプションをいずれも選択しない場合は、コンパイラー・モードは、OBJECT_MODE 環境変数の値によって設定されます。OBJECT_MODE 環境変数も設定されていない場合、コンパイラーは 32 ビット・コンパイル・モードであると見なします。
- アーキテクチャー・ターゲット は、指定された **-qarch** の設定値がコンパイラー・モード の設定値と互換性がある場合は、**-qarch** コンパイラー・オプションの最後に検出されたインスタンスに応じて設定されます。**-qarch** オプションを設定しない場合は、コンパイラーは **-qarch** の設定値が **com** であると見なします。
- アーキテクチャー・ターゲットの調整は、**-qtune** の設定値がアーキテクチャー・ターゲット およびコンパイラー・モード の設定値と互換性がある場合は、**-qtune** コンパイラー・オプションの最後に検出されたインスタンスに応じて設定されます。**-qtune** オプションを設定しない場合は、コンパイラーは、使用中の

-qarch の設定値に応じてデフォルトの **-qtune** の設定値であると見なします。
-qarch を指定しない場合は、コンパイラーは **-qtune** 設定値が **pwr3** であると見なします。

これらのオプションの有効な組み合わせは、コンパイラー・モードおよびプロセッサのアーキテクチャーの有効な組み合わせの表に示します。

以下では、可能性のあるオプションの矛盾およびこれらの矛盾のコンパイラーによる解決について説明します。

- **-q32** または **-q64** の設定値が、ユーザーが選択した **-qarch** オプションと互換性がない。

解決: **-q32** または **-q64** の設定値が **-qarch** オプションをオーバーライドします。コンパイラーは警告メッセージを出し、**-qarch** をそのデフォルト設定値に設定して、**-qtune** オプションをそのデフォルト値に応じて設定します。

- **-q32** または **-q64** の設定値が、ユーザーが選択した **-qtune** オプションと互換性がない。

解決: **-q32** または **-q64** の設定値が **-qtune** オプションをオーバーライドします。コンパイラーは警告メッセージを出し、**-qtune** オプションを **-qarch** の設定値に対するデフォルトの **-qtune** 値に設定します。

- **-qarch** オプションが、ユーザーが選択した **-qtune** オプションと互換性がない。

解決: コンパイラーは警告メッセージを出し、**-qtune** を **-qarch** の設定値に対するデフォルトの **-qtune** 値に設定します。

- 選択した **-qarch** または **-qtune** オプションがコンパイラーに認識されない。

解決: コンパイラーは警告メッセージを出し、**-qarch** および **-qtune** をデフォルトの設定値に設定します。コンパイラー・モード (32 ビットまたは 64 ビット) は、OBJECT_MODE 環境変数か、あるいは **-q32** または **-q64** コンパイラー設定によって判別されます。

関連概念

6 ページの『コンパイラー・オプション』

関連タスク

31 ページの『コンパイラーの起動』

33 ページの『コマンド行におけるコンパイラー・オプションの指定』

28 ページの『64 ビットまたは 32 ビット・モードへの環境変数の設定』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

41 ページの『矛盾するコンパイラー・オプションの解決』

452 ページの『コンパイラー・モードおよびプロセッサのアーキテクチャーの有効な組み合わせ』

矛盾するコンパイラー・オプションの解決

一般に、同じオプションの複数のバリエーションが指定されている場合は (**xref** と **attr** を除く)、コンパイラーは、最後に指定されたオプションの設定値を使用します。コマンド行に指定するコンパイラー・オプションは、コンパイラーに処理させる順序で指定しなければなりません。

矛盾するオプションの規則には、**-ldirectory** オプションおよび **-Ldirectory** オプションの 2 つの例外があります。これらが複数回指定された場合には、その効果が累積されます。

多くの場合、コンパイラーは、以下の順序で、対立するオプションまたは矛盾するオプションを解決します。

1. ソース・コード内のプラグマ・ステートメントは、コマンド行で指定されたコンパイラー・オプションをオーバーライドします。
2. コマンド行に指定されるコンパイラー・オプションは、構成ファイルに指定されたコンパイラー・オプションをオーバーライドします。対立する、または互換性のないコンパイラー・オプションがコマンド行で指定されている場合、コマンド行で後のほうに指定されているオプションが優先されます。
3. 構成ファイルに指定されるコンパイラー・オプションは、コンパイラーのデフォルト設定をオーバーライドします。

すべてのオプション競合が上記の規則で解決するわけではありません。下のテーブルでは、例外およびコンパイラーがそれらの間の競合を処理する方法を要約しています。

オプション	矛盾するオプション	解決
-qhalt	-qhalt によって指定される複数の重大度	指定された最低の重大度
-qnoprint	-qxref -qattr -qsource -qlistopt -qlist	-qnoprint
-qfloat=rsqrt	-qnoignerrno	最後に指定されたオプション
-qxref	-qxref=FULL	-qxref=FULL
-qattr	-qattr=FULL	-qattr=FULL
-p -pg -qprofile	-p -pg -qprofile	最後に指定されたオプション
-qhsflt	-qrndsngl -qspnans	-qhsflt
-qhssngl	-qrndsngl -qspnans	-qhssngl
-E	-P -o -S	-E
-P	-c -o -S	-P
-#	-v	-#
-F	-B -t -W -qpath 構成ファイル設定	-B -t -W -qpath
-qpath	-B -t	-qpath は -B および -t をオーバーライドする
-S	-c	-S

関連概念

6 ページの『コンパイラー・オプション』

関連タスク

31 ページの『コンパイラーの起動』

33 ページの『コマンド行におけるコンパイラー・オプションの指定』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

インクルード・ファイル用のパス名の指定

#include プリプロセッサ・ディレクティブを使用して、あるソース・ファイルを別のファイルに組み込むときには、組み込まれるファイル名を指定しなければなりません。絶対パス名または相対パス名を使用して、ファイル名を指定することができます。

- 絶対パス名を使用したファイルの組み込み

絶対パス名 と呼ばれる絶対パス名 は、ルート・ディレクトリーから始まるファイルの完全な名前です。これらのパス名は、/ (スラッシュ) 文字で始まります。絶対パス名は、ユーザーが現在入っているディレクトリー (作業 ディレクトリーまたは現行 ディレクトリーと呼ばれる) にかかわらず、指定されたファイルを探し出します。

以下の例では、John Doe のサブディレクトリー `example_prog` にあるファイル `mine.h` を指す絶対パスを指定します。

```
/u/johndoe/example_prog/mine.h
```

- 相対パス名を使用したファイルの組み込み

相対パス名 は、現行ソース・ファイルを保持するディレクトリー、または **-ldirectory** オプションを使用して定義されたディレクトリーに関連するファイルを探し出します。

相対パス名を使用したインクルード・ファイルのディレクトリー検索順序

C および C++ 言語では、**#include** プリプロセッサ・ディレクティブの 2 つのバージョンを定義します。IBM XL C/C++ Enterprise Edition は、これらを両方ともサポートします。**#include** ディレクティブで、インクルード・ファイル名は区切り文字 `< >` または `" "` で囲まれています。

選択した区切り文字によって、特定のインクルード・ファイル名を探し出すのに使用される検索パスが決定されます。コンパイラーは、インクルード・ファイルが見つかるまで、以下のように検索パスの全ディレクトリーでそのインクルード・ファイルを検索します。

#include の型	ディレクトリー検索順序
#include <file_name>	<ol style="list-style-type: none"> コンパイラーはまず、-ldirectory コンパイラー・オプションによって指定された各ユーザー・ディレクトリーで、コマンド行に表示される順に <i>file_name</i> を検索します。 C++ コンパイルの場合、コンパイラーはそれから C++ ヘッダーの標準検索パスを検索します。デフォルトの C++ ヘッダー用標準検索パスは /usr/vacpp/include ですが、このパスは -qcpp_stdinc コンパイラー・オプションによって変更することができます。 最後に、コンパイラーは C ヘッダーの標準検索パスを検索します。デフォルトの C ヘッダー用標準検索パスは /usr/include ですが、このパスは -qc_stdinc コンパイラー・オプションによって変更することができます。
#include "file_name"	<ol style="list-style-type: none"> コンパイラーはまず、現行のソース・ファイルが常駐しているディレクトリーでインクルード・ファイルを検索します。現行のソース・ファイルは、ディレクティブ #include "file_name" を含むファイルです。 コンパイラーはそれから、#include <file_name> の上記の検索順序に従ってインクルード・ファイルを検索します。

注:

- file_name* は、組み込まれるファイル名を指定し、必要に応じて、そのファイルへの完全または部分ディレクトリー・パスを含むことができます。
 - ファイル名のみを指定した場合、コンパイラーは、ディレクトリー検索リストの中でそのファイルを探します。
 - ファイル名を部分ディレクトリー・パスと一緒に指定した場合、コンパイラーは、検索パスの中の各ディレクトリーにその部分パスを付加し、完全になったディレクトリー・パスの中でそのファイルの検索を試みます。
 - 絶対パス名を指定した場合、**#include** ディレクティブの 2 つのバージョンの効果は同じです。これは、組み込まれるファイルのロケーションが完全に指定されているからです。
- #include** ディレクティブの 2 つのバージョンの唯一の違いは、“ ” (ユーザー組み込み) バージョンでは、最初に現行ソース・ファイルの常駐するディレクトリーから検索を開始するということです。一般的に、標準ヘッダー・ファイルは < > (システム組み込み) バージョンを使用して組み込み、ユーザーの作成するヘッダー・ファイルは、“ ” (ユーザー組み込み) バージョンを使用して組み込みます。
- qstdinc** および **-qidirfirst** オプションと一緒に、**-ldirectory** オプションを指定することにより、検索順序を変更することができます。

該当する場合は、**-ldirectory** オプションで指定したディレクトリーと、現行のソース・ファイル・ディレクトリーのみを検索するため、**-qnostdinc** オプションを使用します。C プログラムの場合、**/usr/include** ディレクトリーは検索されません。C++ プログラムの場合、**/usr/vacpp/include** および **/usr/include** ディレクトリーは検索されません。

ほかのディレクトリーを検索する前に、**-ldirectory** オプションで指定したディレクトリーを検索するには、**#include "file_name"** ディレクティブと共に **-qidirfirst** オプションを使用します。

ディレクトリー検索パスを指定するには、**-I** オプションを使用します。

関連資料

166 ページの『I』

100 ページの『c_stdinc』

114 ページの『cpp_stdinc』

167 ページの『idirfirst』

304 ページの『stdinc』

プラグマを使用した並列処理の制御

並列処理操作は、ユーザーのプログラム・ソースでプラグマ・ディレクティブによって制御します。プラグマは、**-qsmp** コンパイラー・オプションで並列化が使用可能になっているときにのみ有効です。

IBM SMP または OpenMP ディレクティブは、C プログラムで使用することができます。OpenMP ディレクティブは、C および C++ プログラムで使用できます。それぞれに独自の使用特性があります。

IBM SMP ディレクティブ

▶ C

構文

```
#pragma ibm pragma_name_and_args  
<countable for|while|do loop>
```

プラグマ・ディレクティブは、適用対象のコードのセクションの直前に記述しなければなりません。ほとんどの並列処理プラグマ・ディレクティブの場合は、このコードのセクションはカウント可能なループでなければならない、そのようなループが見つからない場合は、コンパイラーによってエラーが報告されます。

複数の並列処理プラグマ・ディレクティブを 1 つのカウント可能ループに適用することができます。例を以下に示します。

```
#pragma ibm independent_loop  
#pragma ibm independent_calls  
#pragma ibm schedule(static,5)  
<countable for|while|do loop>
```

プラグマ・ディレクティブによっては、同時に指定できないものもあります。同時に指定できないプラグマを同じループに指定した場合は、最後に指定したプラグマがループに適用されます。以下の例では、**parallel_loop** プラグマ・ディレクティブがループに適用され、**sequential_loop** プラグマ・ディレクティブは無視されます。

```
#pragma ibm sequential_loop  
#pragma ibm parallel_loop
```

あるループに対してその他のプラグマを繰り返し指定すると、プラグマの効力が追加されます。例を以下に示します。

```
#pragma ibm permutation (a,b)  
#pragma ibm permutation (c)
```

は、以下と同等です。

```
#pragma ibm permutation (a,b,c)
```

OpenMP ディレクティブ

► C ► C++

構文

```
#pragma omp pragma_name_and_args
statement_block
```

プラグマ・ディレクティブは、一般に、適用対象のコードのセクションの直前に記述します。**omp parallel** ディレクティブは、並列化されるプログラム・コードの領域を定義する場合に使用します。その他の OpenMP ディレクティブは、定義済み並列領域内のデータ変数の可視性、およびその領域内の作業の共用方法と同期方法を定義します。

例えば、以下の例は、**for** ループの反復が並列で実行できる並列領域を定義しています。

```
#pragma omp parallel
{
    #pragma omp for
    for (i=0; i<n; i++)
        ...
}
```

この例は、プログラム・コードの複数の非反復セクションが並列で実行できる並列領域を定義しています。

```
#pragma omp sections
{
    #pragma omp section
    structured_block_1
    ...
    #pragma omp section
    structured_block_2
    ...
    ....
}
```

関連概念

15 ページの『プログラムの並列化』

関連タスク

28 ページの『並列処理ランタイム・オプションの設定』

関連資料

293 ページの『smp』

422 ページの『並列処理を制御するプラグマ』

461 ページの『IBM SMP 並列処理のためのランタイム・オプション』

465 ページの『並列処理のための OpenMP ランタイム・オプション』

467 ページの『並列処理に使用する組み込み関数』

OpenMP 仕様の完全な情報については、以下を参照してください。

- OpenMP Web サイト
- OpenMP 仕様

他のプログラム言語での C および C++ の使用

他のプログラミング言語で作成されたオブジェクトを、C または C++ プログラムにおいて使用できます。このセクションの以下のトピックは、そのような場合に順守すべきプログラミング上の考慮事項を概説します。

- 『言語間呼び出し規則』
- 『対応するデータ型』
- 52 ページの『言語間呼び出しにおけるサブルーチン・リンケージ規約の使用』
- 57 ページの『サンプル・プログラム: Fortran を呼び出す C/C++』

言語間呼び出し規則

別の言語で作成された関数を呼び出す C および C++ コードを作成するときは、以下の勧告に従ってください。

- ID に英大文字を使用しないようにする。Fortran と Pascal では、すべての外部名に英小文字だけを使用する。デフォルトでは、両方とも外部 ID を小文字に畳み込みますが、Fortran コンパイラーは、外部名を大/小文字によって区別するように設定することができます。
- C and C++ 言語ライブラリーの命名規則の矛盾を防ぐため、ID の先頭文字に下線 (_) とドル記号 (\$) を使用しないようにする。
- 長 ID 名を使用しないようにする。ID における有効文字の最大数は 250 文字である。

関連概念

23 ページの『他のプログラム言語での XL C/C++ Enterprise Edition の使用』

関連タスク

『対応するデータ型』

52 ページの『言語間呼び出しにおけるサブルーチン・リンケージ規約の使用』

57 ページの『サンプル・プログラム: Fortran を呼び出す C/C++』

対応するデータ型

以下の表は、C/C++、Fortran、および Pascal で使用できるデータ型の間の対応を示します。C のデータ型の中には、Pascal または Fortran で等価表示のないものがあります。言語間呼び出しのプログラミングの際には、等価表示のないものを使用しないでください。表項目がブランクの部分は、一致するデータ型がないことを示します。

C、C++、Fortran、および Pascal におけるデータ型の対応

C および C++ データ型	Fortran データ型	Pascal データ型
bool (C++) _Bool (C)	LOGICAL*4	—

C および C++ データ型	Fortran データ型	Pascal データ型
char	CHARACTER	CHAR
signed char	INTEGER*1	PACKED -128..127
unsigned char	LOGICAL*1	PACKED 0..255
signed short int	INTEGER*2	PACKED -32768..32767
unsigned short int	LOGICAL*2	PACKED 0..65535
signed long int	INTEGER*4	INTEGER
unsigned long int	LOGICAL*4	—
signed long long int	INTEGER*8	—
unsigned long long int	LOGICAL*8	—
float	REAL REAL*4	SHORTREAL
double	REAL*8 DOUBLE PRECISION	REAL
long double (デフォルト)	REAL*8 DOUBLE PRECISION	REAL
long double (-qlongdouble または -qldbl128 を指定)	REAL*16	—
2 つの float の構造体	COMPLEX COMPLEX*4	2 つの SHORTREALS の RECORD
2 つの double の構造体	COMPLEX*16 DOUBLE COMPLEX	2 つの REALS の RECORD
2 つの long double の構造体 (デフォルト)	COMPLEX*16	—
構造体	—	RECORD (下記の注を参照)
enumeration	INTEGER*4	Enumeration
char[n]	CHARACTER*n	PACKED ARRAY[1..n] OF CHAR
型を示す配列ポインター (*) または型 []	次元変数 (置き換え済み)	ARRAY
関数を示すポインター (*)	関数パラメーター	関数パラメーター
structure (-qalign=packed を 指定)	シーケンス派生型	PACKED RECORD

文字および集約データの特珠な取り扱い

ほとんどの数値データ型には 3 つの言語間で対応するものがあります。文字および集約データ型は、以下の特殊な処理を必要とします。

- 埋め込みと位置合わせの相違のため、C 構造体は Pascal **RECORD** データ型と正しく対応しません。
- C 文字ストリングは、'¥0' 文字で区切られます。Fortran では、すべての文字変数および式の長さは、コンパイル時に判別されます。Fortran が別のルーチンにストリング引き数を渡す場合、引き数リストの最後に、長さを示す隠し引き数を追加します。この長さ引き数は、C で明示的に宣言しなければなりません。C コードは、ヌル終止符を想定しないため、提供または宣言された長さが常に使用さ

れます。C ランタイム・ライブラリーの **strncat**、**strncpm**、および **strncpy** 関数を使用してください。これらの関数の説明は、「*Technical Reference, Volumes 1 and 2: Base Operating System and Extensions*」にあります。

- Pascal の STRING データ型は C 構造体に対応します。例えば、

```
VAR s: STRING(10);
```

は、以下と同等です。

```
struct {  
    int length;  
    char str [10];  
};
```

ここでは、length に STRING の実際の長さが含まれています。

- **-qmacpstr** オプションにより、Pascal のストリング・リテラルがヌル終了のストリングに変換されます。このストリングには、その先頭バイトにストリングの長さが含まれます。
- C と Pascal は、行本位の順番で配列エレメントを保管します (同じ行の配列エレメントが隣接するメモリの位置を占めます)。Fortran は、列本位の順番で昇順の記憶装置に配列エレメントを保管します (同じ列の配列エレメントが隣接するメモリの位置を占めます)。以下の例では、C の A[3][2]、Pascal の A[1..3,1..2]、Fortran の A(3,2) で宣言された二次元配列の保管方法を示します。

二次元配列の保管			
記憶装置	C および C++ エレメント名	Pascal エレメント名	Fortran エレメント名
最下位	A[0][0]	A[1,1]	A(1,1)
	A[0][1]	A[1,2]	A(2,1)
	A[1][0]	A[2,1]	A(3,1)
	A[1][1]	A[2,2]	A(1,2)
	A[2][0]	A[3,1]	A(2,2)
再高位	A[2][1]	A[3,2]	A(3,2)

- 一般的に多次元配列の場合、メモリーに展開されている順に配列エレメントをリストすると、行本位の配列では右端の指標が最も速く変わり、列本位の配列では左端の指標が最も速く変わります。

関連概念

23 ページの『他のプログラム言語での XL C/C++ Enterprise Edition の使用』

関連タスク

49 ページの『言語間呼び出し規則』

52 ページの『言語間呼び出しにおけるサブルーチン・リンケージ規約の使用』

57 ページの『サンプル・プログラム: Fortran を呼び出す C/C++』

関連資料

以下も参照してください。

AIX 5L Version 5.1 Technical Reference: Base Operating System and Extensions,

言語間呼び出しにおけるサブルーチン・リンケージ規約の使用

サブルーチン・リンケージ規約では、サブルーチンの入り口と出口におけるマシン状態を記述します。異なる言語で書き込まれ、別々のオブジェクト・ファイルにコンパイルされたルーチンは、このセクションで説明するサブルーチン・リンケージ規約に準拠する場合、リンクされ、単一プロセスとして実行されることができません。

これらのリンケージ規約は、言語間の高速で効率的なサブルーチン・リンケージを提供します。これらのリンケージ規約は、浮動小数点レジスター (FPR) と汎用レジスター (GPR) をフルに利用したパラメーターの受け渡し方法を指定し、サブルーチンの入り口と出口に関するレジスターの保管と復元を最小化します。

- 『言語間呼び出し - パラメーターの受け渡し』
- 53 ページの『言語間呼び出し - 参照による呼び出しパラメーター』
- 54 ページの『言語間呼び出し - 値パラメーターによる呼び出し』
- 54 ページの『言語間呼び出し - 値によるパラメーターの受け渡し規則』
- 55 ページの『言語間呼び出し - 関数を指すポインター』
- 56 ページの『言語間呼び出し - 関数からの戻り値』
- 56 ページの『言語間呼び出し - スタック・フロア』
- 56 ページの『言語間呼び出し - スタック・オーバーフロー』
- 56 ページの『言語間呼び出し - トレースバック・テーブル』
- 57 ページの『言語間呼び出し - 型のエンコードと検査』
- 57 ページの『サンプル・プログラム: Fortran を呼び出す C/C++』

関連概念

23 ページの『他のプログラム言語での XL C/C++ Enterprise Edition の使用』

関連タスク

49 ページの『言語間呼び出し規則』

49 ページの『対応するデータ型』

言語間呼び出し - パラメーターの受け渡し

リンケージ規約は、パラメーターの受け渡し方式と、FPR、GPR、またはこれらの両方に戻り値を入れるかどうかを指定します。引き数の受け渡しに使用できる GPR と FPR は、R3-R10 と FP1-FP13 という 2 つの固定リストに指定します。

プロトタイピングは、パラメーターの受け渡し方法と、拡大が起こるかどうかに影響を与えます。

非プロトタイプ関数

C 言語の非プロトタイプ関数では、浮動小数点引き数は **double** に拡大され、integral 型は **int** に拡大されます。

プロトタイプ関数

省略符号関数に渡される引き数を除いて、拡大変換は起こりません。浮動小数点 **double** 引き数は、FPR にのみ渡されます。省略符号がプロトタイプに存在する場合、浮動小数点 **double** 引き数は、FPR と GPR の両方に渡されます。

使用できるパラメーターの GPR と FPR よりも多い引き数ワードがあるときに、残りのワードはスタック上のストレージに渡されます。ストレージの値は、それらの値がレジスターにある場合と同様です。引き数のすべてがレジスターに渡された場合であっても、引き数 (浮動または非浮動) の 8 ワードを超えるスペースは、スタック上に予約しておかなければなりません。

パラメーター・エリアのサイズは、スタック・フレームに関連したプロシージャーからの呼び出しステートメントで渡されるすべての引き数を含むには不足しています。特定の呼び出しの引き数すべてが実際にストレージに存在するわけではないけれども、これらの引き数は、各引き数が 1 つまたは複数のワードを占有して、このエリアのリストを形成していると見なすことができます。

パラメーターの受け渡し方法は、以下のとおりです。

- C では、すべての関数引き数は値により渡され、呼び出し先関数は、その関数に渡された値のコピーを受け取ります。
- Fortran では、引き数は参照により渡され (デフォルト)、呼び出し先関数は、その関数に渡された値のアドレスを受け取ります。値による受け渡しを行うには、**%VAL** Fortran 組み込み関数を使うことができます。 **%VAL** および言語間呼び出しの使用方法についての詳細は、「*AIX XL Fortran User's Guide*」を参照してください。
- Pascal では、関数宣言は、値または参照のどちらによりパラメーターを渡すことが期待されているかを判別します。

言語間呼び出し - 参照による呼び出しパラメーター

(Fortran におけるような) 参照による呼び出しパラメーターの場合、パラメーターのアドレスはレジスターで渡されます。

参照によるパラメーターを渡すときには、以下のとおり、C または C++ 関数を作成する場合にそれぞれ注意事項があります。

- Fortran プログラムから呼び出したい C 関数を作成する場合、すべてのパラメーターをポインターとして宣言する。
- Fortran で作成されたプログラムを呼び出す C 関数を作成する場合、すべての引き数は、アドレス演算子を指定したポインターまたはスカラーでなければならない。
- Pascal プログラムから呼び出したい C 関数を作成する場合、Pascal プログラムがすべてのパラメーターを参照パラメーターとして扱うポインターとして宣言する。
- Pascal で作成されたプログラムを呼び出す C 関数を作成する場合、参照パラメーターに対応する引き数はすべて、ポインターでなければならない。

言語間呼び出し - 値パラメーターによる呼び出し

function(...) のように省略符号で指定された、数が可変の引き数を持つプロトタイプ関数では、コンパイラーは、すべての浮動小数点引き数を倍精度に拡大します。整数引き数 (**long int** を除く) は、**int** に広げられます。この拡張により、データ型の中には、明示的な変換をしないと Pascal と C の間で渡すことができないものがあります。また、Pascal ルーチンはあるデータ型の値パラメーターを持つことができません。

以下の情報は、C における値による呼び出しに関することです。以下のリストでは、引き数は浮動値または非浮動値として分類されます。

- それぞれの非浮動スカラー引き数には 1 ワードが必要で、各引き数は GPR (汎用レジスター) に表示されるようにそのワードの中に表示される。言語セマンティクスが指定し、ワード境界に位置合わせされる場合、各引き数は右寄せされる。
- 各浮動値は 1 ワードを占め、倍浮動値はリストの連続する 2 ワードを占め、長倍値は **-qldbl128/-qldouble** オプションの設定に応じて、2 または 4 ワードを占める。
- 構造体値はストレージのどこかにある連続するワードに現れ、該当する位置合わせ要件をすべて満たす。構造体はフルワードに位置合わせされ、 $(\text{sizeof}(\text{struct } X) + (\text{ワード} \cdot \text{サイズ} - 1) / \text{ワード} \cdot \text{サイズ})$ フルワードを占める (最後に埋め込みを行う)。1 ワードよりも小さい構造体は、ワードまたはレジスター内で左寄せされる。大きな構造体は複数のレジスターを占めることができ、部分的に、ストレージとレジスターに渡すことができる。
- ほかの集約値は *val-by-ref* で渡される。つまり、コンパイラーは実際に集約値のアドレスを渡して、起動されたプログラムでコピーが作成されるように調整する。
- 関数ポインターは、ルーチンの関数記述子を指すポインターとして渡される。先頭のワードには、エントリー・ポイントのアドレスが含まれる。詳しくは、言語間呼び出し - 関数を指すポインター を参照してください。

言語間呼び出し - 値によるパラメーターの受け渡し規則

次の例は、32 ビットのプロトタイプ化された関数への呼び出しの例です。

```
int i, j; //32 bits each
long k; //32 bits
double d1, d2;
float f1;
short int s1;
char c;
...
void f(int, int, int, double, float, char, double, short);
f( i, j, k, d1, f1, c, d2, s1 );
```

関数呼び出しの結果、次のストレージ・マッピングが行われます。

受け渡し先		スタック上の PARM 領域の ストレージ・マッピング	
R3	0	i	
R4	4	j	
R5	8	k	
FP1 (R6, R7 未使用)	12	d1	
	16		
FP2 (R8 未使用)	20	f1	
R9	24	//// //// //// c	← 非プロトタイプ C 関数のために右寄せする
	28		
FP3 (R10 未使用)	32	d2	
	36		
Stack		//// //// //// s1	← 非プロトタイプ C 関数のために右寄せする

注:

1. パラメーターのアドレスがとられる場合に限り、パラメーターのマッピングが保証される。
2. フルワード位置合わせより小さいデータは、高位バイトにコピーされる。例の中の関数はプロトタイプ化されているため、パラメーター **c** と **s1** のマッピングは右寄せされる。
3. パラメーター・リストは、ワードのリストを含むストレージの概念的に連続する部分である。効率性のために、リストの先頭 8 ワードは、これらのワード用に予約されたスペースに実際には保管されないが、GPR3-GPR10 に渡される。さらに、先頭の 13 の浮動小数点値パラメーター値は GPR に渡されないが、FPR1-FPR13 に渡される。どの場合でも、リストの先頭 8 ワードより後のパラメーターは、これらのために予約されたスペースに保管される。
4. 呼び出されたプロシーチャーが、ストレージの連続する部分としてパラメーター・リストを扱う場合 (例えば、パラメーターのアドレスが C で処理された場合)、パラメーター・レジスターは、スタック内のレジスター用に予約されたスペースに保管される。
5. レジスター・イメージはスタック上に保管される。
6. 引き数エリア ($P_1 \dots P_n$) は、最大のパラメーター・リストを保持するのに十分な大きさがなければならない。

言語間呼び出し - 関数を指すポインター

関数ポインターは、値が関数アドレスの範囲にわたるデータ型です。この型の変数は、C や Fortran などのいくつかのプログラム言語にあります。Fortran では、**EXTERNAL** ステートメントに現れるダミー引き数が関数ポインターです。関数ポインターは、呼び出しステートメントのターゲット、またはそのようなステートメントの実引き数のターゲットなどのコンテキストでサポートされます。

関数ポインターは、関数記述子のアドレスであるフルワード数量です。関数記述子は 3 ワード・オブジェクトです。最初のワードには、プロシーチャーのエントリー・ポイントのアドレスが含まれ、2 番目のワードには、そのプロシーチャーがバインドされるモジュールの TOC のアドレスが入り、3 番目のワードには、Pascal

などの言語の環境ポインターが入ります。エントリー・ポイントごとに関数記述子は 1 つしかありません。関数が外部の場合、関数記述子は、この記述子が識別する関数と同じモジュールにバインドされます。記述子には、関数名と同じ外部名がありますが、先行の . (ドット) は付いていません。この記述子名は、すべてのインポート・エクスポート命令で使用されます。

言語間呼び出し - 関数からの戻り値

関数は、型に応じて戻り値を渡します。

- 任意の長さの、ポインター、列挙型、および整数値 (**int**、**short**、**long**、**char**、および無符号型) は、R3 に右寄せされて戻される。**long long** 値は、R3 と R4 に戻される。(64 ビット・モードの R3)
- **float** および **double** は、FP1 に戻される。128 ビットの **long double** は、FP1 と FP2 に戻される。
- 呼び出し関数は、呼び出された関数が戻り値を保管するメモリーの位置を示すポインターを提供する。
- **long double** は、R1 と R2 に戻される。

言語間呼び出し - スタック・フロア

スタック・フロア は、スタックがその下の部分まで増大できないシステム定義のアドレスです。

スタックに関連するほかのシステム・インバリエントは、すべてのコンパイラーとアセンブラーで保守しなければなりません。

- スタック・フロアより下のアドレスにデータを保管したり、これらのアドレスからデータにアクセスしない。
- スタック・ポインターは常に有効である。スタック・フレーム・サイズが 32767 バイトを超えるときには、シグナル・ハンドラーが、スタック・データをオーバーレイするか、あるいはスタック・セグメントをオーバーフローさせて正しくない表示をするタイミング・ウィンドウを発生させないようにするため、このスタック・フレーム・サイズの値を 1 つの命令で必ず変更するように注意してください。

言語間呼び出し - スタック・オーバーフロー

リンケージ規約では、オーバーフローに関する明示的なインライン検査は不要です。オペレーティング・システムではストレージ保護メカニズムを使って、スタック・セグメントの最後より後の保管を検出します。

言語間呼び出し - トレースバック・テーブル

コンパイラーはトレースバック・メカニズムをサポートします。このメカニズムは、呼び出しを解明したり、スタックを戻すために、AIX オペレーティング・システムのシンボリック・デバッガーが必要とするものです。各関数には、そのコードの最後のテキスト・セグメントにトレースバック・テーブルがあります。このテーブルには、スタック・フレームやレジスター情報のほかに、関数の型などの関数に関する情報が入っています。

言語間呼び出し - 型のエンコードと検査

プログラムの実行前にエラーを検出することは、IBM XL C/C++ Enterprise Edition の主要な目的です。実行時エラーの検出は難しく、多くの原因は、サブルーチン・インターフェースの不一致やデータ定義の矛盾です。

XL C/C++ Enterprise Edition は、すべての外部シンボルについての情報（データとプログラム）をエンコードする、初期検知のための方式を使用します。 **-qextchk** オプションが指定された場合、外部シンボルに関するこの情報は、バインドまたはロード時に、整合性があるかどうかを検査されます。

「*AIX 5L for POWER-based Systems: Assembler Language Reference*」に、サブルーチン・リンケージ規約の以下の事項について詳しい説明があります。

- レジスター使用法（汎用、浮動小数点、および 特殊目的のレジスター）
- スタック
- 呼び出しルーチンの責任
- 呼び出されたルーチンの責任

サンプル・プログラム: Fortran を呼び出す C/C++

C または C++ プログラムは Fortran の関数またはサブルーチンを呼び出すことができます。

以下の例では、異なる言語で作成されたプログラム単位を組み合わせ、1 つのプログラムを作成する方法を示します。この例では、引き数として別のデータ型を使用する C/C++ と Fortran サブルーチン間のパラメーターの受け渡しも紹介します。

```
#include <iostream>
extern double add(int *, double [],
int *, double []);

double ar1[4]={1.0, 2.0, 3.0, 4.0};
double ar2[4]={5.0, 6.0, 7.0, 8.0};

main()
{
    int x, y;
    double z;

    x = 3;
    y = 3;

    z = add(&x, ar1, &y, ar2); /* Call Fortran add routine */
    /* Note: Fortran indexes arrays 1..n*/
    /* C indexes arrays 0..(n-1) */

    printf("The sum of %1.0f and %1.0f is %2.0f %n",
ar1[x-1], ar2[y-1], z);
}
```

Fortran のサブルーチンは以下のとおりです。

```
C Fortran function add.f - for C/C++ interlanguage call example
C Compile separately, then link to C/C++ program

REAL FUNCTION ADD*8 (A, B, C, D)
REAL*8 B,D
```

```
INTEGER*4 A,C  
DIMENSION B(4), D(4)  
ADD = B(A) + D(C)  
RETURN  
END
```

関連概念

23 ページの『他のプログラム言語での XL C/C++ Enterprise Edition の使用』

関連タスク

49 ページの『言語間呼び出し規則』

49 ページの『対応するデータ型』

52 ページの『言語間呼び出しにおけるサブルーチン・リンケージ規約の使用』

第 3 部 解説

コンパイラー・オプション

本節では、XL C/C++ Enterprise Edition で使用できるコンパイラー・オプションについて説明します。オプションは、本節の以下のトピックに記載しているように、3つの汎用グループに分類されます。

- 『コンパイラーのコマンド行オプション』
- 349 ページの『汎用プラグマ』
- 422 ページの『並列処理を制御するプラグマ』

コンパイラーのコマンド行オプション

本節では、XL C/C++ Enterprise Edition コマンド行オプションをリストして説明します。

リストした任意のオプションの詳細を参照するには、そのオプションの詳しい説明ページを参照してください。これらのページでは、以下を含むコンパイラー・オプションについてそれぞれ説明します。

- コンパイラー・オプションのコマンド行構文。構文の見出しの下にある最初の行は、コマンド行または構成ファイルでの指定方法を示します。2 行目がある場合は、ソース・ファイルで使用する **#pragma options** キーワードです。
- コマンド行、構成ファイル、またはプログラム内の **#pragma** ディレクティブでオプションを指定しない場合のオプションのデフォルトの設定。
- オプションの目的およびその動きに関する追加情報。特に注釈がない限り、オプションはすべて C と C++ のプログラム・コンパイルの両方に適用されます。

関連概念

6 ページの『コンパイラー・オプション』

関連タスク

33 ページの『コマンド行におけるコンパイラー・オプションの指定』

35 ページの『プログラム・ソース・ファイル内のコンパイラー・オプションの指定』

36 ページの『構成ファイル内のコンパイラー・オプションの指定』

39 ページの『アーキテクチャー固有の (32 ビットまたは 64 ビットの) コンパイル用コンパイラー・オプションの指定』

41 ページの『矛盾するコンパイラー・オプションの解決』

関連資料






349 ページの『汎用プラグマ』






422 ページの『並列処理を制御するプラグマ』

452 ページの『コンパイラー・モードおよびプロセッサのアーキテクチャーの有効な組み合わせ』



コマンド行コンパイラー・オプションの要約

オプション名	型	デフォルト	説明
+	(正符号)	-	 任意のファイル <i>filename.nnn</i> を C++ 言語ファイルとしてコンパイルする。ここで、 <i>nnn</i> は、 .o 、 .a 、および .s 以外の任意のサフィックスです。
#	(ポンド記号)	-	何も行わずにコンパイルをトレースする。
32、64	- <i>qopt</i>	32	32 ビットまたは 64 ビットのコンパイラー・モードを選択する。
aggrcopy	- <i>qopt</i>	aggrcopy を参照。	構造体および共用体の破壊コピー操作を使用可能にする。
alias	- <i>qopt</i>	alias を参照。	最適化中に使用する型ベースの別名割り当てを指定する。
align	- <i>qopt</i>	align=full	コンパイラーがファイルのコンパイルに使用する集合体の位置合わせ方式を指定する。
alloca	- <i>qopt</i>	-	 #pragma alloca ディレクティブがソース・コードであるかのように、関数 alloca の呼び出しにインライン・コードを使用する。
ansialias	- <i>qopt</i>	-qansialias を参照。	型ベースの別名割り当てを最適化中に使用するかどうかを指定する。
arch	- <i>qopt</i>	arch=com	実行可能プログラムを実行するアーキテクチャーを指定する。
asm	- <i>qopt</i>	noasm	コンパイラーにアセンブラー言語拡張を受け入れる (または受け入れない) ように指示する。
asm_as	- <i>qopt</i>	—	アセンブラーの呼び出しに使用するパスとフラグを指定する。
assert	- <i>qopt</i>	noassert	 別名割り当てアサーションをコンパイル単位に適用するようにコンパイラーに指示する。
attr	- <i>qopt</i>	noattr	全 ID の属性リストを含むコンパイラー・リストを生成する。
B	- <i>flag</i>	-	コンパイラー、アセンブラー、リンケージ・エディター、およびプリプロセッサに対する代替パス名を判別する。
b	- <i>flag</i>	bdynamic	リンカーに、その後の共用オブジェクトを動的、共用または静的のいずれかとしてプロセスすることを命令する。
bitfields	- <i>qopt</i>	bitfields=unsigned	ビット・フィールドを符号付きとするかどうかを指定する。
bmaxdata	- <i>flag</i>	0	ヒープのサイズをバイト単位で設定する。
brtl	- <i>flag</i>	-	実行時リンクを使用可能にする。
C	- <i>flag</i>	-	プリプロセスされた出力にコメントを保持する。
c	- <i>flag</i>	-	コンパイラーに、コンパイラーのみにソース・ファイルを渡すように指示する。

オプション名	型	デフォルト	説明
c_stdinc	-qopt	-	 C ヘッダーの標準検索ロケーションを変更する。
cache	-qopt	-	特定の実行マシンのキャッシュ構成を指定する。
chars	-qopt	chars=unsigned	コンパイラーに、 char 型の変数をすべて signed または unsigned のいずれかとして処理するように指示する。
check	-qopt	nocheck	特定のタイプの実行時検査を行うコードを生成する。
cinc	-qopt	nocinc	 コンパイラーに extern "C" { } ラッパーをインクルード・ファイルのコンテンツの周りに配置するように指示する。
compact	-qopt	nocompact	最適化とともに使用すると、可能な場合に、実行速度を犠牲にしてコード・サイズが削減される。
cplusplusmt	-qopt	cplusplusmt を参照。	 C++ のコメントをソース・ファイルで認識させたい場合は、このオプションを使用する。
cplusplus_stdinc	-qopt	-	 C++ ヘッダーの標準検索ロケーションを指定する。
D	-flag	-	#define プリプロセッサ・ディレクティブにあるのと同様に ID <i>name</i> を定義する。
dataimported	-qopt	-	インポートされるものとしてデータにマークを付ける。
datalocal	-qopt	-	ローカルとしてデータにマークを付ける。
mbcs, dbcs	-qopt	nodbcs	プログラムにマルチバイト文字が含まれる場合には、 -qdbcs オプションを使用する。
dbxextra	-qopt	nodbxextra	 すべての typedef 宣言、 struct 型定義、 union 型定義、および enum 型定義をデバッガー処理用に組み込むことを指定する。
digraph	-qopt	digraph を参照。	プログラム・ソースでの連字の文字シーケンスの使用を可能にする。
directstorage	-qopt	nodirectstorage	ライトスルー対応ストレージまたはキャッシュ禁止ストレージが参照可能であることをコンパイラーに通知する。
dollar	-qopt	nodollar	\$ シンボルを ID の名前でできるようにする。
dpcl	-qopt	nodpcl	IBM Dynamic Probe Class Library をサポートするブロック・スコープを生成する。
E	-flag	-	ソース・ファイルをプリプロセスするようにコンパイラーに指示する。
e	-flag	-	共用オブジェクトの項目名を指定する。 Id -e name を使用する場合と同等。 Id オプションの追加情報については、システムの資料を参照してください。

オプション名	型	デフォルト	説明
eh	-qopt	eh	 例外処理を制御する。
enum	-qopt	enum を参照。	列挙の占めるストレージの量を指定する。
expfile	-qopt	-	1 つのファイル内のすべてのエクスポートされたシンボルを保管する。
extchk	-qopt	noextchk	バインド時型検査の情報を生成し、コンパイル時の整合性について検査する。
F	-flag	-	コンパイラーの代替構成ファイルの名前を指定する。
f	-flag	-	ファイルを指定してオブジェクト・ファイルのリストを保管する。
fdpr	-qopt	nofdpr	AIX fdpr パフォーマンス・チューニング・ユーティリティとともに使用するためのプログラム情報を収集する。
flag	-qopt	flag=i:i	報告させる診断メッセージの最低の重大度レベルを指定する。
float	-qopt	float を参照。	浮動小数点演算を高速化するか正確度を向上させるために、各種浮動小数点オプションを指定する。
flttrap	-qopt	noflttrap	追加の命令を生成し、浮動小数点例外を検出してトラップする。
fold	-qopt	fold	浮動小数点定数式をコンパイル時に評価するように指定する。
format	-qopt	noformat	ストリング入出力フォーマット指定で起こりうる問題の警告を出す。
fullpath	-qopt	nofullpath	-g オプションを使用するときに、どのようなパス情報をファイル用に保管するかを指定する。
funcsect	-qopt	nofuncsect	別々のオブジェクト・ファイル、制御セクションまたは csect のそれぞれの関数ごとに命令を配置する。
G	-flag	-	リンケージ・エディター (ld コマンド) オプションのみ。動的ライブラリー・ファイルの生成に使用する。
g	-flag	-	分散デバッガーなどのデバッガーが使用するデバッグ情報を生成する。
genproto	-qopt	nogenproto	 K&R 関数定義から ANSI プロトタイプを生成する。
halt	-qopt	 halt=s  halt=e	指定した重大度 以上のエラーが検出された場合に、コンパイル・フェーズ後に停止するようにコンパイラーに指示する。
haltonmsg	-qopt	-	 特定のエラー・メッセージが検出されたときは、コンパイル・フェーズ後に停止するようにコンパイラーに指示する。
heapdebug	-qopt	noheapdebug	デバッグ・バージョンのメモリー管理関数を使用可能にする。

オプション名	型	デフォルト	説明
hot	-qopt	nohot	最適化中に、高位ループ分析および変換の実行をコンパイラーに指示する。
hsflt	-qopt	nohsflt	単精度 float の結果および浮動小数点から整数への変換に関する範囲検査を除去することによって、計算を高速化する。
hssngl	-qopt	nohssngl	結果を float のメモリー位置に保管する場合にのみ単精度式を丸めることを指定する。
I	-flag	-	絶対パスを指定しない #include ファイル名に追加の検索パスを指定する。
idirfirst	-qopt	noidirfirst	#include “ <i>file_name</i> ” ディレクティブで組み込むファイルの検索順序を指定する。
ignerrno	-qopt	noignerrno	コンパイラーが、システム呼び出しによって errno が変更されないと想定して最適化を行うことを許可する。
ignprag	-qopt	-	特定のプラグマ・ステートメントを無視するようにコンパイラーに指示する。
info	-qopt	<div> <div>▶ C</div> <div>noinfo</div> <div>▶ C++</div> <div>info=lan</div> </div>	通知メッセージを作成する。
initauto	-qopt	noinitauto	自動ストレージを指定された 2 桁の 16 進バイト値に初期化する。
inlglue	-qopt	noinlglue	外部関数の呼び出しまたは関数ポインターを介した呼び出しを行うために必要なポインター・グルー・コードをインライン化することによって、高速な外部結合を生成する。
inline	-qopt	inline を参照。	関数の呼び出しを生成する代わりに、関数のインラインを試みる。
ipa	-qopt	ipa を参照。	プロシージャークラス分析 (IPA) と呼ぶクラスの最適化をオンにしたりカスタマイズしたりする。
isolated_call	-qopt	-	ソース・ファイル内の副次作用がない関数を指定する。
keepinlines	-qopt	nokeepinlines	▶ C++ 未参照の extern インライン関数の定義を保持または廃棄するようコンパイラーに指示する。
keepparm	-qopt	nokeepparm	▶ C++ アプリケーションが最適化される場合でも関数仮パラメーターがスタックに保管されているか確認する。
keyword	-qopt	keyword を参照。	指定されたストリングが、キーワードとして処理されるのか、ID として処理されるのかを制御する。
L	-flag	L を参照。	-I オプションによって指定したライブラリー・ファイルについて、指定したディレクトリーを検索する。
l	-flag	l を参照。	リンクのために指定したライブラリーを検索する。
langlvl	-qopt	langlvl を参照。	コンパイル用の C または C++ の言語レベルを選択する。

オプション名	型	デフォルト	説明
largepage	-qopt	nolargepage	AIX v5.1D 以降を実行している POWER4 および POWER5 システムで使用可能な大規模ページ・ヒープを活用するようにコンパイラーに指示する。
ldbl128、longdouble	-qopt	noldbl128	long double 型のサイズを 64 ビットから 128 ビットに増加させる。
libansi	-qopt	nolibansi	ANSI C ライブラリー関数の名前が付いたすべての関数が実際はシステム関数であると見なす。
linedebug	-qopt	nolinedebug	デバッガーのために、省略された行番号およびソース・ファイル名の情報を生成する。
list	-qopt	nolist	オブジェクト・リストを含むコンパイラー・リストを生成する。
listopt	-qopt	nolistopt	有効なオプションをすべて示すコンパイラー・リストを作成する。
longlit	-qopt	nolonglit	サフィックスをはずしたりリテラルを 64 ビット・モードの long 型にする。
longlong	-qopt	longlong を参照。	プログラムで long long 型を許可する。
M	-flag	-	make コマンドの記述ファイルの組み込みに適したターゲットを含む出力ファイルを作成する。
ma	-flag	-	 #pragma alloca ディレクティブがソース・コードであるかのように、関数 alloca の呼び出しにインライン・コードを使用する。
macpstr	-qopt	nomacpstr	Pascal スtring・リテラルを、先頭バイトに String の長さが含まれるヌル終了 String に変換する。
maf	-qopt	maf	浮動小数点乗算・加算命令を生成するかどうかを指定する。
makedep	-qopt	-	make コマンドの記述ファイルの組み込みに適したターゲットを含む出力ファイルを作成する。
maxerr	-qopt	nomaxerr	指定した重大度以上のエラーの件数が指定した数に達した場合に、コンパイルを停止するようにコンパイラーに指示する。
maxmem	-qopt	maxmem=8192	メモリーを大量に消費する特定の最適化のローカル・テーブルに使用するメモリーの量を制限する。
mbcs、dbs	-qopt	nombcs	プログラムにマルチバイト文字が含まれる場合には、 -qmbcs オプションを使用する。
mkshrobj	-qopt	-	生成されたオブジェクト・ファイルから共用オブジェクトを作成する。
namemangling	-qopt	namemangling=ansi	 C++ ソース・コードから生成した外部シンボル名のネーム・マングリング方式を選択する。

オプション名	型	デフォルト	説明
O, optimize	-qopt , <i>-flag</i>	nooptimize	コンパイル中に、選択したレベルでコードを最適化する。
o	<i>-flag</i>	-	コンパイラによって作成されるオブジェクト・ファイル、アセンブラー・ファイル、または実行可能ファイルの出力位置を指定する。
objmodel	-qopt	objmodel=classic	▶ C++ オブジェクト・モデルの型を設定する。
oldpassbyvalue	-qopt	nooldpassbyvalue	▶ C++ <code>const</code> を含むクラスまたは参照メンバーが関数の引き数に渡される方法を指定する。
P	<i>-flag</i>	-	コンパイラ呼び出し時に名前を指定した C または C++ のソース・ファイルをプリプロセスし、プリプロセスされた出力ソース・ファイルを入力ソース・ファイルごとに作成する。
p	<i>-flag</i>	-	コンパイラが作成するオブジェクト・ファイルをプロファイル用に設定する。
pascal	-qopt	nopascal	▶ C 型指定子および関数宣言内でワード pascal を無視する。
path	-qopt	-	代替プログラム名およびパス名を構成する。
pdf1, pdf2	-qopt	nopdf1, nopdf2	プロファイル指示のフィードバックを介して最適化を調整する。
pg	<i>-flag</i>	-	プロファイル用にオブジェクト・ファイルを設定する。 -p オプションよりも多くの情報が提供されます。
phsinfo	-qopt	nophsinfo	各コンパイル・フェーズでかかった時間を報告する。
pic	-qopt	pic=small	共用ライブラリーでの使用に適した位置独立コードを生成するようにコンパイラに指示する。
prefetch	-qopt	prefetch	コンパイルされたコード内でのプリフェッチ指示の生成を使用可能にする。
print	-qopt	print	-qnoprint はリストを抑制する。
priority	-qopt	-	▶ C++ 静的オブジェクトを初期化する場合の優先順位を指定する。
proclocal, procimported, procunknown	-qopt	proclocal を参照。	関数をローカル、インポートされるもの、または不明としてマークする。
proto	-qopt	noproto	▶ C すべての関数がプロトタイプ宣言されていると見なす。
Q	<i>-flag</i>	Q を参照。	関数のインラインを試行する。
r	<i>-flag</i>	-	再配置可能オブジェクトを作成する。
report	-qopt	noreport	プログラム・ループの並列化と最適化の方法を示す変換レポートを作成するようコンパイラに指示する。
ro	-qopt	ro を参照。	ストリング・リテラルの保管型を指定する。
roconst	-qopt	roconst を参照。	定数値の保管場所を指定する。
roptr	-qopt	noroptr	定数ポインタの保管場所を指定する。

オプション名	型	デフォルト	説明
rrm	-qopt	norm	正および負の無限大への実行時丸めモードと互換性がない浮動小数点の最適化を回避する。
rtti	-qopt	nortti	 typeid 演算子および dynamic_cast 演算子のための実行時識別 (RTTI) 情報を生成する。
S	-flag	-	ソース・ファイルごとにアセンブリー言語ファイル (.s) を生成する。
s	-flag	-	シンボル・テーブルをストリップする。
saveopt	-qopt	nosaveopt	コマンド行コンパイラー・オプションをオブジェクト・ファイル内に保管する。
showinc	-qopt	noshowinc	-qsource と共に使用して、プログラム・ソース・リストにユーザー・ヘッダー・ファイル (" " を使用して組み込む) またはシステム・ヘッダー・ファイル (< > を使用して組み込む) を選択的に表示する。
showpdf	-qopt	noshowpdf	-qpdf1 と共に使用して、追加呼び出しおよびブロック数プロファイル情報を実行可能ファイルに追加する。
smallstack	-qopt	nosmallstack	スタック・フレームのサイズを削減するようコンパイラーに指示する。
smp	-qopt	nosmp	プログラム・コードの並列化を使用可能にする。
source	-qopt	nosource	コンパイラー・リストを作成し、ソース・コードを組み込む。
sourcetype	-qopt	sourcetype=default	実際のソース・ファイル名サフィックスとは関係なく、すべてのソース・ファイルを、このオプションによって指定されたソース・タイプであるかのように処理することをコンパイラーに指示する。
spill	-qopt	spill=512	レジスター割り振り予備域のサイズを指定する。
spnans	-qopt	nospnans	追加の命令を生成して、単精度から倍精度への変換時にシグナル型 NaN を検出する。
srcmsg	-qopt	nosrcmsg	 stderr ファイル内の診断メッセージに、対応するソース・コード行を追加する。
staticinline	-qopt	nostaticinline	インライン関数を静的であるとして扱う。
statsym	-qopt	nostatsym	永続的なストレージ・クラスを持つユーザー定義の非外部名を名前リストに追加する。
stdinc	-qopt	stdinc	#include <file_name> および #include "file_name" ディレクティブで組み込むファイルを指定する。
strict	-qopt	strict を参照。	プログラムのセマンティクスを変更する可能性がある -O3 オプションによる積極的な最適化をオフにする。

オプション名	型	デフォルト	説明
strict_induction	-qopt	strict_induction を参照。	プログラムのセマンティクスを変更する可能性があるループ帰納変数の最適化を使用不可にする。
suppress	-qopt	suppress を参照。	抑制するコンパイラー・メッセージ番号を指定する。
symtab	-qopt	-	シンボル・テーブルに示される情報を決定する。
syntaxonly	-qopt	-	▶ C コンパイラーに、オブジェクト・ファイルを生成せずに構文検査を行わせる。
t	-flag	t を参照。	-B オプションによって指定されたプレフィックスを、指定したプログラムに追加する。
tabsize	-qopt	tabsize=8	コンパイラーによって認識されるタブの長さを変更する。
tbtable	-qopt	tbtable を参照。	トレースバック・テーブルの特性を設定する。
tempinc	-qopt	tempinc を参照。	▶ C++ テンプレート関数およびクラス宣言に対する別個のインクルード・ファイルを生成し、オプションで指定できるディレクトリーにこれらのファイルを配置する。
templaterecompile	-qopt	templaterecompile を参照。	▶ C++ -qtemplateregistry コンパイラー・オプションを使用してコンパイルされたコンパイル単位間の依存性の管理に役立つ。
templateregistry	-qopt	templateregistry	▶ C++ ソース内で検出されたときにすべてのテンプレートのレコードを保守し、各テンプレートのインスタンスが 1 つだけ生成されるようにする。
tempmax	-qopt	tempmax=1	▶ C++ tempinc オプションによって生成させるテンプレート・インクルード・ファイルの最大数をヘッダー・ファイルごとに指定する。
threaded	-qopt	threaded を参照。	プログラムがマルチスレッド環境で稼働することを示す。
tmplparse	-qopt	tmplparse=no	▶ C++ 構文解析およびセマンティック検査をテンプレート定義のインプリメンテーションに適用するかどうかを指定する。
tocdata	-qopt	notocdata。	アプリケーションが使用するスレッド局在のストレージ・モデルを指定する。
tocmerge	-qopt	notocmerge。	TOC マージを使用可能にして TOC ポインターのロードを削減し、外部ロードのスケジューリングを改善する。
trigraph	-qopt	trigraph を参照。	プログラム・ソースでの 3 文字表記の文字シーケンスの使用を可能にする。
tune	-qopt	tune を参照。	実行可能プログラムの最適化対象とするアーキテクチャーを指定する。
twolink	-qopt	notwolink	▶ C++ ライブラリーから組み込まれる静的コンストラクターの数を最小にする。

オプション名	型	デフォルト	説明
U	-flag	-	コンパイラーまたは -D オプションによって定義された ID のうち、指定したものを未定義にする。
unique	-qopt	nounique	C++ 静的コンストラクターやデコンストラクターのファイルのコンパイル単位に固有の名を生成する。
unroll	-qopt	unroll=auto	プログラムの内側のループをアンロールする。
unwind	-qopt	unwind	アプリケーションがいずれのプログラム・スタック・アンwind・メカニズムにも依存しないことをコンパイラーに伝える。
upconv	-qopt	noupconv	C 整数拡張を行うときに unsigned の指定を保持する。
utf	-qopt	noutf	UTF リテラル構文の認識を使用可能にする。
V	-flag	-	コンパイルの進行に関する情報をコマンドに似た形式で報告するようにコンパイラーに指示する。
v	-flag	-	コンパイルの進行に関する情報を報告するようにコンパイラーに指示する。
vftable	-qopt	vftable	C++ 仮想関数テーブルの生成を制御する。
W	-flag	-	リストしたワードを、指定したコンパイラー・プログラムに渡す。
w	-flag	-	警告メッセージの抑止を要求する。
warn64	-qopt	nowarn64	32 ビットおよび 64 ビット・コンパイラー・モード間で起こりうるデータ変換問題の検査を使用可能にする。
weaksymbol	-qopt	noweaksymbol	弱いシンボルを生成するようにコンパイラーに指示する。
xcall	-qopt	noxcall	コンパイル単位内の静的ルーチンを扱うコードを、外部呼び出しであるかのように生成する。
xref	-qopt	noxref	すべての ID の相互参照リストを含むコンパイラー・リストを生成する。
y	-flag	-yn	浮動小数点定数式のコンパイル時丸めモードを指定する。
Z	-flag	-	ライブラリー名の検索パスを指定する。

+ (正符号)

► C++

目的

任意のファイル *filename.nnn* を C++ 言語としてコンパイルする。ここで、*nnn* は、**.a**、**.o**、または **.s** 以外の任意のサフィックスです。

構文

►► -+ ————— ◀◀

注

-+ オプションを使用しない場合は、**.C** (大文字の C)、**.cc**、**.cp**、**.cpp**、**.cxx**、または **.c++** がなければ、C++ ファイルとしてコンパイルされません。**.c** (小文字の c) のサフィックスの付いたファイルを **-+** を指定せずにコンパイルするとファイルは C 言語ファイルとしてコンパイルされます。

-+ オプションは、**-qsourcetype** オプションと共に使用してはいけません。

例

ファイル *myprogram.cplspls* を C++ ソース・ファイルとしてコンパイルするには、以下を入力します。

```
xlc -+ myprogram.cplspls
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

297 ページの『sourcetype』

(ポンド記号)

► C ► C++

目的

何も行わずにコンパイルをトレースする。このオプションは、コマンド行に指定されたコンパイルのステップをプレビューします。このオプションと一緒に **xlc** コマンドを出すと、起動されるプリプロセッサ、コンパイラ、およびリンケージ・エディター内のプログラムの名前と、各プログラムに指定されるオプションが示されます。プリプロセッサ、コンパイラ、およびリンケージ・エディターは起動されません。

構文

►► — -# —————►

注

このコマンドを使用して、特定のコンパイルに関連するコマンドおよびファイルを判別します。これにより、ソース・コードのコンパイル、および既存のファイル (.lst ファイルなど) の上書きのオーバーヘッドが回避されます。情報は、標準出力に出力されます。

このオプションは、**-v** と同じ情報を表示しますが、コンパイラは起動しません。**-#** オプションは、**-v** オプションをオーバーライドします。

例

ソース・ファイル myprogram.c をコンパイルするためのステップをプレビューさせるには、以下を入力します。

```
xlc myprogram.c -#
```

関連資料

61 ページの『コンパイラのコマンド行オプション』

339 ページの『v』

32、64

► C ► C++

目的

32 ビットまたは 64 ビットのコンパイラー・モードのいずれかを選択します。

構文

► `-q`  

注

-q32 および **-q64** オプションは、`OBJECT_MODE` 環境変数が存在する場合にその値によって設定されたコンパイラー・モードをオーバーライドします。このオプションがコマンド行で明示的に指定されておらず、`OBJECT_MODE` 環境変数が設定されていない場合、コンパイラーはデフォルトで 32 ビット出力モードになります。

コンパイラーを 64 ビット・モードで起動した場合は、`__64BIT__` プリプロセッサ・マクロが定義されます。

コンパイラーの出力を使用するアーキテクチャーに合わせてその出力を最適化するには、**-q32** および **-q64** オプションを **-qarch** および **-qtune** コンパイラー・オプションとともに使用します。**-q32**、**-q64**、**-qarch**、および **-qtune** コンパイラー・オプションの有効な組み合わせについては、コンパイラー・モードおよびプロセッサのアーキテクチャーの有効な組み合わせの表を参照してください。64 ビット・モードでは、**-qarch=com** は **-qarch=ppc** と同じように扱われます。

-qarch=ppc を使用したり、**-qfloat=hssngl** または **-qfloat=hsflt** を指定した ppc ファミリー・アーキテクチャーを使用すると、rs64II 以降のシステムでは間違った結果が生成される可能性があります。

例

32 ビットの PowerPC® アーキテクチャーのコンピュータで、`myprogram.c` からコンパイルした実行可能プログラムのテスト実行するように指定するには、次のように入力します。

```
xlc -o testing myprogram.c -q32 -qarch=ppc
```

重要!

- 異なるソース・ファイルに対して、32 ビットおよび 64 ビットのコンパイル・モードを混在させると、XCOFF オブジェクトはバインドされません。完全に再コンパイルして、オブジェクトをすべて同じモードにしなければなりません。
- リンク・オプションは、リンクしているオブジェクトのタイプを反映していなければなりません。64 ビット・オブジェクトをコンパイルした場合は、64 ビット・モードを使用してこれらのオブジェクトをリンクしなければなりません。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

84 ページの『arch』

141 ページの『float』

327 ページの『tune』

343 ページの『warn64』

452 ページの『コンパイラー・モードおよびプロセッサのアーキテクチャーの有効な組み合わせ』

aggrcopy

► C ► C++

目的

構造体および共用体の破壊コピー操作を使用可能にします。

構文

►► `-q-aggrcopy=` nooverlap overlap ►►

デフォルト設定

有効な **-qlanglvl=extended** または **-qlanglvl=classic** でコンパイルする場合、このオプションのデフォルト設定は **-qaggrcopy=overlap** です。そうでない場合は、デフォルトは **-qaggrcopy=nooverlap** です。

ソースと宛先の割り当てがオーバーラップしていないために ANSI C 規格に適合しないプログラムは、**-qaggrcopy=overlap** コンパイラー・オプションでコンパイルする必要があります。

注

-qaggrcopy=nooverlap コンパイラー・オプションが使用可能な場合、コンパイラーは、構造体および共用体のソースおよび宛先の割り当てがオーバーラップしないと見なします。この前提事項によって、コンパイラーはより速いコードを生成します。

例

```
xlc myprogram.c -qaggrcopy=nooverlap
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

198 ページの『langlvl』

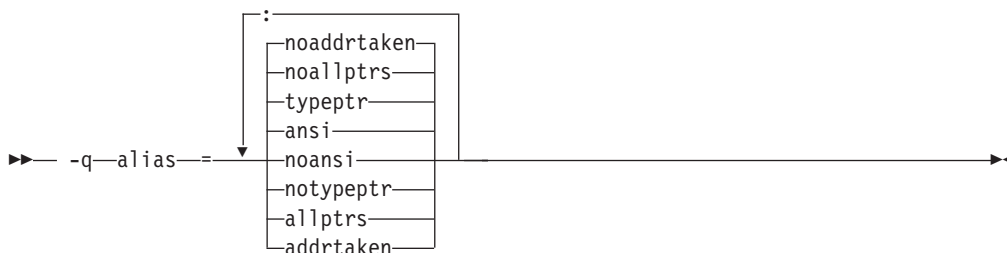
alias

► C ► C++

目的

別名割り当てアサーションをコンパイル単位に適用するようにコンパイラーに指示する。コンパイラーは、他を指定しない限り、別名割り当てアサーションを利用して、可能な位置で最適化の対象を拡大します。

構文



別名割り当ての使用可能なオプションは、以下のとおりです。

[no]typeptr	notypeptr が指定された場合、さまざまな型に対するポインターに別名が割り当てられません。つまり、コンパイル単位では、異なる型の 2 つのポインターが同じ保管場所を指すことはありません。
[no]allptrs	noallptrs が指定された場合、ポインターに別名が割り当てられません (これは -qalias=typeptr も暗黙指定します)。したがって、コンパイル単位では、2 つのポインターが同じ保管場所を指すことはありません。
[no]addrtaken	noaddrtaken が指定された場合、変数は、アドレスが取得されない限り、ポインターから切り離されます。アドレスがコンパイル単位に記録されていない 変数のクラスは、すべてポインターを介した間接アクセスでは結合されないと見なされます。
[no]ansi	ansi が指定された場合、最適化で型ベースの別名割り当てが使用されます。これは、データ・オブジェクトへのアクセスに安全に使用することができる左辺値に制限を加えます。最適化プログラムは、ポインターが同じ型のオブジェクトを指すことしか できないと想定します。これ (ansi) は、 xlc x1C 、および c89 コンパイラー のデフォルトです。このオプションは、 -O オプションも指定しない限り無効です。

noansi を選択すると、最適化プログラムは最悪の事態を考慮して別名割り当てを想定します。これは、型に関係なく、所定の型のポインターが、外部オブジェクトまたはアドレスがすでに取得されている任意のオブジェクトを指すことができると想定します。これが **cc** コンパイラーのデフォルトです。

注

以下は、型ベースの別名割り当ての対象となりません。

- 符号付きまたは符号なしの型。例えば、**signed int** へのポインターは、**unsigned int** を指すことができます。
- 文字ポインター型は、任意の型を指すことができます。

- **volatile** または **const** として修飾された型。例えば、**const int** へのポインターは、**int** を指すことができます。

例

myprogram.c のコンパイル時に最悪の場合を考慮して別名割り当てを想定するように指定するには、以下を入力します。

```
xlc myprogram.c -O -qalias=noansi
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

83 ページの『ansialias』

360 ページの『#pragma disjoint』

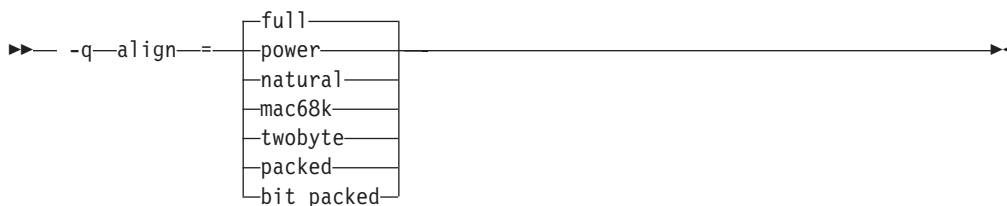
align

➤ C ➤ C++

目的

コンパイラーがファイルのコンパイルに使用する集合体の位置合わせ規則を指定する。このオプションを使用して、ソース・プログラム全体または特定の部分のいずれかについて、クラス型オブジェクトのマップ時に使用する最大の位置合わせを指定します。

構文



使用可能な位置合わせのオプションは、以下のとおりです。

full	コンパイラーは、RS/6000® の位置合わせ規則を使用します。これがデフォルトです。 注: full サブオプションがデフォルトで、既存のオブジェクトとの後方互換性を確実にします。後方互換性が不要でない場合、潜在的なアプリケーション・パフォーマンスを向上させるために、 natural 位置合わせの使用を考慮する必要があります。
power	コンパイラーは、RS/6000 の位置合わせ規則を使用します。 <i>power</i> オプションは、 <i>full</i> と同じです。
mac68k	コンパイラーは、Macintosh** の位置合わせ規則を使用します。このサブオプションは 32 ビット・コンパイルにのみ有効です。
twobyte	コンパイラーは、Macintosh の位置合わせ規則を使用します。 <i>twobyte</i> オプションは、 <i>mac68k</i> と同じです。このサブオプションは 32 ビット・コンパイルにのみ有効です。
packed	コンパイラーは、 パック の位置合わせ規則を使用します。
bit_packed	コンパイラーは、 bit_packed の位置合わせ規則を使用します。 bit_packed に対する位置合わせ規則は、ビット・フィールドのデータがバイト境界とは関係なく、ビット単位でパックされる点を除き、 packed に対する位置合わせ規則と同じです。
natural	コンパイラーは、構造体メンバーを自然な境界にマップします。これには、構造体または共用体の最初のメンバーではない double および long double にも位置合わせ規則を適用することを除いて、 <i>power</i> サブオプションと同じ効力があります。

352 ページの『#pragma align』および 394 ページの『#pragma options』も参照してください。

注

コマンド行で **-qalign** オプションを複数回使用した場合は、最後に指定した位置合わせ規則がファイルに適用されます。

#pragma align=alignment_mode を使用して、コードのサブセットの位置合わせを制御し、**-qalign** コンパイラー・オプションの設定をオーバーライドできます。直前の位置合わせ規則に復帰するには、**#pragma align=reset** を使用します。コンパイラーは、位置合わせディレクティブをスタックします。このため、**#pragma align=reset** ディレクティブを指定することによって、直前の位置合わせディレクティブの内容が不明であっても、その規則に戻して使用することができます。例えば、インクルード・ファイル内にクラス宣言があり、そのクラスに対して指定した位置合わせ規則をクラスの組み込み先に適用したくない場合に、このオプションを使用することができます。

例

例 1 - 集合体定義にのみ影響を与える

コンパイラー呼び出しを使用して、以下を実行します。

```
xlc file2.C /* <-- default alignment rule for file is */
/*          full because no alignment rule specified */
```

ここで、file2.C には以下が含まれます。

```
extern struct A A1;
typedef struct A A2;

#pragma options align=bit_packed /* <-- use bit_packed alignment rules*/
struct A {
    int a;
    char c;
};
#pragma options align=reset /* <-- Go back to default alignment rules */

struct A A1; /* <-- aligned using bit_packed alignment rules since */
A2 A3;      /*      this mode applied when struct A was defined   */
```

例 2 - #pragma の組み込み

コンパイラー呼び出しを使用して、以下を実行します。

```
xlc -qalign=mac68k file.c /* <-- default alignment rule for file is */
/*      Macintosh      */
```

ここで、file.c には以下が含まれます。

```
struct A {
    int a;
    struct B {
        char c;
        double d;
    } BB;
#pragma options align=power /* <-- B will be unaffected by this */
/*      #pragma, unlike previous behavior; */
/*      Macintosh alignment rules still */
/*      in effect */
} AA;
#pragma options align=reset /* <-- A is unaffected by this #pragma; */
/*      Macintosh alignment rules still */
/*      in effect */
```

__align 指定子の使用

__align 指定子を使用すると、データ項目を宣言または定義するときの位置合わせを明示的に指定することができます。

__align 指定子:

目的: **__align** 指定子を使用して、データ項目を宣言または定義する場合の位置合わせおよび埋め込みを明示的に指定する。

構文:

```
declarator __align (int_const) identifier;  
  
__align (int_const) struct_or_union_specifier [identifier] {struct_decln_list}
```

ここで、

int_const は、バイト整合の境界を指定します。 *int_const* は、0 より大きく 2 の累乗と等しい整数定数でなければなりません。

注: **__align** 指定子は、第 1 レベルの変数および集合体定義の宣言でのみ使用することができます。これは、パラメーターおよび自動を無視します。

__align 指定子は、別の集合体定義の範囲内でネストされた集合体定義に使用することができます。

__align 指定子は、以下の状態で使用することはできません。

- 集合体定義内の個々のエレメント。
- 不完全型で宣言された変数。
- 定義なしで宣言された集合体。
- 1 つの配列の個々のエレメント。
- **typedef**、**function**、および **enum** などの、他の型の宣言または定義。
- 変数の位置合わせのサイズが、型の位置合わせのサイズよりも小さい場合。

すべての位置合わせがオブジェクト・ファイル内で表示可能であるとは限りません。

例: 以下は、**__align** を第 1 レベルの変数に適用しています。

```
int __align(1024) varA;          /* varA is aligned on a 1024-byte boundary  
                                and padded with 1020 bytes          */  
  
static int __align(512) varB; /* varB is aligned on a 512-byte boundary  
                                and padded with 508 bytes          */  
  
int __align(128) functionB( ); /* An error                        */  
typedef int __align(128) T;    /* An error                        */  
__align enum C {a, b, c};     /* An error                        */
```

以下は、集合体メンバーに影響せずに **__align** を位置合わせおよび埋め込み集合体の各タグに適用しています。

```
__align(1024) struct structA {int i; int j;}; /* struct structA is aligned  
                                                on a 1024-byte boundary  
                                                with size including padding  
                                                of 1024 bytes          */
```

```
__align(1024) union unionA {int i; int j;}; /* union unionA is aligned
                                             on a 1024-byte boundary
                                             with size including padding
                                             of 1024 bytes */
```

以下は、構造体または共用体を使用している集合体のサイズおよび位置合わせが影響を受けているところで、**__align** を構造体または共用体に適用しています。

```
__align(128) struct S {int i;}; /* sizeof(struct S) == 128 */
struct S sarray[10]; /* sarray is aligned on 128-byte boundary
                     with sizeof(sarray) == 1280 */
struct S __align(64) svar; /* error - alignment of variable is
                           smaller than alignment of type */
struct S2 {struct S s1; int a;} s2; /* s2 is aligned on 128-byte boundary
                                     with sizeof(s2) == 256 */
```

以下は、**__align** を配列に適用しています。

```
AnyType __align(64) arrayA[10]; /* Only arrayA is aligned on a 64-byte
                                boundary, and elements within that array
                                are aligned according to the alignment
                                of AnyType. Padding is applied after the
                                back of the array and does not affect
                                the size of the array member itself. */
```

以下は、変数の位置合わせのサイズが型の位置合わせのサイズと異なるところに**__align** を適用しています。

```
__align(64) struct S {int i;};
struct S __align(32) s1; /* error, alignment of variable is smaller
                        than alignment of type */
struct S __align(128) s2; /* s2 is aligned on 128-byte boundary */
struct S __align(16) s3[10]; /* error */
int __align(1) s4; /* error */
__align(1) struct S {int i;}; /* error */
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

352 ページの『#pragma align』

401 ページの『#pragma pack』

また、「XL C/C++ プログラミング・ガイド」の『集合体内のデータの位置合わせ』も参照してください。

alloca

► C

目的

#pragma alloca ディレクティブがソース・コードにあるかのように、関数 **alloca** の呼び出しにインライン・コードを使用します。

構文

►► -q—alloca—◀◀

注

► C **#pragma alloca** が未指定の場合で、**-ma** を使用しない場合、**alloca** は組み込み関数としてではなく、ユーザー定義 ID として扱われます。

► C++ C++ プログラムでは、**__alloca** 組み込み関数を使用する必要があります。ソース・コードがすでに **alloca** を関数名として参照している場合は、コンパイラの起動時に、コマンド行で次のオプションを使用します。

```
-Dalloca=__alloca
```

alloca の代わりに、C99 可変長配列を使用できます。

例

myprogram.c をコンパイルして、関数 **alloca** の呼び出しをインラインとして扱わせるには、次のように入力します。

```
xlc myprogram.c -qalloca
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

115 ページの『D』

228 ページの『ma』

353 ページの『#pragma alloca』

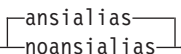
ansialias

► C ► C++

目的

型ベースの別名割り当てを最適化中に使用するかどうかを指定する。型ベースの別名は、データ・オブジェクトへの安全なアクセスに使用することができる左辺値を制限します。

構文

►► -q 

394 ページの『#pragma options』も参照してください。

注

このオプションは廃止されました。新しいアプリケーションでは、**-qalias=** を使用してください。

xlc、**xlC**、**c99**、および **c89** でのデフォルトは **ansialias** です。最適化プログラムは、ポインターが同じ型のオブジェクトを指すことしか できないと想定します。

cc でのデフォルトは **noansialias** です。

このオプションは、**-O** オプションも指定しない限り無効です。

noansialias を選択すると、最適化プログラムは最悪の場合を考慮して別名割り当てを想定します。これは、型に関係なく、所定の型のポインターが、外部オブジェクトまたはアドレスがすでに取得されている任意のオブジェクトを指すことができると想定します。

以下は、型ベースの別名割り当ての対象となりません。

- 符号付きおよび符号なしの型。例えば、**signed int** へのポインターは、**unsigned int** を指すことができます。
- 文字ポインター型は任意の型を指すことができます。
- **volatile** または **const** として限定された型。例えば、**const int** へのポインターは、**int** を指すことができます。

例

myprogram.C のコンパイル時に最悪の場合を考慮して別名割り当てを想定するように指定するには、以下を入力します。

```
xlc myprogram.C -O -qnoansialias
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

76 ページの『alias』

394 ページの『#pragma options』

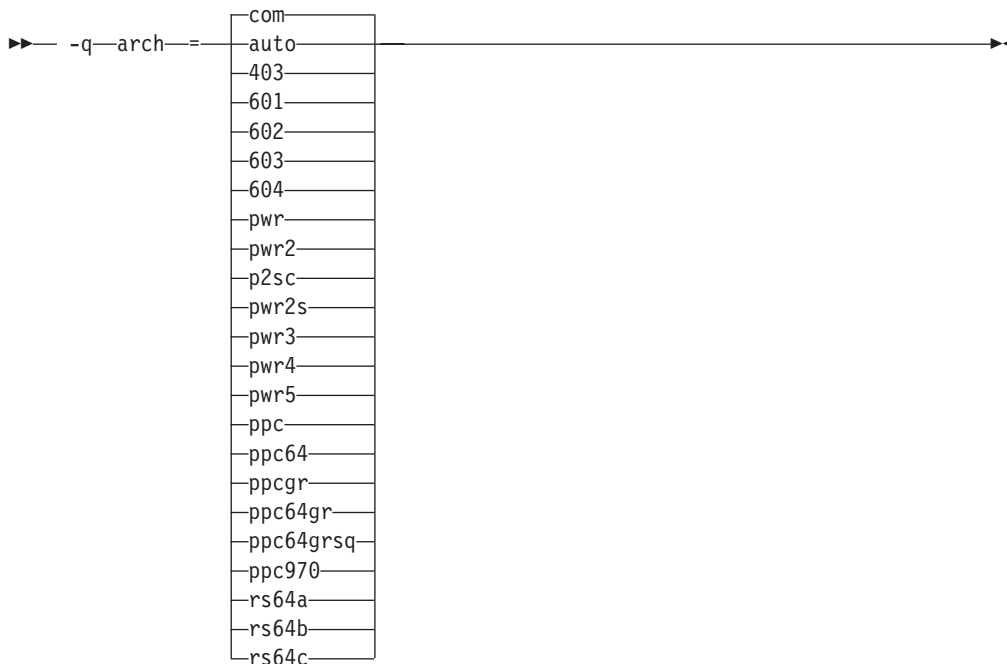
arch

► C ► C++

目的

コード (命令) の生成対象とする一般的なプロセッサのアーキテクチャーを指定する。

構文



使用可能なオプションは、以下で説明するように、プロセッサ・アーキテクチャーの幅広いファミリー、またはそれらのアーキテクチャー・ファミリーのサブグループを指定します。

com

- **-q32** が設定または暗黙指定されている場合、これがデフォルトです。
- 32 ビット実行モードでは、POWER、POWER2*、および PowerPC* ハードウェア・プラットフォームのすべてで実行される命令が含まれるオブジェクト・コードを生成します (つまり、生成される命令はすべてのプラットフォームに共通です)。**-qarch=com** の使用は、共通モードでのコンパイルと呼びます。
- 64 ビット・モードでは、すべての 64 ビット PowerPC ハードウェア・プラットフォームでは稼働し、32 ビットのみのプラットフォームでは稼働しないオブジェクト・コードを生成します。
- `_ARCH_COM` マクロを定義します。
- これは、**-O4** または **-O5** コンパイラー・オプションが指定されていない場合のデフォルト・オプションです。

auto

- **-O4** または **-O5** が設定または暗黙指定されている場合、これは暗黙指定されます。
- コンパイルが行われるハードウェア・プラットフォームで実行される命令を含んだオブジェクト・コードを生成します。

- 403
 - PowerPC 403 ハードウェア・プラットフォームで実行される命令を含んだオブジェクト・コードを生成します。
 - `_ARCH_COM`、`_ARCH_PPC`、および `_ARCH_403` マクロを定義します。
- 601
 - PowerPC 601 ハードウェア・プラットフォームで実行される命令を含んだオブジェクト・コードを生成します。
 - `_ARCH_COM`、`_ARCH_PWR`、`_ARCH_PPC`、および `_ARCH_601` マクロを定義します。
 - **-q64** が設定または暗黙指定されている場合、このオプションは無効です。
- 602
 - PowerPC 602 ハードウェア・プラットフォームで実行される命令を含んだオブジェクト・コードを生成します。
 - `_ARCH_COM`、`_ARCH_PPC`、および `_ARCH_602` マクロを定義します。
 - **-q64** が設定または暗黙指定されている場合、このオプションは無効です。
- 603
 - PowerPC 603 ハードウェア・プラットフォームで実行される命令を含んだオブジェクト・コードを生成します。
 - `_ARCH_COM`、`_ARCH_PPC`、`_ARCH_PPCGR`、および `_ARCH_603` マクロを定義します。
 - **-q64** が設定または暗黙指定されている場合、このオプションは無効です。
- 604
 - PowerPC 604 ハードウェア・プラットフォームで実行される命令を含んだオブジェクト・コードを生成します。
 - `_ARCH_COM`、`_ARCH_PPC`、`_ARCH_PPCGR`、および `_ARCH_604` マクロを定義します。
 - **-q64** が設定または暗黙指定されている場合、このオプションは無効です。
- pwr
 - POWER、POWER2、および PowerPC 601 ハードウェア・プラットフォームのいずれでも実行される命令を含んだオブジェクト・コードを生成します。
 - `_ARCH_COM` および `_ARCH_PWR` マクロを定義します。
 - **-q64** が設定または暗黙指定されている場合、このオプションは無効です。
- pwr2
 - POWER2 プラットフォームで実行される命令を含んだオブジェクト・コードを生成します。
 - `_ARCH_COM`、`_ARCH_PWR`、および `_ARCH_PWR2` マクロを定義します。
 - **-q64** が設定または暗黙指定されている場合、このオプションは無効です。
- pwr2s
 - POWER2 Chip デスクトップ・インプリメンテーションで実行される命令を含んだオブジェクト・コードを生成します。
 - `_ARCH_COM`、`_ARCH_PWR`、`_ARCH_PWR2`、および `_ARCH_PWR2S` マクロを定義します。
 - **-q64** が設定または暗黙指定されている場合、このオプションは無効です。

p2sc	<ul style="list-style-type: none"> POWER2 Super Chip ハードウェア・プラットフォームで実行される命令を含んだオブジェクト・コードを生成します。 _ARCH_COM、_ARCH_PWR、_ARCH_PWR2、および _ARCH_P2SC マクロを定義します。 -q64 が設定または暗黙指定されている場合、このオプションは無効です。
pwr3	<ul style="list-style-type: none"> POWER3™、POWER4™、POWER5™ または、PowerPC 970 ハードウェア・プラットフォームのいずれでも実行される命令を含んだオブジェクト・コードを生成します。 _ARCH_COM、_ARCH_PPC、_ARCH_PPCGR、_ARCH_PPC64、_ARCH_PPC64GR、_ARCH_PPC64GRSQ、および _ARCH_PWR3 マクロを定義します。
pwr4	<ul style="list-style-type: none"> POWER4、POWER5、または PowerPC 970 ハードウェア・プラットフォームで実行される命令を含んだオブジェクト・コードを生成します。 _ARCH_COM、_ARCH_PPC、_ARCH_PPCGR、_ARCH_PPC64、_ARCH_PPC64GR、_ARCH_PPC64GRSQ、_ARCH_PWR3、および _ARCH_PWR4 マクロを定義します。
pwr5	<ul style="list-style-type: none"> POWER5 ハードウェア・プラットフォームで実行される命令を含んだオブジェクト・コードを生成します。 _ARCH_COM、_ARCH_PPC、_ARCH_PPCGR、_ARCH_PPC64、_ARCH_PPC64GR、_ARCH_PPC64GRSQ、_ARCH_PWR3、_ARCH_PWR4、および _ARCH_PWR5 マクロを定義します。
ppc	<ul style="list-style-type: none"> 32 ビット・モードでは、32 ビット PowerPC ハードウェア・プラットフォームのすべてで実行される命令を含んだオブジェクト・コードを成します。このサブオプションによって、コンパイラーは、単精度データとともに使用する単精度命令を生成します。 _ARCH_COM および _ARCH_PPC マクロを定義します。 -qarch=ppc と -q64 を共に指定すると、-qarch=ppc64 を暗黙指定します。
ppc64	<ul style="list-style-type: none"> あらゆる 64 ビット PowerPC ハードウェア・プラットフォームで稼動するオブジェクト・コードを生成します。 このサブオプションは 32 ビット・モードでコンパイルするときに選択できますが、生成されるオブジェクト・コードは 32 ビット PowerPC プラットフォームでは認識されないか、異なる動きをする命令をインクルードしていることがあります。 _ARCH_COM、_ARCH_PPC、および _ARCH_PPC64 マクロを定義します。
ppcgr	<ul style="list-style-type: none"> 32 ビット・モードでは、オプションのグラフィックス命令をサポートする PowerPC プロセッサ用のオブジェクト・コードを生成します。 -q64 コンパイラー・オプションも有効である場合、このオプションは、以下で説明するように ppc64gr として解釈されます。 _ARCH_COM、_ARCH_PPC、および _ARCH_PPCGR マクロを定義します。
ppc64gr	<ul style="list-style-type: none"> オプションのグラフィックス命令をサポートするすべての 64 ビット PowerPC ハードウェア・プラットフォーム用のコードを生成します。 _ARCH_COM、_ARCH_PPC、_ARCH_PPCGR、_ARCH_PPC64、および _ARCH_PPC64GR マクロを定義します。

- ppc64grsq
 - オプションのグラフィックスおよび平方根命令をサポートするすべての 64 ビット PowerPC ハードウェア・プラットフォーム用のコードを生成します。
 - `_ARCH_COM`、`_ARCH_PPC`、`_ARCH_PPCGR`、`_ARCH_PPC64`、`_ARCH_PPC64GR`、および `_ARCH_PPC64GRSQ` マクロを定義します。
- ppc970
 - PowerPC 970 プラットフォーム専用の命令を生成します。
 - `_ARCH_COM`、`_ARCH_PPC`、`_ARCH_PPCGR`、`_ARCH_PPC64`、`_ARCH_PPC970`、`_ARCH_PWR3`、`_ARCH_PWR4`、`_ARCH_PPC64GR`、および `_ARCH_PPC64GRSQ` マクロを定義します。
- rs64a
 - RS64I プラットフォームで実行されるオブジェクト・コードを生成します。
 - `_ARCH_COM`、`_ARCH_PPC`、`_ARCH_PPC64`、および `_ARCH_RS64A` マクロを定義します。
- rs64b
 - RS64II プラットフォームで実行されるオブジェクト・コードを生成します。
 - `_ARCH_COM`、`_ARCH_PPC`、`_ARCH_PPCGR`、`_ARCH_PPC64`、`_ARCH_PPC64GR`、`_ARCH_PPC64GRSQ`、および `_ARCH_RS64B` マクロを定義します。
- rs64c
 - RS64III プラットフォームで実行されるオブジェクト・コードを生成します。
 - `_ARCH_COM`、`_ARCH_PPC`、`_ARCH_PPCGR`、`_ARCH_PPC64`、`_ARCH_PPC64GR`、`_ARCH_PPC64GRSQ`、および `_ARCH_RS64C` マクロを定義します。

注

特定のアーキテクチャーで最高のパフォーマンスになるようにしたい場合に、他のアーキテクチャーでプログラムを使用しないのであれば、適切なアーキテクチャー・オプションを使用してください。

-qarch=ppc を使用したり、**-qfloat=hssngl** または **-qfloat=hsflt** を指定した ppc ファミリー・アーキテクチャーを使用すると、RS64II 以降のシステムでは間違った結果が生成される可能性があります。

-qarch=suboption は、**-qtune=suboption** とともに使用することができます。

-qarch=suboption は、命令の生成対象とするアーキテクチャーを指定し、

-qtune=suboption は、コードの最適化対象とするターゲット・プラットフォームを指定します。

例

32 ビットの PowerPC アーキテクチャーのコンピュータで、myprogram.c からコンパイルされた実行可能プログラム testing を実行するように指定するには、以下を入力します。

```
xlc -o testing myprogram.c -q32 -qarch=ppc
```

関連タスク

39 ページの『アーキテクチャー固有の (32 ビットまたは 64 ビットの) コンパイル用コンパイラー・オプションの指定』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

141 ページの『float』

244 ページの『O, optimize』

327 ページの『tune』

452 ページの『コンパイラー・モードおよびプロセッサのアーキテクチャーの有効な組み合わせ』

asm

► C ► C++

目的

コンパイラーにアセンブラー言語拡張を受け入れるように指示します。

構文

► — -q — noasm — asm — gcc — ◀◀

注

このフィーチャーにより、インライン・アセンブラー・ステートメントをソース・コードに挿入できます。

-qasm=gcc コンパイラー・オプションは、**asm** キーワードを拡張 gcc 構文およびセマンティクスと共に認識するように、コンパイラーに指示します。デフォルトの設定値 **-qnoasm** では、これはサポートされていません。

システム・アセンブラー・プログラムはこのコマンドに使用可能で、有効でなければなりません。詳しくは、**-qasm_as** コンパイラー・オプションを参照してください。

例

以下のコードの断片は、**-qasm** コンパイラー・オプションの簡単な使用法を示しています。

```
int a, b, c;
int main() {
    asm("add %0, %1, %2" : "=r"(a) : "r"(b), "r"(c) );
}
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

90 ページの『asm_as』

asm_as



目的

asm ステートメントでアセンブラー・コードを処理するために、アセンブラーの呼び出しに使用するパスとフラグを指定します。

構文

►► `-qasm_as=invocation_string` ◀◀

invocation_string は、**asm** ステートメントにアセンブラーの呼び出しに使用するパスとフラグです。

注

このオプションを使用して代替アセンブラー・プログラム、およびそのアセンブラーの呼び出しに必要なフラグを指定します。

このオプションは、コンパイラー構成ファイルで定義される **as** コマンドのデフォルト設定をオーバーライドします。

例

`myprog.c` でインライン・アセンブラー・コードを検出した場合、`/bin/as` でアセンブラー・プログラムを使用するようにコンパイラーに指示するには、コマンド行で次のように指定します。

```
xlc myprog.c -qasm_as=/bin/as
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

89 ページの『asm』

assert



目的

別名割り当てアサーションをコンパイル単位に適用するようにコンパイラーに要求する。コンパイラーは、別名割り当てアサーションを利用して、可能な場合に最適化の機会を拡大する。

構文



別名割り当ての使用可能なオプションは、以下のとおりです。

noassert	別名割り当てアサーションは適用されません。
ASsert=TYPePtr	異なる型へのポインターには別名を割り当てません。つまり、コンパイル単位では、異なる型の 2 つのポインターが同じ保管場所を指すことはありません。
ASsert=ALLPtrs	ポインターには別名を割り当てません (これは、 -qassert=typeptr を暗黙指定します)。したがって、コンパイル単位では、2 つのポインターが同じ保管場所を指すことはありません。
ASsert=ADDRtaken	変数は、アドレスが取得されない限り、ポインターに結合されません。アドレスがコンパイル単位に記録されていない 変数のクラスは、すべてポインターを介した間接アクセスでは結合されないと見なされます。

394 ページの『#pragma options』も参照してください。

注

このオプションは廃止されました。新しいアプリケーションでは、**-qalias=** を使用してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

76 ページの『alias』

attr

➤ C ➤ C++

目的

全 ID の属性リストを含むコンパイラー・リストを生成する。

構文



ここで、

-qnoattr	プログラムにある ID の属性リストを生成しません。
-qattr=full	プログラムにある ID をすべて報告します。
-qattr	使用されている ID のみを報告します。

394 ページの『#pragma options』も参照してください。

注

このオプションは、**-qxref** も指定しない限り、相互参照リストを生成しません。

-qnoprint オプションは、このオプションをオーバーライドします。

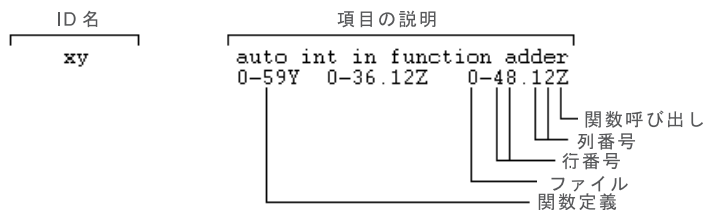
-qattr を **-qattr=full** の後に指定すると、無効になってしまいます。完全なリストが生成されます。

例

プログラム myprogram.C をコンパイルして、全 ID のコンパイラー・リストを生成するには、以下を入力します。

```
x1C myprogram.C -qxref -qattr=full
```

一般的な相互参照リストの形式は、以下のとおりです。



関連資料

61 ページの『コンパイラーのコマンド行オプション』

265 ページの『print』

346 ページの『xref』

394 ページの『#pragma options』

B

► C ► C++

目的

コンパイラー、アセンブラー、リンケージ・エディター、およびプリプロセッサーなどのプログラムの代替パス名を決定する。

構文

```
►► -B [prefix] [-t program]
```

program は、**-t** コンパイラー・オプションが認識する任意のプログラム名となります。プログラムの指定について詳しくは、**t** の資料を参照してください。

注

prefix オプションは、新しいプログラムへのパス名の一部を定義します。コンパイラーは、プレフィックスとプログラム名の間に **/** を追加しません。

プログラムごとに完全パス名を形成するために、IBM XL C/C++ Enterprise Edition は、コンパイラー、アセンブラー、リンケージ・エディター、およびプリプロセッサー用の標準のプログラム名にプレフィックスを追加します。

IBM XL C/C++ Enterprise Edition の実行可能ファイルのいくつかまたはすべてについて複数のレベルを保持し、使用するプログラムを指定できるようにしたい場合には、このオプションを使用します。

-Bprefix が指定されていない場合は、デフォルトのパスが使用されます。

-B -tprograms は、**-B** プレフィックス名を追加するプログラムを指定します。

-Bprefix -tprograms オプションは、**-Fconfig_file** オプションをオーバーライドします。

例

/lib/tmp/mine/ にある代替の **x1C** コンパイラーを使用して **myprogram.C** をコンパイルするには、以下を入力します。

```
x1C myprogram.C -B/lib/tmp/mine/ -tc
```

/lib/tmp/mine/ にある代替のリンケージ・エディターを使用して **myprogram.C** をコンパイルするには、次を入力します。

```
x1C myprogram.C -B/lib/tmp/mine/ -tl
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

256 ページの『path』

311 ページの『t』

b

➤ C ➤ C++

目的

共用オブジェクトをリンケージ・エディターで処理する方法を制御します。

構文

➤ — -b dynamic
shared
static ————— ➤

ここで、オプションは、以下のとおりです。

dynamic、shared	リンカーに、その後の共用オブジェクトを動的モードで処理させます。これはデフォルトです。動的モードでは、共用オブジェクトは、出力ファイルに静的には組み込まれません。代わりに、共用オブジェクトは、出力ファイルのローダー・セクションにリストされます。
static	リンカーに、その後の共用オブジェクトを静的モードで処理させます。静的モードでは、共用オブジェクトは、出力ファイルに静的にリンクされます。

注

デフォルト・オプション **-bdynamic** では、C ライブラリー (**lib.c**) は必ず動的にリンクされます。C ライブラリーにリンクするときに未解決のリンカー・エラーに関して起こる可能性のある問題を回避するには、**-bstatic** オプションを使用するコンパイル・セクションの最後に **-bdynamic** オプションを追加しなければなりません。

これとほかの **ld** オプションの詳細については、「AIX コマンド・リファレンス」を参照してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

97 ページの『brtl』

以下も参照してください。

「コマンド・リファレンス 第 5 巻」の **ld** コマンド

bitfields

► C ► C++

目的

ビット・フィールドを符号付きとするかどうかを指定します。デフォルトでは、ビット・フィールドは符号なしです。

構文

►► — -q—bitfields—=— —————►◄
 └──unsigned──┐
 signed └──

ここで、オプションは、以下のとおりです。

signed	ビット・フィールドは符号付きです。
unsigned	ビット・フィールドは符号なしです。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

bmaxdata

► C ► C++

目的

このオプションは、(初期化済みおよび未初期化の両方の) 静的データによって共用されるエリアの最大サイズを設定し、ヒープを *number* バイトに設定します。この値は、システム・ローダーが `soft ulimit` を設定するのに使用します。

デフォルト設定は **-bmaxdata=0** です。

構文

►► -bmaxdata=0
number►►

注

number の有効値は、0 および 0x10000000 の倍数 (0x10000000、0x20000000、0x30000000, ...) です。システムが許可する最大値は、0x80000000 です。

size の値が 0 の場合、単一の 256MB (0x10000000 バイト) のデータ・セグメント (セグメント 2) が、静的データ、ヒープ、およびスタックによって共用されます。値がゼロ以外の場合は、別の 256 MB データ・セグメント (セグメント 2) がスタックによって使用される一方で、指定されたサイズのデータ域 (セグメント 3 から始まる) が静的データおよびヒープによって共用されます。したがって、0 が指定されたときの合計データ・サイズは 256 MB で、0x10000000 が指定されたときの合計サイズは 512MB です。このうち、256MB はスタック用で、残りの 256MB は静的データおよびヒープ用です。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

brtl

➤ C ➤ C++

目的

出力ファイルに対する実行時リンクを可能にします。

構文

➤— -brtl—➤

注

DCE スレッド・ライブラリーおよびヒープ・デバッグ・ライブラリーには、実行時リンクとの互換性はありません。コンパイラーを **xlc_r4** または **xlc_r4** で起動する場合、または **-qheapdebug** コンパイラー・オプションが指定される場合は、**-brtl** コンパイラー・オプションを指定してはいけません。

実行時リンクとは、プログラムの実行がすでに始まった後で、共用モジュール内の未定義シンボルおよび非据え置きシンボルを解決する機能です。これは、実行時定義（これらの機能定義は、リンク時には使用できません）およびシンボルの再バインド機能を提供するためのメカニズムです。実行時リンクを使用可能にするには、メイン・アプリケーションを作成しなければなりません。実行時リンカーを使って、どのモジュールでも簡単にリンクできるというわけではありません。

プログラムに実行時リンクを組み込むには、**-brtl** コンパイラー・オプションを使用してコンパイルします。これにより、プログラムに実行時リンカーへの参照が追加されます。この実行時リンカーは、プログラム実行の開始時にプログラムの始動コード (`/lib/crt0.o`) によって呼び出されます。共用オブジェクト入力ファイルは、プログラム・ローダー・セクション内の従属として、コマンド行に指定されたときと同じ順序でリストされます。プログラム実行が開始されると、システム・ローダーはこれらの共用オブジェクトをロードし、その定義を実行時リンカーで 사용할できるようにします。

システム・ローダーは、メイン・プログラムおよび呼び出されたモジュールで参照されるすべてのシンボルをロードし、解決できなければなりません。それができない場合、プログラムは実行されません。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

94 ページの『b』

154 ページの『G』

「General Programming Concepts: Writing and Debugging Programs」の『Shared Objects and Run-time Linking』の章も参照してください。

C

► C ► C++

目的

プリプロセスされた出力にコメントを保持する。

構文

►► -C ————— ◀◀

注

-C オプションは、**-E** または **-P** オプションを指定しないと無効になります。 **-E** オプションでは、コメントが標準出力に書き込まれます。 **-P** オプションでは、コメントが出力ファイルに書き込まれます。

例

myprogram.c をコンパイルして、コメントを含むプリプロセスされたプログラム・テキストを含むファイルを生成させるには、以下を入力します。

```
xlc myprogram.c -P -C
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

126 ページの『E』

252 ページの『P』

C

► C ► C++

目的

コンパイラーに、コンパイラーのみにソース・ファイルを渡すように指示する。

構文

►► -c ◀◀

注

コンパイル済みのソース・ファイルは、リンケージ・エディターに送信されません。コンパイラーは、*file_name.c*、*file_name.i*、*file_name.C*、*file_name.cpp* などの有効なソース・ファイルごとに、出力オブジェクト・ファイル *file_name.o* を作成します。

-c オプションは、**-E**、**-P**、または **-qsyntaxonly** オプションのいずれかが指定されている場合にオーバーライドされます。

-c オプションを、**-o** オプションと組み合わせて使用すると、コンパイラーによって作成されたオブジェクト・ファイルの明示的な名前を提供することができます。

例

myprogram.C をコンパイルして、実行可能ファイルではなくオブジェクト・ファイル **myprogram.o** を生成させるには、次のコマンドを入力します。

```
xlc myprogram.C -c
```

myprogram.C をコンパイルして、実行可能ファイルではなくオブジェクト・ファイル **new.o** を生成させるには、次のコマンドを入力します。

```
xlc myprogram.C -c -o new.o
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

126 ページの『E』

248 ページの『o』

252 ページの『P』

310 ページの『syntaxonly』

c_stdinc



目的

C ヘッダーの標準検索ロケーションを変更する。

構文

→ `-q-c_stdinc=`  `path` →

注

C ヘッダーの標準検索パスは、**-qc_stdinc** コンパイラー・オプションによって決定されます。このコンパイラー・オプションが指定されず、または空のストリングを指定する場合、デフォルトのヘッダー・ファイル検索パスが使用されます。このオプションのデフォルトの検索パスは、コンパイラーのデフォルト構成ファイルにあります。

このオプションが複数回指定されている場合、コンパイラーは最後に指定されたオプションのみを使用します。複数のディレクトリーを検索パスに指定するには、このオプションを一度指定し、: (コロン) で複数の検索ディレクトリーを区切ってください。

このオプションは、**-qnostdinc** オプションが有効である場合には無視されます。

例

mypath/headers1 および **mypath/headers2** を標準検索パスの一部として指定するには、以下を入力します。

```
xlc myprogram.c -qc_stdinc=mypath/headers1:mypath/headers2
```

関連タスク

43 ページの『相対パス名を使用したインクルード・ファイルのディレクトリー検索順序』

36 ページの『構成ファイル内のコンパイラー・オプションの指定』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

114 ページの『cpp_stdinc』

304 ページの『stdinc』

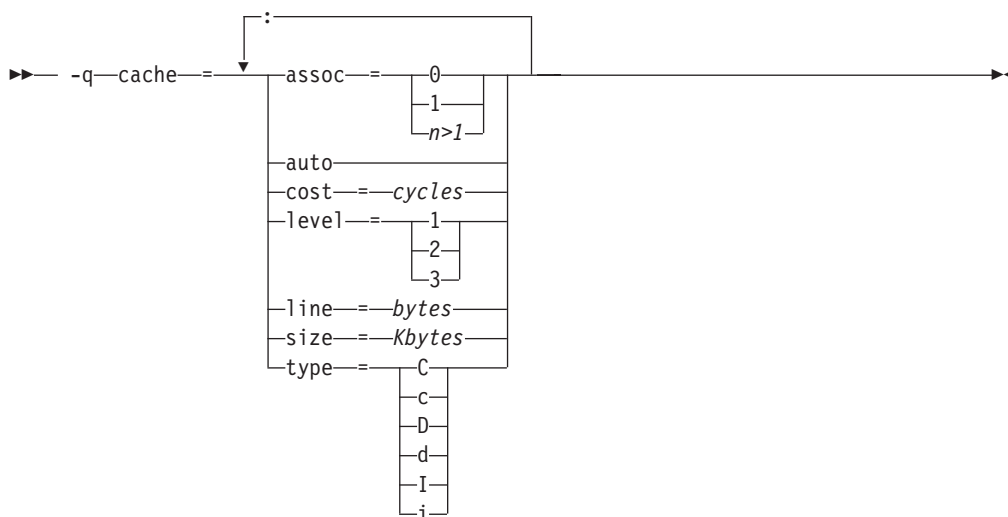
cache

► C ► C++

目的

-qcache オプションは、特定の実行マシンのキャッシュ構成を指定する。プログラムの実行システムの型がわかっていて、システムにデフォルトのケースとは異なる構成の命令またはデータ・キャッシュがある場合は、このオプションを使用して、正確なキャッシュ特性を指定します。コンパイラーは、この情報を使用して、キャッシュに関連する最適化の利点を計算します。

構文



ここで、使用可能なキャッシュ・オプションは、以下のとおりです。

<code>assoc=number</code>	キャッシュの設定の結合順序を指定します。この場合、 <i>number</i> は、以下のいずれかです。 0 直接マップされたキャッシュ 1 完全結合のキャッシュ N>1 <i>n</i> 通りの集合の結合キャッシュ
<code>auto</code>	コンパイル・マシンの特定のキャッシュ構成を自動的に検出します。これは、実行環境がコンパイル環境と同じであることを前提としています。
<code>cost=cycles</code>	キャッシュ・ミスの結果生ずるパフォーマンス・ペナルティを指定します。
<code>level=level</code>	影響されるキャッシュのレベルを指定します。この場合、 <i>level</i> は、以下のいずれかです。 1 基本キャッシュ 2 レベル 2 のキャッシュ、または、レベル 2 のキャッシュがない場合は、テーブル・ルックアサイド・バッファ (TLB) 3 TLB マシンに複数のレベルのキャッシュがある場合は、別々の <code>-qcache</code> オプションを使用します。
<code>line=bytes</code>	キャッシュの行サイズを指定します。

`size=Kbytes` キャッシュの合計サイズを指定します。
`type=cache_type` 指定した型のキャッシュに、設定を適用します。この場合、`cache_type` は、以下のいずれかです。

C または c

データおよび命令キャッシュの結合

D または d

データ・キャッシュ

I または i

命令キャッシュ

注

-qtune 設定は、最も一般的なコンパイル用の、最適なデフォルト **-qcache** 設定を判別します。これらのデフォルト設定は、**-qcache** を使用してオーバーライドすることができます。しかし、間違った値をキャッシュ構成に指定したり、異なる構成のマシン上でプログラムを実行した場合、そのプログラムは正常に稼働しますが、若干遅くなる可能性があります。

-qcache オプションで **-O4**、**-O5**、または **-qipa** を指定しなければなりません。

-qcache サブオプションを指定するときには、以下のガイドラインを使用してください。

- 可能な限り多くの構成パラメーターに対して情報を指定します。
- ターゲットの実行システムに複数のレベルのキャッシュがある場合は、別々の **-qcache** オプションを使用して各キャッシュ・レベルを記述します。
- ターゲットの実行マシン上のキャッシュの正確なサイズがわからない場合は、見積もった小さい方のキャッシュ・サイズを指定します。実際にあるキャッシュ・サイズより大きなサイズを指定して、キャッシュ・ミスやページ不在を経験するよりも、キャッシュ・メモリーをいくらか未使用で残した方がよいでしょう。
- データ・キャッシュは、命令キャッシュよりもプログラム・パフォーマンスにおいて、より大きな影響を及ぼします。異なるキャッシュ構成を試してみる時間的余裕があまりない場合は、まず、そのデータ・キャッシュに対して最適な構成指定を判別します。
- 間違った値をキャッシュ構成に指定したり、または異なる構成のマシン上でプログラムを実行した場合、プログラムのパフォーマンスが低下する場合がありますが、プログラム出力は期待どおりとなります。
- **-O4** および **-O5** 最適化オプションは、コンパイル・マシンのキャッシュ特性を自動的に選択します。**-qcache** オプションを **-O4** または **-O5** オプションとともに指定する場合は、最後に指定されたオプションが優先されます。

例

結合された命令およびデータ・レベル 1 のキャッシュ (2 つの方法で結合されており、サイズが 8 KB で、64 バイトのキャッシュ行を持っているキャッシュ) でシステムのパフォーマンスを調整するには、以下を入力します。

```
xlc -O4 -qcache=type=c:level=1:size=8:line=64:assoc=2 file.C
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

181 ページの『ipa』

244 ページの『O、optimize』

chars

➤ C ➤ C++

目的

コンパイラーに、**char** 型の変数をすべて **signed** または **unsigned** のいずれかとして処理するように指示する。

構文

➤ — -qchars=signed/unsigned ➤

356 ページの『`#pragma chars`』および 394 ページの『`#pragma options`』も参照してください。

注

以下のプリプロセッサ・ディレクティブのいずれかを使用して、ソース・プログラムにおける符号のタイプを指定することもできます。

```
#pragma options chars=sign_type
```

```
#pragma chars (sign_type)
```

ここで、*sign_type* は、**signed** または **unsigned** のいずれかです。

このオプションの設定に関係なく、**char** 型は、型の互換性検査または C++ 多重定義のため、**unsigned char** および **signed char** とは異なると見なされます。

例

myprogram.c をコンパイルするときに、すべての **char** 型を **signed** として扱うには、次のように入力します。

```
xlc myprogram.c -qchars=signed
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

356 ページの『`#pragma chars`』

394 ページの『`#pragma options`』

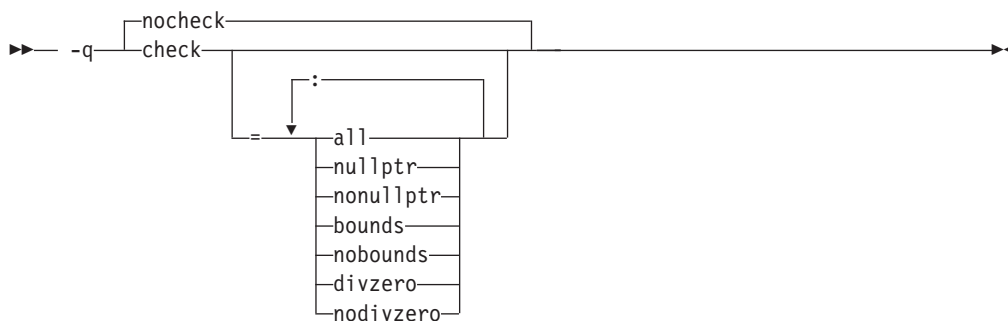
check

► C ► C++

目的

特定のタイプの実行時検査を行うコードを生成する。違反が検出された場合は、**SIGTRAP** シグナルをプロセスに送信することによって、実行時例外が発生します。

構文



ここで、

all

以下のサブオプションをすべてオンに切り替えます。**no...** 形式の 1 つまたは複数の他のオプションとともに、**all** オプションをフィルターとして使用することができます。

例えば、以下を使用すると、

```
xlc myprogram.C -qcheck=all:nonnullptr
```

ストレージの参照に使用するポインター変数に含まれるアドレスを除いて、すべての検査が行われます。

no... 形式のオプションとともに **all** を使用する場合は、**all** は最初のサブオプションでなければなりません。

nullptr | nonnullptr

ストレージの参照に使用するポインター変数に含まれるアドレスの実行時検査を行います。アドレスは、使用する位置で検査されます。値が 512 より小さい場合は、トラップが起こります。

bounds | nobounds

サイズが既知のオブジェクト内の添え字を指定するときに、アドレスの実行時検査を行います。指標が検査され、結果がオブジェクトのストレージの境界内のアドレスになることが確認されます。アドレスがオブジェクトの境界内でない場合は、トラップが起こります。

このサブオプションは、可変長配列へのアクセスに有効ではありません。

divzero | nodivzero

整数除法の実行時検査を行います。0 による除法を行おうとすると、トラップが起こります。

394 ページの『#pragma options』も参照してください。

注

-qcheck オプションには、上記のとおり、幾つかのサブオプションがあります。複数のサブオプションを使用する場合は、コロン (:) でそれぞれを区切ってください。

サブオプション、およびコマンド行の **-qcheck** 以外のバリエーションなしで **-qcheck** オプションを指定すると、すべてのサブオプションがオンになります。

サブオプションを指定して **-qcheck** オプションを使用すると、`no` プレフィックスがない場合は指定したサブオプションがオンになり、`no` プレフィックスがある場合はオフになります。

-qcheck オプションは、複数回指定することができます。サブオプションの設定は累積されますが、後のサブオプションによって前のサブオプションがオーバーライドされます。

-qcheck オプションは、アプリケーションの実行時のパフォーマンスに影響を及ぼします。検査を有効にすると、実行時検査がアプリケーションに挿入されるため、実行が遅くなる場合があります。

例

1. **-qcheck=nullptr:bounds** の場合:

```
void func1(int* p) {
    *p = 42;          /* Traps if p is a null pointer */
}

void func2(int i) {
    int array[10];
    array[i] = 42;     /* Traps if i is outside range 0 - 9 */
}
```

2. **-qcheck=divzero** の場合:

```
void func3(int a, int b) {
    a / b;             /* Traps if b=0 */
}
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

394 ページの『#pragma options』

cinc

▶ C++

目的

コンパイラーに **extern "C" { }** ラッパーをインクルード・ファイルのコンテンツの周りに配置するように指示します。

構文

▶▶ `-q` `nocinc` `cinc` `directory_prefix` ▶▶

ここで、

`directory_prefix` このオプションが作用するファイルが入っているディレクトリを指定します。

注

指定されたディレクトリからのインクルード・ファイルでは、トークン **extern "C" {** がインクルード・ファイルの最初のステートメントの前に挿入され、**}** がインクルード・ファイルの最後のステートメントの後に付加されます。

例

アプリケーション `myprogram.C` がディレクトリ `/usr/tmp` に位置するヘッダー・ファイル `foo.h` を含んでおり、次のコードを含んでいるとします。

```
int foo();
```

以下を使用してアプリケーションをコンパイルします。

```
xlc myprogram.C -qcinc=/usr/tmp
```

アプリケーションに以下のようなヘッダー・ファイル `foo.h` が含まれます。

```
extern "C" {  
int foo();  
}
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

compact

► C ► C++

目的

最適化とともに使用すると、可能な場合に、実行速度を犠牲にしてコード・サイズが削減される。

構文

►► -q nocompact
compact ◀◀

394 ページの『#pragma options』も参照してください。

注

コード・サイズは、インライン化やループのアンロールなど、インラインでのコードの複製や展開による最適化を禁止することによって削減されます。実行時間が増加する可能性があります。

例

myprogram.C をコンパイルしてコード・サイズを削減するには、次のように入力します。

```
xlc myprogram.C -O -qcompact
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

394 ページの『#pragma options』

cplusplus

▶ C

目的

C++ のコメントを C ソース・ファイルで認識させるには、このオプションを使用します。

構文

▶ `-q` `nocplusplus` `cplusplus`

デフォルト

デフォルト設定は、以下のようにそれぞれ異なります。

- **xlc** または **xlc_r** を使用してコンパイラーを呼び出すと、**-qcplusplus** が暗黙的に選択されます。
- **-qlanglvl** が **stdc99** または **extc99** に対して設定されていると、**-qcplusplus** も暗黙的に選択されます。コマンド行で **-qlanglvl** オプションの後に、**-qnocplusplus** を指定することによって、これらの暗黙選択をオーバーライドすることができます。例えば、次のようにします。 **-qlanglvl=stdc99 -qnocplusplus** または **-qlanglvl=extc99 -qnocplusplus**
- そうでない場合は、デフォルト設定は **-qnocplusplus** です。

注

`__C99_CPLUSCMT` コンパイラー・マクロは、**cplusplus** が選択される際に定義されます。

文字シーケンス `//` は、ヘッダー名、文字定数、ストリング・リテラル、またはコメントを除いて、C++ のコメントの開始となります。コメントはネストしません。また、マクロ置き換えは、コメント内では実行されません。以下の文字シーケンスは、次の C++ コメント内で無視されます。

- `//`
- `/*`
- `*/`

C++ のコメントの形式は、`//text` です。文字シーケンスの 2 つのスラッシュ (`//`) は、2 つの間に何もはさまずに隣接していなければなりません。スラッシュの右側から改行文字で示される論理ソース行の最後までが、すべてコメントとして扱われます。`//` 区切り文字は、行内の任意の位置に置くことができます。

`//` コメントは C89 の一部ではありません。**-qcplusplus** を指定すると、以下の有効な C89 プログラムの結果が誤りになります。

```
main() {
    int i = 2;
    printf("%i\n", i /* 2 */
          + 1);
}
```

正しい答えは 2 (2 / 1) です。 **-qcpluscmt** を指定すると、結果は 3 (2 + 1) になります。

プリプロセッサは、以下の方法でコメントすべてを処理します。

- **-C** オプションを指定しない 場合は、コメントがすべて除去され、単一のブランクで置換されます。
- **-C** オプションを指定した 場合は、コメントがプリプロセッサ・ディレクティブまたはマクロ引き数に現れない限り、コメントが出力されます。
- **-E** を指定した場合は、継続シーケンスがすべてのコメントで認識されて出力されます。
- **-P** を指定した場合は、コメントが認識されて出力から除去され、連結された出力行が形成されます。

円記号 (¥) 文字を使用して複数の物理ソース行が 1 つの論理ソース行に結合されている場合には、コメントは、それらの物理ソース行にわたることができます。 3 文字表記 (??/) によって円記号を表すこともできます。

例

1. C++ のコメントの例

以下の例に、C++ のコメントの使用を示します。

```
// A comment that spans two ¥
physical source lines

// A comment that spans two ??/
physical source lines
```

2. プリプロセッサ出力の例 1

以下のソース・コード・フラグメントの場合:

```
int a;
int b; // A comment that spans two ¥
      physical source lines
int c;
      // This is a C++ comment
int d;
```

-P オプションの場合の出力は、以下のとおりです。

```
int a;
int b;
int c;

int d;
```

-P -C オプションの場合の C89 モード出力は、以下のとおりです。

```
int a;
int b; // A comment that spans two    physical source lines
int c;
      // This is a C++ comment
int d;
```

-E オプションの場合の出力は、以下のとおりです。

```
int a;
int b;

int c;

int d;
```

-E -C オプションの場合の C89 モード出力は、以下のとおりです。

```
#line 1 "fred.c"
int a;
int b; // a comment that spans two ¥
        physical source lines
int c;
        // This is a C++ comment
int d;
```

-P -C オプションまたは **-E -C** オプションの場合の拡張モード出力は、以下のとおりです。

```
int a;
int b; // A comment that spans two ¥
        physical source lines
int c;
        // This is a C++ comment
int d;
```

3. プリプロセッサ出力の例 2 - ディレクティブ行

以下のソース・コード・フラグメントの場合:

```
int a;
#define mm 1 // This is a C++ comment on which spans two ¥
              physical source lines
int b;
              // This is a C++ comment
int c;
```

-P オプションの場合の出力は、以下のとおりです。

```
int a;
int b;

int c;
```

-P -C オプションの場合の出力は、以下のとおりです。

```
int a;
int b;
        // This is a C++ comment
int c;
```

-E オプションの場合の出力は、以下のとおりです。

```
#line 1 "fred.c"
int a;
#line 4
int b;

int c;
```

-E -C オプションの場合の出力は、以下のとおりです。

```
#line 1 "fred.c"
int a;
#line 4
int b;

// This is a C++ comment

int c;
```

4. プリプロセッサ出力の例 3 - マクロ関数の引き数

以下のソース・コード・フラグメントの場合:

```
#define mm(aa) aa
int a;
int b; mm(// This is a C++ comment
        int blah);

int c;
// This is a C++ comment
int d;
```

-P オプションの場合の出力は、以下のとおりです。

```
int a;
int b; int blah;
int c;

int d;
```

-P -C オプションの場合の出力は、以下のとおりです。

```
int a;
int b; int blah;
int c;
// This is a C++ comment
int d;
```

-E オプションの場合の出力は、以下のとおりです。

```
#line 1 "fred.c"
int a;
int b;
int blah;
int c;

int d;
```

-E -C オプションの場合の出力は、以下のとおりです。

```
#line 1 "fred.c"
int a;
int b;
int blah;
int c;
// This is a C++ comment
int d;
```

5. コンパイル例

C++ のコメントがコメントとして認識されるように myprogram.c. so をコンパイルするには、次のように入力します。

```
xlc myprogram.c -qcpluscmt
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

98 ページの『C』

126 ページの『E』
198 ページの『langlvl』
252 ページの『P』

cpp_stdinc

➤ C++

目的

C++ ヘッダーの標準検索ロケーションを変更する。

構文

➡ `-q-cpp_stdinc=`  `path` ➡

注

C++ ヘッダーの標準検索パスは、**-qcpp_stdinc** コンパイラー・オプションによって決定されます。このコンパイラー・オプションが指定されず、または空のストリングを指定する場合、デフォルトのヘッダー・ファイル検索パスが使用されます。このオプションのデフォルトの検索パスは、コンパイラーのデフォルト構成ファイルにあります。

このオプションが複数回指定されている場合、コンパイラーは最後に指定されたオプションのみを使用します。複数のディレクトリーを検索パスに指定するには、このオプションを一度指定し、: (コロン) で複数の検索ディレクトリーを区切ってください。

このオプションは、**-qnostdinc** オプションが有効である場合には無視されます。

例

mypath/headers1 および **mypath/headers2** 標準検索パスを作成するには、以下を入力します。

```
xlc myprogram.C -qcpp_stdinc=mypath/headers1:mypath/headers2
```

関連タスク

43 ページの『相対パス名を使用したインクルード・ファイルのディレクトリー検索順序』

36 ページの『構成ファイル内のコンパイラー・オプションの指定』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

100 ページの『c_stdinc』

304 ページの『stdinc』

D



目的

#define プリプロセッサ・ディレクティブにあるのと同様にマクロ *name* を定義します。 *definition* は、 *name* に割り当てるオプションの定義または値です。

構文



注

また、マクロ名がすでに **-D** コンパイラー・オプションによって定義されていない場合は、**#define** プリプロセッサ・ディレクティブを使用してソース・プログラムのマクロ名を定義することができます。

`-Dname=` は、`#define name` と同等です。

-Dname は、#define name 1 と同等です。(これがデフォルトです。)

-D オプションによってすでに定義されているマクロ名を定義するのに **#define** デイレクティブを使用すると、エラー状態になります。

プログラムの移植性および標準への適合性を援助するために、オペレーティング・システムは、**-D** オプションで設定することができるマクロ名を参照するヘッダー・ファイルをいくつか提供します。これらのヘッダー・ファイルのほとんどは、**/usr/include** ディレクトリーまたは **/usr/include/sys** ディレクトリーのいずれかに入っています。詳細については、「*AIX Files Reference*」の『“Header Files Overview”』を参照してください。

構成ファイルは、**-D** オプションを使用して、以下の事前定義マクロを指定します。

マクロ名	AIX v5.2 に適用	AIX v5.1 に適用	AIX v4.3 に適用
AIX	✓	✓	✓
AIX32	✓	✓	✓
AIX41	✓	✓	✓
AIX43	✓	✓	✓
AIX50	✓	✓	
AIX51	✓	✓	
AIX52	✓		
IBMR2	✓	✓	✓
POWER	✓	✓	✓
ANSI_C_SOURCE	✓	✓	✓

ソース・ファイルに対する正しいマクロを確実に定義するには、適切なマクロ名を指定して **-D** オプションを使用してください。ソース・ファイルに

/usr/include/sys/stat.h ヘッダー・ファイルが組み込まれる場合は、
-D_POSIX_SOURCE オプションでコンパイルして、そのファイルの正しい定義を検出しなければなりません。

ソース・ファイルに **/usr/include/standards.h** ヘッダー・ファイルが組み込まれる場合は、**_ANSI_C_SOURCE**、**_XOPEN_SOURCE**、および **_POSIX_SOURCE** が定義されていなければ、これらが定義されます。

-Uname オプションは、**-D** オプションによって定義されたマクロを未定義にするために使用され、その優先順位は **-Dname** オプションよりも上です。

例

1. AIX v4.2 以降は、2 ギガバイトよりも大きいサイズのファイルをサポートしており、大容量のデータを単一ファイルで保管することができます。ユーザーのアプリケーションで大容量のファイルを操作できるようにするには、
-D_LARGE_FILES および **-qlonglong** コンパイラー・オプションを指定してコンパイルします。例を以下に示します。

```
xlc myprogram.c -D_LARGE_FILES -qlonglong
```

2. myprogram.c において、COUNT という名前のインスタンスをすべて 100 で置換するには、以下を入力します。

```
xlc myprogram.c -DCOUNT=100
```

これは、ソース・ファイルの先頭に **#define COUNT 100** があることと同等です。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

331 ページの『U』

473 ページの『付録 A. 事前定義マクロ』

以下も参照してください。

「Files Reference」の Header Files command

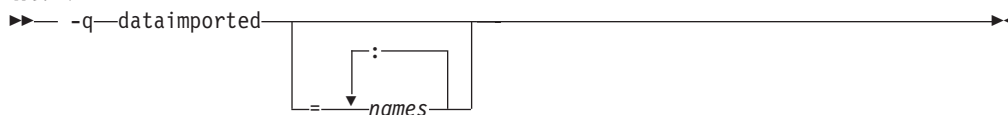
dataimported

➤ C ➤ C++

目的

インポートされるものとしてデータにマークを付ける。


構文



注

このオプションが有効な場合、インポートされる変数は、ライブラリーの共用部分と動的にバインドされます。

- **-qdataimported** を指定すると、変数をすべてインポートするものであると見なすようにコンパイラーに指示します。
- **-qdataimported=names** は、指定された変数にインポートされるものとしてマークを付けます。ここで、*names* は、コロン (:) によって区切られた変数名のリストです。明示的に指定されていない変数は影響を受けません。

注:  C++ プログラムでは、変数の名前 は、マングルされた名前を使用して指定する必要があります。例えば、次のコード・セグメントを想定します。

```
struct C{
    static int i;
}
```

コンパイラー・オプションを次のように指定することによって、変数 **C::i** をインポートするように指定できます。

```
-qdataimported=i 1C
```

オペレーティング・システム **dump -tv** または **nm** ユーティリティを使用して、オブジェクト・ファイルからマングルされた名前を取得することができます。マングルされた名前を検証するには、**c++filt** ユーティリティを使用します。

-qdataimported および **-qdatalocal** データ・マーキング・オプションが矛盾する場合は、以下の方法で解決されます。

変数名をリストするオプション: 特定の変数名に対する最後の明示的指定が使用されます。

この形式は、名前リストを指定しません。指定された最後のオプションが、名前リスト・フォームに明示的にリストされていない変数のデフォルトになります。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

118 ページの『datalocal』

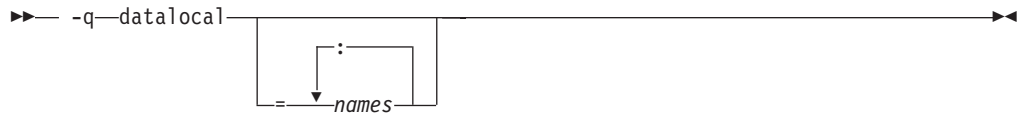
datalocal

➤ C ➤ C++

目的

ローカルとしてデータにマークを付ける。

構文



注

このオプションが有効な場合、ローカル変数は、それを使用する関数と静的にバインドされます。

- **-qdatalocal** を指定すると、変数をすべてローカルであると見なすようにコンパイラーに指示します。
- **-qdatalocal=names** は、指定された変数にローカルとしてマークを付けます。ここで、*names* は、コロン (:) によって区切られた ID のリストです。明示的に指定されていない変数は影響を受けません。

注: ➤ C++ C++ プログラムでは、変数の名前 は、マングルされた名前を使用して指定する必要があります。例えば、次のコード・セグメントを想定します。

```
struct C{
    static int i;
}
```

コンパイラー・オプションを次のように指定することによって、変数 **C::i** をローカル・データとして指定できます。

```
-qdatalocal=i__1C
```

オペレーティング・システム **dump -tv** または **nm** ユーティリティーを使用して、オブジェクト・ファイルからマングルされた名前を取得することができます。マングルされた名前を検証するには、**c++filt** ユーティリティーを使用します。

インポートする変数がローカルであると見なされると、パフォーマンスが低下する場合があります。

-qdataimported および **-qdatalocal** データ・マーキング・オプションが矛盾する場合は、以下の方法で解決されます。

変数名をリストするオプション: 特定の変数名に対する最後の明示的指定が使用されます。

デフォルトを変更するオプション: この形式は、名前リストを指定しません。指定された最後のオプションが、名前リスト・フォームに明示的にリストされていない変数のデフォルトになります。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

117 ページの『dataimported』

dbxextra



目的

すべての **typedef** 宣言、**struct** 型定義、**union** 型定義、および **enum** 型定義をデバッグ用に組み込むことを指定する。

構文

→ -q  →

394 ページの『#pragma options』も参照してください。

注

-g オプションと一緒にこのオプションを使用すると、デバッガーで使用するための追加のデバッグ情報を生成します。

-g オプションを指定すると、デバッグ情報がオブジェクト・ファイルに組み込まれます。オブジェクトおよび実行可能ファイルのサイズを最小にするために、コンパイラーは、参照されるシンボルに関する情報しか組み込みません。デバッグ情報は、**-qdbxextra** を指定しない限り、参照されない配列、ポインター、またはファイル・スコープの変数に対しては生成されません。

-qdbxextra を使用すると、オブジェクトおよび実行可能ファイルのサイズが増加する場合があります。

例

デバッグ用に myprogram.c 内のシンボルをすべて組み込むには、以下を入力します。

```
xlc myprogram.c -g -qdbxextra
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

155 ページの『g』

394 ページの『#pragma options』

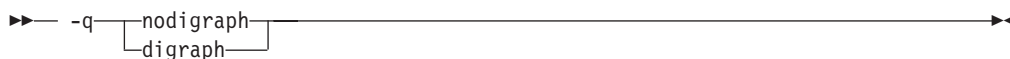
digraph

► C ► C++

目的

キーボードにない文字を表すために、連字キーの組み合わせおよびキーワードを使用できるようにする。

構文



394 ページの『#pragma options』も参照してください。

デフォルト

- **C** `-qlanglvl` が `extc99` または `stdc99` 以外の値に設定されている場合は、`-qnodigraph`。
- **C** `-qlanglvl` が `extc99` または `stdc99` の値に設定されている場合は、`-qdigraph`。
- **C++** `-qdigraph`

注

連字は、すべてのキーボードで利用できるわけではない文字を指定することができるキーワードまたはキーの組み合わせです。

連字キーの組み合わせは、以下のとおりです。

キーの組み合わせ	生成される文字
<%	{
%>	}
<:	[
:>]
%%	#

C++ プログラムでのみ有効な追加のキーワードは、以下のとおりです。

キーワード	生成される文字
bitand	&
and	&&
bitor	
or	
xor	^
compl	~
and_eq	&=
or_eq	=
xor_eq	^=
not	!
not eq	!=

例

プログラムをコンパイルするときに連字の文字シーケンスを使用できないようにするには、以下を入力します。

```
xlc myprogram.C -qnodigraph
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

198 ページの『langlvl』

325 ページの『trigraph』

394 ページの『#pragma options』

directstorage

► C ► C++

目的

ライトスルー対応ストレージまたはキャッシュ禁止ストレージが参照可能であることをコンパイラーに通知する。

構文

```
►► -q[nodirectstorage|directstorage]◀◀
```

注

-qdirectstorage コンパイラー・オプションは、ライトスルー対応またはキャッシュ禁止のストレージが参照される可能性があること、および適切なコンパイラー出力を生成すべきであることを、コンパイラーに通知します。

PowerPC アーキテクチャを使用すると、キャッシュ編成のさまざまなインプリメンテーションが可能になります。確実にアプリケーションをすべてのインプリメンテーションで正しく実行するためには、さまざまな命令およびデータ・キャッシュの存在を想定し、それに従ってアプリケーションをプログラムする必要があります。

関数を確実に正しく実行するため、プログラムで指定されたストレージ管理属性、および実行されている関数に応じて、キャッシュ命令を使用する場合があります。

例えば、**dcbz** 命令は、データのブロックをキャッシュに割り振ってから、それを一連のゼロに初期化します。**dcbz** 命令は、データの大きなブロックをゼロ化する場合にパフォーマンスを上げるために使用できますが、以下のいずれかの条件下で位置合わせエラーが発生するため、慎重に使用する必要があります。

- 命令によって指定されたキャッシュ・ブロックが、キャッシュ禁止とマークされたメモリー領域にある。
- キャッシュがライトスルー・モードである。
- L1 Dcache か L2 キャッシュが使用不可である。

-qdirectstorage を指定すると、**dcbz** 命令の生成を抑制し、上記の位置合わせエラーが回避されます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

dollar

► C ► C++

目的

\$ シンボルを ID の名前で使えるようにする。

構文

►► — -q —  —————►►

394 ページの『#pragma options』も参照してください。

例

\$ をプログラム内の ID で使用できるように、myprogram.c をコンパイルするには、以下を入力します。

```
xlc myprogram.c -qdollar
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

394 ページの『#pragma options』

dpcl

► C ► C++

目的

動的プローブ・クラス・ライブラリー (DPCL) に基づくツールが実行可能ファイルの構造体を参照するために使用できるシンボルを生成します。

構文

```
►► -q  -q
```

注

DPCL はアプリケーション・プログラム・インターフェース (API)、またはライブラリーで、プログラミング・ツールのビルドに関するタスクを単純化できます。アプリケーション・パフォーマンス分析ツールをサポートするように設計されていますが、アプリケーション・ステアリング、ロード・バランシング、およびこのほか多くのタイプのツールにも使用されることがあります。

DPCL はシリアル、共用メモリー、およびメッセージ引き渡しアプリケーションを同様に簡単にサポートします。ツールで新規アプリケーションを作成し、または実行中のアプリケーションに接続することを可能にするクライアント/サーバー・アーキテクチャーを使用します。DPCL は、アプリケーションを再コンパイルすることなく、実行中に、アプリケーションからインストルメンテーションを挿入または除去します。

-qdpcl オプションを指定すると、コンパイラーはプログラム内のコードのブロックを定義するためのシンボルを送信します。これにより、DPCL インターフェースを使用するツールを使用して、このオプションでコンパイルしたオブジェクト・ファイルのメモリー使用量などのパフォーマンス情報を検査することができます。

-qdpcl を **-g** オプションとともに指定し、確実にコンパイラーがデバッグおよびプログラム分析ツールによって必要とされるデバッグ情報を生成するようする必要があります。

-qipa オプションまたは **-qsmp** オプションを **-qdpcl** とともに指定することはできません。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

『dpcl』

155 ページの『g』

181 ページの『ipa』



目的

ソース・ファイルをプリプロセスするようにコンパイラーに指示する。

構文

▶▶ — -E —▶▶

注

-E および **-P** オプションの結果は異なります。**-E** オプションを指定すると、コンパイラーは、入力が **C** または **C++** ファイルであり、出力が何らかの方法で再コンパイルまたは再処理されると見なします。これらの前提事項を以下に示します。

- 元のソースの位置が保持されます。これは、**#line** ディレクティブが生成されるためです。
- すべてのトークンは元のつづりで出力されます。この場合、継続シーケンスが含まれます。つまり、他のツールによる以後のコンパイルまたは再処理において、同じ位置 (例えば、エラー・メッセージの位置) が提供されます。

-P オプションは、汎用のプリプロセスのために使用します。入力または出力の使用目的に関する前提事項はありません。このモードは、**C** または **C++** で作成されていない入力ファイルとともに使用します。このため、プリプロセス固有の構成は、ANSI **C** 標準の記述のとおり処理されます。この場合は、継続シーケンスは、同標準の “Phases of Translation” の記述のとおり除去されます。プリプロセス固有のテキストでないテキストは、すべてそのまま出力されます。

-E を使用すると、トークンのソース位置を保持するために **#line** ディレクティブが生成されます。ブランク行は除去されて、代わりに **#line** ディレクティブで置換されます。

-P オプションでは、行継続シーケンスは除去されて、ソース行が連結されます。**-E** オプションでは、トークンは、ソース位置を保持するために別個の行に出力されます。この場合は、継続シーケンスは除去されます。

-E オプションは、**-P**、**-o**、および **-qsyntaxonly** オプションをオーバーライドして、任意のファイル名を受け入れます。

-E を **-M** オプションとともに使用すると、ファイル名のサフィックスが **.C**、**.cpp**、**.cc** (すべての **C++** ソース・ファイル)、**.c** (**C** ソース・ファイル)、または **.i** (プリプロセスされたソース・ファイル) であるファイルのみが処理されます。認識されないファイル名サフィックスが付いたソース・ファイルは、**C** ファイルと見なされてプリプロセスされ、エラー・メッセージは生成されません。

-C が指定されていない限り、プリプロセスされた出力では、コメントは単一の空白文字で置換されます。コメントが複数のソース行にわたり、**-C** が指定されていない場合は、改行および **#line** ディレクティブが出されます。マクロ関数の引き数にあるコメントは削除されます。

デフォルトでは、ソース・ファイルがプリプロセス、コンパイル、およびリンク・エディットされて、実行可能ファイルが生成されます。

例

myprogram.C をコンパイルしてプリプロセスされたソースを標準出力に送るには、以下を入力します。

```
x1C myprogram.C -E
```

myprogram.C に以下のようなコード・フラグメントがある場合は、

```
#define SUM(x,y) (x + y) ;
int a ;
#define mm 1 ; /* This is a comment in a
                preprocessor directive */
int b ;          /* This is another comment across
                  two lines */
int c ;
                /* Another comment */
c = SUM(a, /* Comment in a macro function argument*/
        b) ;
```

出力は以下のようになります。

```
#line 2 "myprogram.C"
int a;
#line 5
int b;

int c;

c =
(a + b);
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

226 ページの『M』

248 ページの『o』

252 ページの『P』

310 ページの『syntaxonly』

e

► C ► C++

目的

このオプションは、**-qmkshrobj** コンパイラー・オプションを指定した場合のみ使用される。詳しくは、**-qmkshrobj** コンパイラー・オプションの説明を参照してください。

構文

►► — *-e—name—* —————▶▶

関連資料

61 ページの『コンパイラーのコマンド行オプション』

239 ページの『mkshrobj』

eh

► C++

目的

コンパイル中のモジュールで例外処理が使用可能であるかどうかを制御する。

構文



ここで、

- v5 VisualAge C++ V5.0 との互換性を持つ例外処理コードを生成するようにコンパイラーに指示します。
- v6 VisualAge C++ V6.0 との互換性があり、他の catch ブロック内にネストされている try-catch ブロックを正しく処理する新規の例外処理コードを生成するようにコンパイラーに指示します。
- eh 例外処理が使用可能です。-qeh がサブオプションなしで指定された場合、-qeh=v5 が想定されます。
- noeh プログラムが C++ 構造化例外処理を使用しない場合は、-qnoeh を指定してコンパイルし、アプリケーションに必要なコードを生成しないようにしてください。

注

関連資料

61 ページの『コンパイラーのコマンド行オプション』

283 ページの『rtti』

enum

► C ► C++

目的

列挙が占有するストレージの量を指定する。

構文



ここで、有効な **enum** 設定は、以下のとおりです。

- 1 列挙が 1 バイトのストレージを占有することを指定します。列挙値の範囲が **signed char** の限度内にある場合は、列挙の型は **char** 型であり、そうでない場合は **unsigned char** であることを指定します。
 - 2 列挙が 2 バイトのストレージを占有することを指定します。列挙値の範囲が **signed short** の限度内にある場合は、列挙の型は **short** 型であり、そうでない場合は **unsigned short** であることを指定します。
 - 4 列挙が 4 バイトのストレージを占有することを指定します。列挙値の範囲が **signed int** の限度内にある場合は、列挙の型は **int** 型であり、そうでない場合は **unsigned int** であることを指定します。
 - 8 列挙が 8 バイトのストレージを占有することを指定します。
32 ビット・コンパイル・モードでは、列挙値の範囲が **signed long long** の制限内にある場合、その列挙は **long long** 型で、そうでない場合は **unsigned long long** です。
64 ビット・コンパイル・モードでは、列挙値の範囲が **signed long** の制限内にある場合は、その列挙は **long** 型で、そうでない場合は **unsigned long** です。
- int 列挙が 4 バイトのストレージを占有して、**int** で表されることを指定します。値は、C コンパイルにおける **signed int** の範囲を超えることはできません。
- intlong 列挙の値の範囲が **int** の限度を超える場合、列挙が 8 バイトのストレージを占有することを指定します。 **-qenum=8** の説明を参照してください。
列挙の値の範囲が **int** の制限を超えない場合は、列挙は 4 バイトのストレージを占有し、**int** によって示されます。
- small 列挙が列挙内の値の範囲を正確に表せる最小のスペース (1、2、4、または 8 バイトのストレージ) を占有するように指定します。値の範囲に負の値が含まれない限り、符号は *unsigned* です。
8 バイトの **enum** が結果として生じる場合、実際に使用される列挙型はコンパイル・モードに依存しています。 **-qenum=8** の説明を参照してください。

363 ページの『**#pragma enum**』および 394 ページの『**#pragma options**』も参照してください。

注

-qenum=small オプションは、**enum** 定数の範囲を表すことができる最小の事前定義型に必要なストレージの量を **enum** 変数 に割り振ります。デフォルトでは、符号なしの事前定義型が使用されます。**enum** 定数のいずれかが負の場合は、符号付きの事前定義型が使用されます。

-qenum=1|2|4|8 オプションは、**enum** 変数 に特定の量のストレージを割り振ります。指定されたストレージ・サイズが **enum** 変数の範囲が必要とするサイズよりも小さい場合は、重大エラー・メッセージが出され、コンパイルが停止します。

ISO C 1989 および ISO C1999 規格では、列挙範囲が **int** の範囲を超えないことが要求されます。有効な **-qlanglvl=stdc89** または **-qlanglvl=stdc99** でコンパイルすると、列挙の値が **int** の範囲を超える場合、コンパイラーは以下のように振る舞います。

- **-qenum=int** が有効な場合、重大エラー・メッセージが出されてコンパイルが停止する。
- **-qenum** のほかのすべての設定には、情報メッセージが出されてコンパイルが継続する。

以下の表に、事前定義型の選択のための優先順位を示します。この表には、事前定義型、対応する事前定義型の **enum** 定数の最大範囲、およびその事前定義型に必要なストレージの量、つまり、最小サイズの **enum** に適用した場合に、**sizeof** 演算子によって得られる値も示します。

関連資料

61 ページの『コンパイラーのコマンド行オプション』


363 ページの『`#pragma enum`』

394 ページの『`#pragma options`』

Enum サイズと型 - 特に断りがない限り、すべての型が signed です。

enum=1			enum=2		enum=4		enum=8			
							32 ビット・コンパイル・モード		64 ビット・コンパイル・モード	
範囲	var	const	var	const	var	const	var	const	var	const
0..127	char	int	short	int	int	int	long long	long long	long	long
-128..127	char	int	short	int	int	int	long long	long long	long	long
0..255	unsigned char	int	short	int	int	int	long long	long long	long	long
0..32767	ERROR ¹	int	short	int	int	int	long long	long long	long	long
-32768..32767	ERROR ¹	int	short	int	int	int	long long	long long	long	long
0..65535	ERROR ¹	int	unsigned short	int	int	int	long long	long long	long	long
0..2147483647	ERROR ¹	int	ERROR ¹	int	int	int	long long	long long	long	long
-(2147483647+1).. 2147483647	ERROR ¹	int	ERROR ¹	int	int	int	long long	long long	long	long
0..4294967295	ERROR ¹	unsigned int	ERROR ¹	unsigned int	unsigned int	unsigned int	long long	long long	long	long
0..(2 ⁶³ -1)	ERROR ¹	long ^{2b}	ERROR ¹	long ^{2b}	ERROR ¹	long ^{2b}	long long ^{2b}	long long ^{2b}	long ^{2b}	long ^{2b}
-2 ⁶³ ..(2 ⁶³ -1)	ERROR ¹	long ^{2b}	ERROR ¹	long ^{2b}	ERROR ¹	long ^{2b}	long long ^{2b}	long long ^{2b}	long ^{2b}	long ^{2b}
0..2 ⁶⁴	ERROR ¹	unsigned long ^{2b}	ERROR ¹	unsigned long ^{2b}	ERROR ¹	unsigned long ^{2b}	unsigned long long ^{2b}	unsigned long long ^{2b}	unsigned long ^{2b}	unsigned long ^{2b}


注:

- これらの列挙は **-qenum=11214** 設定では大き過ぎます。重大エラーが出されてコンパイルが停止します。
この状態を訂正するには、列挙の範囲を狭めて、より大きな **enum** 設定を選択するか、または **small** または **intlong** などの動的 **enum** 設定を選択する必要があります。
-  列挙型は、C アプリケーションを ISO C 1989 および ISO C 1999 規格にコンパイルするときに、**int** の範囲を超えてはいけません。有効な **-qlanglvl=stdc89** または **-qlanglvl=stdc99** でコンパイルすると、列挙の値が **int** の範囲を超える場合、コンパイラーは以下のように振る舞います。
 - qenum=int** が有効な場合、重大エラー・メッセージが出されてコンパイルが停止する。
 - qenum** のほかのすべての設定には、情報メッセージが出されてコンパイルが継続する。

Enum サイズと型 - 特に断りがない限り、すべての型が **signed** です。

	enum=int			enum=intlong				enum=small			
	enum=int			enum=intlong				enum=small			
範囲	var	const		var	const	var	const	var	const	var	const
0..127	int	int		int	int	int	int	unsigned char	int	unsigned char	int
-128..127	int	int		int	int	int	int	signed char	int	signed char	int
0..255	int	int		int	int	int	int	unsigned char	int	unsigned char	int
0..32767	int	int		int	int	int	int	unsigned short	int	unsigned short	int
-32768..32767	int	int		int	int	int	int	short	int	short	int
0..65535	int	int		int	int	int	int	unsigned short	int	unsigned short	int
0..2147483647	int	int		int	int	int	int	unsigned int	int	unsigned int	int
-(2147483647+1).. 2147483647	int	int		int	int	int	int	int	int	int	int
0..4294967295	unsigned int	unsigned int		unsigned int	unsigned int	unsigned int	unsigned int	unsigned int	unsigned int	unsigned int	unsigned int
0..(2 ⁶³ -1)	ERR ^{2a}	ERR ^{2a}		long long ^{2b}	long long ^{2b}	long long ^{2b}	long ^{2b}	unsigned long long ^{2b}	unsigned long ^{2b}	unsigned long ^{2b}	unsigned long ^{2b}
-2 ⁶³ .. (2 ⁶³ -1)	ERR ^{2a}	ERR ^{2a}		long long ^{2b}	long long ^{2b}	long long ^{2b}	long ^{2b}	long long ^{2b}	long long ^{2b}	long ^{2b}	long ^{2b}
0..2 ⁶⁴	ERR ^{2a}	ERR ^{2a}		unsigned long long ^{2b}	unsigned long long ^{2b}	unsigned long long ^{2b}	unsigned long ^{2b}	unsigned long long ^{2b}	unsigned long long ^{2b}	unsigned long ^{2b}	unsigned long ^{2b}

注:

- これらの列挙は **-qenum=11214** 設定では大き過ぎます。重大エラーが出されてコンパイルが停止します。この状態を訂正するには、列挙の範囲を狭めて、より大きな **enum** 設定を選択するか、または **small** または **intlong** などの動的 **enum** 設定を選択する必要があります。
-  列挙型は、C アプリケーションを ISO C 1989 および ISO C 1999 規格にコンパイルするときに、**int** の範囲を超えてはいけません。有効な **-qlanglvl=stdc89** または **-qlanglvl=stdc99** でコンパイルすると、列挙の値が **int** の範囲を超える場合、コンパイラーは以下のように振る舞います。
 - qenum=int** が有効な場合、重大エラー・メッセージが出されてコンパイルが停止する。
 - qenum** のほかのすべての設定には、情報メッセージが出されてコンパイルが継続する。

expfile

► C ► C++

目的

1 つの指定ファイルにすべてのエクスポートされたシンボルを保管します。

このオプションは、**-qmkshrobj** コンパイラー・オプションを指定した場合のみ使用されます。詳しくは、**-qmkshrobj** コンパイラー・オプションの説明を参照してください。

構文

►► `-q-expfile=filename` ◀◀

関連資料

61 ページの『コンパイラーのコマンド行オプション』

239 ページの『mkshrobj』

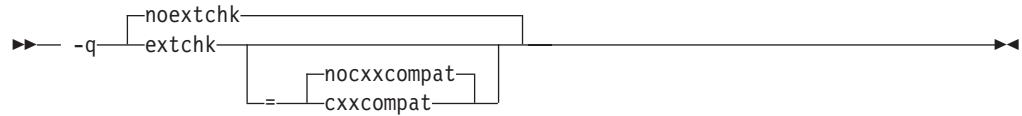
extchk

➤ C ➤ C++

目的

バインド時型検査の情報を生成し、コンパイル時の整合性について検査する。

構文



ここで、以下のサブオプションは、C++ のコンパイルにのみ適用されます。

nocxxcompat	C でコンパイルされたオブジェクト、または -qextchk=nocxxcompat を有効にして C++ でコンパイルされたオブジェクトとの互換性を提供します。コマンド行で -qextchk がサブオプションなしで指定されている場合は、このサブオプションがデフォルトです。
cxxcompat	IBM C++ コンパイラー V3.6 以前のバージョンのコンパイラーでコンパイルされたオブジェクトとの互換性を提供します。

394 ページの『#pragma options』も参照してください。

注

-qextchk は、コンパイル時に整合性を検査して、リンク時にコンパイル単位間での不一致を検出します。

-qextchk は、不完全型への参照を含む関数またはオブジェクトに対する型の検査を行いません。

例

バインド時検査情報を生成するように myprogram.c をコンパイルするには、以下を入力します。

```
x1C myprogram.c -qextchk
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

394 ページの『#pragma options』

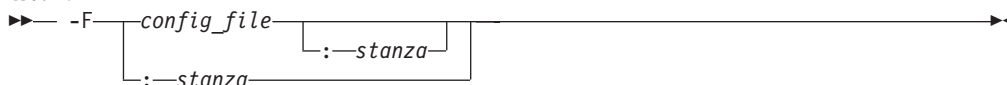
F

➤ C ➤ C++

目的

コンパイラーの代替構成ファイル (.cfg) の名前を指定する。

構文



サブオプションは、以下のとおりです。

<i>config_file</i>	コンパイラー構成ファイルの名前を指定します。
<i>stanza</i>	コンパイラーを起動するために使用するコマンドの名前を指定します。これは、コンパイラーが、コンパイラー環境をセットアップするために、 <i>config_file</i> 内の <i>stanza</i> の記入項目を使用するよう指示します。

注

デフォルトは、インストール時に提供される構成ファイルです。コマンド行またはソース・ファイル内で指定するファイル名またはスタンザは、すべて構成ファイルで指定したデフォルトをオーバーライドします。

構成ファイルの内容に関する詳細については、36 ページの『構成ファイル内のコンパイラー・オプションの指定』を参照してください。

-B、**-t**、および **-W** オプションは、**-F** オプションをオーバーライドします。

例

/usr/tmp/myvac.cfg という構成ファイルを使用して **myprogram.c** をコンパイルするには、以下のように入力します。

```
xlc myprogram.c -F/usr/tmp/myvac.cfg:xlc
```

関連タスク

36 ページの『構成ファイル内のコンパイラー・オプションの指定』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

93 ページの『B』

311 ページの『t』

341 ページの『W』

f

► C ► C++

目的

ファイルを指定して、リンカーへ渡すための **xlc** のオブジェクト・ファイルのリストを保管します。

構文

►► `-f` *filelistname* ◀◀

注

filelistname ファイルには、オブジェクト・ファイルの名前のみを含めるようにしてください。1 行につき 1 つのオブジェクト・ファイルにしてください。

このオプションは、**ld** コマンド用の **-f** オプションと同じです。

例

myobjlistfile に含まれているファイルのリストをリンカーに渡すには、以下を入力してください。

```
xlc -f/usr/tmp/myobjlistfile
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

fdpr

► C ► C++

目的

AIX の **fdpr** (Feedback Directed Program Restructuring) パフォーマンス・チューニング・ユーティリティーで使用するための、プログラムに関する情報を収集します。

構文

```
►► -q  
```

注

fdpr パフォーマンス・チューニング・ユーティリティーを使ってプログラムを最適化する前に、**-qfdpr** を指定してプログラムをコンパイルしてください。最適化データは、オブジェクト・ファイルに保管されます。

fdpr パフォーマンス・チューニング・ユーティリティーの使用に関する詳細については、「AIX バージョン 4 コマンド・リファレンス」を参照するか、以下のコマンドを入力してください。

```
man fdpr
```

例

fdpr ユーティリティーに必要なデータが含まれるように `myprogram.c` をコンパイルするには、以下を入力します。

```
xlc myprogram.c -qfdpr
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

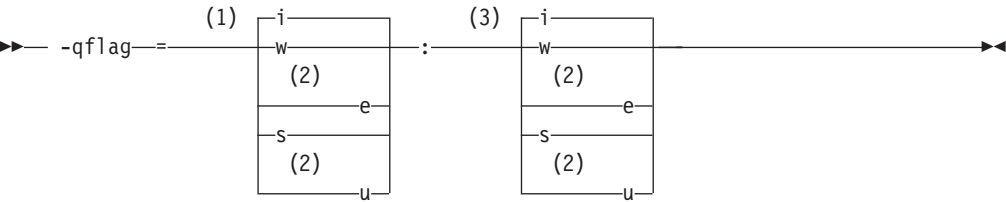
flag

C C++

目的

リストに報告させるとともに端末に表示させる診断メッセージの最低限の重大度レベルを指定する。診断メッセージは、関連するサブメッセージとともに表示されます。

構文



注:

- 1 リストに報告させる、最低限の重大度レベルのメッセージ
- 2 C コンパイルのみ
- 3 端末について報告させる、最低限の重大度レベルのメッセージ

ここで、メッセージ重大度レベルは、以下のとおりです。

重大度	説明
i	通知
w	警告
e	エラー (C コンパイルのみ)
s	重大エラー
u	回復不能エラー (C コンパイルのみ)

394 ページの『#pragma options』も参照してください。

注

リスト報告および端末報告の両方について、最低限のメッセージ重大度レベルを指定しなければなりません。

通知メッセージ・レベルを指定しても、**-qinfo** オプションはオンになりません。

例

myprogram.c をコンパイルし、その際に生成された全メッセージをリストに示し、ワークステーションにはエラー・メッセージと重大度の高いメッセージ (およびエラーの修正に役立つ関連する通知メッセージ) だけが表示されるようにするには、次のように入力します。

```
xlc myprogram.c -qflag=i:e
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

171 ページの『info』

342 ページの『w』

394 ページの『#pragma options』

457 ページの『コンパイラー・メッセージ』

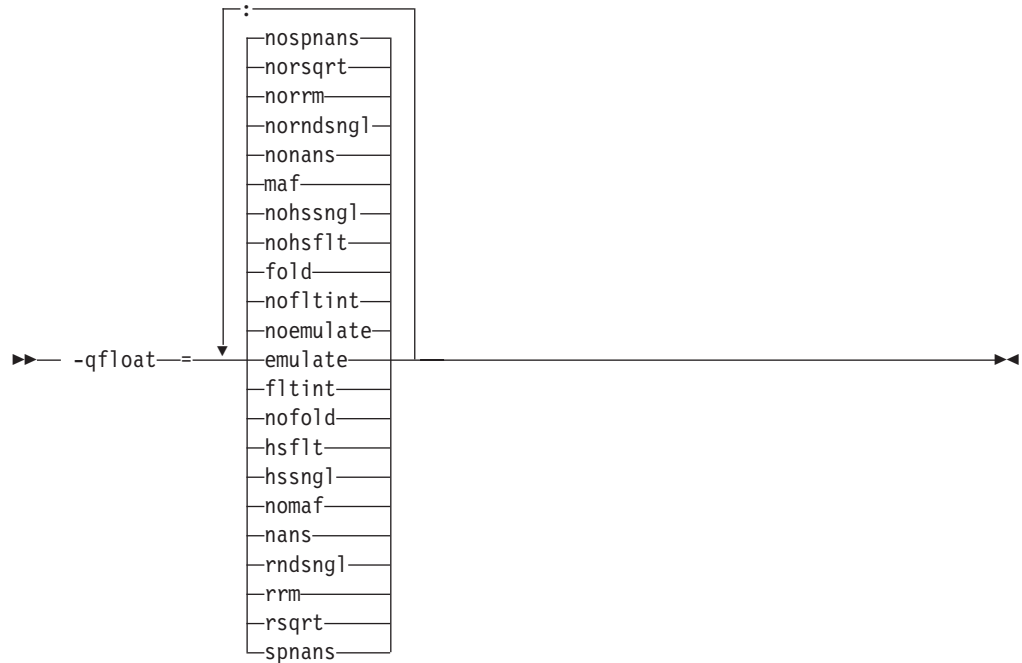
float

► C ► C++

目的

さまざまな浮動小数点オプションを指定する。これらのオプションは、浮動小数点の計算を高速化したり、より正確にしたりするための各種の技法を提供します。

構文



オプション選択については、以下の『注』セクションで説明します。 394 ページの『#pragma options』も参照してください。

注

デフォルト設定以外の **float** サブオプションを使用すると、浮動小数点計算でさまざまな結果を作成することになります。特定のサブオプションに対する必要な条件をすべて満たしていない場合は、誤った計算結果になる場合があります。このため、IEEE 浮動小数点値に関する浮動小数点計算の経験があり、プログラムにエラーが混入する可能性を適切に評価することができる場合でなければ、このオプションを使用してはなりません。

以下の 1 つ以上の **float** サブオプションを指定できます。

emulate PowerPC 403™ プロセッサで省略された浮動小数点命令をエミュレートします。デフォルトは **float=noemulate** です。

noemulate

PowerPC 403 プロセッサの浮動小数点命令をエミュレートするには、**-qfloat=emulate** を使用します。PowerPC 403 の浮動小数点命令の代わりに関数呼び出しが出されます。このオプションは、PowerPC 403 プロセッサをターゲットとした単スレッドの独立型環境でのみ使用してください。

以下とともに **-qfloat=emulate** を使用してはなりません。

- **-qarch=pwr, -qarch=pwr2, -qarch=pwrx**
- **-qlongdouble, -qldbl128**
- **xlC128** または **xlC128** コンパイラ呼び出しコマンド

fltint オーバーフローを検査しない高速のインライン・コードを使用することによって、浮動小数点から整数への変換を高速化します。デフォルトは **float=nofltint** であり、浮動小数点から整数への変換を検査して、範囲外の値がないかどうか調べます。

nofltint

このサブオプションは、最適化オプションとともにのみ使用するようになっています。

- 有効な **-O2** を使用すると、**-qfloat=nofltint** は暗黙設定です。
- 有効な **-O3** 以上を使用すると、**-qfloat=fltint** は暗黙指定されます。

-O3 オプションによって浮動小数点から整数への変換に範囲検査を組み込むには、**-qfloat=nofltint** を指定します。

- **-qnostrict** は、**-qfloat=fltint** を設定します。

fltint サブオプションが既に指定されている場合は、最適化レベルを変更しても、**fltint** の設定は変更されません。

PWR2 および PPC ファミリー・アーキテクチャーでは、範囲外の値を正しく処理する、より高速のインライン・コードが使用されます。

-qstrict | **-qnostrict** オプションと **-qfloat=** オプションが矛盾する場合は、最後の設定が使用されます。

fold 浮動小数点定数式を、実行時ではなくコンパイル時に評価することを指定します。

nofold

-qfloat=fold オプションは、廃止された **-qfold** オプションに代わるものです。新しいアプリケーションでは、**-qfloat=fold** を使用してください。

hsflt
nohsflt

注: **hsflt** サブオプションは、浮動小数点計算の特性が既知である特定のアプリケーションのためのものです。他のアプリケーション・プログラムをコンパイルするときにこのオプションを使用すると、誤った結果になる場合がありますが、警告は出されません。

-qfloat=rndsngl、**-q64**、または **-qarch=ppc**、あるいは他の PPC ファミリー・アーキテクチャー設定を指定してこのオプションを使用すると、rs64b 以降のシステムで間違った結果が生成される可能性があります。

hsflt オプションは、保管の前および浮動小数点から整数への変換時に、計算値を単精度に丸めるのではなく切り捨てることによって、計算を高速化します。 **nohsflt** サブオプションは、式の評価後に単精度式を丸め、範囲外の値かどうかについて浮動小数点から整数への変換を検査することを指定します。

hsflt サブオプションは、 **rndsngl**、**nans**、および **spnans** サブオプションをオーバーライドします。

-qfloat=hsflt オプションは、廃止された **-qhsflt** オプションに代わるものです。新しいアプリケーションでは、**-qfloat=hsflt** を使用してください。

このオプションは、**-qarch** オプションが **pwr**、**pwr2**、**pwr_x**、**pwr2s**、または **com** (32 ビット・モードの場合) に設定されていない場合にはほとんど効果がありません。PPC ファミリー・アーキテクチャーでは、すべての単精度 (float) 演算が丸められます。このオプションの影響を受けるのは、単精度 (float) への倍精度 (double) 式のキャストだけです。

hssngl
nohssngl

注: **-qfloat=rndsngl**、**-q64**、または **-qarch=ppc**、あるいは他の PPC ファミリー・アーキテクチャー設定を指定してこのサブオプションを使用すると、rs64b 以降のシステムで間違った結果が生成される可能性があります。

hssngl オプションは、結果を **float** のメモリー位置に保管するときのみ単精度式を丸めることを指定します。 **nohssngl** オプションは、式の評価後に単精度式を丸めることを指定します。 **hssngl** を使用すると、実行時のパフォーマンスを向上させることが可能であり、**-qfloat=hsflt** を使用するよりも安全です。

このサブオプションは、**-qarch** オプションが **pwr**、**pwr2**、**pwr_x**、**pwr2s**、または **com** (32 ビット・モードの場合) に設定されていない場合にはほとんど効果がありません。PPC ファミリー・アーキテクチャーでは、すべての単精度 (float) 演算が丸められます。このオプションの影響を受けるのは、単精度 (float) への倍精度 (double) 式のキャストだけで、このオプションは **store** 命令が生成される代入演算子で使用されます。

-qfloat=hssngl オプションは、廃止された **-qhssngl** オプションに代わるものです。新しいアプリケーションでは、**-qfloat=hssngl** を使用してください。

maf nomaf	<p>適切な箇所で浮動小数点乗算・加算命令を使用することによって、より高速でより正確に浮動小数点計算を行います。この結果は、コンパイル時に行われる類似の計算の結果または他のタイプのコンピュータでの結果と正確に同じにならない場合があります。負のゼロ結果が作成されることがあります。このオプションは、浮動小数点の中間結果の精度に影響を及ぼす場合があります。</p> <p>-qfloat=maf オプションは、廃止された -qmaf オプションに代わるものです。新しいアプリケーションでは、-qfloat=maf を使用してください。</p>
nans nonans	<p>追加の命令を生成して、実行時に単精度から倍精度に変換するときシグナル型 NaN (Not-a-Number) を検出します。オプション nonans は、この変換を検出する必要がないことを指定します。</p> <p>-qfloat=nans は、IEEE 754 規格に完全に準拠するために必要です。</p> <p>hsflt オプションは、nans オプションをオーバーライドします。</p> <p>-qflttrap または -qflttrap=invalid オプションとともに使用すると、コンパイラーは、オペランドのいずれかがシグナル型 NaN である場合に引き起こされる比較演算における無効演算例外を検出します。</p> <p>-qfloat=nans オプションは、廃止された -qfloat=spnans オプションおよび -qspnans オプションに代わるものです。新しいアプリケーションでは、-qfloat=nans を使用してください。</p>
rndsngl norndsngl	<p>各単精度 (float) 演算の結果を単精度に丸めることを指定します。-qfloat=norndsngl は、式全体を評価した後でなければ単精度への丸めを行わないことを指定します。このオプションを使用すると、他のタイプのコンピュータでの類似した計算の結果と整合させるために、速度が犠牲になる場合があります。</p> <p>hsflt サブオプションは、rndsngl オプションをオーバーライドします。</p> <p>このサブオプションは、-qarch オプションが pwr、pwr2、pwrx、pwr2s、または com (32 ビット・モードの場合) に設定されていない場合には効果がありません。PPC ファミリー・アーキテクチャーでは、すべての単精度 (float) 演算が丸められます。</p> <p>-qfloat=hssngl または -qfloat=hsflt を指定してこのオプションを使用すると、rs64b 以降のシステムでは間違った結果が生成される可能性があります。</p> <p>-qfloat=rndsngl オプションは、廃止された -qrndsngl オプションに代わるものです。新しいアプリケーションでは、-qfloat=rndsngl を使用してください。</p>

rrm
norrm

浮動小数点の最適化が、正および負の無限大への実行時丸めモードと非互換にならないようにします。浮動小数点の丸めモードが実行時に変更される可能性があること、または浮動小数点の丸めモードが実行時に最も近い値に丸めるモードでないことをコンパイラーに通知します。

浮動小数点状況レジスターおよび制御レジスターを実行時に変更する場合は、**-qfloat=rrm** を指定しなければなりません (例外トラッピングの初期化についても同様です)。

-qfloat=rrm オプションは、廃止された **-qrrm** オプションに代わるものです。新しいアプリケーションでは、**-qfloat=rrm** を使用してください。

rsqrt
norsqrt

平方根の逆数を計算して乗算することによって、平方根の結果による除法を含むコードのシーケンスを置換することができるかどうかを指定します。この置換を可能にすると、コードの実行が高速化されます。

- **-O2** の場合、デフォルトは **-qfloat=norsqrt** です。
- **-O3** の場合、デフォルトは **-qfloat=rsqrt** です。このデフォルトをオーバーライドするには、**-qfloat=norsqrt** を使用します。
- **-qnostrict** は、**-qfloat=rsqrt** を設定します。(**-qfloat=rsqrt** は、**errno** が **sqrt** 関数呼び出しの場合に設定されないことを意味するので注意してください。)
- **-qarch=pwr2** も指定している場合、**-qfloat=rsqrt** は無効です。
- **-qignerrno** も指定しない限り、**-qfloat=rsqrt** は無効です。

rsqrt がすでに指定してある場合は、最適化レベルを変更しても、**rsqrt** オプションの設定は変更されません。**-qstrict** | **-qnostrict** オプションと **-qfloat=** オプション が矛盾する場合は、最後の設定が使用されます。

spnans
nospnans

追加の命令を生成して、単精度から倍精度への変換時にシグナル型 NaN を検出します。オプション **nospnans** は、この変換を検出する必要がないことを指定します。

hsflt サブオプションは、**spnans** サブオプションをオーバーライドします。

-qfloat=spnans オプションは、廃止された **-qfloat=spnans** および **-qspnans** オプションに代わるものです。新しいアプリケーションでは、**-qfloat=spnans** を使用してください。

例

コンパイル時に定数浮動小数点式を評価し、乗算・加算命令を生成しないように **myprogram.C** をコンパイルするには、以下を入力します。

```
xlc myprogram.C -qfloat=fold:nomaf
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

84 ページの『arch』

147 ページの『flttrap』

217 ページの『ldbl128、longdouble』

282 ページの『rm』

305 ページの『strict』

394 ページの『#pragma options』

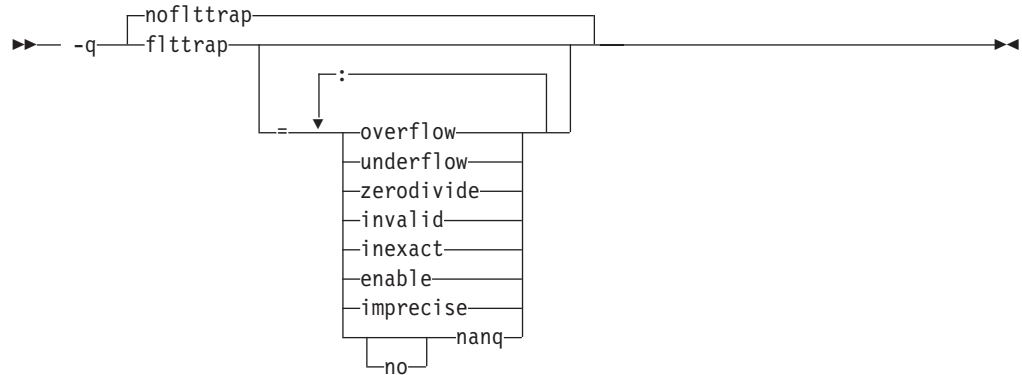
fltttrap

➤ C ➤ C++

目的

追加の命令を生成して、実行時の浮動小数点例外を検出およびトラップします。

構文



ここで、サブオプションでは以下が行われます。

ENable	メインプログラムの開始時点で指定された例外を使用可能にします。 (以下で説明する) nanq を除き、ソース・コードを変更せずに、下記の例外トラッピング・オプションをオンにする場合は、このサブオプションが必要です。
OVerflow	浮動小数点のオーバーフローを検出およびトラップするためのコードを生成します。
UNDeRflow	浮動小数点のアンダーフローを検出およびトラップするためのコードを生成します。
ZEROdivide	ゼロによる浮動小数点除法を検出およびトラップするためのコードを生成します。
INValid	浮動小数点無効演算例外を検出およびトラップするためのコードを生成します。
INEXact	浮動小数点精度低下例外を検出およびトラップするためのコードを生成します。
IMPrecise	指定された例外の不正確な検出のためのコードを生成します。例外が引き起こされた場合は、例外が検出されますが、例外の正確な位置は判別されません。
nanq	浮動小数点演算で処理または生成される NaNQ (Not a Number Quiet) 例外を検出およびトラップするためのコードを生成します。 nanq および nonanq 設定は、 -qnofltttrap 、 -qfltttrap 、または -qfltttrap=enable の影響を受けません。

394 ページの『#pragma options』も参照してください。

注

このオプションは、リンク時に認識されます。 **-qnofltttrap** は、これらの追加の命令を生成する必要がないことを指定します。

サブオプションを指定せずに **-qfllttrap** オプションを指定することは、**-qfllttrap=overflow:underflow:zerodivide:invalid:inexact** を指定することと同等です。例外が自動的に使用可能にされることはなく、全浮動小数点演算が検査されて、正確な例外の位置に関する情報が提供されます。

#pragma options とともに指定する場合、**-qnofllttrap** オプションは、指定する最初のオプションでなければなりません。

プログラムにシグナル型 NaN が含まれる場合は、すべての例外をトラップするために、**-qfllttrap** とともに **-qfloat=nans** を使用してください。

-qfllttrap オプションを **-qoptimize** オプションとともに指定する場合、コンパイラーは、以下の例に示すように振る舞います。

- **-O2** の場合:
 - 1/0 は、**div0** 例外を生成し、結果は無限大になります。
 - 0/0 は、無効演算を生成します。
- **-O3** 以上の場合:
 - 1/0 は、**div0** 例外を生成し、結果は無限大になります。
 - 0/0 は、直前の除法の結果によって乗算されたゼロに戻ります。

例

浮動小数点のオーバーフローとアンダーフロー、および 0 による除法を検出するように myprogram.c をコンパイルするには、以下を入力します。

```
xlc myprogram.c -qfllttrap=overflow:underflow:zerodivide:enable
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

141 ページの『float』

244 ページの『O, optimize』

fold

► C ► C++

目的

浮動小数点定数式をコンパイル時に評価するように指定する。

構文

►► — -q — 

394 ページの『#pragma options』も参照してください。

注

このオプションは廃止されました。新しいアプリケーションでは、**-qfloat=fold** を使用してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

141 ページの『float』

394 ページの『#pragma options』

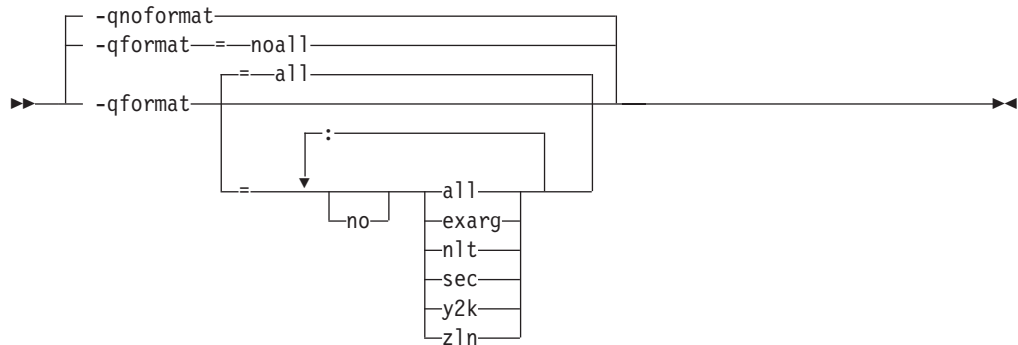
format

► C ► C++

目的

文字列入出力フォーマット指定で起こりうる問題の警告を出します。診断された関数は、`printf`、`scanf`、`strftime`、`strfmon` ファミリー関数、およびフォーマット属性でマークされた関数です。

構文



サブオプションは、以下のとおりです。

- all** すべてのフォーマット診断メッセージをオンにします。
- exarg** 過剰な引き数が `printf` および `scanf` スタイル関数呼び出しに表示された場合に警告を出します。
- nlt** フォーマット関数とそのフォーマット引き数を `va_list` として取らない限り、書式制御文字列が文字列・リテラルではない場合に警告を出します。
- sec** フォーマット関数の使用中に起こりうるセキュリティ問題の警告を出します。
- y2k** 2桁の年数を作成する `strftime` フォーマットの警告を出します。
- zln** 長さがゼロのフォーマットの警告を出します。

注: 上記のサブオプションのいずれかの前に **no** を指定すると、診断メッセージのグループが使用不可になります。例えば、`y2k` 警告の診断メッセージをオフにするには、コマンド行で **-qformat=noy2k** を指定します。

注

コマンド行で **-qformat** が指定されていない と、コンパイラーはデフォルト設定を **-qnoformat** と見なします。これは、**-qformat=noall** と同等です。

サブオプションなしで、コマンド行で **-qformat** が指定されると、コンパイラーはデフォルト設定を **-qformat=all** と見なします。

例

1. すべての書式制御文字列診断を使用可能にするには、以下のいずれかを入力します。

```
xlc myprogram.C -qformat=all
```

```
xlc myprogram.C -qformat
```

2. y2k データ診断検査以外のすべてのフォーマット診断検査を使用可能にするには、以下を入力します。

```
xlc myprogram.C -qformat=all:ny2k
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

fullpath

➤ C ➤ C++

目的

-g コンパイラー・オプションを使用するときに、どのようなパス情報をファイル用に保管するかを指定する。

構文

➤➤ — -q —  —➤➤

注

-qfullpath を使用すると、コンパイラーは、**-g** オプションで指定したソース・ファイルの絶対 (フル) パス名を保持します。

ファイルの相対パス名は、**-qnofullpath** を使用すると保持されます。

-qfullpath は、実行可能ファイルを他のディレクトリーに移動した場合に便利です。**-qnofullpath** を指定すると、デバッガーに検索パスを指定しない限り、デバッガーはファイルを検出することができなくなります。**-qfullpath** を使用すると、ファイルを正常に探すことができます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

155 ページの『g』

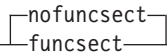
funcsect

► C ► C++

目的

個別のオブジェクト・ファイル、制御セクション (csect) のそれぞれの関数ごとに命令を配置する。デフォルトでは、各オブジェクト・ファイルは、対応するソース・ファイル内に定義されたすべての関数を結合して、単一の制御セクションで構成されます。

構文

►► — -q — 

注

複数の csects を使用すると、オブジェクト・ファイルのサイズが大きくなりますが、リンケージ・エディターが、呼び出されていない関数を除去したり、または呼び出されるすべての場所で最適化プログラムがインラインした関数を除去できるようにすることによって、最終の実行可能ファイルのサイズが削減されることがあります。

ファイルが、初期化された静的データ、または以下のプラグマ・ステートメント

```
#pragma comment (copyright)
```

を含んでいる場合、1 マシン語分大きくなる関数があります。

このオプションが **#pragma options** で指定されているとき、プラグマ・ディレクティブはコンパイル単位最初のステートメントの前で指定しなければなりません。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

329 ページの『twolink』

394 ページの『#pragma options』

G



目的

実行時リンクで利用できる共用オブジェクトを作成するようにリンケージ・エディターに指示します。

構文



注

コンパイラーは、**-bE:**、**-bexport:**、**-bexpall**、または **-bnoexpall** を使用してエクスポートするシンボルを指定しない限り、共用オブジェクトからすべてのグローバル・シンボルを自動的にエクスポートします。

-G を使用して共用ライブラリを作成する場合、コンパイラは次のことを実行します。

1. ユーザーが **-bE:**、**-bexport:**、**-bexpall**、または **-bnoexpall** を指定しない場合は、CreateExportList スクリプトを使用して、すべてのグローバル・シンボルを含むエクスポート・リストを作成します。 **-tE/-B** または **-qpath=E:** オプションを使って別のスクリプトを指定することができます。
2. エクスポート・リストの作成に CreateExportList ユーティリティが使用されていたか、**-qmkshrobj** および **-qexpfile** コンパイラ・オプションが使用されていた場合、エクスポート・リストを保管してください。
3. 適切なオプションおよびオブジェクト・ファイルを使ってリンカーを呼び出し、共用オブジェクトをビルドします。

これは、リンケージ・エディター (**ld**) オプションです。 **ld** コマンドの使用法および構文に関する詳細については、使用中のオペレーティング・システムの資料を参照してください。

CreateExportList ユーティリティおよび共用オブジェクトの作成について詳しくは、「プログラミング・ガイド」の『ライブラリーの作成』を参照してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

93 ページの『B』

94 ページの『b』

97 ページの『brtl』

134 ページの『expfile』

256 ページの『path』

311 ページの『t』

また、Web で以下を参照してください。

「General Programming Concepts: Writing and Debugging Programs」の『Shared Objects and Runtime Linking』の章

「コマンド・リファレンス 第3巻」の『ld コマンド』セクション

244 ページの『O、optimize』
271 ページの『Q』

genproto

► C

目的

K&R 関数定義から ANSI プロトタイプを生成する。これは、K&R から ANSI に簡単に変遷するのに役立ちます。

構文

```
► -q [nogenproto | genproto [==parmnames]] ◀
```

注

parmnames を指定せずに **-qgenproto** を使用すると、プロトタイプはパラメーター名なしで生成されます。**parmnames** を指定すると、パラメーター名がプロトタイプに組み込まれます。

例

以下の関数が `foo.c` にある場合に、

```
foo(a,b,c)
float a;
int *b;
int c;
```

以下を指定すると、

```
xlc -c -qgenproto foo.c
```

以下が生成されます。

```
int foo(double, int*, int);
```

パラメーター名は除去されます。一方、以下を指定すると、

```
xlc -c -qgenproto=parmnames foo.c
```

以下が生成されます。

```
int foo(double a, int* b, int c);
```

この場合、パラメーター名は保持されます。

K&R 関数に渡される前に狭い型の引き数 (**char**、**short**、**float** など) すべてが拡張されることが ANSI で規定されているため、**float a** は、プロトタイプでは **double** または **double a** として示されることに注意してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

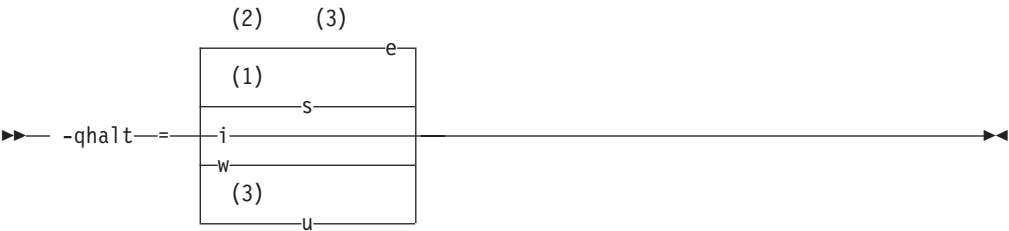
halt

C C++

目的

指定した重大度 以上のエラーが検出された場合に、コンパイル・フェーズ後に停止するようにコンパイラーに指示する。

構文



注:

- 1 C++ コンパイルのデフォルトです。
- 2 C コンパイルのデフォルトです。
- 3 C コンパイルのみ

ここで、重大度が増大する順での重大度レベルは、以下のとおりです。

重大度	説明
i	通知
w	警告
e	エラー (C コンパイルのみ)
s	重大エラー
u	回復不能エラー (C コンパイルのみ)

394 ページの『#pragma options』も参照してください。

注

-qhalt オプションの結果、コンパイラーが停止するときは、コンパイラーの戻りコードはゼロ以外です。

-qhalt を複数回指定した場合は、最低の重大度レベルが使用されます。

-qhalt オプションは、**-qmaxerr** オプションでオーバーライドすることができます。

診断メッセージは、**-qflag** オプションで制御することができます。

例

`myprogram.c` をコンパイルし、警告以上のレベルのメッセージが出された場合にコンパイルを停止させるには、以下を入力します。

```
xlc myprogram.c -qhalt=w
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

139 ページの『flag』

234 ページの『maxerr』

394 ページの『#pragma options』

haltonmsg

► C++

目的

指定した *msg_number* が検出されたときに、コンパイル・フェーズ後に停止するよう、コンパイラーに指示する。

構文

```
►► -qhaltonmsg=msg_number ◀◀
```

注

-qhaltonmsg オプションの結果、コンパイラーが停止するときは、コンパイラーの戻りコードはゼロ以外です。

-qhaltonmsg オプションで複数のメッセージ番号を指定することができます。追加のメッセージ番号はコンマで分離される必要があります。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

457 ページの『コンパイラー・メッセージ』

heapdebug

➤ C ➤ C++

目的

デバッグ・バージョンのメモリー管理関数を使用可能にする。

構文

➤ ➤ -q noheapdebug
heapdebug ➤ ➤

注

デフォルトでは、コンパイラーは、通常のメモリー管理関数 (**calloc**、**malloc**、**new** など) を使用し、それぞれのローカル・ストレージを事前に初期化しません。通常のメモリー管理関数のヘッダー・ファイルは、システムのインクルード・ディレクトリーに入っています。

_debug_calloc、**_debug_malloc**、**new** などのメモリー管理関数のデバッグ・バージョンを宣言するヘッダー・ファイルは、製品のインクルード・ディレクトリー `usr/vacpp/include` に入っています。

-qheapdebug を指定すると、ヘッダー・ファイルの検索順序を変更し、コンパイラーはメモリー管理関数のデバッグ・バージョンを宣言するヘッダー・ファイルを見つけることができます。コンパイラーは、メモリー管理関数のデバッグ・バージョンのヘッダー・ファイルが保管されている製品のインクルード・ディレクトリーでまずヘッダー・ファイルを検索し、その後、システムのインクルード・ディレクトリーで検索します。

-qheapdebug を指定すると、**__DEBUG_ALLOC__** マクロも定義します。

例

デバッグ・バージョンのメモリー管理関数で `myprogram.c` をコンパイルするには、以下を入力します。

```
x1C -qheapdebug myprogram.C -o testing
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

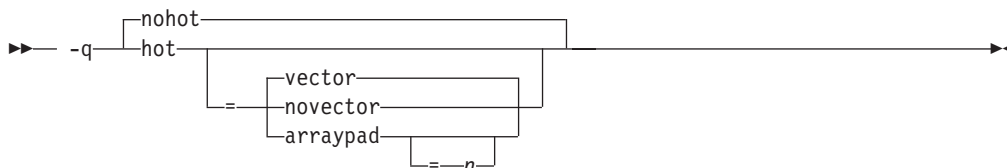
hot

C C++

目的

最適化中に、高位のループ分析と変換を実行するようコンパイラーに指示します。

構文



ここで、

arraypad	コンパイラーは、利点があると考えられる個所に、任意の配列を適宜選択した分量ごとに埋め込みます。すべての配列が埋め込まれる必要はなく、いろいろな異なる配列が異なる分量ごとに埋め込まれる場合があります。
arraypad= <i>n</i>	コンパイラーは、すべての配列をコードに埋め込みます。埋め込みの分量は、正整数の値でなければならず、各配列は、エレメントの整数値で埋め込まれます。 <i>n</i> は整数値であるため、埋め込み値を配列エレメントの最大サイズの倍数 (典型的な値は 4、8、または 16) にすることをお勧めします。
vector novector	コンパイラーは、連続する配列のエレメント (例えば、平方根、逆数平方根) に関してループ内で実行される特定の操作をライブラリー・ルーチンの呼び出しに変換します。この呼び出しは、同時にいくつかの結果を計算します。これは、各結果を逐次計算するよりも高速です。

-qhot=novector を指定すると、コンパイラーは、ループおよび配列について高位の変換を実行しますが、呼び出しで特定のコードをベクトル・ライブラリー・ルーチンに置き換える個所の最適化は行いません。

-qhot=vector オプションは、使用中のプログラムの結果の精度に影響する可能性があるため、精度の変更を受け入れられない場合は、

-qhot=novector または **-qstrict** のいずれかを指定してください。

デフォルト

-qhot=vector サブオプションは、**-qhot**、**-qsmp**、**-O4**、または **-O5** オプションを指定したときに、デフォルトでオンになります。

注

同様に、コマンド行で **-qhot** を指定するときに、最低でもレベル 2 の最適化を指定しなければ、コンパイラーは **-O2** と見なします。

キャッシュ・アーキテクチャーのインプリメンテーションにより、2 の累乗の配列次元はキャッシュの使用効率の低下を招く可能性があります。オプションの **arraypad** サブオプションにより、コンパイラーは、配列の次元を増加できるようになります。そうすることにより、配列処理ループの効率が向上する可能性があります。2 の累乗のいくつかの次元で大規模な配列 (特に最初の配列) がある場合、ま

たは配列処理プログラムがキャッシュ・ミスまたはページ不在によってスローダウンしている場合は、**-qhot=arraypad** を指定することを考慮に入れてください。

vector サブオプションは配列データを最適化し、適用できる場合、並列して数学演算を実行します。コンパイラーは、ベクター・サイズの制限がない標準のレジスターを使用します。**vector** サブオプションは単精度および倍精度の浮動小数点数値をサポートするため、大きな数値処理要求があるアプリケーションでは役立ちます。

-qhot=arraypad および **-qhot=arraypad=*n*** は両方とも安全なオプションではありません。これらは、埋め込みが実行された場合にコードの中断の原因となりうる再シェーピングまたは同等性の検査を実行しません。

例

以下の例は、**-qhot=arraypad** オプションをオンにします。

```
xlc -qhot=arraypad myprogram.c
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

98 ページの『C』

244 ページの『O、optimize』

293 ページの『smp』

hsflt

➤ C ➤ C++

目的

単精度 **float** の結果と浮動小数点から整数への変換とに関する範囲検査を除去することによって、計算を高速化します。**-qnohsflt** は、式の評価後に単精度式を丸め、範囲外の値かどうかについて浮動小数点から整数への変換を検査することを指定します。

構文

➤ — -q  

394 ページの『#pragma options』も参照してください。

注

このオプションは廃止されました。新しいアプリケーションでは、**-qfloat=hsflt** を使用してください。

-qhsflt オプションは、**-qrndsngl** および **-qspnans** オプションをオーバーライドします。

-qhsflt オプションは、浮動小数点計算の特性が既知である特定のアプリケーションを対象としています。その他のアプリケーション・プログラムのコンパイル時にこのオプションを使用すると、誤った結果が生成される可能性があります。警告は出されません。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

141 ページの『float』

300 ページの『spnans』

394 ページの『#pragma options』

hssngl

► C ► C++

目的

結果を **float** のメモリー位置に保管する場合にのみ単精度式を丸めることを指定する。**-qnohssngl** は、式の評価後に単精度式を丸めることを指定します。**-qhssngl** を使用すると、実行時パフォーマンスを向上させることができます。

構文

►► — -q —  —►►

394 ページの『#pragma options』も参照してください。

注

このオプションは廃止されました。新しいアプリケーションでは、**-qfloat=hssngl** を使用してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

141 ページの『float』

394 ページの『#pragma options』

目的

絶対パスを指定しない **#include** ファイル名に追加の検索パスを指定する。

構文

▶— **-I**—*directory*————▶

注

directory の値は、有効なパス名 (**/u/golnaz**、**/tmp**、**./subdir** など) でなければなりません。コンパイラーは *directory* にスラッシュ (**/**) を付加し、検索前にファイル名と連結させます。パス・ディレクトリーは、名前がスラッシュ (**/**) で始まらない **#include** ファイルについて、最初にコンパイラーが検索するディレクトリーです。ディレクトリーを指定しない場合は、デフォルトで、標準のディレクトリーが検索されます。

構成ファイルおよびコマンド行の両方で **-I directory** オプションを指定した場合は、構成ファイルで指定したパスが最初に検索されます。

-I directory オプションは、コマンド行で複数回指定することができます。複数の **-I** オプションを指定した場合は、ディレクトリーは、コマンド行に指定された順序で検索されます。ディレクトリーの検索に関する詳細については、『**相対パス名を使用したインクルード・ファイルのディレクトリー検索順序**』を参照してください。

#include ディレクティブにフル (絶対) パス名を指定した場合、このオプションの影響はありません。

例

myprogram.C をコンパイルし、インクルード・ファイルについて **/usr/tmp** と **/oldstuff/history** を順に検索するには、次のように入力します。

```
xlc myprogram.C -I/usr/tmp -I/oldstuff/history
```

関連タスク

43 ページの『**相対パス名を使用したインクルード・ファイルのディレクトリー検索順序**』

関連資料

61 ページの『**コンパイラーのコマンド行オプション**』

idirfirst

➤ C ➤ C++

目的

#include “file_name” ディレクティブで組み込むファイルの検索順序を指定する。

構文

➡ -q  _____ ➡

394 ページの『#pragma options』も参照してください。

注

-qidirfirst は **-I** オプションと一緒に使用してください。

idirfirst オプションを使用しない 場合に、**#include** “file_name” ディレクティブで組み込まれるファイルの標準の検索順序は、以下のとおりです。

1. 現行のソース・ファイルがあるディレクトリーを検索します。
2. **-I** オプションで指定した 1 つ以上のディレクトリーを検索します。
3. 標準の組み込みディレクトリーは、以下のとおりです。
 - C プログラムの場合は **/usr/include**
 - C++ プログラムの場合は、**/usr/vacpp/include** および **/usr/include**

-qidirfirst を指定すると、**-I** オプションで指定したディレクトリーが検索されてから、現行のファイルがあるディレクトリーが検索されるようになります。

-qidirfirst は、**#include <file_name>** ディレクティブの検索順序には影響を及ぼしません。

-qidirfirst は、**#include “file_name”** および **#include <file_name>** の両方について検索順序を変更する **-qnostdinc** オプションとは無関係です。

ファイルの検索順序については、**相対パス名を使用したインクルード・ファイルのディレクトリー検索順序** で説明しています。

最後の有効な **#pragma options [no]idirfirst** は、それ以後の **#pragma options [no]idirfirst** によって置き換えられるまで、有効なままです。

例

myprogram.c をコンパイルして、インクルード・ファイルについて **/usr/tmp/myinclude** を検索させてから現行ディレクトリー（ソース・ファイルがある）を検索させるには、以下を入力します。

```
xlc myprogram.c -I/usr/tmp/myinclude -qidirfirst
```

関連タスク

43 ページの『相対パス名を使用したインクルード・ファイルのディレクトリー検索順序』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

166 ページの『I』

304 ページの『stdinc』

394 ページの『#pragma options』

ignerrno

➤ C ➤ C++

目的

コンパイラーが、システム呼び出しによって **errno** が変更されないと想定して最適化を行うことを許可する。

構文

➤➤ — -q — noignerrno
ignerrno — ➤➤

394 ページの『#pragma options』も参照してください。

注

例外が発生すると、一部のシステム・ライブラリー・ルーチンは **errno** を設定します。この設定および以後の **errno** の副次作用は、**-qignerrno** を指定することによって無視することができます。

-O3 以上の最適化オプションを指定すると、**-qignerrno** も設定します。最適化と **errno** を設定する機能の両方が必要な場合は、コマンド行で最適化オプションの後に **-qnoignerrno** を指定する必要があります。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

244 ページの『O、optimize』

394 ページの『#pragma options』

ignprag

► C ► C++

目的

特定のプラグマ・ステートメントを無視するようにコンパイラーに指示する。

構文



ここで、このオプションの影響を受けるプラグマ・ステートメントは、以下のとおりです。

disjoint	ソース・ファイルのすべての #pragma disjoint ディレクティブを無視します。
isolated	ソース・ファイルのすべての #pragma isolated_call ディレクティブを無視します。
all	ソース・ファイルのすべての #pragma isolated_call ディレクティブおよび #pragma disjoint ディレクティブを無視します。
ibm	► C #pragma ibm parallel_loop 、 #pragma ibm schedule など、ソース・ファイルのすべての IBM 並列処理ディレクティブを無視します。
omp	#pragma omp parallel 、 #pragma omp critical など、ソース・ファイルのすべての OpenMP 並列処理ディレクティブを無視します。

394 ページの『#pragma options』も参照してください。

注

このオプションは、別名割り当てプラグマのエラーの検出に役立ちます。別名割り当てが誤っていると、診断が困難な実行時エラーが発生します。実行時エラーが発生するものの、**-O** オプションとともに **-qignprag** オプションを使用するとエラーが消失するときは、別名割り当てプラグマで指定した情報が誤っている可能性があります。

例

myprogram.c をコンパイルして、**#pragma isolated_call** ディレクティブをすべて無視させるには、以下を入力します。

```
xlc myprogram.c -qignprag=isolated
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

360 ページの『#pragma disjoint』

379 ページの『#pragma isolated_call』

394 ページの『#pragma options』

422 ページの『並列処理を制御するプラグマ』

info

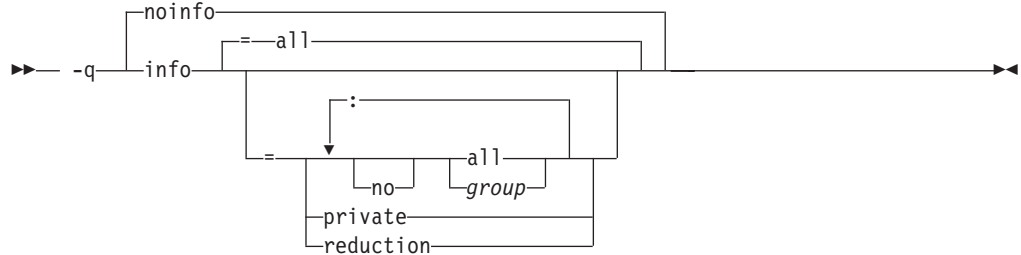
► C ► C++

目的

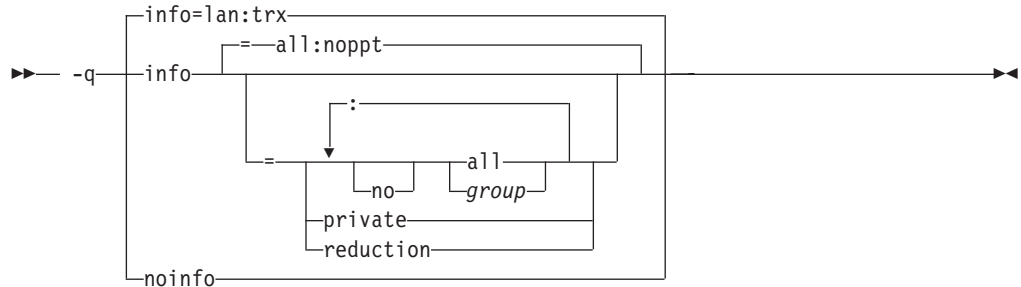
通知メッセージを作成する。

構文

► C



► C++



-qinfo オプションおよび診断メッセージ・グループについては、以下の『注』セクションで説明します。 373 ページの『#pragma info』および 394 ページの『#pragma options』も参照してください。

デフォルト

コマンド行で **-qinfo** を指定しなければ、コンパイラーは以下のことを想定します。

1. ► C **-qnoinfo**
2. ► C++ **-qinfo=lan:trx**

コマンド行で、サブオプションなしで **-qinfo** を指定すると、コンパイラーは以下のことを想定します。

1. ► C **-qinfo=all**
2. ► C++ **-qinfo=all:noppt**

注

サブオプションなしで **-qinfo=all** または **-qinfo** を指定すると、C++ コードの **ppt** (プリプロセッサ・トレース) グループを除き、すべてのグループのすべての診断メッセージがオンになります。


-qnoinfo または **-qinfo=noall** を指定すると、すべてのグループのすべての診断メッセージがオフになります。

このコンパイラー・オプションの **#pragma options info=suboption[:suboption ...]** 形式、または **#pragma options noinfo** 形式を使用して、プログラム・コードの 1 つ以上の特定のセクションにあるメッセージを一時的に使用可能または使用不可にすることができます。

-qinfo オプションの有効な形式は、以下のとおりです。

all すべてのグループのすべての診断メッセージをオンにします。

 **-qinfo** 形式と **-qinfo=all** 形式の効果は同じです。

 オプションの **-qinfo** および **-qinfo=all** 形式は両方とも同じ効果がありますが、**ppt** グループ (プリプロセッサ・トレース) を組み込みません。

lan 言語レベルの影響を通知する診断メッセージを使用可能にします。これは C++ コンパイルのデフォルトです。

noall プログラムの特定の部分について、すべての診断メッセージをオフにします。

private 並列ループに対して **private** になった共用変数をリストします。

reduction 並列ループの中で縮約変数として認識されたすべての変数をリストします。

group

特定のメッセージのグループをオンまたはオフにします。ここで、*group* は以下の 1 つまたは複数です。

group	戻される (抑制される) メッセージのタイプ
c99 noc99	C89 言語レベルと C99 言語レベルの間で異なる動きをする場合がある C コード。
cls nocls	C++ クラス
cmp nocmp	無符号比較における起こりうる冗長度
cnd nocnd	条件式における起こりうる冗長度または問題
cns nocns	定数を含む命令
cnv nocnv	変換
dcl nodcl	宣言の整合性
eff noeff	効果のないステートメントおよびプラグマ
enu noenu	enum 変数の整合性
ext noext	未使用の外部定義
gen nogen	汎用診断メッセージ
gnr nognr	一時変数の生成
got nogot	goto 文の使用
ini noini	初期設定で起こりうる問題
inl noinl	インラインされていない関数
lan nolan	言語レベル効果
obs noobs	旧フィーチャー
ord noord	評価の未指定順序
par nopar	未使用パラメーター
por nopor	移送不能言語構造体
ppc noppc	プリプロセッサの使用で起こりうる問題
ppt noppt	プリプロセッサ・アクションのトレース
pro nopro	欠落関数プロトタイプ
rea norea	到達できないコード
ret noret	戻りステートメントの整合性
trd notrd	データまたは精度の起こりうる切り捨てまたは欠落
tru notru	コンパイラーで切り捨てられた変数名
trx notrx	16 進浮動小数点定数の丸め
uni nouni	未初期化変数
upg noupg	前のリリースと比べて、現在のコンパイラー・リリースの新しい振る舞いを示すメッセージを生成する。
use nouse	未使用の自動および静的変数
vft novft	C++ プログラムで仮想関数テーブルの生成
zea nozea	ゼロ範囲の配列。

例

myprogram.C をコンパイルして、変換および到達しないステートメントを除くすべての項目について通知メッセージを生成させるには、以下を入力します。

```
xlc myprogram.C -qinfo=all -qinfo=nocnv:norea
```

関連概念

19 ページの『並列化ループ内の縮約操作』

19 ページの『並列環境での共用変数および Private 変数』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

158 ページの『halt』

160 ページの『haltonmsg』

308 ページの『suppress』

373 ページの『#pragma info』

394 ページの『#pragma options』

410 ページの『#pragma report』

initauto

► C ► C++

目的

自動変数を 2 桁の 16 進バイト値 *hex_value* に初期化します。

構文

```
►► -q[no]initauto[=hex_value]
```

394 ページの『#pragma options』も参照してください。

注

このオプションは、追加のコードを生成して、自動変数値を初期化します。これによって、プログラムの実行時のパフォーマンスが低下するので、デバッグ用にのみ使用してください。

-qinitauto の初期値にはデフォルト設定がありません。明示的な値を設定しなければなりません (**-qinitauto=FA** など)。

例

myprogram.c をコンパイルして、自動変数を 16 進値 FF (10 進値 255) に初期化させるには、以下を入力します。

```
xlc myprogram.c -qinitauto=FF
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

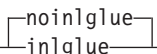

inlglue

► C ► C++

目的

外部関数の呼び出しまたは関数ポインターを介した呼び出しを行うために必要なポインター・グルー・コードをインライン化することによって、高速な外部結合を生成する。

構文

►► -q  

394 ページの『#pragma options』も参照してください。

注

グルー・コード はリンカーによって生成され、 2 つの外部関数の間で制御を渡す目的で使用したり、ポインターを介して関数を呼び出すときに使用します。また、C++ の仮想関数呼び出しをインプリメントするのにも使用します。したがって、**-qinlglue** オプションは、ポインターを介した関数呼び出しまたは外部コンパイル単位の呼び出しにしか影響を及ぼしません。外部関数の呼び出しの場合、**-qprocimported** オプションなどを使用することによって、関数がインポートされることを指定してください。

グルー・コードのインラインによって、コードのサイズが増加する場合があります。これは、**-qcompact** オプションを指定することによってオーバーライドすることができます。オーバーライドすると、**-qinlglue** オプションは無効になります。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

108 ページの『compact』

267 ページの『proclocal、 procimported、 procunknown』

394 ページの『#pragma options』

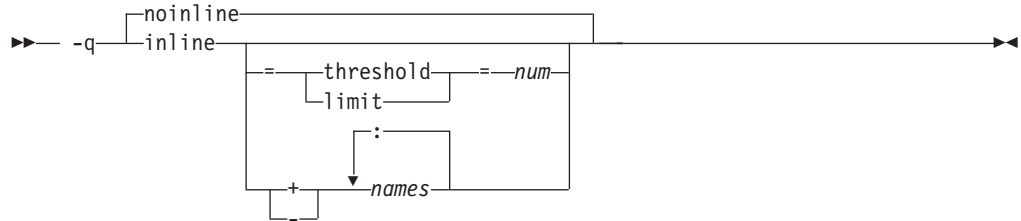
inline

► C ► C++

目的

これらの関数の呼び出しを生成する代わりに、関数のインラインを試みます。インラインは可能であれば実行されますが、実行する最適化によってはインラインされない関数もあります。

構文



► C C 言語では、以下の **-qinline** オプションが適用されます。

-qinline

コンパイラーは、**-qinline** オプションに対するサブオプションのすべての他の設定に従って、実行可能なソース・ステートメントが 20 個以下の適切な関数をすべてインラインしようとしています。**-qinline** が最後に指定された場合は、すべての関数がインラインされます。

-qinline=threshold=num

インラインする関数に対してサイズの制限を設定します。実行可能ステートメントの数は、関数をインラインする場合は、*num* 以下でなければなりません。*num* は正の整数でなければなりません。デフォルト値は 20 です。*threshold* の値に **0** を指定すると、**__inline**、**_Inline**、または **_inline** キーワードでマークされた関数以外の関数はインラインされません。

num 値は、論理 C ステートメントに適用されます。以下の例に示すように、宣言はカウントされません。

```
increment()
{
    int a, b, i;
    for (i=0; i<10; i++) /* statement 1 */
    {
        a=i;             /* statement 2 */
        b=i;             /* statement 3 */
    }
}
```

`-qinline-names`

コンパイラーは、*names* にリストされた関数をインラインしません。 *name* の各関数名は、それぞれコロンの (:) で分離します。他のすべての適切な関数はインラインされます。このオプションは、 **-qinline** を暗黙指定します。

例を以下に示します。

`-qinline-salary:taxes:expenses:benefits`

これによって、**salary**、**taxes**、**expenses**、または **benefits** という名前の関数以外のすべての関数が、可能であればインラインされます。

ソース・ファイルで定義されていない関数については、警告メッセージが出されます。

`-qinline+names`

names にリストされた関数およびすべての他の適切な関数をインラインしようとします。 *name* の各関数名は、コロンの (:) で分離しなければなりません。このオプションは、 **-qinline** を暗黙指定します。

例を以下に示します。

`-qinline+food:clothes:vacation`

これによって、インラインに適格なすべての他の関数とともに、可能な場合は、**food**、**clothes**、または **vacation** という名前の関数がすべてインラインされます。

ソース・ファイルで定義されていない関数、または定義はされているがインラインできない関数については、警告メッセージが出されます。

このサブオプションは、*threshold* 値の設定をすべてオーバーライドします。 **-qinline+names** と一緒に *threshold* 値にゼロを使用して、特定の関数をインラインすることができます。

例を以下に示します。

`-qinline=threshold=0`

これに以下を続けて指定します。

`-qinline+salary:taxes:benefits`

これによって、**salary**、**taxes**、または **benefits** という名前の関数のみが、可能な場合にインラインされ、それ以外はインラインされません。

`-qinline=limit=num`

インラインによる関数サイズの増加を許容する最大サイズを (生成されるコードに対するバイト数で) 指定します。この制限は、ユーザーが指定した関数のインラインには影響を及ぼしません。

`-qnoinline`

関数を一切インラインしません。 **-qnoinline** が最後に指定された場合は、関数は一切インラインされません。

 以下の **-qinline** オプションが C++ 言語に適用されます。

<code>-qinline</code>	コンパイラーは、可能な関数をすべてインラインします。
<code>-qnoinline</code>	コンパイラーは関数を一切インラインしません。

デフォルト

デフォルトでは、インライン指定はコンパイラーに対する手掛かりとして扱われます。結果は、ユーザーが選択した他のオプションに応じて以下ようになります。

- **-O** コンパイラー・オプションの 1 つを使用してプログラムを最適化する場合、コンパイラーは、インラインとして宣言されているすべての関数のインライン化を試みます。そうでない場合、コンパイラーは、インラインとして宣言されている比較的簡単な関数宣言のいくつかのみをインライン化しようとします。

注

-qinline オプションは、**-Q** オプションと機能的に同等です。

-g オプションを (デバッグ情報を生成させるために) 指定した場合、インライン化は影響を受けることがあります。 **-g** コンパイラー・オプションの情報を参照してください。

インラインによって必ずしも実行時間が改善されるとは限らないため、コードに対するこのオプションの効果はユーザー自身がテストしなければなりません。再帰的な関数または相互に再帰的な関数はインラインしないでください。

通常、最適化を要求する (**-O** オプション) と、アプリケーションのパフォーマンスが最適化され、最適化を要求しないとコンパイラーのパフォーマンスが最適化されます。

インライン化を最大化する場合は、最適化 (**-O**) に加え、適切な **-qinline** オプションを指定します。

XL C/C++ Enterprise Edition (**inline**、**_inline**、**_Inline**、および **__inline**) の C 言語キーワードは、**-qnoinline** 以外の **-qinline** オプションをすべてオーバーライドします。コンパイラーは、その他の **-qinline** オプションの設定に関係なく、これらのキーワードでマークされた関数をインラインしようとします。

inline、**_Inline**、**_inline**、および **__inline** 関数指定子: コンパイラーでは、コンパイラーにインラインさせたい関数を指定するために使用できるキーワード・セットを提供します。機能的に同等なキーワードは、以下のとおりです。

- **inline** (C99 および C++ のみ)
- **_Inline** (C のみ)
- **_inline** (C のみ)
- **__inline** (C および C++)

例を以下に示します。

```
_Inline int catherine(int a);
```

関数 **catherine** をインラインすることをコンパイラーに指示します。これは、関数呼び出しではなく関数に対してコードが生成されることを示します。

データとともにインライン指定子を使用する、または **main()** 関数を宣言すると、エラーが生成されます。

デフォルトでは、関数のインラインはオフであり、インライン指定子で修飾した関数は、単に `extern` 関数として扱われます。関数のインラインをオンにするには、**-qinline** または **-Q** コンパイラー・オプションのいずれかを指定します。**-O** または **-qoptimize** コンパイラー・オプションを指定して、最適化をオンにすると、インライン化もオンになります。

再帰的関数 (その関数そのものを呼び出す関数) は、最初のカレンスでのみインラインされます。関数内部からのその関数への呼び出しはインラインされません。

指定したサイズよりも小さい関数すべてを自動的にインラインするには、**-qinline** または **-Q** コンパイラー・オプションを指定することもできます。しかし、最高のパフォーマンスを得るには、自動インラインを使用するのではなく、インライン・キーワードを使ってインラインしたい関数を選ぶようにしてください。

インライン関数は、宣言と定義を同時に行うことができます。インライン・キーワードの 1 つを指定してインライン関数を宣言する場合、インライン・キーワードなしで定義することができます。以下のコードの断片では、インライン関数定義を示します。インライン・キーワードは関数定義および関数宣言の両方で指定されることに注意してください。

```
_inline int add(int i, int j) { return i + j; }

inline double fahr(double t);
```

注: インライン指定子の使用は関数の意味を変更しませんが、関数のインライン拡張は、実引き数の計算の順序を保存しないことがあります。

例

`myprogram.C` をコンパイルし、関数を一切インラインしないようにするには、以下を入力します。

```
xlc myprogram.C -O -qnoinline
```

`myprogram.c` をコンパイルして、12 行未満の関数のインラインをコンパイラーに試行させるには、以下を入力します。

```
xlc myprogram.c -O -qinline=12
```

関連資料

- 61 ページの『コンパイラーのコマンド行オプション』
- 155 ページの『g』
- 244 ページの『O、optimize』
- 271 ページの『Q』

ipa

➤ C ➤ C++

目的

プロシージャ間分析 (IPA) と呼ぶクラスの最適化をオンにしたりカスタマイズしたりする。

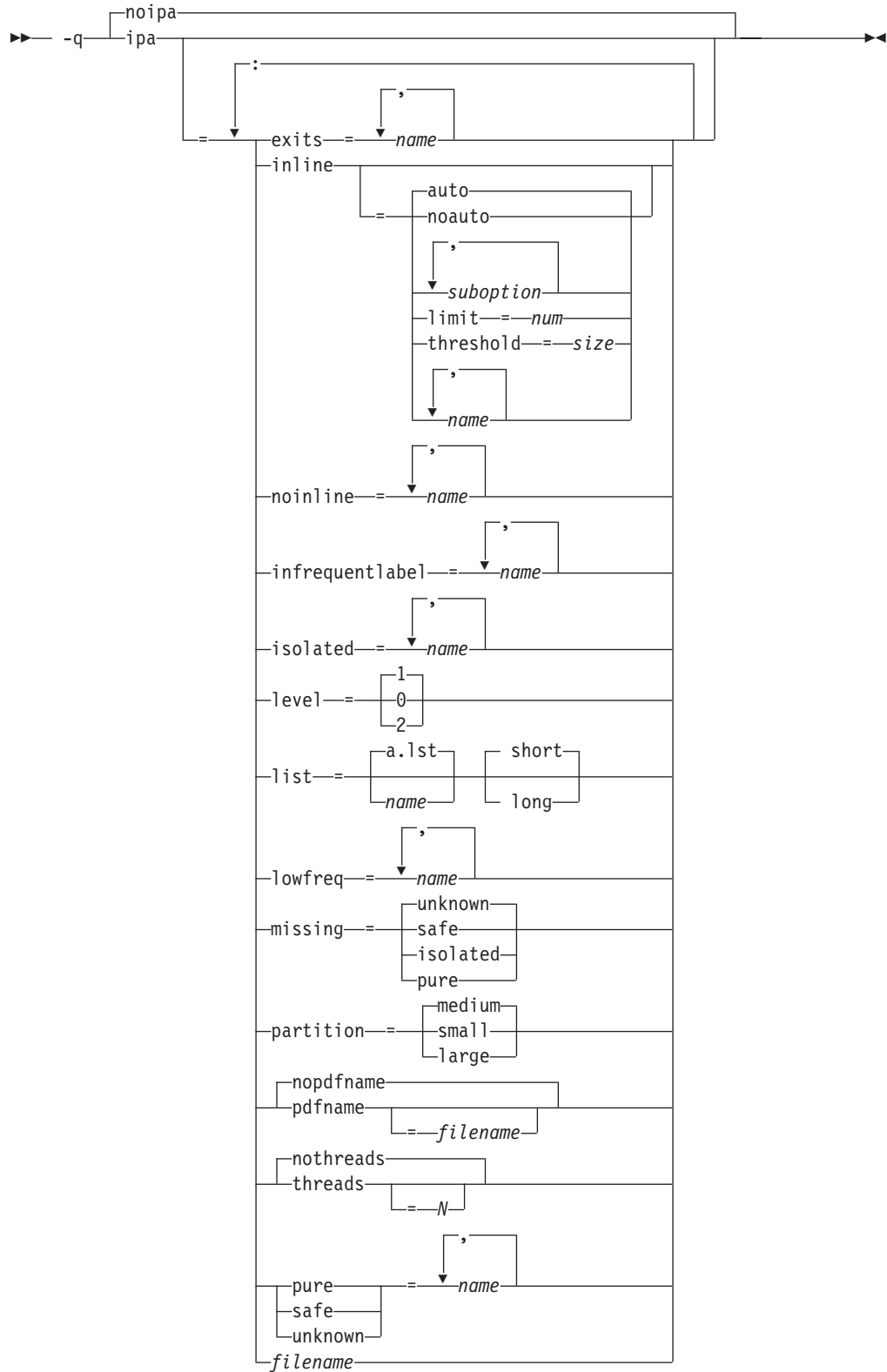
コンパイル時の構文



ここで、

-qipa の コンパイル時の オプション	説明
-qipa	以下の -qipa suboption のデフォルトで、プロシージャ間分析を活動化します。 <ul style="list-style-type: none">• inline=auto• level=1• missing=unknown• partition=medium
-qipa=object -qipa=noobject	標準のオブジェクト・コードをオブジェクト・ファイルに入れるかどうかを指定します。 noobject サブオプションを指定すると、最初の IPA フェーズ中にオブジェクト・コードが生成されないため、全体のコンパイル時間を大幅に短縮することができます。 -S コンパイラー・オプションを noobject と一緒に指定した場合は、 noobject は無視されます。 コンパイルおよびリンクを同じステップで実行し、 -S もリスト・オプションもまったく指定していない場合は、デフォルトで -qipa=noobject が暗黙指定されます。 -qipa でリンクする際に使用されるオブジェクト・ファイルのいずれかが -qipa=noobject オプションで作成された場合は、エントリー・ポイント (実行可能プログラムのメインプログラム、またはライブラリー用にエクスポートされた任意の関数) を含むファイルをすべて -qipa でコンパイルしなければなりません。

リンク時の構文



ここで、

-qipa のリンク時のオプション	説明
-qnoipa	プロシージャーク間分析を非活動化します。
-qipa	以下の -qipa suboption のデフォルトで、プロシージャーク間分析を活動化します。 <ul style="list-style-type: none">• inline=auto• level=1• missing=unknown• partition=medium

以下に示す 1 つ以上の形式をサブオプションに含めることもできます。複数のサブオプションはコンマで区切ります。

リンク時のサブオプション	説明
exits=name{,name}	プログラム出口を表す関数の名前を指定します。プログラム出口は、戻ることができず、IPA パス 1 でコンパイルされたプロシージャークを呼び出すこともできない呼び出しです。
inline=auto inline=noauto	自動インラインだけを有効にしたり無効にしたりします。コンパイラーは、ユーザーが指定した関数をインラインの候補として引き続き受け入れます。
inline[=suboption]	-qinline コンパイラー・オプションの指定と同じです。 <i>suboption</i> は、任意の有効な -qinline サブオプションです。
inline=limit=num	実行するインライン展開の量を判別するために -Q オプションが使用するサイズの制限を変更します。ここで設定する制限は、呼び出し側プロシージャークのサイズの上限です。 <i>num</i> は、最適化プログラムの概算による、生成されるコードのバイト数です。この数値を大きくすると、コンパイラーは、より大きいサブプログラム、より多くのサブプログラム呼び出し、またはその両方をインラインできるようになります。この引き数は、 inline=auto がオンになっている場合にのみインプリメントされます。
inline=threshold=size	インラインする関数のサイズの上限を指定します。ここで、 <i>size</i> は inline=limit で定義された値です。この引き数は、 inline=auto がオンになっている場合にのみインプリメントされます。
inline=name{,name}	インラインしようとする関数をコンマで区切ったリストを指定します。関数は <i>name</i> によって識別されます。
noinline=name{,name}	インラインしなければならない関数をコンマで区切ったリストを指定します。関数は <i>name</i> によって識別されます。
infrequentlabel=name{,name}	プログラムの通常の実行では分岐される頻度の低いユーザー定義のラベルのリストを指定します。

リンク時の サブオプション	説明
<code>isolated=name,{name}</code>	IPA でコンパイルされない分離された 関数のリストを指定します。分離された関数やそれらの呼び出しチェーン内の関数は、いずれもグローバル変数を参照することができません。
<code>level=0</code> <code>level=1</code> <code>level=2</code>	<p>プロシージャーク間分析の最適化レベルを指定します。デフォルトのレベルは 1 です。有効なレベルは以下のとおりです。</p> <ul style="list-style-type: none"> • Level 0 - 最小限のプロシージャーク間分析および最適化しか行いません。 • Level 1 - インライン、限定された別名分析、および限定された呼び出し位置調整をオンにします。 • Level 2 - 完全なプロシージャーク間のデータの流れおよび別名の分析を実行します。
<code>list</code> <code>list=[name] [short long]</code>	<p>リンク・フェーズ中にリスト・ファイルを生成するように指定します。リスト・ファイルには、IPA によって実行される変換および分析をはじめ、区画ごとにバックエンドによって生成されるオプションのオブジェクト・リストに関する情報が入ります。このオプションを使用して、リスト・ファイルの名前を指定することもできます。</p> <p>(-qlist か -qipa=list オプションのいずれかを使用して) リストを要求した場合に <i>name</i> が指定されていない場合は、リスト・ファイルの名前はデフォルトで a.lst になります。</p> <p>long および short サブオプションを使用して、リスト・ファイルの詳細化や簡略化を要求することができます。short サブオプションはデフォルトであり、リストの Object File Map、Source File Map、および Global Symbols Map セクションが生成されます。long サブオプションでは、short サブオプションで生成されるすべてのセクションに加えて、Object Resolution Warnings、Object Reference Map、Inliner Report、および Partition Map セクションが生成されます。</p>
<code>lowfreq=name,{name}</code>	頻繁に呼び出されないと考えられる関数の名前を指定します。一般には、エラー処理、トレース、または初期化の関数です。これらの関数の呼び出しの最適化を小規模にすることによって、コンパイラーが、プログラムのその他の部分がより高速に実行されるようにできる場合があります。

リンク時のサブオプション	説明
missing=attribute	<p>-qipa でコンパイルされず、 unknown、safe、isolated、および pure サブオプションのいずれによっても明示的に指定されていないプロシージャースの間の動きを指定します。</p> <p>以下の属性を使用して、この情報の詳細を指定することができます。</p> <ul style="list-style-type: none"> • safe - 直接呼び出し関数ポインターのいずれによっても可視の (欠落していない) 関数を間接的に呼び出さない関数。 • isolated - 可視の関数からアクセスできるグローバル変数を直接参照しない関数。共用ライブラリーからバインドされた関数は分離された (<i>isolated</i>) ものとな見なされます。 • pure - 安全 (<i>safe</i>) かつ分離された (<i>isolated</i>) 関数であり、可視の関数からアクセスできるストレージを間接的に変更しない関数。また、純粋な (<i>pure</i>) 関数には、取得できる内部状態はありません。 • unknown - デフォルトの設定。このオプションは、不明の (<i>unknown</i>) 関数の呼び出しに対するプロシージャース間の最適化の量を大幅に制限します。欠落している関数が安全 (<i>safe</i>)、分離された (<i>isolated</i>)、または純粋な (<i>pure</i>) 関数として認識されないように指定します。
partition=small partition=medium partition=large	<p>パス 2 中に IPA によって作成される各プログラム区画のサイズを指定します。</p>
nopdfname pdfname pdfname=filename	<p>PDF プロファイル情報を含むプロファイル・データ・ファイルの名前を指定します。 <i>filename</i> を指定しない場合、デフォルト・ファイル名は ._pdf になります。</p> <p>プロファイルは、現在の作業ディレクトリーか、PDFDIR 環境変数によって指定されたディレクトリーに配置されます。これにより、同じ PDFDIR を使用して複数の実行可能ファイルを同時に実行することが可能になります。これは、動的ライブラリー上の PDF でのチューニングに役立ちます。</p>

リンク時の サブオプション	説明
nothreads threads threads= <i>N</i>	<p>コンパイラーがコード生成に割り当てるスレッド数を指定します。</p> <p>nothreads を指定することは、1 つのシリアル処理を実行することと同等です。これはデフォルトです。</p> <p>threads を指定すると、コンパイラーが、使用可能なプロセッサの数に応じて、使用するスレッドの数を判別できるようになります。</p> <p>threads=<i>N</i> を指定すると、<i>N</i> スレッドを使用するようプログラムに指示します。<i>N</i> は、1 ～ MAXINT の範囲の任意の整数値にすることができますが、効率上、ユーザーのシステム上で使用可能なプロセッサの数に限定されます。</p>
pure= <i>name</i> {, <i>name</i> }	<p>-qipa でコンパイルされない純粋な関数のリストを指定します。純粋として指定された関数は、すべて分離された および安全な 関数でなければならず、内部状態を変更したり、副次作用を持ったりしてはなりません。ここで、副次作用は、呼び出し元に対して可視のデータのいずれかを変更する可能性があることと定義します。</p>
safe= <i>name</i> {, <i>name</i> }	<p>-qipa でコンパイルされず、プログラムのほかのパートを呼び出さない 安全な 関数のリストを指定します。安全な関数は、グローバル変数を変更することができますが、-qipa でコンパイルされた関数を呼び出すことはできません。</p>
unknown= <i>name</i> {, <i>name</i> }	<p>-qipa でコンパイルされない不明の 関数のリストを指定します。不明として指定したすべての関数は、-qipa でコンパイルされるプログラムの他の部分を呼び出したり、グローバル変数およびダミー引き数を変更したりすることができます。</p>

リンク時のサブオプション	説明
<i>filename</i>	<p>特別な形式のサブオプション情報を含むファイルの名前を指定します。</p> <p>ファイルの形式は以下のとおりです。</p> <pre># ... comment attribute{, attribute} = name{, name} missing = attribute{, attribute} exits = name{, name} lowfreq = name{, name} inline [= auto = noauto] inline = name{, name} [from name{, name}] inline-threshold = unsigned_int inline-limit = unsigned_int list [= file-name short long] noinline noinline = name{, name} [from name{, name}] level = 0 1 2 prof [= file-name] noprof partition = small medium large unsigned_int</pre> <p>ここで、<i>attribute</i> は以下のいずれかです。</p> <ul style="list-style-type: none"> • exits • lowfreq • unknown • safe • isolated • pure

注

このオプションは、プロシージャーク間分析 (IPA) と認識されているクラスの最適化をオンにしたりカスタマイズしたりします。

- IPA は、**-qipa=noobject** オプションを指定した場合でもコンパイル時間を大幅に増加させる場合があるので、IPA の使用は、開発の最終的なパフォーマンス調整フェーズに限ってください。
- **-qipa** オプションは、アプリケーション全体、あるいはそのできるだけ多くの部分の、コンパイルおよびリンク・ステップで指定してください。少なくとも、**main** を含むファイル、またはライブラリーをコンパイルしている場合はエントリー・ポイントの少なくとも 1 つをコンパイルしなければなりません。
- IPA によるプロシージャーク間の最適化によって、プログラムのパフォーマンスを大幅に向上させることができますが、以前は誤っていても機能していたプログラムが失敗する可能性があります。以下のリストに、積極的な最適化を行わなくても偶然に機能するが、IPA で判明するプログラミングの例をいくつか示します。
 1. 割り振り順序または自動変数の位置に依存するもの。例えば、自動変数のアドレスを取得してから別のローカル変数のアドレスと後で比較して、スタックの拡大方向を判別するものです。C 言語では、自動変数が割り振られる位置や、他の自動変数に対する相対位置は保証されていません。そのような関数を IPA でコンパイルしてはなりません (機能しません)。

2. 無効なポインターのアクセス、または配列の境界を超えた位置のアクセス。

IPA はグローバル・データ構造を再編成する場合があります。参照先が不正なポインターによって未使用のメモリーを以前に変更した可能性がある場合は、ユーザーが割り振ったストレージを破壊する場合があります。

- IPA でコンパイルするためのリソースが十分にあることを確認してください。
IPA は、従来のコンパイルよりもかなり大きなオブジェクト・ファイルを生成する場合があります。このため、これらの中間ファイルを保持するために使用する一時保管場所 (慣習的に `/tmp`) が小さすぎる場合があります。大規模なアプリケーションをコンパイルしている場合は、`TMPDIR` 環境変数による一時記憶域の宛先変更を考慮してください。
- IPA を実行するために十分な仮想メモリー・スペースが存在することを確認してください。例えば、小さなプログラムでは 200MB で十分な場合がありますが、複雑で大規模なアプリケーションでは、正常に IPA でリンクするためには数ギガバイトの仮想メモリーが必要になる場合があります。ない場合は、オペレーティング・システムによってシグナル 9 で IPA が強制終了されます。これをトラップすることはできず、IPA はその一時ファイルをクリーンアップすることができません。
- 異なるリリースのコンパイラーで作成されたオブジェクトをリンクすることはできますが、少なくとも、リンクするオブジェクトの作成に使用した新しい方のコンパイラーと同じリリース・レベルのリンカーを使用しなければなりません。
- ソース・コードに明らかに参照されている、または設定されているいくつかのシンボルは、IPA によって最適化される場合があります、デバッグ、`nm`、またはダンプ出力に対して失われる場合があります。 **-g** コンパイラーとともに IPA を使用すると、その結果、通常、非ステップ可能出力となります。

次のサブオプションで *name* を指定するとき、正規表現構文を使用できます。

- `exits`
- `inline`、`noinline`
- `isolated`
- `lowfreq`
- `pure`
- `safe`
- `unknown`

正規식을指定するための構文規則については以下で説明します。

式	説明
<i>string</i>	<i>string</i> で指定された任意の文字と一致します。例えば、 <i>test</i> は、 <i>testimony</i> 、 <i>latest</i> 、および <i>intestine</i> と一致します。
^ <i>string</i>	<i>string</i> で指定されたパターンが行の先頭にある場合にのみ、そのパターンと一致します。
<i>string</i> \$	<i>string</i> で指定されたパターンが行の終わりにある場合にのみ、そのパターンと一致します。
<i>str.i</i> <i>ng</i>	ピリオド (.) は任意の 1 文字と一致します。例えば、 <i>t.st</i> は、 <i>test</i> 、 <i>tast</i> 、 <i>tZst</i> 、および <i>tlst</i> と一致します。
<i>string</i> \ <i>special_char</i>	円記号 (¥) は、特殊文字をエスケープするために使用できます。例えば、ピリオドで終わる行を検索したい場合、式 <i>.\$</i> を指定するだけで、任意の文字を少なくとも 1 つ含むすべての行が表示されます。 <i>¥.\$</i> を指定すると、ピリオド (.) がエスケープされ、突き合わせでは通常の文字として扱われます。
[<i>string</i>]	<i>string</i> で指定された任意の文字と一致します。例えば、 <i>t[a-g123]st</i> は、 <i>tast</i> および <i>test</i> と一致しますが、 <i>t-st</i> または <i>tAst</i> とは一致しません。
[^ <i>string</i>]	<i>string</i> で指定されたどの文字とも突き合わせを行いません。例えば、 <i>t[^a-zA-Z]st</i> は、 <i>tlst</i> 、 <i>t-st</i> 、および <i>t,st</i> と一致しますが、 <i>test</i> または <i>tYst</i> とは一致しません。
<i>string</i> *	<i>string</i> で指定されたパターンの 0 回以上のオカレンスと一致します。例えば、 <i>te*st</i> は、 <i>tst</i> 、 <i>test</i> 、および <i>teeeeeest</i> と一致します。
<i>string</i> +	<i>string</i> で指定されたパターンの 1 回以上のオカレンスと一致します。例えば、 <i>t(es)+t</i> は、 <i>test</i> および <i>tesest</i> と一致しますが、 <i>tt</i> とは一致しません。
<i>string</i> ?	string で指定されたパターンの 0 または 1 回のオカレンスと一致します。例えば、 <i>te?st</i> は、 <i>tst</i> または <i>test</i> と一致します。
<i>string</i> { <i>m,n</i> }	<i>string</i> で指定されたパターンの、 <i>m</i> ~ <i>n</i> 回のオカレンスと一致します。例えば、 <i>a{2}</i> は <i>aa</i> と一致し、 <i>b{1,4}</i> は <i>b</i> 、 <i>bb</i> 、 <i>bbb</i> 、および <i>bbbb</i> と一致します。
<i>string</i> 1 <i>string</i> 2	<i>string1</i> または <i>string2</i> のいずれかで指定されたパターンと一致します。例えば、 <i>s o</i> は文字 <i>s</i> と <i>o</i> の両方と一致します。

IPA を使用するために必要なステップは、以下のとおりです。

1. IPA 分析は、コンパイルおよびリンクの時間を増加させる 2 パス・メカニズムを使用するため、予備パフォーマンス分析および調整を行ってから **-qipa** オプションでコンパイルします。 **-qipa=noobject** オプションを使用することによって、コンパイルおよびリンクのオーバーヘッドをいくらか削減することができます。
2. **-qipa** オプションは、アプリケーション全体のコンパイルとリンクの両方のステップ、またはできるだけ多くのステップにおいて指定します。サブオプションを使用して、**-qipa** でコンパイルされない プログラムの部分に関する前提を示します。コンパイル中に、コンパイラーは、プロシーチャー間分析情報を **.o** ファ

イルに保管します。リンク中には、 **-qipa** オプションによって、アプリケーション全体の完全な再コンパイルが行われます。

注: コンパイル中に重大エラーが起こった場合は、 **-qipa** は RC=1 を戻して終了します。パフォーマンス分析も終了します。

例

ファイルのセットをプロシージャ間分析でコンパイルするには、以下を入力します。

```
xlc -c -O3 *.C -qipa
xlc -o product *.o -qipa
```

同じファイルのセットをコンパイルして、 2 番目のコンパイルの最適化および最初のコンパイル・ステップの速度を改善する方法を以下に示します。ほとんど実行されることのない 2 つの関数 *trace_error* および *debug_dump* が存在するとしています。

```
xlc -c -O3 *.C -qipa=noobject
xlc -c *.o -qipa=lowfreq=trace_error,debug_dump
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

177 ページの『inline』

219 ページの『libansi』

221 ページの『list』

257 ページの『pdf1、pdf2』

285 ページの『S』

isolated_call

► C ► C++

目的

ソース・ファイル内の副次作用がない関数を指定する。

構文

```
►— -q-isolated_call== — function_name —►
```

ここで、

function_name 副次作用がない関数 (ポインターまたは参照パラメーターが指す変数の値を変更する場合は除く)、あるいは副次作用がある関数や処理に依存しない関数の名前です。

副次作用 とは、実行時環境の状態の変更です。このような変更の例としては、揮発オブジェクトへのアクセス、外部オブジェクトの変更、ファイルの変更、またはこれらのいずれかを行う別の関数の呼び出しなどが挙げられます。副次作用のない関数は、外部変数および静的変数を変更しません。

function_name には、関数をコロン (:) で区切ってリストすることができます。

379 ページの『`#pragma isolated_call`』および 394 ページの『`#pragma options`』も参照してください。

注

関数に分離としてマークを付けた場合は、次のことを最適化プログラムに示すことによって、最適化されたコードの実行時のパフォーマンスを向上させることができます。

- 外部変数および静的変数が呼び出し先関数によって変更されていない
- ループ不変パラメーター付きの関数呼び出しをループの外に移動できる
- 同じパラメーターでの複数の関数呼び出しを 1 つの呼び出しにまとめることができる
- 結果値が必要ない場合には、関数呼び出しを廃棄できる

`#pragma options isolated_call` ディレクティブは、ファイルの先頭で、最初の C または C++ ステートメントの前に指定しなければなりません。 **`#pragma isolated_call`** ディレクティブは、ソース・ファイルの任意の位置で使用することができます。

例

`myprogram.c` をコンパイルして、関数 `myfunction(int)` および `classfunction(double)` に副次作用がないことを指定するには、以下を入力します。

```
xlc myprogram.c -qisolated_call=myfunction:classfunction
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

379 ページの『`#pragma isolated_call`』

394 ページの『`#pragma options`』

keepinline

► C++

目的

参照されていない `extern` インライン関数の定義を保持または廃棄するようにコンパイラーに指示する。

構文

►► `-q` nokeepinline
keepinline ◄◄

注

デフォルトの **-qnokeepinline** 設定は、参照されていない `extern` インライン関数の定義を廃棄するようコンパイラーに指示します。これによって、オブジェクト・ファイルのサイズを削減することができます。

-qkeepinline 設定は、参照されていない `extern` インライン関数の定義を保持します。この設定は、v5.0.2.1 更新レベルより前の VisualAge C++ コンパイラーと同じ動きを提供し、共用ライブラリーと、旧リリースのコンパイラーで作成されたオブジェクト・ファイルとの互換性を保つようにします。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

177 ページの『`inline`』

179 ページの『`inline`、`_Inline`、`_inline`、および `__inline` 関数指定子』

keepparm

C C++

目的

アプリケーションが最適化される場合でも関数仮パラメーターがスタックに保管されているか確認します。

構文

Diagram illustrating the merge sort algorithm. The array is divided into two halves, each of size 4. The left half is further divided into two halves of size 2, and the right half is also divided into two halves of size 2. The process continues until the array is fully sorted.

注

関数は通常、エントリー・ポイントでスタックにその着信パラメーターを保管します。ただし、コードを使用可能な最適化オプションでコンパイルすると、コンパイラーがスタックからこれらのパラメーターを除去することで最適化に利点があると見なす場合、これらのパラメーターを除去することがあります。

-qkeepparm を指定すると、最適化時であっても確実にパラメーターをスタックに保管します。このコンパイラー・オプションは、スタックにこれらの値を保持することによって、着信パラメーターの値をデバッガーなどのツールに確実に使用可能にします。ただし、そうすることで、アプリケーション・パフォーマンスに悪影響が及ぶことがあります。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

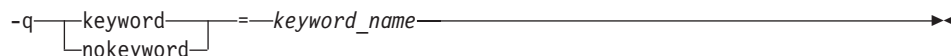
keyword

► C ► C++

目的

このオプションは、指定された名前がプログラム・ソースに現れるたびに、キーワードとして処理するのか、ID として処理するのかを制御します。

構文

►► -q =keyword_name

注

デフォルトでは、C および C++ 言語標準に定義されている組み込みキーワードはすべてキーワードとして予約されています。このオプションを使用しても、キーワードを言語に追加することはできません。しかし、**-qnokeyword=keyword_name** を使用して、組み込みキーワードを使用不可にし、**-qkeyword=keyword_name** を使用して、これらのキーワードを復元することができます。

このオプションは、すべての C++ 組み込みキーワードで使用できます。

このオプションは、また、以下の C 組み込みキーワードで使用することができます。

- asm
- __complex__ (C99)
- __imag__ (C99)
- inline
- __real__ (C99)
- restrict
- typeof

例

以下を起動することにより、bool を復元することができます。

```
xlc -qkeyword=bool
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

L

► C ► C++

目的

-lkey オプションによって指定したライブラリー・ファイルのパス・ディレクトリーを検索する。

構文

►► **-L***directory*◄◄

デフォルト

デフォルトでは、標準のディレクトリーしか検索されません。

注

LIBPATH 環境変数が設定されている場合、コンパイラーは **LIBPATH** が指定したディレクトリー・パスでまずライブラリーを検索し、それから **-L** コンパイラー・オプションが指定したディレクトリー・パスで検索します。

構成ファイルとコマンド行の両方で **-Ldirectory** オプションを指定した場合は、構成ファイルで指定した検索パスが最初に検索されます。

例

myprogram.c をコンパイルして、ライブラリー **libspfiles.a** についてディレクトリー **/usr/tmp/old** および **-l** オプションで指定したすべてのディレクトリーを検索させるには、次のように入力します。

```
xlc myprogram.c -lspfiles -L/usr/tmp/old
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』
197 ページの『1』

➤ C ➤ C++

目的

ダイナミック・リンクの場合は指定したライブラリー・ファイル `libkey.so` を検索してから `libkey.a` を検索し、静的リンクの場合は単に `libkey.a` を検索する。

構文

➤— `-lkey`—➤

デフォルト

コンパイラーのデフォルトでは、いくつかのコンパイラー・ランタイム・ライブラリーのみが検索されます。デフォルトの構成ファイルは、**-l** コンパイラー・オプションで検索するためにデフォルトのライブラリー名を指定し、**-L** コンパイラー・オプションでデフォルトのライブラリー用検索パスを指定します。

注

また、デフォルトの検索パスにない追加のライブラリー用検索パス情報を提供する必要があります。検索パスは、**-Ldirectory** または **-Z** オプションで変更することができます。(静的リンクまたはダイナミック・リンクの場合に) 検索されるライブラリーのタイプの指定に関する詳細については、**-B**、**-brtl**、および **-bstatic**、**-bdynamic** を参照してください。

C および C++ 実行時ライブラリーは自動的に追加されます。

-l オプションは、後から認識されたものが累積されます。コマンド行で後に指定された **-l** オプションは、前の **-l** によって指定されたライブラリーのリストを置換するのではなく、そのリストへの追加を行います。ライブラリーはコマンド行に表示される順序で検索されるため、ライブラリーを指定する順序は、アプリケーション内のシンボル・レゾリューションに影響を与える可能性があります。

詳しくは、オペレーティング・システムの **ld** に関する資料を参照してください。

例

`myprogram.C` をコンパイルし、`/usr/mylibdir` ディレクトリーにあるライブラリー `mylibrary` (`libmylibrary.a`) とリンクするには、以下を入力します。

```
xlc myprogram.C -lmylibrary -L/usr/mylibdir
```

関連タスク

36 ページの『構成ファイル内のコンパイラー・オプションの指定』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

93 ページの『B』

94 ページの『b』

97 ページの『brtl』

196 ページの『L』

348 ページの『Z』

langlvl

► C ► C++

目的

コンパイル用の言語レベルと言語オプションを選択します。

構文

►► -q—langlvl—=  *suboption* ►►

ここで、*suboption* については、以下の『注』セクションで説明します。

381 ページの『#pragma langlvl』および 394 ページの『#pragma options』も参照してください。

デフォルト

デフォルトの言語レベルは、コンパイラーの呼び出しに使用するコマンドに応じてそれぞれ異なります。

呼び出し	デフォルトの言語レベル
xlC/xlc++	extended
xlC	extc89
cc	extended
c89	stdc89
c99	stdc99

注

► **C** 以下のプラグマ・ディレクティブのいずれかを使用して、C 言語ソース・プログラムで言語レベルを指定することもできます。

```
#pragma options langlvl=suboption
#pragma langlvl(suboption)
```

pragma ディレクティブは、ソース・コードのどの非コメント行よりも前に指定しなければなりません。

► **C** C プログラムの場合、以下の **-qlanglvl** サブオプションを *suboption* に使用できます。

classic	非 stdc89 プログラムのコンパイルを許可し、K&R レベルのプリプロセッサに厳密に適合させます。この言語レベルは、 <code>math.h</code> などの AIX v5.1 システム・ヘッダー・ファイルではサポートされていません。AIX v5.1 システム・ヘッダー・ファイルを使用しなければならない場合は、プログラムを stdc89 言語レベルまたは extended 言語レベルへコンパイルすることを検討してください。
extended	RT コンパイラーおよび classic との互換性を提供します。この言語レベルは C89 に基づいています。

saa	現行の SAA [®] C CPI 言語定義に適合するコンパイル。これは現在は SAA C レベル 2 です。
saal2	SAA C レベル 2 CPI 言語定義に適合するコンパイル。これにはいくつかの例外があります。
stdc89	ANSI C89 標準に適合するコンパイルで、ISO C90 としても知られています。
stdc99	ISO C99 標準に適合するコンパイル。 注: C99 の要求するヘッダー・ファイルと実行時ライブラリーは、すべてのオペレーティング・システム・リリースでサポートされているわけではありません。
extc89	コンパイルは、ANSI C89 標準に適合しており、インプリメンテーション固有の言語拡張を受け入れます。
extc99	コンパイルは、ISO C99 標準に適合しており、インプリメンテーション固有の言語拡張を受け入れます。 注: C99 の要求するヘッダー・ファイルと実行時ライブラリーは、すべてのオペレーティング・システム・リリースでサポートされているわけではありません。
[no]c99complex	このサブオプションは、C99 複合データ型および関連キーワードを認識するようにコンパイラーに指示します。 注: 複合データ型に対するサポートは、各種のコンパイラー間でそれぞれ異なるため、移植性の問題が起きる場合があります。このコンパイラー・オプションを -qinfo=por と共に指定すると、コンパイラーは移植性に関する警告メッセージを出します。
[no]c99complexheader	このサブオプションは、C99 complex.h ヘッダー・ファイルを使用するようにコンパイラーに指示します。
[no]compatzea	このオプションは、すべての C 言語レベルで使用できます。デフォルトは -qlanglvl=nocompatzea です。 -qlanglvl=compatzea の指定は、 -qlanglvl=zeroextarray も有効である場合にのみ効果があります。

XL C/C++ v7.0 for AIX より前のバージョンでは、範囲が 0 の配列の基盤ディメンションは 1 でした。範囲が 0 の配列に sizeof 演算子を適用すると、1 に要素型のサイズを掛けた値を戻します。

XL C/C++ v7.0 for AIX 以降のバージョンでは、範囲が 0 の配列の基盤ディメンションが変更され、基盤ディメンション 0 がサポートされます。この振る舞いは、XL C/C++ コンパイラーの Linux および Mac OS X バージョンの振る舞いと一貫性を持ちます。

この変更により、既存のコードの振る舞いが影響を受ける可能性があります。 **-qlanglvl=compatzea** オプションは互換性を保つためのオプションであり、これを使用して、XL C/C++ v7.0 for AIX でコンパイルされた AIX アプリケーションにおいて、ディメンション 1 を持つ範囲 0 の配列を引き続き使用することができます。

[no]ucs

言語レベル **stdc99** および **extc99** では、デフォルトは **-qlanglvl=ucs** です。

このオプションは、ユニコード文字が、プログラム・ソース・コードの ID、ストリング・リテラル、および文字リテラルで許可されるかどうかを制御します。

ユニコード文字セットは、C 標準でサポートされています。この文字セットには、北アメリカおよび西欧のすべての言語を含む、幅広い範囲の言語で使用する文字の全セット、数字、およびその他の文字が含まれています。ユニコード文字は、16 ビットまたは 32 ビットが可能です。ASCII の 1 バイト文字は、ユニコード文字セットのサブセットです。

このオプションが、yes に設定されている場合は、ソース・ファイルに直接か、またはエスケープ・シーケンスに類似した表記を使用して、ユニコード文字を挿入することができます。ユニコード文字の多くが、画面に表示できない、またはキーボードから入力できないため、通常は、後者の方法が好まれています。ユニコード文字の表記形式は、16 ビット文字の場合は `¥uhhhh`、32 ビット文字の場合は `¥Uhhhhhhh` です。*h* は、16 進数字を表します。文字の短縮 ID は、ISO/IEC 10646 によって指定されます。

[no]zeroextarray

このオプションを使用すると、柔軟な配列要素および範囲 0 の配列がサポートされます。デフォルトは、**-qlanglvl=nozeroextarray** です。

このサブオプションはすべての C 言語レベルでサポートされていますが、言語レベル **stdc89** のコンパイルで柔軟な配列要素を使用する場合は、コンパイラーから警告メッセージが出されます。コンパイラーはまた、言語レベル **stdc89** または **stdc99** のコンパイルで範囲 0 の配列があると、重大エラー・メッセージを出します。これらのメッセージを表示させないようにするには、柔軟な配列要素または範囲 0 の配列の定義個所の前に、`__extension__` キーワードを用います。

以下の **-qlanglvl** サブオプションは、C コンパイラーによって受け入れられますが、無視されます。これらのサブオプションが暗黙指定する関数を使用可能にするには、**-qlanglvl=extended**、**-qlanglvl=extc99**、または **-qlanglvl=extc89** を使用してください。 **-qlanglvl** に他の値を指定すると、これらのサブオプションによって暗黙指定される関数は使用不可になります。

[no]gnu_assert	GNU C 移植性オプション。
[no]gnu_explicitregvar	GNU C 移植性オプション。
[no]gnu_include_next	GNU C 移植性オプション。
[no]gnu_locallabel	GNU C 移植性オプション。
[no]gnu_warning	GNU C 移植性オプション。

▶ C++ C++ プログラムでは、以下の 1 つ以上の **-qlanglvl** サブオプションを *suboption* に指定できます。

ansi	コンパイルでは、ANSI/ISO 1998 規格がサポートされます。これにより、 <code>_cplusplus</code> マクロが "199711L" にアクティブ化されます。2003 規格では、小さな問題点の変更のみが追加されています。ここでは、2003 規格に準拠します。
compat366	コンパイルは、IBM C および C++ コンパイラ V 3.6 フィーチャーの一部 (すべてではない) に準拠します。
extended	コンパイルは strict98 に基づいて行われますが、拡張言語フィーチャーを適合させるために若干の相違があります。
strict98	コンパイルでは、ANSI/ISO 1998 規格がサポートされます。これにより、 <code>_cplusplus</code> マクロが "199711L" にアクティブ化されます。2003 規格では、小さな問題点の変更のみが追加されています。ここでは、2003 規格に準拠します。
[no]anonstruct	このサブオプションは、匿名の構造体および匿名のクラスが C++ ソースで許可されるかどうかを制御します。

デフォルトでは、コンパイラは匿名の構造体を許可します。これは C++ 標準の拡張で、Microsoft Visual C++ が提供している C++ コンパイラと互換性のある動きをします。

匿名の構造体は、以下のコード・フラグメントにあるように、たいていの場合は共用体で使用されます。

```
union U {
    struct {
        int i:16;
        int j:16;
    };
    int k;
} u;
// ...
u.j=3;
```

このサブオプションが設定されていると、コードが匿名の構造体を宣言し、**-qinfo=por** が指定されている場合は、警告が出ます。

-qlanglvl=noanonstruct でビルドすると、匿名の構造体にエラーのフラグが付けられます。標準 C++ に適合している場合は、

noanonstruct を指定してください。

[no]anonunion	このサブオプションは、どのメンバーが匿名の共用体で許可されるかを制御します。
---------------	--

このサブオプションが **anonunion** に設定されていると、匿名の共用体は、標準 C++ が匿名の共用体以外で許可するすべての型のメンバーを持つことができます。例えば、`struct`、`typedef`、および `enumeration` などの非データ・メンバーは許可されます。

メンバー関数、仮想関数、または単純ではないデフォルト・コンストラクター、コピー・コンストラクター、またはデストラクターを持つクラスのオブジェクトは、このオプションの設定にかかわらず共用体のメンバーにはなりません。

デフォルトでは、コンパイラは匿名の共用体の非データ・メンバーを許可します。これは標準 C++ の拡張子で、VisualAge C++ の以前のバージョンや以前に使用されていた製品、および Microsoft Visual C++ が提供している C++ コンパイラと互換性のある振る舞いをします。

このオプションが **anonunion** に設定されていると、コードがその拡張子を使用した場合、**-qsuppress** オプションで警告メッセージを抑止しない限り、警告が出ます。

標準 C++ に適合している場合は、**noanonunion** を設定してください。

[no]ansifor

このサブオプションは、C++ 標準で定義されているスコープ規則を for-init ステートメントで宣言されている名前に適用するかどうかを制御します。

デフォルトでは、標準 C++ 規則が使用されます。例えば、以下のコードでは名前検索エラーが起こります。

```
{
    //...
    for (int i=1; i<5; i++) {
        cout << i * 2 << endl;
    }
    i = 10; // error
}
```

エラーの理由は、i が、あるいは for-init ステートメント内で宣言されたいずれかの名前が、for ステートメント内でのみ可視であるからです。エラーを訂正するには、i をループの外で宣言するか、あるいは ansiForStatementScopes を no に設定します。

[no]ansisinit

noansifor を設定して、古い言語の動きを許可してください。初期のバージョンの VisualAge C++ や先行の製品が提供したコンパイラー、および Microsoft Visual C++ などのその他の製品を使用して開発したコードの場合、これを行う必要が生じることがあります。このサブオプションは、古いコンパイラー (v3.6 以前) および現在のコンパイラー (v5.0 以降) のオブジェクト互換性を選択するのに使用できます。

このサブオプションは、v3.6 以前のバージョンの IBM C/C++ コンパイラーでビルドされた既存の共用ライブラリーが含まれているアプリケーションをビルドする場合に役立ちます。 **noansisinit** サブオプションを指定することにより、新規にコンパイルしたオブジェクト内にデストラクターを持つ (静的ローカルを含む) グローバル・オブジェクトの動きが、以前のコンパイラーでビルドされたオブジェクトと確実に互換性を保つようにします。デフォルト設定は **ansisinit** です。

[no]c99__func__

このサブオプションは、C99 **__func__** ID を認識するようコンパイラーに指示します。 **__func__** ID は、以下のような暗黙宣言があるかのように振る舞います。

```
static const char __func__[] = function_name;
ここで、function_name は、__func__ ID が表示される関数の名前です。
```

__func__ ID の効果は、以下のコード・セグメントで見ることができます。

```
void this_function()
{
    printf("__func__ appears in %s", __func__);
}
```

実行時に、以下を出力します。

```
__func__ appears in this_function
```

-qlanglvl=extended が有効な場合、**c99__func__** サブオプションはデフォルトで使用可能です。 **-qlanglvl=c99__func__** を指定することによってすべての言語レベルについて使用可能にし、**-qlanglvl=noc99__func__** を指定することによってすべての言語レベルについて使用不可にできます。

c99__func__ が有効な場合、**__C99_FUNC__** マクロは 1 に定義され、その他の場合は定義されません。

[no]c99complex	このサブオプションは、C99 複合データ型および関連キーワードを認識するようにコンパイラーに指示します。 注: 複合データ型に対するサポートは、各種のコンパイラー間でそれぞれ異なるため、移植性の問題が起きる場合があります。このコンパイラー・オプションを -qinfo=por と共に指定すると、コンパイラーは移植性に関する警告メッセージを出します。
[no]c99complexheader	このサブオプションは、C99 complex.h ヘッダー・ファイルを使用するようコンパイラーに指示します。
[no]c99compoundliteral	このサブオプションは、C99 複合リテラル・フィーチャーをサポートするようコンパイラーに指示します。
[no]c99hexfloat	このオプションは、C++ アプリケーションで C99 スタイル 16 進浮動小数点定数のサポートを使用可能にします。このサブオプションは、 -qlanglvl=extended のためにデフォルトでオンです。有効な場合は、コンパイラーは macro <code>__C99_HEX_FLOAT_CONST</code> を定義します。
[no]c99vla	c99vla が有効な場合、コンパイラーは C++ アプリケーションで C99 型可変長配列の使用をサポートします。 macro <code>__C99_VARIABLE_LENGTH_ARRAY</code> は値 1 で定義されます。 注: C++ アプリケーションでは、可変長配列によって使用するために割り振られるストレージは、常駐する関数が実行を完了するまでリリースされません。
[no]compatzea	デフォルトは -qlanglvl=nocompatzea です。 -qlanglvl=compatzea の指定は、 -qlanglvl=zeroextarray も有効である場合にのみ効果があります。 XL C/C++ v7.0 for AIX より前のバージョンでは、範囲が 0 の配列の基盤ディメンションは 1 でした。範囲が 0 の配列に <code>sizeof</code> 演算子を適用すると、1 に要素型のサイズを掛けた値を戻します。 XL C/C++ v7.0 for AIX 以降のバージョンは、範囲が 0 の配列の基盤ディメンションが変更され、基盤ディメンション 0 がサポートされます。この振る舞いは、XL C/C++ コンパイラーの Linux および MacOS X バージョンの振る舞いと一貫性を持ちます。 この変更により、既存のコードの振る舞いが影響を受ける可能性があります。 -qlanglvl=compatzea オプションは互換性を保つためのオプションであり、これを使用して、XL C/C++ v7.0 for AIX でコンパイルされた AIX アプリケーションにおいて、ディメンション 1 を持つ範囲 0 の配列を引き続き使用することができます。
[no]gnu_assert	以下の GNU C システム識別アサーションのサポートを使用可能または使用不可にするための GNU C 移植性オプションです。 <ul style="list-style-type: none">• <code>#assert</code>• <code>#unassert</code>• <code>#cpu</code>• <code>#machine</code>• <code>#system</code>
[no]gnu_complex	このサブオプションは、GNU 複合データ型および関連キーワードを認識するようにコンパイラーに指示します。 注: 複合データ型に対するサポートは、各種の C++ コンパイラー間でそれぞれ異なるため、移植性の問題が起きる場合があります。このコンパイラー・オプションを -qinfo=por と共に指定すると、コンパイラーは移植性に関する警告メッセージを出します。
[no]gnu_computedgoto	計算済み <code>goto</code> のサポートを使用可能にする GNU C 移植性オプション。このサブオプションは -qlanglvl=extended に使用可能で、macro <code>__IBM_COMPUTED_GOTO</code> を定義します。
[no]gnu_explicitregvar	変数の明示的レジスターの指定をコンパイラーが受け入れて無視するかどうかを制御するための GNU C 移植性オプションです。

[no]gnu_externtemplate このサブオプションは、extern テンプレートのインスタンス化を使用可能または使用不可にします。

デフォルト設定は、拡張言語レベルへコンパイルするときは **gnu_externtemplate** です、**strict98** または **compat366** 言語レベルへコンパイルするときは **nognu_externtemplate** になります。

gnu_externtemplate が有効な場合、明示的 C++ テンプレート・インスタンス化の前にキーワード **extern** を追加することによってテンプレートのインスタンス化が **extern** であると宣言することができます。**extern** キーワードは、宣言内の最初のキーワードでなければならず、**extern** キーワードは 1 つしか使用できません。

これは、クラスまたは関数のインスタンスを生成しません。クラスおよび関数の両方で、extern テンプレートのインスタンス化が、extern テンプレート・インスタンス化に先行するコードによってトリガーされておらず、明示的にインスタンスを生成されているのでも、明示的に特殊化されているのでもなければ、その extern テンプレートのインスタンス化はテンプレートのパーツのインスタンス化を妨げます。

クラスの場合、静的データ・メンバーおよびメンバー関数のインスタンスは生成されませんが、クラスをマップするために必要であれば、クラス自体のインスタンスは生成されます。必要コンパイラー生成関数 (例えば、デフォルト・コピー・コンストラクター) のインスタンスは生成されます。関数の場合、プロトタイプのインスタンスは生成されますが、テンプレート関数の本体のインスタンスは生成されません。

以下の例を参照してください。

```
template < class T > class C {
    static int i;
    void f(T) { }
};

template < class U > int C<U>::i = 0;
extern template class C<int>; // extern explicit
                                // template
                                // instantiation
C<int> c; // does not cause instantiation of
          // C<int>::i or C<int>::f(int) in
          // this file but class is
          // instantiated for mapping
C<char> d; // normal instantiations

=====

template < class C > C foo(C c) { return c; }

extern template int foo<int>(int); // extern explicit
                                    // template
                                    // instantiation
int i = foo(1); // does not cause instantiation
                // of body of foo<int>
```

[no]gnu_include_next GNU C **#include_next** プリプロセッサ・ディレクティブのサポートを使用可能または使用不可にするための GNU C 移植性オプションです。

[no]gnu_labelvalue 値としてのラベルのサポートを使用可能または使用不可にするための GNU C 移植性オプションです。このサブオプションは **-qlanglvl=extended** のためにデフォルトでオンであり、macro **__IBM_LABEL_VALUE** を定義します。

[no]gnu_locallabel ローカル宣言ラベルのサポートを使用可能または使用不可にするための GNU C 移植性オプションです。

gnu_membernamereuse	メンバー・リストのテンプレート名を typedef として再使用可能にする GNU C 移植性オプション。
[no]gnu_suffixij	GCC スタイル複合メンバーのサポートを使用可能または使用不可にするための GNU C 移植性オプションです。 gnu_suffixij が指定される場合、複素数はサフィックス <i>i</i> / <i>I</i> または <i>j</i> / <i>J</i> で終わることができます。このオプションは -q^{lang}lvi=varargmacros と同様です。主な違いは以下のとおりです。
[no]gnu_varargmacros	<ul style="list-style-type: none"> • オプションの可変引き数 ID が省略符号の前に置かれ、macro <code>__VA_ARGS__</code> の代わりに ID を使用可能にする。空白が ID と省略符号の間に表示されることがあります。 • 可変引き数を省略できる。 • トークン貼り付け演算子 (##) がコンマと可変引き数の間に表示される場合、可変引き数が提供されていないと、プリプロセッサはぶら下がりコンマ (,) を除去する。 • macro <code>__IBM_MACRO_WITH_VA_ARGS</code> が 1 に定義される。 <p>例 1 - 単純な置換:</p> <pre>#define debug(format, args...) printf(format, args) debug("Hello %s\n", "Chris");</pre> <p>以下のようにプリプロセスします。</p> <pre>printf("Hello %s\n", "Chris");</pre> <p>例 2 - 可変引き数の省略:</p> <pre>#define debug(format, args...) printf(format, args) debug("Hello\n");</pre> <p>以下のようにプリプロセスし、ぶら下がりコンマを残します。</p> <pre>printf("Hello\n",);</pre> <p>例 3 - トークン貼り付け演算子を使用して、可変引き数が削除されたときにぶら下がりコンマを除去する:</p> <pre>#define debug(format, args...) printf(format, ## args) debug("Hello\n");</pre> <p>以下のようにプリプロセスします。</p> <pre>printf("Hello\n");</pre>
[no]gnu_warning	GNU C #warning プリプロセッサ・ディレクティブのサポートを使用可能または使用不可にするための GNU C 移植性オプションです。

[no]illptom

このサブオプションは、メンバーへのポインターを形成するのにどの式が使用できるかを制御します。XL C/C++ は、一般的に使用されているが C++ 標準に準拠していないいくつかの形式を受け入れることができます。

デフォルトで、コンパイラーはこれらの形式を許可します。これは標準 C++ の拡張子で、VisualAge C++ の初期のバージョンや以前に使用されていた製品、および Microsoft Visual C++ が提供している C++ コンパイラーと互換性のある振る舞いをします。

このサブオプションが **illptom** に設定されていると、コードがその拡張子を使用した場合、**-qsuppress** オプションで警告メッセージを抑制しない限り、警告が出ます。

例えば、以下のコードは、関数のメンバー **p** へのポインターを定義し、それを古いスタイルで **C::foo** のアドレスへ初期設定します。

```
struct C {  
    void foo(int);  
};  
  
void (C::*p) (int) = C::foo;
```

C++ 標準に適合している場合は、**noillptom** を設定してください。上記のコード例では、**&** 演算子を使用するよう変更しなければなりません。

```
struct C {  
    void foo(int);  
};  
  
void (C::*p) (int) = &C::foo;
```

[no]implicitint

このサブオプションは、コンパイラーが、欠落している、または部分的に指定されている型を `int` の暗黙的な指定として受け入れるかどうかを制御します。これは、標準では現在受け入れられてはいませんが、既存のコードには存在する場合があります。

サブオプション・セットが **noimplicitint** に設定されている場合は、すべての型が完全に指定されていなければなりません。

サブオプション・セットが **implicitint** に設定されている場合は、ネーム・スペース・スコープでの関数宣言、またはメンバー・リスト内の関数宣言は、`int` を戻すよう暗黙的に宣言されます。また、型を完全に指定しない宣言指定子のシーケンスは、いずれも、整数型を暗黙的に指定します。あたかも `int` 指定子が存在しているかのような効果がありますので、注意してください。これは、つまり、指定子 `const` が自ら定数整数を指定することを意味します。

以下の指定子は、型を完全に指定しません。

- `auto`
- `const`
- `extern`
- `extern "<literal>"`
- `inline`
- `mutable`
- `friend`
- `register`
- `static`
- `typedef`
- `virtual`
- `volatile`
- プラットフォーム特定型 (例えば、`_cdecl`)

型が指定されている状態は、いずれも、このサブオプションの影響を受けるので、注意してください。これには、例えば、テンプレート型およびパラメーター型、例外指定、式における型 (eg, `casts`、`dynamic_cast`、`new`)、および変換関数の型が含まれます。

デフォルトでは、コンパイラーは **-qianglvi=implicitint** を設定します。これは C++ 標準に対する拡張で、VisualAge C++ の初期バージョンや先行の製品、および Microsoft Visual C++ が提供している C++ コンパイラーと互換性のある動きをします。

例えば、関数 `MyFunction` の戻りの型は、以下のコードで省略されたため、`int` です。

```
MyFunction()  
{  
    return 0;  
}
```

標準 C++ に適合している場合は、**-qianglvi=noimplicitint** を設定してください。例えば、上記の関数宣言は、以下のように変更されなければなりません。

```
int MyFunction()  
{  
    return 0;  
}
```

[no]newexcp

このサブオプションは、C++ の **new** 演算子が例外を throw するか、あるいはしないかを指定します。 **-qlanglvl=newexcp** オプションが指定されている場合に、要求されたメモリー割り振りが失敗すると、標準例外 **std::bad_alloc** が throw されます。このオプションは、**new** 演算子の **nothrow** バージョンには適用されません。

クラス固有の **new** 演算子、および配置引き数付きの **new** 演算子は、このオプションに影響されません。

new 演算子の標準インプリメンテーションは、例外を完全にサポートします。 VisualAge C++ の以前のバージョンとの互換性については、これらの演算子はデフォルトで 0 を戻します。

[no]offsetnonpod

このサブオプションは、**offsetof** マクロをデータのみではないクラスへ適用できるかどうかを制御します。 C++ のプログラマーは、データのみクラスを“プレーンな古いデータ” (POD) のクラスと普段よく呼んでいます。

デフォルトでは、コンパイラーは **offsetof** を non-POD クラスで使用することを許可します。これは、C++ 標準の拡張子で、VisualAge C++ for OS/2 3.0、VisualAge for C++ for Windows、バージョン 3.5、および Microsoft Visual C++ が提供している C++ コンパイラーに互換性のある振る舞いをします。

このオプションが設定されていると、コードがその拡張子を使用した場合、**-qsuppress** オプションで警告メッセージを抑止しない限り、警告が出ます。

標準 C++ に適合している場合は、**-qlanglvl=nooffsetnonpod** を設定してください。

コードが以下のいずれか 1 つを含むクラスへ **offsetof** を適用する場合は、**-qlanglvl=offsetnonpod** を設定してください。

- ユーザー宣言のコンストラクターまたはデストラクター
- ユーザー宣言の代入演算子
- **private** または **protected** 非静的データ・メンバー
- 基底クラス
- 仮想関数
- メンバーへの型ポインターの非静的データ・メンバー
- 非データ・メンバーを持つ構造体または共用体
- 参照

[no]olddigraph

このオプションは、古いスタイルの **digraph** が C++ ソースで許可されるかどうかを制御します。これは、**-qdigraph** も設定されている場合にのみ適用されます。

デフォルトでは、コンパイラーは、C++ 標準で指定された **digraph** のみをサポートします。

以下の **digraph** のうち少なくとも 1 つがコードに含まれている場合は、**-qlanglvl=olddigraph** を設定します。

Digraph	結果の文字
%%	# (ポンド記号)
%%%%	## (ダブル・ポンド記号、プリプロセッサー・マクロの連結演算子として使用される)

VisualAge C++ の以前のバージョンおよび以前に使用されていた製品でサポートされている標準 C++ および拡張 C++ 言語レベルと互換の場合は、**-qlanglvl=noolddigraph** を設定します。

[no]oldfriend

このオプションは、詳述されたクラス名なしでクラスを指定するフレンド宣言を C++ エラーとして取り扱うかどうかを制御します。

デフォルトでは、コンパイラーは、キーワード・クラスを持つクラスの名前を詳述せずにフレンド・クラスを宣言できるようになっています。これは C++ 標準に対する拡張で、VisualAge C++ の初期バージョンや先行の製品、および Microsoft Visual C++ が提供している C++ コンパイラーと互換性のある動きをします。

例えば、下記のステートメントは、クラス **IFont** がフレンド・クラスになるように宣言し、**oldfriend** サブオプションが **specified** に設定されている場合に有効です。

```
friend IFont;
```

標準 C++ に適合している場合は、**nooldfriend** サブオプションを設定してください。上記の宣言例は、下記のステートメントに変更するか、または **-qsuppress** オプションで警告メッセージを抑止しない限り、警告を出します。

```
friend class IFont;
```

[no]oldmath

このサブオプションは、**<math.h>** を組み込み済みソース・ファイルまたは基本ソース・ファイルとして指定する場合に、**math.h** の数学関数宣言のどのバージョンが組み込まれるかを制御します。

デフォルトでは、新しい標準の数学関数が使用されます。C++ 標準に厳密に適合している場合は、**-qlanglvl=nooldmath** でビルドしてください。

VisualAge C++ の初期バージョンや先行の製品でビルドされたモジュールとの互換性を保持するため、**-qlanglvl=oldmath** でビルドする必要がある場合があります。

[no]oldtempacc

このサブオプションは、一時オブジェクトの作成が回避される場合でも、一時オブジェクトの作成のためのコピー・コンストラクターへのアクセスを常に検査するかどうかを制御します。

デフォルトでは、コンパイラーはアクセス検査を抑制します。これは、C++ 標準の拡張子で、VisualAge C++ for OS/2[®] 3.0、VisualAge for C++ for Windows、Version 3.5、および Microsoft Visual C++ が提供している C++ コンパイラーと互換性のある振る舞いをします。

このサブオプションが yes に設定されていると、コードがその拡張子を使用した場合、**-qsuppress** で警告メッセージを使用不可にしない限り、警告が出ます。

標準 C++ に適合している場合は、**-qlanglvl=nooldtempacc** を設定してください。例えば、以下のコードの throw ステートメントは、コピー・コンストラクターがクラス C の protected メンバーであるため、エラーの原因となります。

```
class C {
public:
    C(char *);
protected:
    C(const C&);
};

C foo() {return C("test");} // return copy of C object

void f()
{
    // catch and throw both make implicit copies of
    // the thrown object
    throw C("error"); // throw a copy of a C object
    const C& r = foo(); // use the copy of a C object
                        // created by foo()
}
```

上記のコード例では、3 個所にコピー・コンストラクター C(const C&) の誤った形式が使用されています。

[no]oldtmplalign

このサブオプションでは、バージョン 5.0 以前のバージョンのコンパイラー (xlC) に実装されている位置合わせ規則を指定します。これらの以前のバージョンの xlC コンパイラーは、ネスト・テンプレート用に指定された位置合わせ規則を無視します。VisualAge C++ 4.0 以降のデフォルトでは、これらの位置合わせ規則は無視されません。例えば、次のテンプレートでは、A<char>::B のサイズは **-qlanglvl=nooldtmplalign** の場合は 5、**-qlanglvl=oldtmplalign** の場合は 8 となります。

```
template <class T>
struct A {
#pragma options align=packed
    struct B {
        T m;
        int m2;
    };
#pragma options align=reset
};
```

[no]oldtmpspec

このサブオプションは、C++ 標準に準拠していないテンプレートの特殊化を許可するかどうかを制御します。

デフォルトでは、コンパイラーはこれらの古い特殊化を許可します (**-qlanglvl=nooldtmpspec**)。これは、標準 C++ の拡張子で、VisualAge C++ for OS/2 3.0、VisualAge for C++ for Windows、バージョン 3.5、および Microsoft Visual C++ が提供している C++ コンパイラーに互換性のある振る舞いをします。

-qlanglvl=oldtmpspec が設定されていると、コードがその拡張子を使用した場合、**-qsuppress** オプションで警告メッセージを抑止しない限り、警告が出ます。

例えば、以下の行を使用して、型 `char` のテンプレート・クラス・リボンを明示的に特殊化することができます。

```
template<class T> class ribbon { /*...*/};  
class ribbon<char> { /*...*/};
```

標準 C++ に適合している場合は、**-qlanglvl=nooldtmpspec** を設定してください。上記の例では、テンプレートの特殊化は、以下のように変更されなければなりません。

```
template<class T> class ribbon { /*...*/};  
template<> class ribbon<char> { /*...*/};
```

[no]redefmac

前の `#undef` または `undefine()` ステートメントなしでマクロを再定義できるかどうかを指定します。

[no]trailenum

このサブオプションは、`enum` 宣言で末尾のコンマを許可するかどうかを制御します。

デフォルトでは、コンパイラーは、列挙子リストの最後に 1 つまたはそれ以上の末尾のコンマを許可します。これは C++ 標準の拡張子で、Microsoft Visual C++ との互換性を提供します。以下の `enum` 宣言は、この拡張子を使用します。

```
enum grain { wheat, barley, rye,, };
```

標準 C++ に適合している場合、または VisualAge C++ の以前のバージョンおよび先行の製品でサポートされている **stdc89** 言語レベルに適合している場合は、**-qlanglvl=notrailenum** を設定します。

[no]typedefclass

このサブオプションは、VisualAge C++ および先行の製品の以前のバージョンとの後方互換性を提供します。

現在の C++ 標準は、クラス名を指定するはずの個所に `typedef` の名前を指定することを許可しません。このオプションは、この制限を緩和します。基本指定子リストおよびコンストラクター初期化指定子リストの `typedef` 名の使用を許可するには、**-qlanglvl=typedefclass** を設定します。

デフォルトでは、`typedef` 名はクラス名を指定するはずの個所に指定することはできません。

[no]ucs

このサブオプションは、ユニコード 文字が、C++ ソースの ID、ストリング・リテラル、および文字リテラルで許可されるかどうかを制御します。デフォルト設定は **-qlanglvl=noucs** です。

ユニコード文字セットは、C++ 標準でサポートされています。この文字セットには、北アメリカおよび西欧のすべての言語を含む、幅広い範囲の言語で使用される文字の全セット、数字、およびその他の文字が含まれています。ユニコード文字は、16 ビットまたは 32 ビットが可能です。ASCII の 1 バイト文字は、ユニコード文字セットのサブセットです。

-qlanglvl=ucs が設定されている場合は、ソース・ファイルに直接、あるいはエスケープ・シーケンスに類似した表記を使用して、ユニコード文字を挿入することができます。ユニコード文字の多くが、画面に表示できない、またはキーボードから入力できないため、通常は、後者の方法が好まれています。ユニコード文字の表記形式は、16 ビット文字の場合は `¥uhhhh`、32 ビット文字の場合は `¥Uhhhhhhhh` です。`h` は、16 進数字を表します。文字の短縮 ID は、ISO/IEC 10646 によって指定されます。

[no]varargmacros

この C99 フィーチャーは、C++ アプリケーションの関数のようなマクロにある可変引き数リストの使用を許可します。構文は、可変引き数関数に似ており、`printf` のマスキング・マクロとして使用することができます。

例を以下に示します。

```
#define debug(format, ...) printf(format, __VA_ARGS__)  
  
debug("Hello %s¥n", "Chris");
```

以下のようにプリプロセスします。

```
printf("Hello %s¥n", "Chris");
```

置換リストのトークン `__VA_ARGS__` は、パラメーターの省略符号に対応します。省略符号は、マクロ呼び出しの可変引き数を示します。

varargmacros を指定すると、macro `__C99_MACRO_WITH_VA_ARGS` を値 1 に定義します。

[no]zeroextarray

このサブオプションは、柔軟な配列要素と範囲 0 の配列をアプリケーションで使用できるかどうかを制御します。

デフォルトでは、コンパイラーはゼロ・エレメントを持つ配列を許可します。これは C++ 標準の拡張子で、Microsoft Visual C++ との互換性を提供します。下記の宣言例は、無次元配列 `a` および `b` を定義します。

```
struct S1 { char a[0]; };  
struct S2 { char b[]; };
```

柔軟な配列要素と範囲 0 の配列は、**-qlanglvl** が **extended** または **zeroextarray** に設定されている場合にサポートされます。他の言語レベルの場合は、コンパイラーがエラー・メッセージを出します。

標準 C++ に適合している場合、または VisualAge C++ の以前のバージョンおよび以前に使用されていた製品でサポートされている ANSI 言語レベルに適合している場合は、**nozeroextarray** を設定します。

classic によって示される **stdc89** モードに対する例外は以下のとおりです。

トークン化

マクロ展開により導入されるトークンは、場合によっては隣接するトークンと結合されることがあります。これは、従来は、旧式のプリプロセッサのテキスト・ベースのインプリメンテーションで意図的に行われていました。これは、従来のインプリメンテーションでは、プリプロセッサが別個のプログラムであり、その出力がコンパイラーに渡されていたためです。

同様の理由から、コメントによってのみ分けられたトークンが、1 つのトークンを形成するように結合されることもあります。ここでは、**classic** モードでコンパイルされたプログラムのトークンを行う方法の要約を示します。

1. ソース・ファイルの該当ポイントで、次のトークンが、トークンを形成することのできる可能性のある文字の最長シーケンスです。例えば、`i ++ + ++ j` は正しいプログラムになりますが、`i+++++j` は `i ++ ++ + j` としてトークン化されます。
2. 形成されたトークンが ID およびマクロ名である場合は、そのマクロは、その **#define** ディレクティブで指定されたトークンのテキストで置き換えられます。各パラメーターは、対応する引き数のテキストで置き換えられます。コメントは、引き数とマクロ・テキストの両方から取り除かれます。
3. スキャンは、元のプログラムの一部であるかのように、マクロが置き換えられたポイントの最初のステップから再開されます。
4. プログラム全体のプリプロセスが終わると、その結果は、最初のステップのようにコンパイラーによって再度スキャンされます。置き換えを行うマクロがないため、2 番目と 3 番目のステップはここでは適用されません。プリプロセス・ディレクティブに似ている最初の 3 ステップによって生成される構成体は、そのようには処理されません。

新しいトークンを形成するため、隣接しているものの事前に分けられているトークンのテキストを結合するのは、3 番目および 4 番目のステップで行います。

行継続用の `¥` 文字は、ストリング、文字リテラル、およびプリプロセス・ディレクティブでしか受け入れられません。

以下の構成体があるとしします。

```
#if 0
    "unterminated
#endif
#define US "Unterminating string
char *s = US terminated now"
```

上記の構成体は、先頭が `FALSE` ブロック内の終了しないリテラルで、2 番目がマクロ展開後に完了するため、診断メッセージを生成しません。しかし、

```
char *s = US;
```

このとおり指定すると、`US` 内のストリング・リテラルが行の終わりまでに完了しないため、診断メッセージが生成されます。

空の文字リテラルは使用できます。このリテラルの値はゼロです。

プリプロセッサ・
ディレクティブ

行の先頭列に、# トークンがなければなりません。# の直後に続くトークンは、マクロ展開に使用できます。ディレクティブの名前、および以下の例の (の場合にのみ、¥ を使って行を継続することができます。

```
#define f(a,b) a+b
f¥
(1,2)      /* accepted */

#define f(a,b) a+b
f(¥
1,2)      /* not accepted */
```

¥ に関する規則は、ディレクティブが有効であるかどうかに適用されます。例を以下に示します。

```
#¥
define M 1  /* not allowed */

#def¥
ine M 1     /* not allowed */

#define¥
M 1         /* allowed */

#dfine¥
M 1         /* equivalent to #define M 1, even
              though #dfine is not valid */
```

以下に、**classic** モードと **stdc89** モードの間の、プリプロセッサ・ディレクティブの相違点を示します。ここにリストしていないディレクティブは、どちらのモードでも同じように振る舞います。

#ifdef/#ifndef

最初のトークンが ID でないと、診断メッセージは生成されず、条件は FALSE です。

#else 余計なトークンがあると、診断メッセージは生成されません。

#endif 余計なトークンがあると、診断メッセージは生成されません。

#include

< と > は別のトークンです。ヘッダーは、< と > のつづりと、これらの間のトークンを結合することにより、形成されます。そのため、/* と // はコメントとして認識されて (常に除去され)、” および ’ が < および > 内のリテラルを開始します。(C プログラムでは、**-qcplusplus** を指定すると、C++ 形式のコメント // が認識されます。)

#line 行番号の一部ではないトークンすべてのつづりは、新しいファイル名を形成します。これらのトークンは、ストリング・リテラルである必要はありません。

#error **classic** モードでは認識されません。

#define

有効なマクロ・パラメーター・リストは、ID なしで構成されるか、またはコンマで区切られた任意の数の ID で構成されます。コンマは無視され、パラメーター・リストは、コンマが指定されていないかのように構成されます。パラメーター名を固有にする必要はありません。競合する場合は、最後に指定された名前が認識されます。

無効パラメーター・リストの場合、警告が発行されます。マクロ名を新しい定義で再定義すると、警告が発行され、その新しい定義が使用されます。

#undef 余計なトークンがあると、診断メッセージは生成されません。

マクロ展開

- マクロ起動に関する引き数の数がパラメーターの数に一致しないと、警告が発行されます。
- 関数類似マクロのマクロ名の後に (トークンがある場合、これは引き数が少なすぎる (上記と同様)) として処理され、警告が発行されます。
- パラメーターは、ストリング・リテラルと文字リテラルで置き換えられます。
- 例:

```
#define M()    1
#define N(a)   (a)
#define O(a,b) ((a) + (b))

M(); /* no error */
N(); /* empty argument */
O(); /* empty first argument
      and too few arguments */
```

テキスト出力

コメントの置き換え用に生成されるテキストはありません。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

95 ページの『bitfields』

104 ページの『chars』

121 ページの『digraph』

139 ページの『flag』

171 ページの『info』

177 ページの『inline』

226 ページの『M』

278 ページの『ro』

308 ページの『suppress』

381 ページの『#pragma langlvl』

394 ページの『#pragma options』

「C/C++ ランゲージ・リファレンス」の『IBM C 言語拡張機能』 および『IBM C++ 言語拡張機能』セクションも参照してください。

largepage

► C ► C++

目的

AIX v5.1D 以降を実行している POWER4 および POWER5 システムで使用可能な大規模ページ・ヒープを活用するようにコンパイラーに指示する。

構文

►► — -q —  —►►

注

-qlargepage を指定してコンパイルすると、その結果として、プログラムのパフォーマンスが向上する可能性があります。このオプションは、AIX v5.1D 以降を実行している POWER4 および POWER5 システム上でのみ有効です。

このオプションは、IPA (**-qipa**、**-O4**、**-O5** コンパイラー・オプション) とともに使用したときに有効です。

例

myprogram.c をコンパイルして大規模ページ・ヒープを使用するには、次のように入力します。

```
xlc myprogram.c -qlargepage
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

181 ページの『ipa』

ldbl128、longdouble

➤ C ➤ C++

目的

long double 型のサイズを 64 ビットから 128 ビットに増加させる。

構文



394 ページの『#pragma options』も参照してください。

注

-q1ongdouble オプションは、**-qldbl128** オプションと同じです。

128 ビットの **long double** 型をサポートする別個のライブラリーが提供されます。 **128** のサフィックス (**xlC128**、**xlC128**、**cc128**、**xlC128_r**、**xlC128_r**、または **cc128_r**) を付けて、いずれかの呼び出しコマンドを使用した場合は、これらのライブラリーが自動的にリンクされます。以下の表に示すように、**-lkey** オプションを使用して、128 ビット・バージョンのライブラリーを手操作でリンクすることもできます。

デフォルト (64 ビット) の long double		128 ビットの long double	
ライブラリー	-lkey オプションの形式	ライブラリー	-lkey オプションの形式
libC.a	-lC	libC128.a	-lC128
libC_r.a	-lC_r	libC128_r.a	-lC128_r

プログラムが 128 ビットの **long double** を使用するとき (例えば、**-qldbl128** のみを指定した場合) に 128 ビット・バージョンのライブラリーないでリンクすると、予測できない結果になる場合があります。

-qldbl128 オプションは、**__LONGDOUBLE128** を定義します。

#pragma options ディレクティブは、ソース・ファイルにある最初の C または C++ ステートメントの前で指定しなければなりません。このオプションはファイル全体に適用されます。

例

myprogram.c をコンパイルして **long double** 型を 128 ビットにするには、以下を入力します。

```
x1C myprogram.c -qldbl128 -lC128
```

または

```
x1C128 myprogram.c
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

197 ページの『1』

394 ページの『#pragma options』

libansi

► C ► C++

目的

ANSI C ライブラリー関数の名前が付いたすべての関数が実際はシステム関数であると見なす。

構文

►► -q  

394 ページの『#pragma options』も参照してください。

注

これによって、最適化プログラムは、指定された関数について副次作用があるかどうかなどの動きを認識するため、より優れたコードを生成することができます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

394 ページの『#pragma options』

linedebug

➤ C ➤ C++

目的

デバッガー用に行番号およびソース・ファイル名の情報を生成する。

構文

➤ ➤ -q  ➤ ➤

注

このオプションは最小限のデバッグ情報しか生成しないため、結果として得られるオブジェクトのサイズは、**-g** デバッグ・オプションを指定した場合に生成されるオブジェクトよりも小さくなります。デバッガーを使用してソース・コードのステップを実行することはできますが、変数情報を表示させたり照会したりすることはできません。トレースバック・テーブルを生成させると、行番号が組み込まれます。

このオプションは、**-O** (最適化) オプションとともに使用しないでください。生成される情報が不完全であったり、誤解のもととなる場合があります。

-qlinedebug オプションを指定した場合は、インライン・オプションがデフォルトで **-Q!** (関数をインラインにしない) になります。

-g オプションは、**-qlinedebug** オプションをオーバーライドします。コマンド行で **-g -qnoline debug** を指定した場合は、**-qnoline debug** が無視され、以下の警告が出されます。

1506-... (W) Option -qnoline debug is incompatible with option -g and is ignored

例

myprogram.c をコンパイルして実行可能プログラム **testing** を作成し、デバッガーでステップ実行できるようにするには、次のように入力します。

```
xlc myprogram.c -o testing -qlinedebug
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

155 ページの『g』

244 ページの『O、optimize』

271 ページの『Q』

394 ページの『#pragma options』

list

► C ► C++

目的

オブジェクト・リストを含むコンパイラー・リストを生成する。

構文

►► -q  

394 ページの『#pragma options』も参照してください。

注

-qnoprint コンパイラー・オプションは、このオプションをオーバーライドします。

例

myprogram.C をコンパイルしてオブジェクト・リストを生成させるには、以下を入力します。

```
xlc myprogram.C -qlist
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

265 ページの『print』

394 ページの『#pragma options』

listopt

► C ► C++

目的

コンパイラ呼び出し時に有効なすべてのオプションを示すコンパイラ・リストを作成する。

構文

►► -q  

注

リストには、コンパイラのデフォルト、デフォルト構成ファイル、およびコマンド行設定によって設定された有効なオプションが示されます。プログラム・ソースにある **#pragma** ステートメントによるオプション設定は、コンパイラ・リストには示されません。

-qnoprint を指定すると、このコンパイラ・オプションがオーバーライドされます。

例

myprogram.C をコンパイルして、有効なオプションをすべて示すコンパイラ・リストを作成させるには、以下を入力します。

```
xlc myprogram.C -qlistopt
```

関連資料

61 ページの『コンパイラのコマンド行オプション』

265 ページの『print』

41 ページの『矛盾するコンパイラ・オプションの解決』

longlit

➤ C ➤ C++

目的

サフィックスをはずしたリテラルを long 型に 64 ビット・モードで作成する。

構文

➡➡ — -q — no longlit — longlit — ➡➡

注

下記の表は、**stdc89**、**extc89**、または **extended** 言語レベルでコンパイルするときの 64 ビット・モードでの定数の暗黙の型を示したものです。

	デフォルトの 64 ビット・モード	qlonglit での 64 ビット・モード
サフィックスがない 10 進数	signed int signed long unsigned long	signed long unsigned long
サフィックスをはずした 8 進数または 16 進数	signed int unsigned int signed long unsigned long	signed long unsigned long
サフィックス u/U 付き	unsigned int unsigned long	unsigned long
サフィックス l/L 付き	signed long unsigned long	signed long unsigned long
サフィックス ul/UL 付き	unsigned long	unsigned long

下記の表は、**stdc99**、**extc99**、または **extended** 言語レベルでコンパイルするときの 64 ビット・モードでの定数の暗黙の型を示したものです。

	10 進定数	10 進定数への -qlonglit の効果
サフィックスを はずした場合	int long int	long int
u または U	unsigned int unsigned long int	unsigned long int
l または L	long int	long int
u または U、および l または L の両方	unsigned long int	unsigned long int
ll または LL	long long int	long long int
u または U、および ll または LL の両方	unsigned long long int	unsigned long long int

	8 進または 16 進定数	8 進または 16 進定数への -qlonglit の効果
サフィックスを はずした場合	int unsigned int long int unsigned long int	long int unsigned long int
u または U	unsigned int unsigned long int	unsigned long int
l または L	long int unsigned long int	long int unsigned long int
u または U 、および l または L の両方	unsigned long int	unsigned long int
ll または LL	long long int unsigned long long int	long long int unsigned long long int
u または U 、および ll または LL の両方	unsigned long long int	unsigned long long int

関連資料

61 ページの『コンパイラーのコマンド行オプション』

198 ページの『langlvl』

longlong

► C ► C++

目的

プログラムにおいて **long long** 整数型を許可する。

構文

►►  

デフォルト

xlc、**xlc**、および **cc** でのデフォルトは **-qlonglong** であり、**_LONG_LONG** を定義します (**long long** 型がプログラムで機能するようになります)。**c89** でのデフォルトは、**-qolonglong** です (**long long** 型はサポートされません)。

注

► **C** このオプションは、選択した言語レベルが **stdc99** または **extc99** のときには指定できません。これは、C89 標準への拡張として提供されている **long long** のサポートを制御するために使用します。この拡張は、C99 標準の一部である **long long** のサポートとはわずかに異なります。

例

1. **myprogram.c** をコンパイルして **long long int** を許可しないようにするには、次のように入力します。

```
xlc myprogram.c -qolonglong
```

2. AIX v4.2 以降は、2 ギガバイトよりも大きいサイズのファイルをサポートしており、大容量のデータを単一ファイルで保管することができます。ユーザーのアプリケーションで大容量のファイルを操作できるようにするには、**-D_LARGE_FILES** および **-qlonglong** コンパイラ・オプションを指定してコンパイルします。例を以下に示します。

```
xlc myprogram.c -D_LARGE_FILES -qlonglong
```

関連資料

61 ページの『コンパイラのコマンド行オプション』



目的

make コマンドの記述ファイルの組み込みに適したターゲットを含む出力ファイルを作成する。

構文

▶▶ — **-M** —▶▶

注

-M オプションは、**-qmakedep** オプションと機能的に同じです。

.u ファイルは **make** ファイルではありません。**.u** ファイルを編集しなければ、**make** コマンドで使用することはできません。このコマンドに関する詳細については、ご使用のオペレーティング・システムの資料を参照してください。

出力ファイルには、入力ファイルに対する行、および各インクルード・ファイルに対する項目が入ります。この一般形式は次のとおりです。

```
file_name.o:file_name.c
file_name.o:include_file_name
```

インクルード・ファイルは、**#include** プリプロセッサ・ディレクティブの検索順序の規則に従ってリストされます。この規則については、『**相対パス名を使用したインクルード・ファイルのディレクトリ検索順序**』で説明しています。インクルード・ファイルが見つからない場合、**.u** ファイルには追加されません。

include 文がないファイルについては、入力ファイル名のみをリストする 1 行を含む出力ファイルが生成されます。

例

-o オプションを指定しない場合、**-M** によって生成される出力ファイルは、現行ディレクトリに作成されます。サフィックスとして **.u** が付きます。例えば、以下のコマンドでは、

```
xlc -M person_years.c
```

出力ファイル **person_years.u** が生成されます。

.u ファイルは、**.c**、**.C**、**.cpp**、または **.i** サフィックスが付いた入力ファイルごとに作成されます。また、**-+** コンパイラ・オプションを有効にした状態で、**C++** プログラムをコンパイルすると、ファイル・サフィックスは受け入れられ、**.u** ファイルが生成されます。そうでない場合は、出力の **.u** ファイルは、他のファイルについては一切作成されません。

例えば、以下のコマンドでは、

```
xlc -M conversion.c filter.c /lib/libm.a
```

conversion.u および **filter.u** の 2 つの出力ファイル、および実行可能ファイルが作成されます。**.u** ファイルは、ライブラリーについては作成されません。

現行ディレクトリーが書き込み可能でない場合、**.u** ファイルは作成されません。
-o file_name を **-M** とともに指定する場合、**.u** ファイルは、**-o file_name** で暗黙指定されるディレクトリーに置かれます。例えば、以下の呼び出しでは、

```
xlc -M -c t.c -o /tmp/t.o
```

.u 出力ファイルが **/tmp/t.u** に配置されます。

関連タスク

43 ページの『相対パス名を使用したインクルード・ファイルのディレクトリー検索順序』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

71 ページの『+ (正符号)』

232 ページの『makedep』

248 ページの『o』

297 ページの『sourcetype』

ma



目的

呼び出し用インライン・コードを組み込み関数 **alloca** に置換します。

構文

▶▶ — -ma —▶▶

注

#pragma alloca が未指定の場合、あるいは **-ma** を使用しない場合、**alloca** は組み込み関数としてではなく、ユーザー定義 ID として扱われます。

このオプションは、C++ プログラムには適用されません。C++ プログラムでは、**alloca** 関数宣言を組み込むには、代わりに **#include <malloc.h>** を指定しなければなりません。

例

myprogram.c をコンパイルして、関数 **alloca** の呼び出しをインラインとして扱わせるには、次のように入力します。

```
xlc myprogram.c -ma
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

82 ページの『alloca』

353 ページの『#pragma alloca』

macpstr

► C ► C++

目的

Pascal スtring・リテラルを、先頭バイトにStringの長さが含まれるヌル終了Stringに変換する。

構文

```
►► — -q — nomacpstrmacpstr —►►
```

394 ページの『#pragma options』も参照してください。

注

Pascal String・リテラルには、必ず文字 “**¥p**” が含まれます。Stringの途中にある文字 **¥p** は Pascal String・リテラルを形成せず、この文字の直前に ” (二重引用符) 文字がなければなりません。

Pascal String・リテラルの最終的な長さは、255 バイト (バイトに収まる最大の長さ) 以下です。

例えば、**-qmacpstr** オプションが有効な場合、コンパイラーは以下のように変換します。

```
"¥pABC"
```

以下のようになります。

```
'¥03' , 'A' , 'B' , 'C' , '¥0'
```

Pascal String・リテラルの処理は 1 バイト文字にしか有効でないため、**-qmbcs** または **-qdbcs** オプションが有効である場合は、コンパイラーは **-qmacpstr** オプションを無視します。

#pragma options のキーワード **macpstr** は、すべての C または C++ ソース・ステートメントの前のソース・ファイルの先頭でのみ有効です。ソース・ファイルの途中で使用すると、指定は無視され、コンパイラーによってエラー・メッセージが出されます。

String・リテラルの処理: ここでは、Pascal String・リテラルの処理方法について説明します。

- Pascal String・リテラルを通常のStringに連結すると、非 Pascal Stringが作成されます。例えば、Stringを連結すると、以下のようになります。

```
"ABC" "¥pDEF"
```

これは以下のようになります。

```
"ABCpDEF"
```

- Pascal String・リテラルを**ワイド・String・リテラル**と連結することはできません。

- (長さバイトおよび終端のヌルを除いて) 255 バイトより長い Pascal スtring・リテラルは、コンパイラによって 255 文字に切り捨てられます。
- コンパイラは、**-qmbcs** または **-qdbcs** が使用されている場合には、**-qmacpstr** オプションを無視し、警告メッセージを出します。
- ワイド・Stringについては Pascal String・リテラル処理は行われなため、**-qmacpstr** オプションを指定しても、ワイド・String・リテラルでエスケープ・シーケンス **¥p** を使用すると、警告メッセージが生成されてそのエスケープ・シーケンスが無視されます。
- Pascal String・リテラルは、他の C または C++ のString・リテラルと異なる基本型ではありません。Pascal String・リテラル処理の完了後は、結果として得られたStringは、他のすべてのStringと同様に扱われます。Pascal Stringを前提とする関数にプログラムが C Stringを渡す場合、またはこの逆の場合の動きは未定義です。
- 2 つの Pascal String・リテラルを連結しても (**strcat()** など)、結果は 1 つの Pascal String・リテラルにはなりません。しかし、前述のとおり、2 つの隣接する Pascal String・リテラルを連結して、先頭バイトが新しいString・リテラルの長さである 1 つの Pascal String・リテラルを形成することはできます。例えば、Stringを連結すると、以下のようになります。

```
¥p ABC" ¥p DEF"
```

または

```
¥p ABC" "DEF"
```

以下のようになります。

```
¥06ABCDEF"
```

- 処理の完了後に Pascal String・リテラルの任意のバイトを変更しても、先頭バイトの元の長さの値は変更されません。例えば、String **¥06ABCDEF"** で、Stringの中の既存文字の 1 つとヌル文字を置換すると、Stringの最初のバイト値を変更せず、これによってStringの長さを含みます。
- 処理済みの Pascal String・リテラルのバイトを変更しても、エラーや警告は出されません。
- 以下の文字を入れる場合について考えます。

```
'¥p' , 'A' , 'B' , 'C' , '¥0'
```

これを文字配列に入れても、Pascal String・リテラルは形成されません。

例

mypascal.c をコンパイルしてString・リテラルをヌル終了Stringに変換するには、次のように入力します。

```
xlc mypascal.c -qmacpstr
```

関連資料

61 ページの『コンパイラのコマンド行オプション』

238 ページの『mbcs、dbcs』

394 ページの『#pragma options』

maf

► C ► C++

目的

浮動小数点乗算・加算命令を生成するかどうかを指定する。このオプションは、浮動小数点の中間結果の精度に影響を及ぼします。

構文

►► — -q —  —►

394 ページの『#pragma options』も参照してください。

注

このオプションは廃止されました。新しいアプリケーションでは、**-qfloat=maf** を使用してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

141 ページの『float』

394 ページの『#pragma options』

makedep

► C ► C++

目的

make コマンドの記述ファイルの組み込みに適したターゲットを含む出力ファイルを作成する。

構文

►► — -q—makedep—◀◀

注

-qmakedep オプションは、**-M** オプションと機能的に同じです。

.u ファイルは **make** ファイルではありません。**.u** ファイルを編集しなければ、**make** コマンドで使用することはできません。このコマンドに関する詳細については、ご使用のオペレーティング・システムの資料を参照してください。

-o オプションを指定しない場合は、**-qmakedep** オプションで生成される出力ファイルは、現行ディレクトリーに作成されます。サフィックスとして **.u** が付きます。例えば、以下のコマンドでは、

```
x1C -qmakedep person_years.C
```

出力ファイル **person_years.u** が生成されます。

.u ファイルは、**.c**、**.C**、**.cpp**、または **.i** サフィックスが付いた入力ファイルごとに作成されます。また、**-+** コンパイラ・オプションを有効にした状態で、**C++** プログラムをコンパイルすると、ファイル・サフィックスは受け入れられ、**.u** ファイルが生成されます。そうでない場合は、出力の **.u** ファイルは、他のファイルについては一切作成されません。

例えば、以下のコマンドでは、

```
x1C -qmakedep conversion.C filter.C /lib/libm.a
```

conversion.u および **filter.u** の 2 つの出力ファイル (および実行可能ファイル) が作成されます。**.u** ファイルは、ライブラリーについては作成されません。

現行ディレクトリーが書き込み可能でない場合、**.u** ファイルは作成されません。

-o file_name を **-qmakedep** とともに指定する場合、**.u** ファイルは、**-ofile_name** で暗黙指定されるディレクトリーに置かれます。例えば、以下の呼び出しでは、

```
x1C -qmakedep -c t.C -o /tmp/t.o
```

.u 出力ファイルが **/tmp/t.u** に配置されます。

出力ファイルには、入力ファイルに対する行、および各インクルード・ファイルに対する項目が入ります。この一般形式は次のとおりです。

```
file_name.o:include_file_name
file_name.o:file_name.C
```

インクルード・ファイルは、**#include** プリプロセッサ・ディレクティブの検索順序の規則に従ってリストされます。この規則については、43 ページの『相対パス名を使用したインクルード・ファイルのディレクトリ検索順序』で説明しています。インクルード・ファイルが見つからない場合、**.u** ファイルには追加されません。

include 文がないファイルについては、入力ファイル名のみをリストする 1 行を含む出力ファイルが生成されます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

226 ページの『M』

248 ページの『o』

43 ページの『相対パス名を使用したインクルード・ファイルのディレクトリ検索順序』

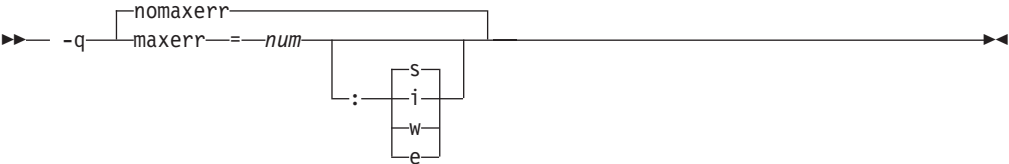
maxerr

C C++

目的

指定した重大度レベルまたはそれより高い重大度レベルのエラーの件数が *num* に達したときに、コンパイルを停止するよう、コンパイラーに指示する。

構文



ここで、*num* は整数でなければなりません。重大度レベルは、以下のいずれかを選択できます。

<i>sev_level</i>	説明
i	通知
w	警告
e	エラー (C のみ)
s	重大エラー

注

重大度レベルを指定していない場合は、**-qhalt** オプションの現行の値が使用されます。

-qmaxerr オプションを複数回指定した場合は、最後に指定した **-qmaxerr** オプションによって、オプションの処理が決まります。**-qmaxerr** および **-qhalt** オプションの両方を指定した場合は、最後に指定した **-qmaxerr** または **-qhalt** オプションによって、**-qmaxerr** オプションで使用する重大度レベルが決まります。

エラーの数が指定した制限に達すると、回復不能エラーが起こります。出されるエラー・メッセージは以下のようなものです。

1506-672 (U) The number of errors has reached the limit of ...

-qnomaxerr を指定すると、検出されたエラーの数に関係なく、ソース・ファイル全体がコンパイルされます。

診断メッセージは、**-qflag** オプションで制御することができます。

例

- 10 件の警告が検出されたときに myprogram.c のコンパイルを停止するには、以下を入力します。

```
xlc myprogram.c -qmaxerr=10:w
```

2. 現行の **-qhalt** オプション値が **S** (重大) であるとして、5 件の重大エラーが検出されたときに `myprogram.c` のコンパイルを停止するには、以下のコマンドを入力します。

```
xlc myprogram.c -qmaxerr=5
```

3. 3 件の通知メッセージが出された場合に `myprogram.c` のコンパイルを停止するには、以下のコマンドを入力します。

```
xlc myprogram.c -qmaxerr=3:i
```

または

```
xlc myprogram.c -qmaxerr=3 -qhalt=i
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

139 ページの『flag』

158 ページの『halt』

457 ページの『メッセージ重大度レベルとコンパイラー応答』

maxmem

► C ► C++

目的

メモリーを大量に消費する特定の最適化のローカル・テーブルに最適化プログラムが使用するメモリーの量を制限します。メモリー・サイズの制限はキロバイトで指定されます。

構文

►► `-qmaxmem=size` ◀◀

デフォルト

- -O2 最適化が有効な場合、maxmem=8192。
- -O3 以上の最適化が有効な場合、maxmem=-1。

注

- *size* の値を -1 にすると、制限について検査されることなく、最適化ごとに必要な量のメモリーが使用できるようになります。コンパイル中のソース・ファイル、ソース内のサブプログラムのサイズ、マシン構成、およびシステムのワークロードによっては、この量が使用可能なシステム・リソースを超えてしまう場合があります。
- **-qmaxmem** によって設定される制限は、コンパイラー全体ではなく、特定の最適化のためのメモリーの量です。コンパイル処理全体で必要となるテーブルは、この制限の影響を受けないため、この制限には含まれません。
- コンパイラーが制限以下のメモリーしか必要としない場合は、大きい制限を設定してもソース・ファイルのコンパイルに悪影響は及びません。
- 最適化の範囲を制限しても、結果として得られるプログラムが必ずしも遅くなるわけではありません。単に、パフォーマンスを向上させるすべての機会を検出する前にコンパイラーが終了する場合があります。
- 制限を増加させても、結果として得られるプログラムが必ずしも高速になるわけではありません。単に、パフォーマンスを向上させる機会がある場合にコンパイラーがその機会を検出しやすくなるだけです。

コンパイル中のソース・ファイル、ソース内のサブプログラムのサイズ、マシン構成、およびシステムのワークロードによっては、制限を高く設定しすぎるとページ・スペースがすべて使用されてしまう場合があります。特に、**-qmaxmem=-1** を指定すると、コンパイラーがストレージを無制限に使用できるようになるので、最悪の場合には、最も優れた装備を持つマシンのリソースでさえも使用し尽くしてしまうおそれがあります。

例

ローカル・テーブルに指定するメモリーが **16384** キロバイトになるように、myprogram.C をコンパイルするには、以下のように入力します。

```
xlc myprogram.C -qmaxmem=16384
```


関連資料

61 ページの『コンパイラーのコマンド行オプション』

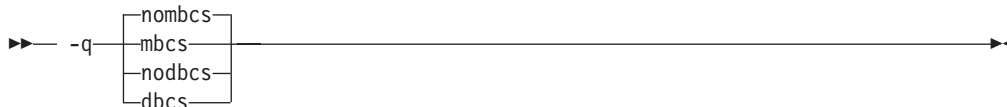
mbcs、dbcs

➤ C ➤ C++

目的

プログラムにマルチバイト文字が含まれる場合には、**-qmbcs** オプションを使用する。**-qmbcs** オプションは **-qdbcs** と同等です。

構文



394 ページの『`#pragma options`』も参照してください。

注

マルチバイト文字は、中国語、日本語、韓国語などの特定の言語で使用されます。

-qmbcs または **-qdbcs** コンパイラー・オプションを指定した場合、マルチバイト文字はコメントでも許可されます。

ソース・ファイルがマルチバイト文字リテラルを含み、デフォルトの **-qnombcs** または **-qnodbcs** コンパイラー・オプションが有効な場合、コンパイラーはすべてのリテラルを単一バイト・リテラルとして扱います。

例

myprogram.c がマルチバイト文字を含む場合、これをコンパイルするには、次のように入力します。

```
xlc myprogram.c -qmbcs
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

394 ページの『`#pragma options`』

493 ページの『付録 C. XL C/C++ Enterprise Edition における各国語サポート』

mkshrobj

➤ C ➤ C++

目的

生成されたオブジェクト・ファイルから共用オブジェクトを作成する。

構文

```
➤➤ -q—mkshrobj [—priority]
```

ここで、*priority* はファイル内のすべての静的オブジェクトの初期化の優先順位を指定します。*priority* は、-214782623 (優先順位が最も高い - 最初に初期化される) から 214783647 (優先順位が最も低い - 最後に初期化される) までのいずれかの番号になります。-214783648 から -214782624 までの数字は、システム使用のために予約されています。優先順位が指定されていない場合は、デフォルトの優先順位 0 が使用されます。C で書かれている共用オブジェクトをリンクする (**xlc** コマンドを使用) 場合は、優先順位は使用されません。

注

このオプションは、後述する関連オプションとともに、共用オブジェクトを作成するために **makeC++SharedLib** コマンドの代わりに使用しなければなりません。このオプションを使用する利点は、コンパイラーが **tempinc** ディレクトリー内のテンプレートのインスタンス生成を自動的に組み込んでコンパイルすることです。

以下の表では、**makeC++SharedLib** と **-qmkshrobj** との同等オプションを示します。

makeC++SharedLib オプション	-qmkshrobj および関連オプション
-p <i>nnn</i>	-qmkshrobj= <i>nnn</i>
-e <i>file_name</i>	-qexpfile= <i>file_name</i>
-E <i>export_file</i>	-bE: <i>export_file</i>
-I <i>import_file</i>	-bI: <i>import_file</i>
-x	-qnolib
-x 32	-q32
-x 64	-q64
-n <i>entry_point</i>	-e <i>entry_point</i>

コンパイラーは、**-bE:**、**-bexport:**、または **-bexpall** オプションでエクスポートするシンボルを明示的に指定しない限り、共用オブジェクトからのすべてのグローバル・シンボルを自動的にエクスポートします。

priority サブオプションは、C コンパイラーを使用してリンクする場合、または共用オブジェクトに静的初期設定がない場合は、効果がありません。

-qmkshrobj を指定すると、**-qplic** を暗黙指定します。

また、以下の関連オプション は、**-qmkshrobj** コンパイラー・オプションと共に使用できます。

-o <i>shared_file</i>	共用ファイルの情報を保持するファイルの名前。デフォルトは shr.o です。
-qexpfile= <i>filename</i>	すべてのエクスポートされたシンボルを <i>filename</i> に保管します。 このオプションは、xLC が自動的にエクスポート・リストを作成しない限り、無視されます。
-e <i>name</i>	共用実行可能ファイルの項目名を <i>name</i> に設定する。デフォルトは -enentry です。

-qmkshrobj を使用して共用ライブラリーを作成する場合は、コンパイラーは以下のように行います。

1. ユーザーが **-bE:**、**-bexport:**、**-bexpall**、または **-bnoexpall** を指定しない場合は、CreateExportList コマンド (以下で説明します) を使用して、すべてのグローバル・シンボルを含むエクスポート・リストを作成します。 **-tE/-B** または **-qpath=E:** オプションを使って別のスクリプトを指定することができます。
2. エクスポート・リストの作成に CreateExportList ユーティリティーが使用されていたか、**-qmkshrobj** および **-qexpfile** コンパイラー・オプションが使用されていた場合、エクスポート・リストを保管してください。
3. 適切なオプションおよびオブジェクト・ファイルを使ってリンカーを呼び出し、共用オブジェクトをビルドします。

CreateExportList ユーティリティーおよび共用オブジェクトの作成について詳しくは、「プログラミング・ガイド」の『ライブラリーの作成』を参照してください。

関連資料

- 61 ページの『コンパイラーのコマンド行オプション』
- 73 ページの『32、64』
- 94 ページの『b』
- 128 ページの『e』
- 134 ページの『expfile』
- 248 ページの『o』
- 256 ページの『path』
- 263 ページの『pic』
- 266 ページの『priority』
- 406 ページの『#pragma priority』

また、「XL C/C++ プログラミング・ガイド」のセクション『ライブラリーの作成』を参照してください。

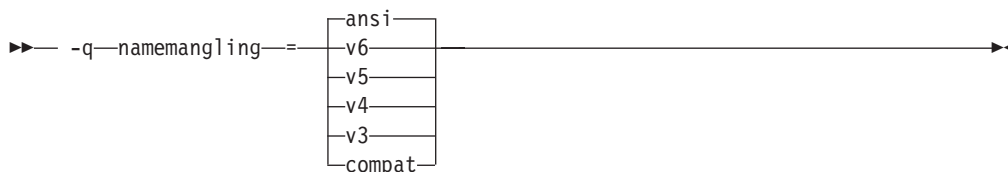
namemangling

➤ C++

目的

C++ ソース・コードから生成した外部シンボル名のネーム・マングリング方式を選択する。

構文



ここで、マングリング方式に使用できる選択項目は以下のとおりです。

- | | |
|------|--|
| ansi | ネーム・マングリング方式は、標準 C++ の各種の言語フィーチャーを、関数テンプレートの多重定義も含めて、完全にサポートしています。 |
| v6 | ネーム・マングリング方式は、VisualAge C++ バージョン 6.0 との互換性があります。 |
| v5 | ネーム・マングリング方式は、VisualAge C++ バージョン 5.0 との互換性があります。 |
| v4 | ネーム・マングリング方式は、VisualAge C++ バージョン 4.0 との互換性があります。 |
| v3 | この方式は、VisualAge C++ のバージョン 4.0 より前にリリースされたバージョンで作成したリンク・モジュール、あるいは #pragma namemangling または -qnamemangling=compat コンパイラー・オプションを指定して作成されたリンク・モジュールとの互換性のために、使用してください。 |

関数の名前および関数仮パラメーター・リストが、関数テンプレートの特異化と同じである場合、この方式は使用できません。例えば、以下の場合に **-qnamemangling=compat** を使用可能にすると、XL C/C++ は診断メッセージを発行します。

```
int f(int) {
    return 42;
}

template < class T > int f(T) {
    return 43;
}

int main() {
    f < int > (3); // instantiate int f < int > (int)
    return f(4);
}
```

戻されるエラー・メッセージは以下のようなものです。

```
(S) The definitions of "int f < int > (int)" and
    "int f(int)" have the same linkage signature "f__Fi".
```

compat 上記の **v3** サブオプションと同様。

389 ページの『#pragma namemangling』も参照してください。

注

同じ関数仮パラメーター・タイプの反復パラメーターをマングリングする関数仮パラメーター・リストは、**-qnamemangling=ansi** オプションにより変更されました。**-qnamemangling=ansi** の方式変更により、反復関数仮パラメーターは、同じタイプの有無を判別するための比較時に、トップレベルの `cv` 修飾子を無視するようになりました。同じ関数仮パラメーター・タイプの反復パラメーターは、以下のようにより圧縮エンコード方式によりマングルされます。

```
<parameter> ::= T <param number> [_]    #single repeat of a previous parameter
               ::= N <repetition digit> <param number> [_]    #2 to 9 repetitions
```

ここで、

<code><param number></code>	反復された、前のパラメーター番号を示します。 <code><param number></code> に複数の数字が含まれている場合には、 <code>'_'</code> が後に続きます。
<code><repetition digit></code>	1 より大で 10 より小でなければなりません。引き数が 9 より多く反復される場合、この規則は複数回適用されます。例えば、パラメーター 1 と同じ 38 パラメーターのシーケンスは、 <code>"N91N91N91N91N21"</code> にマングルされます。

以前には、non-type 整数テンプレート引き数のマングリングは、32 ビットの符号なし 10 進数として書き込まれ、`'SP'` というプレフィックスが付きました。この方式で 64 ビット値をマングリングする際に生じるあいまいさのため、この方式は、以下の例で示すように **-qnamemangling=ansi** に変更されました。

```
<non-type template argument> ::= SM          #single repeat of a previous parameter
                               ::= SP <number>  #positive internal argument
                               ::= SN <number>  #negative internal argument
```

non-type 整数テンプレート引き数が正の場合、数値にはプレフィックス `SP` が付きます。non-type 整数テンプレート引き数が負の場合、数値にはプレフィックス `SN` が付き、10 進数は負符号 (-) なしで書き込まれます。表示可能な 10 進数の範囲には制限がありません。

これにより、32 ビット数で表すことのできない正整数のほか、`INT_MIN` 以外の負の non-type 整数引き数を使用するテンプレートとのバイナリー互換性が断ち切られます。バイナリー互換性を保持するには、**-qnamemangling=v6** または **#pragma namemangling(v6)** を指定することによって、この変更以前に使用していたマングリング方式を使用するようにコンパイラーに指示することができます。

#pragma namemangling ディレクティブを使用して、ヘッダー・ファイルをネーム・マングリング方式の変更から保護します。例えば、特定のヘッダー・ファイルが `v5` ネーム・マングリング方式を常に使用するようにするには、ヘッダー・ファイルの先頭に **#pragma namemangling(v5)** を挿入し、ファイルの最後に **#pragma namemangling(pop)** を追加します。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

389 ページの『#pragma namemangling』

391 ページの『#pragma nameManglingRule』

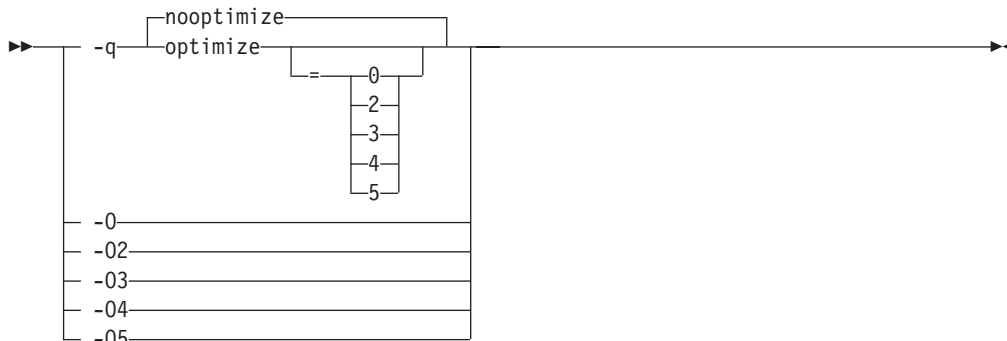
0、optimize



目的

コンパイル中に、選択したレベルでコードを最適化する。

構文



ここで、最適化の設定は以下のとおりです。

-qNOOPTimize -qOPTimize=0	<p>定数の折り込みおよびローカルな共通部分副次式の除去などの、高速でローカルな最適化のみを実行します。</p> <p>この設定は、-qnostrict_induction を明示的に指定していない限り、-qstrict_induction を暗黙指定します。</p>
-O -qOPTimize	<p>コンパイル速度と実行時のパフォーマンスに最適な組み合わせであるとコンパイラ開発者が考えた最適化を実行します。最適化は、製品のリリースによって異なる場合があります。特定のレベルの最適化が必要な場合は、適切な数値を指定してください。</p> <p>この設定は、-qnostrict_induction または -qnostrict によって明示的に否定されない限り、-qstrict と -qstrict_induction を暗黙指定します。</p>
-O2 -qOPTimize=2	<p>-O と同じ。</p>
-O3 -qOPTimize=3	<p>メモリー、コンパイル時間、またはこれら両方を大量に消費する追加の最適化を実行します。このレベルは、コンパイル・リソースの最小化よりも実行時の向上を図りたい場合に有効です。</p> <p>-O3 は -O2 レベルの最適化に適用しますが、時間およびメモリーが無制限です。-O3 はまた、プログラムのセマンティクスを多少変更する可能性がある、より積極的な最適化を実行します。コンパイラは、-O2 ではこれらの最適化を実行しません。</p> <p>-O3 を指定して -qstrict オプションを使用し、プログラムのセマンティクスを変更する可能性がある積極的な最適化をオフにしてください。 -qstrict を -O3 とともに指定すると、 -O2 で実行されるすべての最適化に加えて、さらにループの最適化も行われます。 -qstrict コンパイラ・オプションは、-O3 オプションの後に指定しないと無視されます。</p>

<p>-O3 -qOPTimize=3 (続き)</p>	<p>-O3 を指定したときに実行される積極的な最適化は、以下のとおりです。</p> <p>1. 積極的なコードの移動、および例外を発生させる可能性がある計算に関するスケジューリングが許可されます。</p> <p>ロードおよび浮動小数点計算は、このカテゴリーに分類されます。この最適化が積極的なのは、命令がプログラムの実際のセマンティクスに一致していない可能性があるときに命令を実行する実行パスに、それらの命令を配置することがあるためです。</p> <p>例えば、ループ内で不変の浮動小数点計算がループを通るいくつかのパスで見つかったとしても、すべてのパスを通らない場合は、その計算が例外を発生させる場合があるため、-O2 では移動されません。-O3 では、例外が発生することが確実ではないので、コンパイラーはその計算を移動させます。ロードの移動についても同じです。ポインターを介したロードが移動されることはありませんが、静的またはスタック基底レジスターを介さないロードは、-O3 では移動可能であると見なされます。プログラムに 10 個の要素がある静的配列 <code>a</code> の宣言を入れて、<code>a[600000000003]</code> をロードすると、セグメント違反が発生する可能性があるため、一般に、-O2 でのロードは絶対に安全であるとはいえません。</p> <p>同様の概念がスケジューリングにも適用されます。</p> <p>例:</p> <p>以下の例において、-O2 では、<code>b+c</code> の計算は、以下の 2 つの理由でループからは移動されません。</p> <ul style="list-style-type: none"> 浮動小数点演算であるため、危険であると見なされる。 ループを通るすべてのパスで行われるわけではない。 <p>-O3 では、コードは移動されます。</p> <pre> ... int i ; float a[100], b, c ; for (i = 0 ; i < 100 ; i++) { if (a[i] < a[i+1]) a[i] = b + c ; } ... </pre> <p>2. IEEE 規則に対する適合性が緩和されます。</p> <p>-O2 では、ある特定の最適化は実行されません。これは、そのような最適化によって、結果がゼロの場合に誤った符号が生成されたり、なんらかのタイプの浮動小数点例外を発生させる可能性のある算術演算が除去されるためです。</p> <p>例えば、<code>X + 0.0</code> は <code>X</code> には折り畳まれません。これは、IEEE 規則では <code>-0.0 + 0.0 = 0.0</code> であり、これは <code>-X</code> になるからです。他の例として、最適化の中には、符号が誤ったゼロの結果を生成する最適化を実行するものもあります。例えば、<code>X - Y * Z</code> の結果は <code>-0.0</code> になります。これは、元の計算では <code>0.0</code> になります。</p> <p>ほとんどの場合、結果の相違はアプリケーションにとって重要ではないため、-O3 ではこれらの最適化が許可されます。</p> <p>3. 浮動小数点式が書き換えられる場合があります。</p> <p>再配置によって共通部分副次式を抽出する機会が得られる場合などには、<code>a*b*c</code> などの計算を <code>a*c*b</code> に書き換える場合があります。浮動小数点計算の再配置の別の例として、除法を逆数による乗算で置き換えることが挙げられます。</p>
--------------------------------------	--

<p>-O3 -qOPTimize=3 (続き)</p>	<p>注</p> <ul style="list-style-type: none"> • -qfloat=fltint:rsqrt は、デフォルトでは -O3 で設定されます。 • -qmaxmem=1 は、デフォルトでは -O3 で設定され、コンパイラーは最適化を実行するときにメモリーを必要なだけ使うことができます。 • 組み込み関数は、-O3 では errno を変更しません。 • 積極的な最適化には、浮動小数点サブオプション -qfloat=hsflt、hssngl、-qfloat=rndsngl、またはプログラムの精度モードに影響を与える他のすべてのオプションは含まれません。 • 整数除法命令の最適化は、-O3 であっても非常に危険であると見なされます。 • デフォルトの -qmaxmem 値は、-O3 では -1 です。 • optimize オプションを flttrap オプションと一緒に指定したときのコンパイラーの動きについては、-qflttrap を参照してください。 • -qstrict および -qstrict_induction コンパイラー・オプションを使用して、プログラムのセマンティクスを変更する可能性がある -O3 の影響をオフにすることができます。-qstrict コンパイラー・オプションを有効にするには、コマンド行で -O3 オプションの後に指定する必要があります。 • -O3 コンパイラー・オプションの後に、-O オプションを指定すると、-qignerrno がオンのままになります。
<p>-O4 -qOPTimize=4</p>	<p>このオプションは -O3 と同じですが、以下の点が異なります。</p> <ul style="list-style-type: none"> • コンパイルを行うマシンのアーキテクチャーに対して、-qarch および -qtune オプションを設定する。 • コンパイル・マシンの特性に最も適した -qcache オプションを設定する。 • -qhot オプションが設定される。 • -qipa オプションが設定される。 <p>注: -O、-qcache、-qhot、-qipa、-qarch、および -qtune オプションを後で設定すると、-O4 オプションで暗黙指定された設定がオーバーライドされます。</p>
<p>-O5 -qOPTimize=5</p>	<p>このオプションは -O4 と同じですが、以下の点が異なります。</p> <ul style="list-style-type: none"> • -qipa=level=2 オプションを設定して、完全なプロシージャーク間のデータの流れおよび別名の分析を実行します。 <p>注: -O、-qcache、-qipa、-qarch、および -qtune オプションを後で設定すると、-O5 オプションによって暗黙指定された設定がオーバーライドされます。</p>

注

-qoptimize... は、**-qopt...** に省略できます。例えば、**-qnoot** は **-qnootoptimize** と同等です。

最適化のレベルを上げても、追加の分析によって最適化の機会がさらに検出されるかどうかに応じて、さらにパフォーマンスが向上する場合と向上しない場合があります。

最適化を伴うコンパイルでは、他のコンパイルよりも多くの時間およびマシンのリソースが必要になる場合があります。

最適化によってステートメントが移動または削除される場合があるため、通常は、デバッグ・プログラムのための **-g** フラグと一緒に最適化を指定しないでください。作成されるデバッグ情報が正確でなくなる場合があります。

例

myprogram.C をコンパイルして最大限に最適化させるには、以下を入力します。

```
xlc myprogram.C -O3
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

84 ページの『arch』

101 ページの『cache』

141 ページの『float』

147 ページの『flttrap』

155 ページの『g』

162 ページの『hot』

169 ページの『ignerrno』

170 ページの『ignprag』

181 ページの『ipa』

198 ページの『langlvl』

236 ページの『maxmem』

305 ページの『strict』

307 ページの『strict_induction』

327 ページの『tune』

「*Getting Started with XL C/C++ Enterprise Edition for AIX*」の『最適化について』セクションも参照してください。

目的

コンパイラによって作成されるオブジェクト・ファイル、アセンブラー・ファイル、または実行可能ファイルの出力位置を指定する。コンパイラ呼び出し中に **-o** オプションを使用するときには、*filespec* をファイルかディレクトリーのいずれかの名前にすることができます。リンケージ・エディターの直接呼び出し中に **-o** オプションを使用するときには、*filespec* はファイルの名前にしかすることができません。

構文

▶— **-o** *filespec* —▶

注

-o をコンパイラ呼び出しの一部として指定するときには、*filespec* をディレクトリーまたはファイルの相対パス名か絶対パス名にすることができます。

1. *filespec* がディレクトリーの名前である場合、コンパイラによって作成されるファイルはそのディレクトリーに置かれます。
2. *filespec* という名前のディレクトリーが存在しない場合は、**-o** オプションは、コンパイラによって作成されるファイルの名前を *filespec* と指定します。例えば、以下のコンパイラ呼び出しでは、

```
xlc test.c -c -o new.o
```

オブジェクト・ファイル **new.o** が、**test.o** の代わりに作成されます。また、以下の場合は、

```
xlc test.c -o new
```

new という名前のディレクトリーがなければ、オブジェクト・ファイル **new** が、**a.out** の代わりに作成されます。そうでない場合は、デフォルトのオブジェクト名 **a.out** が使用され、**new** ディレクトリーに配置されます。

C または C++ ソース・ファイルのサフィックス (**.C**、**.c**、**.cpp**、または **.i**) が付いた *filespec* (myprog.c、myprog.i など) はエラーとなり、コンパイラもリンケージ・エディターも呼び出されません。

-c と **-o** を一緒に使用して、*filespec* がディレクトリーを指定していない場合は、一度に 1 つのソース・ファイルしかコンパイルすることはできません。この場合、コンパイラ呼び出し時に複数のソース・ファイル名がリストされる場合、コンパイラは、警告メッセージを出して **-o** を無視します。

-E、**-P**、および **-qsyntaxonly** オプションは、**-ofilename** オプションをオーバーライドします。

例

myaccount という名前のディレクトリーが存在しないとすると、myprogram.c をコンパイルした結果として **myaccount** というファイルを作成させるには、以下のように入力します。

```
xlc myprogram.c -o myaccount
```

ディレクトリー **myaccount** が存在する場合は、コンパイラーは実行可能ファイル **a.out** を作成し、それを **myaccount** ディレクトリーに配置します。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

99 ページの『c』

126 ページの『E』

252 ページの『P』

310 ページの『syntaxonly』

objmodel

➤ C++

目的

オブジェクト・モデルの型を設定する。

構文

➡ — -qobjmodel= — classic
ibm —————▶

ここで、オブジェクト・モデルの選択項目は以下のとおりです。

-qobjmodel=classic	IBM C および C++ コンパイラーのバージョン 3.6 と互換性のあるオブジェクト・モデルを使用する。このサブオプションはまた、 -qobjmodel=compat のレガシー・サブオプション名を使用して指定されますが、このレガシー・サブオプション名のサポートはコンパイラーの今後のリリースで除去される可能性があります。
-qobjmodel=ibm	VisualAge C++ for AIX のバージョン 5.0 で導入されたオブジェクト・モデルを使用します。このオブジェクト・モデルでコンパイルされたオブジェクトは、少ないメモリーを使用し、仮想基底での深い継承に対するパフォーマンスが良好です。

393 ページの『#pragma object_model』も参照してください。

例

ibm オブジェクト・モデルを使って myprogram.C をコンパイルするには、以下を入力します。

```
xlc myprogram.C -qobjmodel=ibm
```

関連概念

6 ページの『オブジェクト・モデル』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

393 ページの『#pragma object_model』

oldpassbyvalue

➤ C++

目的

`const` または参照メンバーを含むクラスを関数引き数に渡す方法を指定する。コンパイル単位のクラスはすべて、このオプションの影響を受けます。

構文

➤ — `-q` nooldpassbyvalue
oldpassbyvalue ➤

404 ページの『`#pragma pass_by_value`』も参照してください。

注

VisualAge C++ v5.0 以降のコンパイラーは、コピー・コンストラクターおよびデストラクターが小さい場合は、`const` または参照データ・メンバーに関係なく、値による受け渡しを使用します。IBM C/C++ コンパイラー v3.6 は、クラスに `const` も参照データ・メンバーもなく、コピー・コンストラクターおよびデストラクターが小さい場合は、値による受け渡しのみを使用します。

`-qoldpassbyvalue` を指定したときには、コンパイラーは IBM C/C++ v3.6 コンパイラーと同様に、`const` または参照メンバーを含むクラスが関数引き数として渡される場合には値による受け渡しを行いません。これは、コンパイル単位のこの種のクラスすべてに適用されます。

`#pragma pass_by_value` ディレクティブは、`-qoldpassbyvalue` をオーバーライドし、これによって、このフィーチャーを使用可能にする際にこれまで以上の制御ができるようになります。詳しくは、`#pragma pass_by_value` の説明を参照してください。

IBM C/C++ コンパイラー v3.6 以前でコンパイルされるライブラリーにリンクする場合、このオプションを使用します。そうでない場合は、この種類のクラス・パラメーター・タイプを持つ関数は、そのコンパイラー以降のバージョンでコンパイルされるモジュールから呼び出されるとき、間違った振る舞いをします。これらのライブラリーのユーザーがクラス・パラメーターを使用するこれらのライブラリーの関数用の正しい呼び出し規約を取得するために、IBM C/C++ コンパイラー v3.6 以前でコンパイルされるライブラリー・ヘッダーは、`#pragma pass_by_value` ディレクティブで保護される必要があります。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

404 ページの『`#pragma pass_by_value`』

P

➤ C ➤ C++

目的

コンパイラーの呼び出し時に指定された C または C++ ソース・ファイルをプリプロセスして、プリプロセスされた出力ソース・ファイル *file_name.i* を、入力ソース・ファイル *file_name.c*、*file_name.C*、または *file_name.cpp* ごとに作成します。

構文

➤ — -P ————— ➤

注

-P オプションは、以下の例外を除き、改行文字を含むすべての空白文字を保存します。

- コメントはすべて単一の空白に縮小される (**-C** が指定されていない場合)。
- プリプロセス・ディレクティブの末尾にある改行は保存されない。
- 関数形式のマクロに対する引き数の前後にある空白文字は保存されない。

#line ディレクティブは出されません。

-P オプションは、プリプロセスされたソース・ファイル *file_name.i* などを、入力として受け入れることはできません。コンパイラーがエラー・メッセージを発行します。

認識されないファイル名サフィックスの付いたソース・ファイルは、C ファイルと見なされてプリプロセスされ、エラー・メッセージは生成されません。

拡張モードでは、以下の場合には、プリプロセッサは改行文字が後に続く円記号を行の継続として解釈します。

- マクロ置き換えテキスト
- マクロ引き数
- プリプロセッサ・ディレクティブと同じ行にあるコメント

これ以外の箇所での行の継続は、**ANSI** モードでなければ処理されません。

-P オプションは、**-E** オプションによってオーバーライドされます。**-P** オプションは、**-c**、**-o**、および **-qsyntaxonly** オプションをオーバーライドします。**-C** オプションは、**-E** および **-P** オプションの両方と一緒に使用することができます。

デフォルトでは、C または C++ のソース・ファイルがコンパイルおよびリンク・エディットされて、実行可能ファイルが作成されます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

98 ページの『C』

99 ページの『c』

126 ページの『E』

248 ページの『o』

310 ページの『syntaxonly』

p

➤ C ➤ C++

目的

コンパイラーが作成するオブジェクト・ファイルをプロファイル用に設定する。

構文

➤ — -p — ➤

注

-qtbtable オプションを設定しない場合は、**-p** オプションによって完全なトレースバック・テーブルが生成されます。

別々のステップでコンパイルおよびリンクを行うときには、両方のステップで **-p** オプションを指定しなければなりません。

例

myprogram.c をコンパイルして、オペレーティング・システムの **prof** コマンドでできるようにするには、次のように入力します。

```
xlc myprogram.C -p
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

314 ページの『tbtable』

また、Web で以下を参照してください。

プロファイルについては、「コマンド・リファレンス 第 4 巻」の『prof コマンド』の節。

pascal



目的

デフォルトの **-qnopascal** は、**pascal** が型指定子や関数宣言に表示されたら、無視するようコンパイラーに指示します。

構文

▶▶ — -q —  —▶▶

注

-qpascal コンパイラー・オプションを指定すると、型指定子および関数宣言のキーワード **pascal** を認識して受け入れるようコンパイラーに指示します。

このオプションを使用すると、他のいくつかのシステムで、IBM XL C/C++ Enterprise Edition プログラムの互換性を向上させることができます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

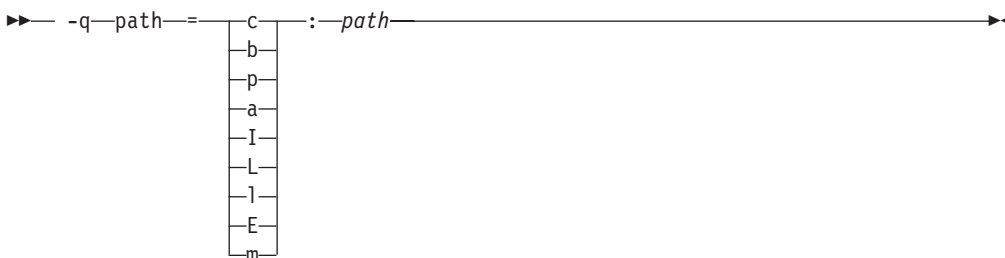
path

► C ► C++

目的

コンパイラー・コンポーネントに代替プログラム名を構成します。このオプションで指定するプログラムとディレクトリー *path* は、正規のプログラムの代わりに使用されます。

構文



ここで、プログラム名は以下のとおりです。

プログラム	説明
c	コンパイラーのフロントエンド
b	コンパイラーのバックエンド
p	コンパイラー・プリプロセッサ
a	アセンブラ
I	プロシージャ間分析ツール (コンパイル・フェーズ)
L	プロシージャ間分析ツール (リンク・フェーズ)
l	リンケージ・エディター
E	CreateExportList ユーティリティー
m	リンケージ・ヘルパー (munch ユーティリティー)

注

-qpath オプションは、**-Fconfig_file**、**-t**、および **-B** オプションをオーバーライドします。

例

/lib/tmp/mine/ にある代替の **x1C** コンパイラーを使用して **myprogram.C** をコンパイルするには、以下を入力します。

```
x1C myprogram.C -qpath=c:/lib/tmp/mine/
```

/lib/tmp/mine/ にある代替のリンケージ・エディターを使用して **myprogram.C** をコンパイルするには、次を入力します。

```
x1C myprogram.C -qpath=l:/lib/tmp/mine/
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

93 ページの『B』

136 ページの『F』

311 ページの『t』

pdf1、pdf2

► C ► C++

目的

プロファイル指示フィードバック (PDF) を使用して最適化を調整する。ここでは、サンプル・プログラムの実行から得られた結果が使用されて、条件付き分岐の付近および頻繁に実行されるコード・セクションでの最適化が改善されます。

構文



注

PDF を使用するには、以下のステップに従ってください。

1. **-qpdf1** オプションを指定して、プログラム内のソース・ファイルの一部または全部をコンパイルする。少なくとも **-O2** 最適化オプションを指定する必要があります。また同様に、少なくとも有効な **-O2** でリンクする必要もあります。後で同じオプションを使用する必要があるため、ファイルのコンパイルに使用するコンパイラ・オプションには特に注意してください。

大規模なアプリケーションでは、最適化から最も利益を得ることのできるコードの領域に集中してください。アプリケーションのコードのすべてを **-qpdf1** オプションでコンパイルする必要はありません。

2. 一般的なデータ・セットを使用してプログラムを一通り実行する。プログラムは、終了時にプロファイル情報を記録します。プログラムは、異なるデータ・セットで複数回実行することができます。また、プロファイル情報が累積されるため、分岐の頻度およびコードのブロックの実行頻度が正確にカウントされます。

重要: 終了したプログラムの通常の実行中に使用されるデータのうち、代表的なデータを使用してください。

3. 前と同じコンパイラ・オプションを使用してプログラムを再リンクする。ただし、**-qpdf1** は **-qpdf2** に変更してください。 **-L**、**-I**、およびその他いくつかのオプションはリンカー・オプションであり、それらはこの時点で変更できます。この 2 番目のコンパイルでは、累積されたプロファイル情報を使用して最適化が微調整されます。結果として得られるプログラムにはプロファイルのオーバーヘッドが含まれないため、フルスピードで実行されます。

最高のパフォーマンスを得るには、PDF を使用するとき、すべてのコンパイルで **-O3**、**-O4**、または **-O5** オプションを使用します。

プロファイルは、現行作業ディレクトリーに配置されるか、PDFDIR 環境変数が設定されている場合はその変数によって指定されたディレクトリーに配置されます。

コンパイルと実行の時間を節約するために、PDFDIR 環境変数を必ず絶対パスに設定してください。そうでない場合、誤ったディレクトリーからアプリケーションが実行され、プロファイル・データ・ファイルが見つからなくなる可能性があります。これが起こると、プログラムが正しく最適化されないか、またはセグメンテー

ション障害によって停止する場合があります。PDF プロセスの終了前に PDFDIR 変数の値を変更してアプリケーションを実行した場合にも、セグメンテーション障害が起こる場合があります。

このオプションではアプリケーション全体を 2 回コンパイルする必要があるため、他のデバッグおよび調整が終了した後の、アプリケーションを実動に供する前の最後のステップの 1 つとして使用するようにしてください。

制約事項

- PDF 最適化には、少なくとも **-O2** の最適化レベルが必要です。
- 実行時にプロファイル情報を収集させるには、メイン・プログラムを PDF を使用してコンパイルしなければなりません。
- プロファイル情報のセットを区別するために **-qipa=pdfname** サブオプションを使用していない限り、同じ PDFDIR ディレクトリーを使用する 2 つの異なるアプリケーションを同時にコンパイルまたは実行しないでください。
- 特定のプログラムのすべてのコンパイル・ステップで、同じコンパイラ・オプションのセットを使用しなければなりません。そうでない場合、PDF がプログラムを正しく最適化できないだけでなく、プログラムの速度が遅くなる場合もあります。構成ファイルによって提供されるすべての設定も含めて、コンパイラ設定はすべて同じでなければなりません。
- 現行バージョン・レベルの XL C/C++ Enterprise Edition によって作成された PDF ファイルと、別のバージョンのコンパイラによって作成された PDF ファイルとを混合しないでください。
- **-qipa** が直接的にも別のオプションを介しても呼び出されなかった場合、**-qpdf1** と **-qpdf2** が **-qipa=level=0** オプションを呼び出します。
- **-qpdf1** でプログラムをコンパイルすると、実行時にプロファイル情報が生成され、パフォーマンス上のオーバーヘッドをいくらか伴うので注意してください。このオーバーヘッドは、**-qpdf2** で再コンパイルするか、PDF なしで再コンパイルするとなくなります。

以下のユーティリティー・プログラムは、**/usr/xlopt/bin** に存在し、PDFDIR ディレクトリーの管理に使用できます。

cleanpdf cleanpdf [*pathname*]

pathname ディレクトリーからプロファイル情報をすべて除去します。 *pathname* を指定しない場合は **PDFDIR** ディレクトリーから、 **PDFDIR** を設定していない場合は現行ディレクトリーから除去します。プログラムを変更してから PDF 処理を再度行う場合は、プロファイル情報を除去すると実行時のオーバーヘッドが減少します。

特定のアプリケーション用 PDF 処理を終了した場合にのみ、**cleanpdf** を実行します。そうでない場合、そのアプリケーションで PDF の使用を再開したい場合は、**-qpdf1** で再度すべてのファイルを再コンパイルする必要があります。

mergepdf mergepdf [-r *scaling*] input {[-r *scaling*] input} ... -o *output* [-n] [-v]

2 つ以上の PDF レコードを単一 PDF 出力レコードにマージします。

- r *scaling*** PDF レコード・ファイルのスケーリング率を指定します。この値は、ゼロ以上でなければならない、整数または浮動小数点値のいずれかにできます。指定されていない場合は、比率は 1.0 と見なします。
- record*** PDF 入力レコード・ファイル、または PDF レコード・ファイルを含むディレクトリーの名前を指定します。
- o *output*** PDF 出力レコード・ファイル、またはマージされる出力が書き込まれるディレクトリーの名前を指定します。
- n** 指定されると、PDF レコード・ファイルは正規化されません。指定されない場合は、**mergepdf** が、ユーザー定義の位取り係数を適用する前に、内部で計算した比率に基づいてレコードを正規化します。
- v** 冗長モードを指定し、内部およびユーザー指定のスケーリング率をスクリーンに表示します。

resetpdf resetpdf [*pathname*]

上記の **cleanpdf** [*pathname*] と同様。

showpdf showpdf

プログラムの実行中に実行されたすべてのプロシーチャーの呼び出しおよびブロック数を表示します。このコマンドを使用するには、最初にコマンド行で **-qpdf1** および **-qshowpdf** コンパイラー・オプションを両方とも指定して、使用しているアプリケーションをコンパイルする必要があります。

例

簡単な例を以下に示します。

```
/* Set the PDFDIR variable.                */
export PDFDIR=$HOME/project_dir

/* Compile all files with -qpdf1.          */
x1C -qpdf1 -O3 file1.C file2.C file3.C

/* Run with one set of input data.        */
a.out <sample.data

/* Recompile all files with -qpdf2.        */
x1C -qpdf2 -O3 file1.C file2.C file3.C

/* The program should now run faster than
   without PDF if the sample data is typical. */
```

具体的な例を以下に示します。

```
/* Set the PDFDIR variable.                */
export PDFDIR=$HOME/project_dir

/* Compile most of the files with -qpdf1.  */
x1C -qpdf1 -O3 -c file1.C file2.C file3.C

/* This file is not so important to optimize.
```

```
x1C -c file4.C

/* Non-PDF object files such as file4.o can be linked in. */
x1C -qpdf1 file1.o file2.o file3.o file4.o

/* Run several times with different input data. */
a.out <polar_orbit.data
a.out <elliptical_orbit.data
a.out <geosynchronous_orbit.data

/* No need to recompile the source of non-PDF object files (file4.C). */
x1C -qpdf2 -O3 file1.C file2.C file3.C

/* Link all the object files into the final application. */
x1C file1.o file2.o file3.o file4.o
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

181 ページの『ipa』

244 ページの『O、optimize』

290 ページの『showpdf』

pg

► C ► C++

目的

プロファイル用にオブジェクト・ファイルを設定する。**-p** オプションよりも多くの情報が提供されます。

-qtbtable オプションを設定しない場合は、**-pg** オプションによって完全なトレースバック・テーブルが生成されます。

構文

►► — -pg ————— ◀◀

例

ご使用のオペレーティング・システムの **gprof** コマンドで使用するために `myprogram.c` をコンパイルするには、次のように入力します。

```
xlc myprogram.c -pg
```

コンパイルおよび リンクするには、必ず **-pg** オプションを指定してください。例を以下に示します。

```
xlc myprogram.c -pg -c  
xlc myprogram.o -pg -o program
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

314 ページの『ttable』

また、Web で以下を参照してください。

プロファイルについては、「コマンド・リファレンス 第 2 巻」の『gprof コマンド』の節。

phsinfo

► C ► C++

目的

各コンパイル・フェーズでかかった時間を報告する。フェーズ情報は標準出力に送られます。

構文

►► -q ►►

注

出力は、各フェーズごとに *number1/number2* の形式を取ります。ここで、*number1* は、コンパイラーによって使用された CPU 時間を表し、*number2* は、コンパイラー時間と、CPU がシステム呼び出しの処理に費やす時間の合計を表します。

例

myprogram.C をコンパイルして、コンパイルの各フェーズでかかった時間を報告させるには、以下のように入力します。

```
xlc myprogram.C -qphsinfo
```

出力は以下のようになります。

```
Front End - Phase Ends; 0.004/ 0.005
W-TRANS  - Phase Ends; 0.010/ 0.010
OPTIMIZ   - Phase Ends; 0.000/ 0.000
REGALLO   - Phase Ends; 0.000/ 0.000
AS         - Phase Ends; 0.000/ 0.000
```

-O4 で同じプログラムをコンパイルすると、次のようになります。

```
Front End - Phase Ends; 0.004/ 0.006
IPA       - Phase Ends; 0.040/ 0.040
IPA       - Phase Ends; 0.220/ 0.280
W-TRANS   - Phase Ends; 0.030/ 0.110
OPTIMIZ   - Phase Ends; 0.030/ 0.030
REGALLO   - Phase Ends; 0.010/ 0.050
AS        - Phase Ends; 0.000/ 0.000
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

pic

► C ► C++

目的

共用ライブラリーでの使用に適した位置独立コードを生成するようにコンパイラーに指示する。

構文

►► -q-pic [=-small] [=-large] ►►

ここで、

- | | |
|-------|--|
| small | 目次のサイズが 64 Kb 以下であると想定するようにコンパイラーに指示します。 |
| large | サイズが 64 Kb よりも大きい目次を許可します。これにより、テーブルにより多くのアドレスを保管できます。通常、このオプションを指定して生成されたコードは、 -qpica=small を指定して生成されたコードよりも大きくなります。 |

注

-qpica がサブオプションなしで指定された場合、**-qpica=small** が想定されます。

-G または **-qmkaahrobj** コンパイラー・オプションが指定された場合、**-qpica** オプションが暗黙指定されます。

-qpica=large を指定すると、**-bbigtoc** を **ld** に渡す場合と同じ効果があります。

例

共用ライブラリー **libmylib.so** をコンパイルするには、次のコマンドを使用します。

```
xlc mylib.c -qpica=small -Wl, -shared, -soname="libmylib.so.1" -o libmylib.so.1
```

-shared および **-soname** オプションについての詳細は、ご使用のオペレーティング・システムの資料の **ld** コマンドを参照してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

73 ページの『32、64』

154 ページの『G』

239 ページの『mkshrobj』

prefetch

► C ► C++

目的

コンパイルされたコード内で、`dcbt` および `dcbz` などのプリフェッチ命令の生成を使用可能にします。

構文

►► — `-q` — `prefetch` — `noprefetch` — ◀◀

注

デフォルトでは、コンパイラーは、コンパイルされたコードにプリフェッチ命令を挿入することがあります。 **-qnoprefetch** オプションを使用すると、このフィーチャーを使用不可にすることができます。

-qnoprefetch オプションは、`__prefetch_by_stream()` などの組み込み関数がプリフェッチ命令を生成することを防ぎません。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

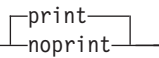
print

► C ► C++

目的

リストを使用可能にするか、または抑制する。 **-qnoprint** を指定すると、すべてのリスト作成オプションが、指定された位置に関係なくオーバーライドされ、リストが抑制されます。

構文

►► -q  ►►

注

デフォルトの **-qprint** では、他のコンパイラー・オプションによって要求された場合にリストが使用可能になります。これらのオプションは以下のとおりです。

- -qattr
- -qlist
- -qlistopt
- -qsource
- -qxref

例

いくつかのファイルに **#pragma options source** および類似したディレクティブがある場合でも `myprogram.C` をコンパイルしてすべてのリストを抑制するには、以下を入力します。

```
xlc myprogram.c -qnoprint
```

関連資料

- 61 ページの『コンパイラーのコマンド行オプション』
- 92 ページの『attr』
- 221 ページの『list』
- 222 ページの『listopt』
- 296 ページの『source』
- 346 ページの『xref』

priority

➤ C++

目的

静的オブジェクトを初期化する場合の優先順位を指定します。

構文

➤ — -q—priority—=—number— ➤

406 ページの『#pragma priority』および 394 ページの『#pragma options』も参照してください。

注

number ファイル内の静的オブジェクトに割り当てられた初期化の優先順位、あるいは共用または非共用のファイルやライブラリーの優先順位です。

優先順位には、-(2147483647 + 1) (最も高い優先順位) から +2147483647 (最も低い優先順位) までを指定できます。

指定しない場合、デフォルト優先順位は 0 になります。

例

ファイル myprogram.C をコンパイルしてオブジェクト・ファイル myprogram.o を作成し、そのファイル内のオブジェクトの初期化の優先順位を 2000 にするには、次のように入力します。

```
xlc myprogram.C -c -qpriority=2000
```

異なる優先順位レベルを指定する **#pragma priority(*number*)** ディレクティブがソース・ファイルに含まれていない場合は、結果として得られるオブジェクト・ファイル内のすべてのオブジェクトの初期化の優先順位に 2000 が指定されます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

394 ページの『#pragma options』

406 ページの『#pragma priority』

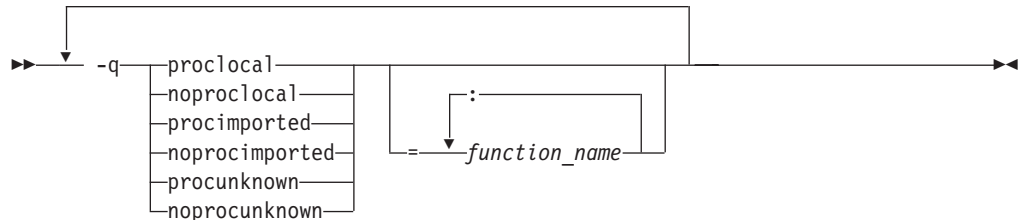
proclocal、procimported、procunknown

► C ► C++

目的

関数をローカル、インポートされるもの、または不明としてマークします。

構文



394 ページの『#pragma options』も参照してください。

デフォルト

デフォルトでは、定義が現行のコンパイル単位内にある関数はすべてローカル **proclocal** であり、その他の関数はすべて不明 **procunknown** であると見なします。ローカルとしてマークされた関数が共用ライブラリー関数に解決される場合は、リンケージ・エディターは、エラーを検出して警告を出します。

注

使用できるサブオプションは以下のとおりです。

ローカルな関数	ローカル関数は、それを呼び出す関数と静的にバインドされます。 -qproclocal を指定すると、すべての関数をローカルであると見なすようにデフォルトが変更されます。 -qproclocal=names は、指定された関数をローカルとしてマークします。ここで、 <i>names</i> は、コロン (:) で区切った関数 ID のリストです。デフォルトは変更されません。
---------	--

ローカルとしてマークされた関数への呼び出しに対しては、サイズがより小さくて高速なコードが生成されます。

インポートされる関数 インポートされる関数は、ライブラリーの共用部分と動的にバインドされます。 **-qprocimported** は、すべての関数がインポートされるものと見なすようにデフォルトを変更します。 **-qprocimported=names** を指定すると、指定された関数がインポートされるものとしてマークされます。ここで、*names* は、コロン (:) で区切った関数 ID のリストです。デフォルトは変更されません。

インポートされるものとしてマークされた関数の呼び出しのために生成されるコードは、不明としてマークされた関数のために生成されるデフォルトのコード・シーケンスよりサイズが大きくなる場合がありますが、高速になります。マークされた関数が、静的にバインドされたオブジェクトに解決される場合は、生成されるコードは、不明の関数のために生成されるデフォルトのコード・シーケンスよりサイズが大きく、実行が遅くなる場合があります。

不明の関数 不明の関数は、リンク・エディット中に、静的または動的のいずれかでバインドされたオブジェクトに解決されます。 **-qprocunknown** を指定すると、すべての関数を不明であると見なすようにデフォルトが変更されます。 **-qprocunknown=names** は、指定された関数を不明としてマークします。ここで、*names* は、コロン (:) で区切った関数 ID のリストです。デフォルトは変更されません。

▶ C++ C++ プログラムでは、関数の名前 は、マングルされた名前を使用して指定する必要があります。

プロシージャをマークするオプションの間の矛盾は、以下の方法で解決されます。

関数名をリストするオプション	特定の関数名に対する最後の明示的指定が使用される。
デフォルトを変更するオプション	この形式は、名前リストを指定しません。最後に指定されたオプションが、名前リスト・フォームに明示的にリストされていない関数に対するデフォルトとなります。

例

1. myprogram.c をアーカイブ・ライブラリー **oldprogs.a** とともにコンパイルして、以下のようにするには、
 - 関数 **fun** および **sun** を、ローカルと指定する。
 - 関数 **moon** および **stars** を、インポートされるものとして指定する。さらに、
 - 関数 **venus** を不明と指定する。

以下のように入力します。

```
xlc myprogram.c oldprogs.a -qprolocal=fun(int):sun()  
-qprocimported=moon():stars(float) -qprocunknown=venus()
```

2. 次の例は、ローカルとしてマークされた関数が、ローカルではなく共用ライブラリー関数に解決されたときに、結果として表示される一般的なエラー・メッセージです。


```
int main(void)
{
    printf("Just in function fool()¥n");
    printf("Just in function fool()¥n");
}

ld: 0711-768 WARNING: Object t.o, section 1, function .printf:
The branch at address 0x18 is not followed by a recognized no-op or
TOC-reload instruction. The unrecognized instruction is 0x83E1004C.
```

実行可能ファイルが作成されますが、動作はしません。エラー・メッセージには、オブジェクト・ファイル **t.o** 内の **printf** の呼び出しで問題が発生したことが示されます。呼び出されるルーチンが共用オブジェクトからインポートされるものであることを確認したら、警告の原因となったソース・ファイルを再コンパイルして、インポートされるものとして **printf** を明示的にマークしてください。例を以下に示します。

```
xlc -c -qprocimported=printf t.c
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

proto



目的

このオプションが設定されると、コンパイラーは、すべての関数がプロトタイプ化されているものと想定する。

構文

►► `-q` noprototype ►►

注

このオプションは、プロシージャーがプロトタイプ宣言されていない場合でも、プロシージャー呼び出し点がその宣言に一致することを明示します。

呼び出し元は、汎用レジスター (GPR) ではなく、浮動小数点レジスターのみで浮動小数点引き数を渡すことができます。コンパイラーは、プロシージャー呼び出し時の引き数の型が、プロシージャー定義の対応するパラメーターと同じであると想定します。

コンパイラーはプロトタイプを持たない関数については、警告を出します。

例

`my_c_program.c` をコンパイルして全関数がプロトタイプ宣言されていることを想定させるには、以下を入力します。

```
xlc my_c_program.c -qproto
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

Q

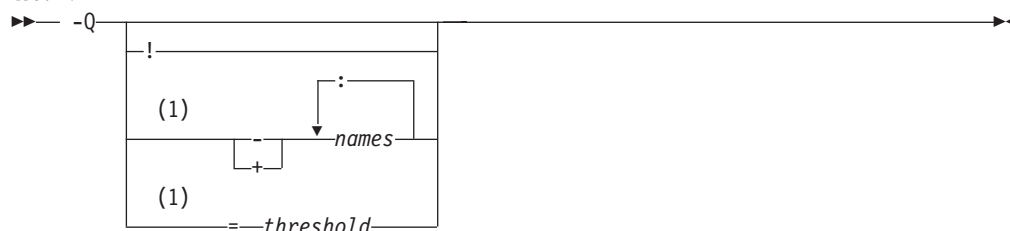
► C ► C++

目的

C++ 言語アプリケーションでは、このオプションは関数のインラインを試行するようコンパイラーに指示します。インラインは可能であれば実行されますが、実行する最適化によってはインラインされない関数もあります。

C 言語アプリケーションでは、このオプションは、コンパイラーがインラインを試行する必要がある特定の関数を指定します。

構文



注:

1 C のみ

► C++ C++ 言語では、以下の **-Q** オプションが適用されます。

- Q コンパイラーは、可能な関数をすべてインラインします。
- Q! コンパイラーは関数を一切インラインしません。

► C C 言語では、以下の **-Q** オプションが適用されます。

- Q **-Q** オプションに対する任意のサブオプションの設定に従って、実行可能なソース・ステートメントが 20 個以下の該当する関数をすべてインラインしようとします。 **-Q** を最後に指定した場合は、すべての関数がインラインされます。
- Q! 関数を一切インラインしません。 **-Q!** を最後に指定した場合は、関数は一切インラインされません。
- Q-names *names* にリストされた関数をインラインしません。 *names* の関数名は、それぞれコロンの (:) で分離します。他のすべての適切な関数はインラインされます。このオプションは、**-Q** を暗黙指定します。

例を以下に示します。

-Q-salary:taxes:expenses:benefits

これによって、**salary**、**taxes**、**expenses**、または **benefits** という名前の関数以外のすべて関数が、可能であればインラインされます。

ソース・ファイルで定義されていない関数については、警告メッセージが出されます。

-Q+names *names* にリストされた関数およびすべての他の適切な関数をインラインしようとします。 *names* の各関数名は、コロン (:) で分離しなければなりません。このオプションは、**-Q** を暗黙指定します。

例を以下に示します。

```
-Q+food:clothes:vacation
```

これによって、インラインに適格なすべての他の関数とともに、可能な場合は、**food**、**clothes**、または **vacation** という名前の関数がすべてインラインされます。

ソース・ファイルで定義されていない関数、または定義はされているがインラインできない関数については、警告メッセージが出されます。

このサブオプションは、*threshold* 値の設定をすべてオーバーライドします。**-Q+names** と一緒に *threshold* 値にゼロを使用して、特定の関数をインラインすることができます。例を以下に示します。

```
-Q=0
```

これに以下を続けて指定します。

```
-Q+salary:taxes:benefits
```

これによって、**salary**、**taxes**、または **benefits** という名前の関数のみが、可能な場合にインラインされ、それ以外はインラインされません。

-Q=threshold インラインする関数に対してサイズの制限を設定します。実行可能ステートメントの数は、関数をインラインする場合は *threshold* 以下でなければなりません。 *threshold* は正の整数でなければなりません。デフォルト値は 20 です。 *threshold* の値に **0** を指定すると、**__inline**、**_Inline**、または **_inline** キーワードでマークされた関数以外の関数はインラインされません。

threshold の値は、論理 C ステートメントに適用されます。以下の例に示すように、宣言はカウントされません。

```
increment()
{
    int a, b, i;
    for (i=0; i<10; i++) /* statement 1 */
    {
        a=i;             /* statement 2 */
        b=i;             /* statement 3 */
    }
}
```

デフォルト

デフォルトでは、インライン指定はコンパイラーに対する手掛かりとして扱われます。インライン化が行われるかどうかは、選択するほかのオプションにも依存することがあります。

- プログラムを最適化する場合、(**-O** オプションを指定) コンパイラーは、インラインとして宣言されている関数をインライン化しようとします。

注

-Q オプションは、**-qinline** オプションと機能的に同等です。

-g オプションを (デバッグ情報を生成させるために) 指定した場合、インライン化は影響を受けることがあります。**-g** コンパイラー・オプションの情報を参照してください。

インラインによって必ずしも実行時間が改善されるとは限らないため、コードに対するこのオプションの効果はユーザー自身がテストしなければなりません。

再帰的な関数または相互に再帰的な関数はインラインしないでください。

通常、最適化を要求する (**-O** オプション) と、アプリケーションのパフォーマンスが最適化され、最適化を要求しないとコンパイラーのパフォーマンスが最適化されます。

inline、**_inline**、**_Inline**、および **__inline** 言語キーワードは、**-Q!** 以外の **-Q** オプションをすべてオーバーライドします。コンパイラーは、その他の **-Q** オプションの設定に関係なく、これらのキーワードでマークされた関数をインラインしようとします。

最大限にインラインさせるには、以下を行います。

- C プログラムの場合は、最適化 (**-O**) に加え、C 言語用の適切な **-Q** オプションを指定します。
- C++ プログラムの場合は、最適化 (**-O**) は指定するが **-Q** オプションは指定しません。

例

プログラム `myprogram.c` をコンパイルし、関数を一切インラインしないようにするには、以下を入力します。

```
xlc myprogram.c -O -Q!
```

プログラム `my_c_program.c` をコンパイルして、12 行以下の関数のインラインをコンパイラーに試行させるには、以下を入力します。

```
xlc my_c_program.c -O -Q=12
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

155 ページの『g』

177 ページの『inline』

244 ページの『O、optimize』

271 ページの『Q』

179 ページの『inline、_Inline、_inline、および __inline 関数指定子』

report

► C ► C++

目的

プログラム・ループの並列化および/または最適化の方法を示す変換レポートを作成するようにコンパイラーに指示する。変換レポートは、コンパイラー・リストの一部として組み込まれます。

構文

►► -q  

注

このオプションは、**-qhot** または **-qsmp** も有効でないと、有効ではありません。

-qreport を、**-qhot** と一緒に指定すると、コンパイラーに、疑似 C のコード・リストと、ループの変換方法の要約とを作成するように指示が出されます。この情報を使用して、プログラムでのループの実行を調整することができます。

-qreport を **-qsmp** と一緒に指定すると、コンパイラーに、プログラムがデータを処理する方法や、プログラムでのループの自動並列化を示すレポートも作成するように指示が出されます。この情報を使用して、プログラムでのループを並列化する方法、またはしない方法を決定することができます。

疑似 C のコード・リストは、コンパイルできるようにはなっていません。プログラムには疑似 C のコードは組み込まないでください。また、疑似 C のコード・リストに名前が表示される可能性のある内部ルーチンを、明示的に呼び出すことはしないでください。

例

myprogram.C をコンパイルして、コンパイラー・リストにループの最適化の過程を示すレポートを組み込むには、以下のように入力してください。

```
x1C -qhot -O3 -qreport myprogram.C
```

myprogram.C をコンパイルして、コンパイラー・リストに並列化されたループを変換する過程を示すレポートを組み込むには、以下のように入力してください。

```
x1C -qsmp -O3 -qreport myprogram.C
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

162 ページの『hot』

293 ページの『smp』

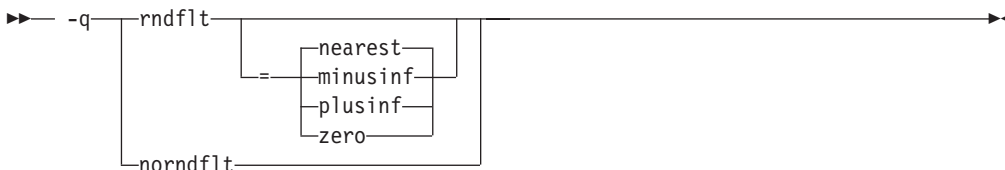
rndflt

➤ C ➤ C++

目的

このオプションは、定数浮動小数点式のコンパイル時丸めモードを制御する。これは、実行時の丸めには影響しません。

構文



ここで、使用可能な丸めオプションは以下のとおりです。

オプション 効果

nearest	表現可能な近似値まで丸める。これはデフォルトです。
minusinf	負の無限大の方向に丸める。
plusinf	正の無限大の方向に丸める。
zero	ゼロの方向に丸める。

注

コンパイル時浮動小数点演算は、プログラムの結果に以下の 2 つの効果を与える可能性があります。

- 特定のケースでは、コンパイル時の計算結果が、実行時に計算された場合の結果とわずかに異なることがあります。コンパイル時にはより多くの丸め演算が発生することが、その理由です。例えば、実行時には乗算 - 加算 浮動小数点演算が使用され、コンパイル時には乗算と加算の各演算が別々に使用される場合があります、これがわずかに異なる結果を生成します。
- 例外を生成する計算は、実行時の演算でデフォルトによって生成された IEEE 結果に畳み込むことができます。これによって、実行時に例外が発生するのを防ぎます。 **-qflltrp** オプションを使用すると、浮動小数点例外を検出およびトラップする命令を生成することができます。

一般には、実行時の丸めモードに影響するコードは、その丸めモードと一致するオプションを使ってコンパイルする必要があります。例えば、以下のプログラムがコンパイルされる場合、式 $1.0/3.0$ はコンパイル時に倍精度の結果に畳み込まれます。

```
main()
{
    float x, y;
    int i;
    x = 1.0/3.0;
    i = *(int *)&x;
    printf("1/3 = %.8x¥n", i);
    x = 1.0;
    y = 3.0;
```



```

x = x/y;
i = *(int *)&x;
printf("1/3 = %.8x\n", i);
}

```

この結果は、この後で、単精度に変換され、float x に保管されます。

-qfloat=nofold オプションを指定すると、浮動小数点計算のコンパイル時の折り畳みを抑制することができます。例えば、以下のコード・フラグメントは、**-qfloat** オプションおよびその他のオプションの設定によって、コンパイル時か実行時のいずれかで評価することができます。

```

x = 1.0;
y = 3.0;
x = x/y;

```

-qrndflt オプションは、浮動小数点計算のコンパイル時の丸めにのみ影響します。このコードが実行時に評価される場合、デフォルトの実行時丸め “round to nearest” が依然として有効となり、コンパイル時丸めモードより優先されます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

141 ページの『float』

147 ページの『flttrap』

347 ページの『y』

ro

➤ C ➤ C++

目的

ストリング・リテラルの保管型を指定する。

構文

➤ — -q —  — ➤

394 ページの『#pragma options』も参照してください。

デフォルト

cc およびその派生物を除く、すべてのコンパイラー呼び出しのデフォルトは **-qro** です。**cc** コンパイラー呼び出しのデフォルトは、**-qnoro** です。

注

-qro を指定した場合は、コンパイラーによってストリング・リテラルが読み取り専用ストレージに配置されます。**-qnoro** を指定した場合は、ストリング・リテラルは読み取り/書き込みストレージに配置されます。

以下を使用して、ソース・プログラムで保管型を指定することもできます。

```
#pragma strings storage_type
```

ここで、*storage_type* は、**read-only** または **writable** です。

ストリング・リテラルを読み取り専用メモリーに配置すると、実行時のパフォーマンスが向上し、ストレージを節約できますが、コードが読み取り専用のストリング・リテラルを変更しようとし、メモリー・エラーが生成されることがあります。

例

myprogram.c をコンパイルして保管型を **writable** にするには、以下のように入力します。

```
xlc myprogram.c -qnoro
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

279 ページの『roconst』

281 ページの『roptr』

394 ページの『#pragma options』

414 ページの『#pragma strings』

roconst

➤ C ➤ C++

目的

定数値の保管場所を指定する。

構文

➤ — -q — roconst — noroconst — ➤

394 ページの『#pragma options』も参照してください。

デフォルト

xlc、**xlc**、および **c89** でのデフォルトは **-qroconst** です。**cc** でのデフォルトは **-qnoroconst** です。

注

-qroconst を指定した場合は、コンパイラーによって定数が読み取り専用ストレージに配置されます。**-qnoroconst** を指定した場合は、定数値は読み取り/書き込みストレージに配置されます。

定数値を読み取り専用メモリーに配置することによって、実行時のパフォーマンスを向上させてストレージを節約し、共用アクセスを提供することができます。読み取り専用の定数値をコードが変更しようとする、メモリー・エラーが生成されます。

-qroconst オプションのコンテキストでは、定数値とは、**const** によって修飾された変数 (**const** によって修飾された文字、整数、浮動小数点、列挙、構造体、共用体、および配列を含む) を指します。以下の変数には、このオプションは適用されません。

- **volatile** で修飾された変数、および **volatile** 変数を含む集合体 (**struct**、**union** など)
- ポインター、およびポインター・メンバーを含む複集合体
- ブロック・スコープを持つ自動型または静的型
- 初期化されていない型
- **const** によってすべてのメンバーが修飾された通常の構造体
- アドレスである初期化指定子、または非アドレス値へのキャストである初期化指定子

-qroconst オプションでは、**-qro** オプションは暗黙指定されません。ストリング・リテラル (**-qro**) と定数値 (**-qroconst**) の両方の保管特性を指定したい場合は、これらのオプションを両方とも指定しなければなりません。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

278 ページの『ro』

281 ページの『roptr』

394 ページの『#pragma options』

roptr

➤ C ➤ C++

目的

定数ポインタの保管場所を指定する。

構文

➤ — -q noroptr
roptr ➤

注

このオプションには AIX v5.2 が必須です。

定数ポインタは、アドレス定数と同等です。例を以下に示します。

```
int* const p = &n;
```

デフォルト値 **-qnoroptr** が有効な場合、定数ポインタ、仮想関数テーブル、および仮想型テーブルが読み取り/書き込みストレージに配置され、エラーを生成することなく値を変更することができます。

-qroptr を指定した場合は、コンパイラによって定数ポインタ、仮想関数テーブル、および仮想型テーブルが読み取り専用ストレージに配置されます。これにより、ランタイム・パフォーマンスの向上、ストレージの節約、および共用アクセスの提供が可能となりますが、コードが読み取り専用の定数値を変更しようとする、メモリー・エラーが生成されます。例えば、**c1_ptr** が示すアドレスを変更しようとする次のコードがあるとします。

```
char c1 = 10;
char c2 = 20;
char* const c1_ptr = &c1;

int main() {
    *(char**)&c1_ptr = &c2;
}
```

このコードを **-qroptr** オプションを指定してコンパイルすると、実行時にセグメンテーション障害が起こります。

共用ライブラリーの一部となるコンパイルされたコードに対して、**-qroptr** を使用してはいけません。

関連資料

61 ページの『コンパイラのコマンド行オプション』

278 ページの『ro』

279 ページの『roconst』

394 ページの『#pragma options』

rrm

► C ► C++

目的

正および負の無限大への実行時丸めモードと互換性がない浮動小数点の最適化を回避する。

構文

►► -q  

394 ページの『#pragma options』も参照してください。

注

このオプションは廃止されました。新しいアプリケーションでは、**-qfloat=rrm** を使用してください。

このオプションによって、実行時に、浮動小数点の丸めモードが変更される可能性があること、あるいはモードが **-yn** (表現可能な最近値への丸め) に設定されていないことが、コンパイラーに通知されます。

浮動小数点状況レジスターおよび制御レジスターを実行時に変更する場合には、**-qrrm** も指定しなければなりません。

デフォルトの **-qnorrm** では、実行時丸めモードの最近値 (**nearest**)、およびゼロ (**zero**) と互換性のあるコードが生成されます。丸めモードのオプションのリストについては、**-y** コンパイラー・オプションを参照してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

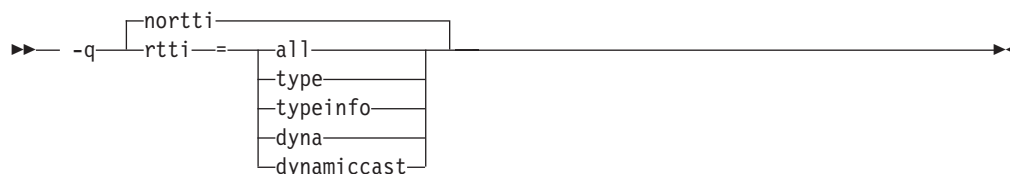
141 ページの『float』

394 ページの『#pragma options』

目的

このオプションを使用して、typeid 演算子と dynamic_cast 演算による例外処理および使用のための実行時型識別 (RTTI) 情報を生成する。

構文



ここで、使用できるサブオプションは以下のとおりです。

すべての	コンパイラーは、 RTTI の typeid および dynamic_cast 演算子に必要な情報を生成します。
	サブオプションなしで -qrtti を指定する場合は、これがデフォルトのサブオプションです。
type typeinfo	コンパイラーは、 RTTI の typeid 演算子に必要な情報を生成しますが、 dynamic_cast 演算子に必要な情報は生成しません。
dyna	コンパイラーは、 RTTI の dynamic_cast 演算子に必要な情報を生成しますが、 typeid 演算子に必要な情報は生成しません。
dynamiccast	

注

実行時のパフォーマンスを最高にするには、デフォルトの **-qnortti** の設定によって RTTI 情報の生成を抑止します。

C++ 言語は、実行時にオブジェクトのクラスを判別するための RTTI 機構を提供します。これは、以下の 2 つの演算子から構成されます。

- オブジェクトの実行時の型を判別する演算子 (typeid)。
- 実行時に検査される型変換を行うための演算子 (dynamic_cast)。

type_info クラスには、使用可能な RTTI が記述されており、 typeid 演算子によって戻される型が定義されています。

-qrtti コンパイラー・オプションを指定する場合は、以下の影響に注意してください。

- **-qrtti** が指定されているときには、仮想関数テーブルの内容は異なります。
- オブジェクトをまとめてリンクするとき、対応するソース・ファイルはすべて、正しい **-qrtti** オプションを指定してコンパイルしなければなりません。
- ライブラリーを混合オブジェクト (いくつかのオブジェクトには **-qrtti** が指定されており、その他のオブジェクトには **-qnortti** が指定されている) でコンパイルすると、未定義シンボル・エラーとなる場合があります。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

129 ページの『eh』

目的

ソース・ファイルごとにアセンブラー言語ファイル (**.s**) を生成する。結果として得られる **.s** ファイルをアセンブルして、オブジェクト **.o** ファイル、または実行可能ファイル (**a.out**) を作成することができます。

構文

▶— -S—▶

注

アセンブラーは、 **x1C** コマンドで起動できます。例を以下に示します。

```
x1C myprogram.s
```

これによって、アセンブラーが起動され、成功した場合はローダーが起動され、実行可能ファイル **a.out** が作成されます。

-E または **-P** と一緒に **-S** を指定した場合は、**-E** または **-P** が優先されます。この優先順位は、コマンド行に指定された順序に関係なく保持されます。

ソース・ファイルを 1 つしか提供しない場合に限り、**-o** オプションを使用して、作成されるファイルの名前を指定することができます。例えば、以下は無効 です。

```
x1C myprogram1.C myprogram2.C -o -S
```

制約事項

生成されるアセンブラー・ファイルには、**-g** または **-qipa** オプションによって **.o** ファイルに組み込まれるデータが、すべて入っているわけではありません。

例

1. myprogram.C をコンパイルしてアセンブラー言語ファイル **myprogram.s** を作成するには、以下のように入力します。

```
x1C myprogram.C -S
```

2. このプログラムをアセンブルしてオブジェクト・ファイル **myprogram.o** を作成するには、以下のように入力します。

```
x1C myprogram.s -c
```

3. myprogram.C をコンパイルしてアセンブラー言語ファイル **asmprogram.s** を作成するには、以下のように入力します。

```
x1C myprogram.C -S -o asmprogram.s
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

126 ページの『E』

155 ページの『g』

181 ページの『ipa』

248 ページの『o』
252 ページの『P』
314 ページの『ttable』

また、Web で以下を参照してください。

- AIX 5L for POWER-based Systems: Assembler Language Reference
- Files Reference

S

► C ► C++

目的

このオプションは、出力ファイルからのシンボル・テーブル、行番号情報、および再配置情報をストリップする。**-s** を指定するとスペースの節約になりますが、**-g** などのオプションを使用してデバッグ情報を生成するときには、従来のデバッグ・プログラムの実用性は制限されます。

構文

► **-s** ◀

注

strip コマンドの使用にも同じ影響があります。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

155 ページの『g』

saveopt

► C ► C++

目的

コマンド行コンパイラー・オプションをオブジェクト・ファイル内に保管する。


構文

►► -q 

注

このオプションにより、コンパイル中のオブジェクト・ファイルにコマンド行コンパイラー・オプションを保管できます。このオプションは、オブジェクト (**.o**) ファイルに対してコンパイルしている場合にのみ、有効です。

ストリングは、次の形式で保管されます。

►► @(#)opt-B 

ここで、

- B** スペースを示します。
- f** FORTRAN 言語コンパイルを示します。
- c** C 言語コンパイルを示します。
- C** C++ 言語コンパイルを示します。
- stanza** コンパイルに使用するドライバー、例えば、c89 または xlc を指定します。
- options** コマンド行で指定されるコマンド行オプションのリスト。個々のオプションをスペースで区切られています。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

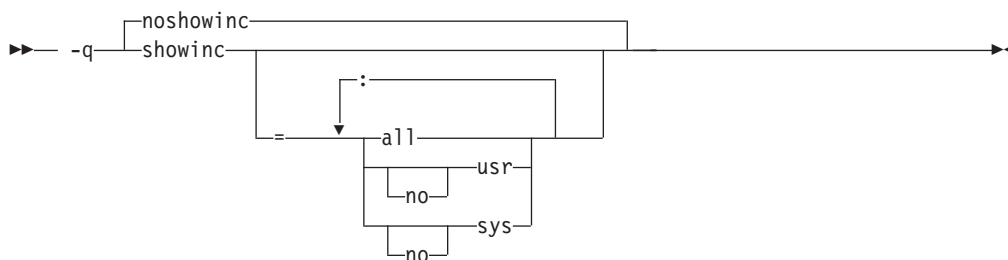
showinc

► C ► C++

目的

-qsource と共に使用して、プログラム・ソース・リストにユーザー・ヘッダー・ファイル (" " を使用して組み込む) またはシステム・ヘッダー・ファイル (< > を使用して組み込む) を選択的に表示する。

構文



ここで、オプションは、以下のとおりです。

noshowinc	ユーザー・インクルード・ファイルとシステム・インクルード・ファイルのどちらもプログラム・ソース・リストに表示しません。これは、 -qshowinc=nousr:nosys の指定と同じです。
showinc	ユーザー・インクルード・ファイルとシステム・インクルード・ファイルの両方をプログラム・ソース・リストに表示します。これは、 -qshowinc=usr:sys または -qshowinc=all の指定と同じです。
all	ユーザー・インクルード・ファイルとシステム・インクルード・ファイルの両方をプログラム・ソース・リストに表示します。これは、 -qshowinc または -qshowinc=usr:sys の指定と同じです。
usr	ユーザー・インクルード・ファイルをプログラム・ソース・リストに表示します。
sys	システム・インクルード・ファイルをプログラム・ソース・リストに表示します。

394 ページの『#pragma options』も参照してください。

注

-qlist または **-qsource** コンパイラー・オプションが有効な場合にのみ、このオプションは有効です。

例

myprogram.C をコンパイルしてすべてのインクルード・ファイルがソース・リストに出力されるようにするには、以下を入力します。

```
xlc myprogram.C -qsource -qshowinc
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

296 ページの『source』

394 ページの『#pragma options』

showpdf

► C ► C++

目的

-qpdf1 と共に使用して、追加呼び出しおよびブロック数プロファイル情報を実行可能ファイルに追加する。

構文

►►  `-q` `showpdf` ►►

注

このオプションは、**-qpdf1** コンパイラー・オプションと共に指定した場合にのみ有効です。

-qpdf1 と共に指定した場合、コンパイラーは、追加のプロファイル情報を、コンパイルされるアプリケーションに挿入して、アプリケーション内のすべてのプロシージャに対する呼び出しおよびブロック数を収集します。コンパイルされたアプリケーションを実行すると、呼び出しおよびブロック数がファイル **._pdf** に記録されます。

トレーニング・データを使用してアプリケーションを実行した後、**showpdf** ユーティリティを使用して **._pdf** ファイルの内容を取り出すことができます。このユーティリティは、**-qpdf** ページに説明されています。

例

例では、プログラム・ファイル **hello.c** に対して次のソースを想定しています。

```
#include <stdio.h>

void HelloWorld()
{
    printf("Hello World");
}

main()
{
    HelloWorld();
}
```

以下を使用してソースをコンパイルします。

```
xlc -qpdf1 -qshowpdf hello.c
```

結果として得られるプログラム実行可能ファイルを実行します。

```
a.out
```

showpdf ユーティリティを実行して、実行可能ファイルに対する呼び出し数およびブロック数を表示します。

```
showpdf
```

showpdf ユーティリティが、以下のようなデータを戻します。

```

HelloWorld(4):  1 (hello.c)

Call Counters:
  5 | 1  printf(6)

Call coverage = 100% ( 1/1 )

Block Counters:
  3-5 | 1
  6   |
  6   | 1

Block coverage = 100% ( 2/2 )

-----
main(5):  1 (hello.c)

Call Counters:
 10 | 1  HelloWorld(4)

Call coverage = 100% ( 1/1 )

Block Counters:
  8-11 | 1
 11   |

Block coverage = 100% ( 1/1 )

Total Call coverage = 100% ( 2/2 )
Total Block coverage = 100% ( 3/3 )

```

関連資料

61 ページの『コンパイラーのコマンド行オプション』
 257 ページの『pdf1、pdf2』

smallstack

➤ C ➤ C++

目的

スタック・フレームのサイズを削減するようコンパイラーに指示する。

構文

➤➤ -q  nosmallstack
smallstack ➤➤

注

AIX では、スタック・サイズが 256 MB までに制限されています。スレッド化プログラムなど、スタックに大量のデータを割り振るプログラムでは、スタック・オーバーフローが発生する可能性があります。このオプションを使用すると、スタック・フレームのサイズを縮小して、オーバーフローを避けることができます。

このオプションは、IPA (**-qipa**、**-O4**、**-O5** コンパイラー・オプション) とともに使用したときに有効です。

このオプションを指定すると、プログラムのパフォーマンスが低下する場合があります。

例

myprogram.c をコンパイルしてスタック・フレームを小さくするには、以下のように入力します。

```
xlc myprogram.c -qipa -qsmallstack
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

155 ページの『g』

181 ページの『ipa』

244 ページの『O, optimize』

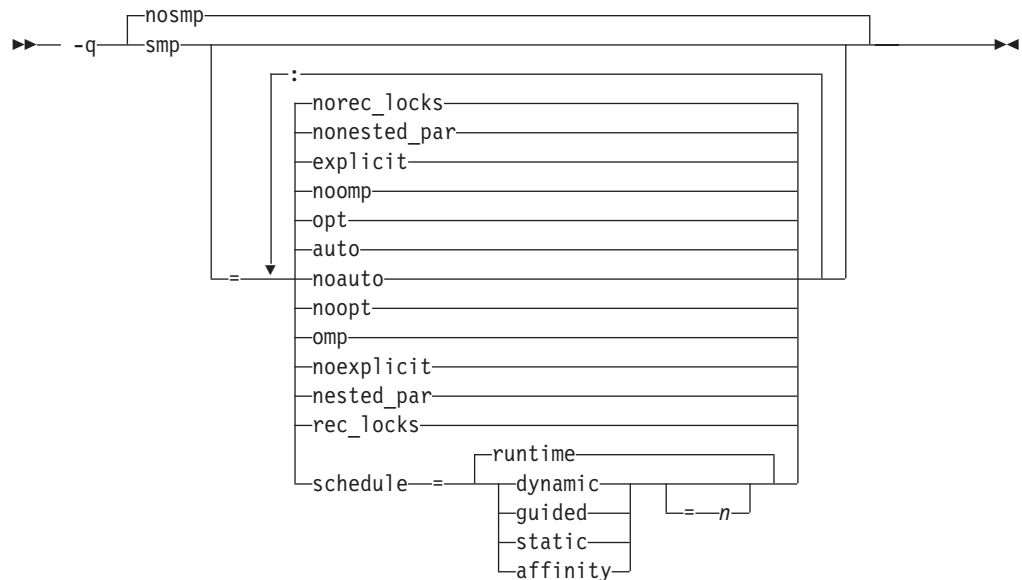
smp

► C ► C++

目的

プログラム・コードの並列化を使用可能にする。

構文



ここで、

auto	プログラム・コードの自動並列化および最適化を使用可能にします。
noauto	プログラム・コードの自動並列化を使用不可にします。 SMP または OpenMP プラグマ・ステートメントによって明示的に並列化されるプログラム・コードが最適化されます。
opt	プログラム・コードの自動並列化および最適化を使用可能にします。
noopt	自動並列化は使用可能にしますが、並列化されたプログラム・コードの最適化は使用不可にします。並列化されたプログラム・コードをデバッグするときは、この設定を使用してください。
omp	OpenMP 2.0 標準に厳密に準拠できるようにします。自動並列化は使用不可になります。並列化されたプログラム・コードは最適化されます。認識されるのは、OpenMP 並列化プラグマのみです。
noomp	プログラム・コードの自動並列化および最適化を使用可能にします。
explicit	ループの明示並列化を制御するプラグマを使用可能にします。
noexplicit	ループの明示並列化を制御するプラグマを使用不可にします。

nested_par

これを指定すると、ネストされた並列構成はシリアルライズされません。**nested_par** は、スレッドの新規のチームをネストされた並列領域用に作成させる原因とはならないため、真のネストされた並列性を提供しません。代わりに、現在使用可能なスレッドが再利用されます。

このオプションを使用する場合は、注意が必要です。外部ループ内の使用可能なスレッドの数および作業の量によっては、このオプションが有効な場合でも、内部ループが順次実行される可能性があります。並列化オーバーヘッドは、必ずしもプログラムのパフォーマンス向上によって相対位置変更されるとは限りません。

nonested_par rec_locks

ネストされた並列構成の並列化を使用不可にします。これを指定すると、再帰的ロックが使用され、ネストされたクリティカル・セクションがデッドロックの原因となることはありません。

norec_locks schedule=sched_type[=n]

これを指定すると、再帰的ロックは使用されません。他のスケジューリング・アルゴリズムがソース・コードに明示的に割り当てられていないループに使用されている、スケジューリング・アルゴリズムの種類およびチャンク・サイズ(*n*)を指定します。*sched_type* が指定されていない場合、**schedule=runtime** がデフォルト設定と見なされます。

注

- **-qsmp** の場合は、**xlc_r** のようなスレッド・セーフ・コンパイラー・モードの起動でのみ使用しなければなりません。これらの起動により、**pthread**s、**xlsmp**、およびすべてのデフォルトのランタイム・ライブラリーのスレッド・セーフ・バージョンが、その結果生じる実行可能ファイルと確実にリンクされます。
- **-qnosmp** オプションのデフォルト設定では、構文検査が実行されることになっていても、並列化ディレクティブのためのコードを生成しないように指定されています。**-qignprag=omp:ibm** を使用して、並列化ディレクティブを完全に無視します。
- サブオプションを指定しない **-qsmp** の指定は、**-qsmp=auto:explicit:noomp:norec_locks:nonested_par:schedule=runtime**、または **-qsmp=opt:explicit:noomp:norec_locks:nonested_par:schedule=runtime** を指定するのと同じです。
- **-qsmp** を指定すると、**-O2** が暗黙的に設定されます。**-qsmp** オプションは **-qnooptimize** をオーバーライドしますが、**-O3**、**-O4** または **O5** はオーバーライドしません。並列化されたプログラム・コードをデバッグするときには、**qsmp=noopt** を指定して、並列化されたプログラム・コードの最適化を使用不可にすることができます。
- **-qsmp** を指定すると、**_IBMSMP** プリプロセス・マクロが定義されます。

関連概念

15 ページの『プログラムの並列化』

関連タスク

28 ページの『並列処理ランタイム・オプションの設定』

47 ページの『プリAGMAを使用した並列処理の制御』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

244 ページの『O, optimize』

321 ページの『threaded』

422 ページの『並列処理を制御するプリAGMA』

430 ページの『#pragma ibm schedule』

461 ページの『IBM SMP 並列処理のためのランタイム・オプション』

465 ページの『並列処理のための OpenMP ランタイム・オプション』

467 ページの『並列処理に使用する組み込み関数』

source

➤ C ➤ C++

目的

コンパイラー・リストを作成し、ソース・コードを組み込む。

構文

➤ — -q 

394 ページの『#pragma options』も参照してください。

注

-qnoprint オプションは、このオプションをオーバーライドします。

#pragma options source および **#pragma options nosource** プリプロセッサ・ディレクティブのペアを、ソース・プログラムにわたって使用することによって、ソースの一部を選択的に出力させることができます。 **#pragma options source** と **#pragma options nosource** の間にあるソースが出力されます。

例

myprogram.C をコンパイルして、ソースを **myprogram.C** に組み込むコンパイラー・リストを作成するには、以下を入力します。

```
x1C myprogram.C -qsource
```

コンパイラー・リストにプログラム・ソースの選択された部分のみを表示したい場合は、**-qsource** コンパイラー・オプションを使用しないでください。以下のコードによって、**#pragma options source** と **#pragma options nosource** ディレクティブの間にあるソースのみがコンパイラー・リストに入ります。

```
#pragma options source
...
/* Source code to be included in the compiler listing
   is bracketed by #pragma options directives.
*/
...
#pragma options nosource
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

265 ページの『print』

394 ページの『#pragma options』

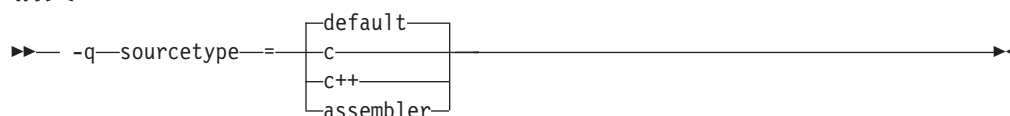
sourcetype

► C ► C++

目的

実際のソース・ファイル名サフィックスとは関係なく、すべてのソース・ファイルを、このオプションによって指定されたソース・タイプであるかのように処理することをコンパイラーに指示する。

構文



ここで、

default	コンパイラーは、ソース・ファイルのプログラム言語がそのファイル名サフィックスによって暗黙指定されているものと見なします。
c	コンパイラーは、このオプションに従い、すべてのソース・ファイルを C 言語ソース・ファイルであるかのようにコンパイルします。
c++	コンパイラーは、このオプションに従い、すべてのソース・ファイルを C++ 言語ソース・ファイルであるかのようにコンパイルします。
assembler	コンパイラーは、このオプションに従い、すべてのソース・ファイルをアセンブラー言語ソース・ファイルであるかのようにコンパイルします。

注

-qsourcetype オプションは、**-+** オプションと共に使用してはいけません。

-qsourcetype オプションはコンパイラーに対し、ファイル名サフィックスに依存するのではなく、ソース・タイプをオプションによって指定されたものと見なすように指示します。

通常、コンパイラーは、コマンド行で指定されたソース・ファイルのファイル名サフィックスを使用して、ソース・ファイルのタイプを判別します。例えば、**.c** サフィックスは通常 C ソース・コードを暗黙指定し、**.C** サフィックスは通常 C++ ソース・コードを暗黙指定し、コンパイラーはそれらを以下のように処理します。

hello.c	ファイルは C ファイルとしてコンパイルされます。
hello.C	ファイルは C++ ファイルとしてコンパイルされます。

これは、ファイル・システムが大/小文字を区別するかどうかにかかわらず適用されます。しかし、大/小文字を区別しないファイル・システムでは、上記の 2 つのコンパイルは同じ物理ファイルを参照します。つまり、コンパイラーは、まだコマンド行上のファイル名引き数の大/小文字の違いを認識し、それに応じてソース・タイプを判別しますが、ファイル・システムからファイルを取り出すときに、大/小文字の区別を無視します。

例

ソース・ファイル **hello.C** を C 言語ソース・ファイルとして処理するには、以下を入力します。

```
xlc -qsrcetype=c hello.C
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

71 ページの『+ (正符号)』

spill

► C ► C++

目的

レジスター割り振り予備域を *size* バイトに指定する。

構文

►► -qspill= size

394 ページの『#pragma options』も参照してください。

注

プログラムが非常に複雑な場合、あるいは計算が多過ぎてレジスター内に一度に保持できないためにプログラムに一時記憶域が必要な場合には、この領域を増加させる必要がある場合があります。コンパイラーが予備域を拡大するように要求するメッセージを出さない限り、この予備域は拡大しないでください。競合がある場合には、指定された最大の予備域が使用されます。

例

myprogram.c のコンパイル時に警告メッセージを受け取ったものの、 **900** 項目の予備域を指定してコンパイルしたい場合には、以下のように入力します。

```
xlc myprogram.c -qspill=900
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

394 ページの『#pragma options』

spnans

► C ► C++

目的

追加の命令を生成して、単精度から倍精度への変換時にシグナル型 NaN を検出する。**-qnospnans** オプションは、この変換の検出が不要であることを指定します。

構文

►► -q  

394 ページの『#pragma options』を参照してください。

注

このオプションは廃止されました。新しいアプリケーションでは、**-qfloat=nans** を使用してください。

-qhsflt オプションは、**-qsfnans** オプションをオーバーライドします。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

141 ページの『float』

164 ページの『hsflt』

394 ページの『#pragma options』

srcmsg



目的

stderr ファイル内の診断メッセージに、対応するソース・コード行を追加する。

構文

→ -q  →

394 ページの『#pragma options』も参照してください。

注

コンパイラーは、診断メッセージが参照するソース行またはソース行の一部を再構成し、診断メッセージの前に表示します。エラーがある列位置を指すポインターが表示される場合もあります。 **-qnosrcmsg** を指定すると、ソース行とフィンガー行の両方の生成が抑制され、エラー・メッセージには単にエラーが発生したファイル、行、および列が表示されるようになります。

再構成されたソース行は、マクロ展開後の状態を表します。時には、行は一部しか再構成されない場合もあります。表示された行の先頭または末尾に文字“...”がある場合は、ソース行の一部が表示されていないことを示します。

デフォルト (**-qnosrcmsg**) では、解析可能な簡潔なメッセージが表示されます。エラーごとにソース行およびポインターが提供されない代わりに、単一の行が表示され、エラーがあるソース・ファイルの名前、エラーの行と文字の列位置、およびメッセージ自体を示す単一の行が表示されます。

例

myprogram.c をコンパイルして、エラーの発生時に診断メッセージとともにソース行を表示させるには、以下を入力します。

```
xlc myprogram.c -qsrcmsg
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

394 ページの『#pragma options』

staticinline

► C ► C++

目的

このオプションは、インライン関数を静的関数として扱うか、`extern` として扱うかを制御する。デフォルトでは、XL C/C++ Enterprise Edition は、インライン関数を `extern` として扱います。

構文

►► `-q` nostaticinline
staticinline ◀◀

例

-qstaticinline オプションを使用すると、以下の宣言における関数 **f** は、たとえ明示的に静的であると宣言していなくても、静的関数として扱われます。

```
inline void f() { /*...*/};
```

デフォルトの **-qnostaticinline** を使用すると、**f** に外部結合が与えられます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

statsym

► C ► C++

目的

永続的なストレージ・クラスを持つユーザー定義の非外部名 (初期化される静的変数または初期化されない静的変数など) を、名前リスト (**xcoff** オブジェクトのシンボル・テーブル) に追加します。

構文

►► -q nostatsym
statsym ◀◀

デフォルト

デフォルトでは、静的変数はシンボル・テーブルに追加されません。ただし、静的関数はシンボル・テーブルに追加されます。

例

myprogram.C をコンパイルして静的シンボルがシンボル・テーブルに追加されるようにするには、以下を入力します。

```
xlc myprogram.C -qstatsym
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

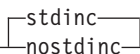
stdinc

➤ C ➤ C++

目的

#include *<file_name>* および **#include** “*file_name*” ディレクティブによって組み込まれるファイルに使用するディレクトリーを指定する。**-qnostdinc** オプションは、標準のインクルード・ディレクトリーを検索パスから除外します。

構文

➤ — -q  ➤

394 ページの『#pragma options』も参照してください。

注

-qnostdinc を指定した場合、**-I** *directory* オプションを使用して明示的にデフォルト検索パス・ディレクトリーを追加していない限り、コンパイラーは、これらのディレクトリーを検索しません。

フル (絶対) パス名を指定した場合は、このオプションはそのパス名に影響を与えません。しかし、相対パス名にはすべて影響します。

-qnostdinc は、**-qidirfirst** から独立しています。(**-qidirfirst** は、現行のソース・ファイルが置かれているディレクトリーを検索する前に、**-I** *directory* で指定されたディレクトリーを検索します。)

ファイルの検索順序については、相対パス名を使用したインクルード・ファイルのディレクトリー検索順序 で説明しています。

最後の有効な **#pragma options [NO]STDINC** は、それ以後の **#pragma options [NO]STDINC** によって置き換えられるまで、有効なままになります。

例

myprogram.c をコンパイルして、**#include** “myinc.h” ディレクティブで myprogram.c に組み込まれるファイルについて、ディレクトリー **/tmp/myfiles** を検索させるには、以下のように入力します。

```
xlc myprogram.c -qnostdinc -I/tmp/myfiles
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

166 ページの『I』

167 ページの『idirfirst』

394 ページの『#pragma options』

43 ページの『相対パス名を使用したインクルード・ファイルのディレクトリー検索順序』

strict

► C ► C++

目的

プログラムのセマンティクスを変更する可能性がある積極的な最適化をオフにする。

構文

►► -q nostrict
strict ►►

394 ページの『#pragma options』も参照してください。

デフォルト

- 最適化レベル **-O3** 以上では **-qnostrict**
- それ以外では **-qstrict**

注

-qstrict は、以下の最適化をオフにします。

- コードを移動させて、例外を起動する可能性があるロードや浮動小数点計算などの計算をスケジューリングする。
- IEEE 規則への適合性を緩和する。
- 浮動小数点式の再関連付けを行う。

このオプションは、**-O2** 以上の最適化レベルでのみ有効です。

-qstrict は **-qfloat=noftint:norsqrt** を設定します。

-qnostrict は **-qfloat=fltint:rsqrt** を設定します。

-qfloat=fltint および **-qfloat=rsqrt** を使用して **-qstrict** の設定値をオーバーライドできます。

例を以下に示します。

- O3 -qstrict -qfloat=fltint** を使用すると、**-qfloat=fltint** は有効になりますが、これ以外の積極的な最適化は行われません。
- O3 -qnostrict -qfloat=norsqrt** を使用すると、コンパイラーは、**-qfloat=rsqrt** 以外のすべての積極的な最適化を行います。

-qnostrict で設定されたオプションと、**-qfloat=options** で設定されたオプションが矛盾する場合は、最後に指定されたオプションが認識されます。

例

-O3 の積極的な最適化がオフになり、範囲検査がオフになり (**-qfloat=fltint**)、逆数 (**-qfloat=rsqrt**) による乗算によって平方根の結果による除法が置換されるように、myprogram.C をコンパイルするには、次のように入力します。

```
xlc myprogram.C -O3 -qstrict -qfloat=fltint:rsqrt
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

141 ページの『float』

244 ページの『O, optimize』

394 ページの『#pragma options』

strict_induction

C C++

目的

プログラムのセマンティクスを変更する可能性があるループ帰納変数の最適化を使用不可にする。ループ帰納変数の切り捨てや符号の拡張子が、結果として変数のオーバーフロー、または循環を起こす場合、このような最適化はプログラムの結果を変更することができます。

構文

```

>> -q nostrict_induction
    |
    | strict_induction
    |
    |----->>

```

デフォルト

- 最適化レベル 3 以上の **-qnostrict_induction**
- それ以外は、**-qstrict_induction**

注

このオプションは、パフォーマンスを非常に低下させる可能性があるため、通常は使用しないようにしてください。プログラムが帰納変数のオーバーフローや循環に依存していない場合は、**-qnostrict_induction** を、**-O2** 最適化オプションとともに使用することを考慮に入れてください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

244 ページの『O, optimize』

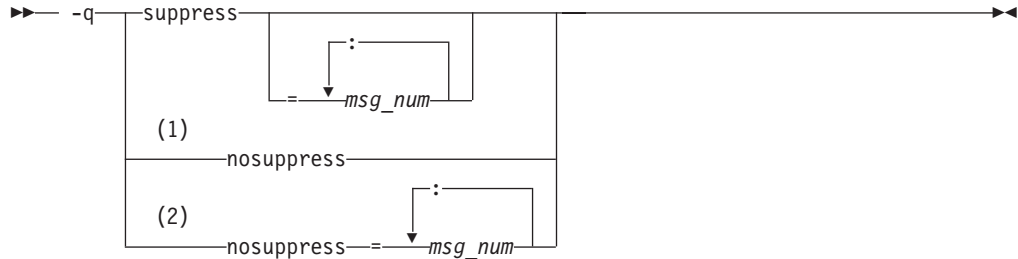
suppress

► C ► C++

目的

指定したコンパイラー・メッセージやドライバー通知メッセージ、または警告メッセージが表示されたり、またはリストに追加されたりするのを防ぐ。

構文



注:

- 1 C のみ
- 2 C++ のみ

注

このオプションは、コンパイラー・メッセージのみを抑制します。リンカー・メッセージまたはオペレーティング・システム・メッセージには影響しません。

IPA メッセージを抑制するには、コマンド行で **-qipa** の前に、**-qsuppress** を入力してください。

(S) および (U) レベルのメッセージなどの、コンパイルを停止させる原因となるコンパイラー・メッセージを非表示にすることはできません。

-qnosuppress コンパイラー・オプションは、**-qsuppress** の直前の設定を取り消します。

例

プログラムが、通常、結果として以下を出力する場合、

```
"myprogram.C", line 1.1:1506-224 (I) Incorrect #pragma ignored
```

以下をコンパイルすることによって、このメッセージを抑制することができます。

```
x1C myprogram.C -qsuppress=1506-224
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

158 ページの『halt』

181 ページの『ipa』

symtab

► C ► C++

目的

このオプションの設定値は、シンボル・テーブルに示される情報を決定する。

構文

► -q-symtab=
└─unref
└─static

ここで、

- | | |
|--------|---|
| unref | すべての typedef 宣言、 struct 型定義、 union 型定義、および enum 型定義を、IBM 分散デバッガーによる処理用に組み込むことを指定します。 |
| | -g オプションと一緒にこのオプションを使用して、デバッガーで使用するための追加のデバッグ情報を生成します。 |
| | -g オプションを指定すると、デバッグ情報がオブジェクト・ファイルに組み込まれます。オブジェクトおよび実行可能ファイルのサイズを最小にするために、コンパイラーは、参照されるシンボルに関する情報しか組み込みません。デバッグ情報は、 -qsymtab=unref を指定しない限り、参照されない配列、ポインター、またはファイル・スコープ変数については生成されません。 |
| | -qsymtab=unref を使用すると、オブジェクトおよび実行可能ファイルのサイズが大きくなる可能性があります。 |
| static | 永続的なストレージ・クラスを持つ、ユーザー定義の非外部名（初期化される静的変数または初期化されない静的変数など）を、名前リスト（ xcoff オブジェクトのシンボル・テーブル）に追加します。 |
| | デフォルトでは、静的変数はシンボル・テーブルに追加されません。 |

例

myprogram.c をコンパイルして静的シンボルがシンボル・テーブルに追加されるようにするには、次のように入力します。

```
xlc myprogram.c -qsymtab=static
```

デバッガーで使用するために、myprogram.c 内のすべてのシンボルをシンボル・テーブルに組み込むには、次のように入力します。

```
xlc myprogram.c -g -qsymtab=unref
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

155 ページの『g』

syntaxonly



目的

コンパイラーに、オブジェクト・ファイルを生成せずに構文検査を行わせる。

構文

▶— -q—syntaxonly—▶

注

-P、**-E**、および **-C** オプションは、**-qsyntaxonly** オプションをオーバーライドします。これによって、**-c** および **-o** オプションもオーバーライドされます。

-qsyntaxonly オプションは、オブジェクト・ファイルの生成しか抑制しません。その他のすべてのファイル (リストなど) は、それらに対応するオプションが設定されている限り作成されます。

例

オブジェクト・ファイルを生成せずに myprogram.c の構文を検査するには、以下を入力します。

```
xlc myprogram.c -qsyntaxonly
```

または

```
xlc myprogram.c -o testing -qsyntaxonly
```

2 番目の例では、**-qsyntaxonly** オプションによって **-o** オプションがオーバーライドされるため、オブジェクト・ファイルは作成されません。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

98 ページの『C』

99 ページの『c』

126 ページの『E』

248 ページの『o』

252 ページの『P』

C C++

-B オプションで指定したプレフィックスを、指定したプログラムに追加します。

プログラム	説明
c	コンパイラーのフロントエンド
b	コンパイラーのバックエンド
p	コンパイラー・プリプロセッサ
a	アセンブラ
I	プロシージャ間分析ツール (コンパイル・フェーズ)
L	プロシージャ間分析ツール (リンク・フェーズ)
l	リンケージ・エディター
E	CreateExportList ユーティリティ
m	リンケージ・ヘルパー (munch ユーティリティ)

```
xlc myprogram.c -B/u/newones/compilers/ -tca
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

93 ページの『B』

tabsize

► C ► C++

目的

コンパイラーによって認識されるタブの長さを変更する。

構文

►► — `-q—tabsize=`  `n` —►►

n は、ソース・プログラム内のタブを表す文字スペースの数です。

注

このオプションは、エラーが発生した列番号を示すエラー・メッセージのみに影響します。例えば、**-qtabsize=1** を指定した場合、コンパイラーはタブの幅が 1 文字であると見なします。この場合、ユーザーは、1 文字位置 (ここでは、タブの長さに関係なく文字とタブはそれぞれ 1 つの文字位置に等しい) が、1 文字分の列と同等であると考えることができます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

tbtable

► C ► C++

目的

関数の型、スタック・フレーム、およびレジスター情報を含め、各関数についての情報を含むトレースバック・テーブルを生成する。トレースバック・テーブルは、そのコードの終わりのテキスト・セグメントに配置されます。

構文

```
►► -q—tbtable—= none
                        |
                        |—full—
                        |
                        |—small—
```

サブオプションは、以下のとおりです。

none	トレースバック・テーブルを生成しません。スタック・フレームをアンワインドすることはできないので、例外処理は使用不可となります。
full	名前およびパラメーターの情報が入った完全なトレースバック・テーブルを生成します。 -qnoopt または -g を指定した場合には、これがデフォルトです。
small	生成されるトレースバック・テーブルには名前やパラメーターの情報は入りませんが、トレースバックとして完全に機能します。最適化は指定したが -g は指定していない場合には、これがデフォルトです。

394 ページの『`#pragma options`』も参照してください。

注

#pragma options ディレクティブは、コンパイル単位の最初のステートメントより前に指定しなければなりません。

多くのパフォーマンス測定ツールでは、最適化されたコードを適切に分析するために完全なトレースバック・テーブルが必要です。コンパイラー構成ファイルには、この要件に合った項目が入っています。最適化されたコードに完全なトレースバック・テーブルが不要な場合は、コンパイラー構成ファイルを以下のように変更することによって、ファイル・スペースを節約することができます。

1. C または C++ コンパイル・スタンザの **options** 行から、**-qtbtable=full** オプションを除去する。
2. **DFLT** スタンザの **xlCopt** 行から、**-qtbtable=full** オプションを除去する。

これらの変更を行うと、**tbtable** オプションのデフォルトが以下のようにになります。

- 最適化オプションを設定してコンパイルするときは **-qtbtable=small**
- 最適化オプションを設定せずにコンパイルするときは **-qtbtable=full**

トレースバック・テーブルの概要については、「言語間呼び出し - トレースバック・テーブル」を参照してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

155 ページの『g』

244 ページの『O, optimize』

394 ページの『#pragma options』

56 ページの『言語間呼び出し - トレースバック・テーブル』

以下も参照してください。

「コマンド・リファレンス 第 5 巻」の ld コマンド

tempinc

➤ C++

目的

テンプレート関数およびクラス宣言用に別個の `tempinc` ファイルを生成し、オプションで指定できるディレクトリーにこれらのファイルを配置します。

構文



注

-qtempinc コンパイラー・オプションと、**-qtemplateregistry** コンパイラー・オプションは、同時に指定できません。**-qtempinc** を指定すると、**-qnottemplateregistry** が暗黙指定されます。同様に、**-qtemplateregistry** を指定すると、**-qnottempinc** が暗黙指定されます。ただし、**-qnottempinc** を指定しても、**-qtemplateregistry** が暗黙指定されるわけではありません。

-qtempinc を指定すると、コンパイラーは、**__TEMPINC__** マクロに値 1 を割り当てます。この割り当ては、**-qnottempinc** を指定した場合には行われません。

例

ファイル `myprogram.c` をコンパイルして、テンプレート関数について生成されたインクルード・ファイルを **/tmp/mytemplates** ディレクトリーに配置するには、以下のように入力します。

```
xlc myprogram.C -qtempinc=/tmp/mytemplates
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

318 ページの『templateregistry』

「XL C/C++ プログラミング・ガイド」のセクション『C++ テンプレートの使用』も参照してください。

templaterecompile

➤ C++

目的

-qtemplateregistry コンパイラー・オプションを使用してコンパイルされたコンパイル単位間の依存性の管理に役立つ。

構文

➤➤ — -q  —➤➤

注

-qtemplaterecompile オプションは、**-qtemplateregistry** オプションを使用してコンパイルされたコンパイル単位間の依存性の管理に役立ちます。複数のコンパイル単位が同一のテンプレート・インスタンス化を参照するプログラムを指定すると、**-qtemplateregistry** オプションは、インスタンス化を含めるための単一のコンパイル単位を指定します。このインスタンス化は、他のコンパイル単位には含まれず、オブジェクト・コードの重複が回避されます。

以前コンパイルされたソース・ファイルが再度コンパイルされる場合、

-qtemplaterecompile オプションは、テンプレート・レジストリーに問い合わせ、このソース・ファイルへの変更が他のコンパイル単位の再コンパイルを必要とするかどうかを判断します。これは、指定されたインスタンス化、およびそのインスタンス化を以前含んでいた対応オブジェクト・ファイルを今後参照しないなどの変更がソース・ファイルに行われたときに発生します。この場合、影響を受けるコンパイル単位は自動的に再コンパイルされます。

-qtemplaterecompile オプションでは、コンパイラーによって生成されたオブジェクト・ファイルが、最初に書き込まれたサブディレクトリー内に残ることが必要となります。自動ビルド・プロセスがオブジェクト・ファイルを元のサブディレクトリーから移動した場合、**-qtemplateregistry** が使用可能になっているときは、必ず **-qnottemplaterecompile** オプションを使用してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

318 ページの『templateregistry』

316 ページの『tempinc』

「*XL C/C++ プログラミング・ガイド*」のセクション『C++ テンプレートの使用』も参照してください。

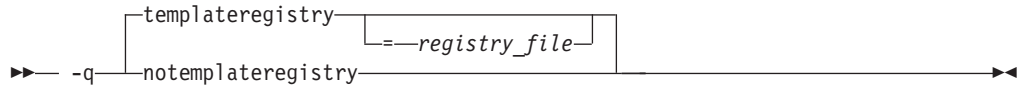
templateregistry

➤ C++

目的

ソース内で検出されるすべてのテンプレートについて、そのレコードを保守し、テンプレートごとに、一度だけインスタンス化を行うようにする。

構文



注

-qtempinc コンパイラー・オプションと、**-qtemplateregistry** コンパイラー・オプションは、同時に指定できません。**-qtempinc** を指定すると、**-qnottemplateregistry** が暗黙指定されます。同様に、**-qtemplateregistry** を指定すると、**-qnottempinc** が暗黙指定されます。ただし、**-qnottempinc** を指定しても、**-qtemplateregistry** が暗黙指定されるわけではありません。

-qtemplateregistry オプションは、ソース内で検出されるすべてのテンプレートについて、そのレコードを保守し、テンプレートごとに、一度だけインスタンス化を行うようにします。コンパイラーが最初にテンプレートのインスタンス化の参照を検出すると、そのインスタンスが生成され、それに関連したオブジェクト・コードが、現行オブジェクト・ファイルに配置されます。別のコンパイル単位で、同じテンプレートの同一のインスタンス化への参照がさらにある場合、それらの参照は記録はされますが、インスタンスが重複して生成されることはありません。

-qtemplateregistry オプションを使用するのに、特別なファイル編成は必要ありません。

場所を指定しない場合、コンパイラーは、現行作業ディレクトリーに保管されている **templateregistry** ファイルにすべてのテンプレート・レジストリー情報を保管します。テンプレート・レジストリー・ファイルは、異なるプログラム間で共有してはいけません。ソースが同じディレクトリーにある 2 つ以上のプログラムがある場合、現行作業ディレクトリーに保管されているデフォルトのテンプレート・レジストリー・ファイルに依存すると、この状況が起こり、間違った結果を出すことになります。

-qtemplateregistry をプロファイル指示フィードバックと併せて使用し、**-qpdf2** を指定してコンパイルする場合は、**-qpdf1** を使用して生成されるテンプレート・レジストリー・ファイルを使用しなければなりません。

例

ファイル myprogram.C をコンパイルして、テンプレート登録情報を **/tmp/mytemplateregistry** ファイルに配置するには、以下のように入力します。

```
xlc myprogram.C -qtemplateregistry=/tmp/mytemplateregistry
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

317 ページの『`templaterecompile`』

316 ページの『`tempinc`』

「*XL C/C++ プログラミング・ガイド*」のセクション『*C++ テンプレートの使用*』も参照してください。

tempmax

➤ C++

目的

-qtempinc オプションによって各ヘッダー・ファイルに対して生成される、テンプレート・インクルード・ファイルの最大数を指定する。

構文

➤ — -q—tempmax—=  —➤

注

number に 1 ～ 99999 の値を指定することによって、テンプレート・ファイルの最大数を指定します。

インスタンス化は、複数のテンプレート・インクルード・ファイルにわたって行われます。

このオプションは、**-qtempinc** オプションによって生成されるファイルのサイズが大きくなり過ぎて、新しいインスタンスの作成時の再コンパイルに膨大な時間がかかるときに使用してください。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

316 ページの『tempinc』

372 ページの『#pragma implementation』

「XL C/C++ プログラミング・ガイド」のセクション『C++ テンプレートの使用』も参照してください。

threaded

► C ► C++

目的

プログラムが複数のスレッドを使用するようコンパイラーに指示します。マルチスレッド・アプリケーションをコンパイルまたはリンクする場合は、このオプションを必ず使用してください。このオプションは、すべての最適化がスレッド・セーフであることを保証します。

構文

►► -q┌nothreaded┐
└threaded┘

デフォルト

デフォルトは、**r** 呼び出しモードでコンパイルしたときは **-qthreaded**、その他の呼び出しモードでコンパイルしたときは **-qnothreaded** です。

注

このオプションは、コンパイルおよびリンケージ・エディターの両方の命令に適用します。

スレッド・セーフティーを保守するには、**-qthreaded** オプションでコンパイルされたファイルは、オプション選択によって明示的にコンパイルされた場合でも、**_r** コンパイラー呼び出しモードを選択して暗黙的にコンパイルされた場合でも、**-qthreaded** オプションでリンクしなければなりません。

このオプションは、コードのスレッド・セーフの作成は行いませんが、すでにスレッド・セーフのコードは、コンパイルおよびリンク後もスレッド・セーフのまま残るようにします。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

293 ページの『smp』

tmplparse

➤ C++

目的

このオプションでは、構文解析およびセマンティック検査をテンプレート定義 (クラス・テンプレート定義、関数本体、メンバー関数本体、および静的データ・メンバー初期化指定子) に適用するか、テンプレートのインスタンス化にのみ適用するかを指定します。コンパイラーは、テンプレート定義内の関数本体と変数初期化指定子を検査して、エラー・メッセージまたは警告メッセージを生成することができます。

構文



サブオプションは、以下のとおりです。

なし	テンプレート定義を構文解析しません。これにより、以前のバージョンの VisualAge C++ および以前の製品用に作成されたコードで発生するエラーの数を減らすことができます。これはデフォルトです。
warn	テンプレート定義を構文解析し、意味エラーの場合に警告メッセージを発行します。
error	テンプレートのインスタンスが生成されない場合でも、テンプレート定義内の問題をエラーとして扱います。

注

このオプションは、テンプレートのインスタンス化ではなく、テンプレート定義に適用されます。このオプションの設定に関係なく、定義の外側で問題が起こるとエラー・メッセージが生成されます。例えば、次のように構成の構文解析またはセマンティック検査中にエラーが見つかったと、必ずエラー・メッセージが生成されます。

- 関数テンプレートの戻りの型
- 関数テンプレートのパラメーター・リスト

関連資料

61 ページの『コンパイラーのコマンド行オプション』

「XL C/C++ プログラミング・ガイド」のセクション『C++ テンプレートの使用』も参照してください。

tocdata

C C++

目的

ローカルとしてデータにマークを付ける。

構文



注

ローカル変数は、それを使用する関数と静的にバインドされます。 **-qtocdata** によって、変数をすべてローカルであると見なすようにコンパイラーに指示します。

インポートされる変数がローカル変数であると見なされた場合、パフォーマンスが低下することがあります。インポートされる変数は、ライブラリーの共用部分と動的にバインドされます。 **-qnotocdata** によって、変数をすべてインポートするものであると見なすようにコンパイラーに指示します。

データにマークを付けるオプションが矛盾する場合は、以下の方法で解決されます。

変数名をリストするオプション	特定の変数名に対する最後の明示的指定が使用されます。
デフォルトを変更するオプション	この形式は、名前リストを指定しません。指定された最後のオプションが、名前リスト・フォームに明示的にリストされていない変数のデフォルトになります。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

tocmerge

► C ► C++

目的

TOC マージを使用可能にして TOC ポインターのロードを削減し、外部ロードのスケジューリングを改善する。

構文

►► — -q — notocmerge — tocmerge — ►►

注

-qtocmerge を指定した場合、コンパイラーは、**-blmportfile** リンカー・オプションに指定されているファイルから読み取ります。**-qtocmerge** を指定していてもインポート・ファイル名が提供されていなければ、このオプションは無視され、警告メッセージが発行されます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

trigraph

► C ► C++

目的

キーボードにない文字を表すために使用される、3 文字表記キーの組み合わせを認識するようコンパイラーに指示します。

構文

► `-q` notrigraph
trigraph

デフォルト

-qtrigraph のデフォルト設定は、コンパイラーの起動に使用したコマンドに応じてそれぞれ異なります。デフォルトは、以下のとおりです。

- **-qnotrigraph** は、**xlC++**、**xlC++_r**、**xlC**、**xlC_r**、**xlC**、**xlC_r**、**cc**、および **cc_r** に対応します。
- **-qtrigraph** は、**c89**、**c89_r**、**c99**、および **c99_r** に対応します。

注

3 文字表記は、すべてのキーボードで使用できるとは限らない文字を指定することができます 3 つのキー文字の組み合わせです。

3 文字表記のキーの組み合わせは、以下のとおりです。

キーの組み合わせ	生成される文字
??=	#
??([
??)]
??/	¥
??'	^
??<	{
??>	}
??!	
??~	~

► **C** デフォルトの **-qtrigraph** 設定は、コマンド行で **-q[no]trigraph** オプションを明示的に設定することによってオーバーライドすることができます。

コマンド行で明示的に指定された **-q[no]trigraph** は、**-q[no]trigraph** がコマンド行で指定された場所に関係なく、通常、指定された **-qlanglvl** コンパイラー・オプションに関連付けられている **-q[no]trigraph** 設定に優先します。

► **C++** C++ プログラムについても同じですが、例外が 1 つあります。コマンド行上で **-qlanglvl=strict98** が **-q[no]trigraph** の 後ろ に出現すると、コマンド行での **-q[no]trigraph** オプションの明示的設定に関係なく、コンパイラーは **-qtrigraph** が設定されているものと見なします。

例

1. プログラムをコンパイルするときに 3 文字表記の文字シーケンスを使用できないようにするには、以下を入力します。

```
xlc myprogram.C -qnotrigraph
```

2. 次のコマンド行呼び出しでは、**-qnotrigraph** コンパイラー・オプションの明示的設定に関係なく、3 文字表記が使用可能になります。

```
xlc myprogram.C -qnotrigraph -qlanglvl=strict98
```

注: この振る舞いは、**strict98** 言語レベルに限られたものです。他のすべての言語レベルの場合、**-q[no]trigraph** コンパイラー・オプションの明示的設定は、指定された言語レベルに対する 3 文字表記サポートのデフォルト設定をオーバーライドします。例えば、次の呼び出しを行うと、3 文字表記サポートが使用不可になります。

```
xlc myprogram.C -qnotrigraph -qlanglvl=extended
```

3. **strict98** 言語レベルに対してコンパイルする際に、3 文字表記の文字シーケンスを使用不可にするには、コマンド行上で **strict98** 言語レベルを指定した後に **-qnotrigraph** を指定します。例えば、以下を入力します。

```
xlc myprogram.C -qlanglvl=strict98 -qnotrigraph
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

121 ページの『digraph』

198 ページの『langlvl』

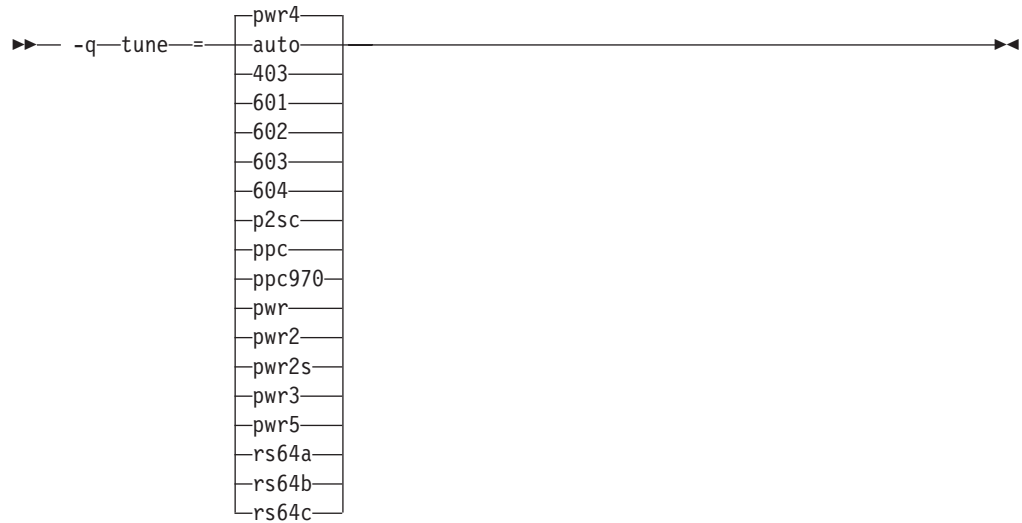
tune

► C ► C++

目的

実行可能プログラムの最適化対象とするアーキテクチャー・システムを指定する。

構文



アーキテクチャーのサブオプションは、以下のとおりです。

403	PowerPC 403 プロセッサ用に最適化されたオブジェクト・コードを生成します。
601	PowerPC 601 プロセッサ用に最適化されたオブジェクト・コードを生成します。
602	PowerPC 602 プロセッサ用に最適化されたオブジェクト・コードを生成します。
603	PowerPC 603 プロセッサ用に最適化されたオブジェクト・コードを生成します。
604	PowerPC 604 プロセッサ用に最適化されたオブジェクト・コードを生成します。
auto	コンパイル先のプラットフォーム用に最適化されたオブジェクト・コードを生成します。
p2sc	PowerPC P2SC プロセッサ用に最適化されたオブジェクト・コードを生成します。
pwr	POWER ハードウェア・プラットフォーム用に最適化されたオブジェクト・コードを生成します。
pwr2	POWER2 ハードウェア・プラットフォーム用に最適化されたオブジェクト・コードを生成します。
pwr2s	POWER2 ハードウェア・プラットフォーム用に最適化されたオブジェクト・コードを生成しますが、プログラムのパフォーマンスを遅くする特定の4倍精度命令は避けます。
pwr3	POWER3 ハードウェア・プラットフォーム用に最適化されたオブジェクト・コードを生成します。
pwr4	POWER4 ハードウェア・プラットフォーム用に最適化されたオブジェクト・コードを生成します。

pwr5	POWER5 ハードウェア・プラットフォーム用に最適化されたオブジェクト・コードを生成します。
pwrx	POWER2 ハードウェア・プラットフォーム用に最適化されたオブジェクト・コードを生成します (-qtune=pwr2 と同じ)。
rs64a	RS64I プロセッサ用に最適化されたオブジェクト・コードを生成します。
rs64b	RS64II プロセッサ用に最適化されたオブジェクト・コードを生成します。
rs64c	RS64III プロセッサ用に最適化されたオブジェクト・コードを生成します。
ppc970	PowerPC 970 プロセッサ用に最適化されたオブジェクト・コードを生成します。

394 ページの『#pragma options』も参照してください。

デフォルト

-qtune オプションのデフォルト設定は、**-qarch** オプションの設定に応じて異なります。

- **-qarch** を指定せずに **-qtune** を指定している場合、コンパイラーは **-qarch=com** を使用します。
- **-qtune** を指定せずに **-qarch** を指定している場合、コンパイラーは、指定されたアーキテクチャーにデフォルトのチューニング・オプションを使用します。リストには、次しか表示されません。 TUNE=DEFAULT

特定の **-qarch** 設定でのデフォルトの **-qtune** 設定については、452 ページの『コンパイラー・モードおよびプロセッサのアーキテクチャーの有効な組み合わせ』で説明されています。

注

-qtune=suboption は、**-qarch=suboption** と共に使用できます。

- **-qarch=suboption** は、命令の生成対象とするアーキテクチャーを指定します。
- **-qtune=suboption** は、コードの最適化対象であるターゲット・プラットフォームを指定します。

例

myprogram.C からコンパイルした実行可能プログラム testing を POWER ハードウェア・プラットフォーム用に最適化するように指定するには、以下を入力します。

```
xlc -o testing myprogram.C -qtune=pwr
```

関連タスク

39 ページの『アーキテクチャー固有の (32 ビットまたは 64 ビットの) コンパイル用コンパイラー・オプションの指定』

関連資料

61 ページの『コンパイラーのコマンド行オプション』

84 ページの『arch』

452 ページの『コンパイラー・モードおよびプロセッサのアーキテクチャーの有効な組み合わせ』

twolink

► C++

目的

ライブラリーおよびオブジェクト・ファイルから組み込まれる静的コンストラクターの数を最小にする。

構文

► `-q` `notwolink` `twolink` ►

注

コンパイラーは、通常は、オブジェクト (.o) ファイルおよびライブラリー (.a) ファイル内の任意の位置で定義されているすべての静的コンストラクターをリンクします。 **-qtwolink** オプションを指定するとリンクに時間がかかりますが、このリンクには古いバージョンの C または C++ コンパイラーと互換性があります。

-qtwolink を使用する前に、アーカイブに配置された .o ファイルによってどのプログラムの動きも変更されないことを確認してください。

デフォルト

デフォルトは **-qnotwolink** です。 .o ファイルおよびオブジェクト・ファイルにある静的コンストラクターがすべて呼び出されます。これによって、生成される実行可能ファイルが大きくなりますが、 .o ファイルをライブラリーに配置してもプログラムの動きは一切変更されなくなります。

例

インクルード・ファイル foo.h が以下のとおりであるとしてします。

```
#include <stdio.h>
struct foo {
    foo() {printf ("in foo\n");}
    ~foo() {printf ("in ~foo\n");}
};
```

また、C++ プログラム t.C が以下のとおりであるとしてします。

```
#include "foo.h"
foo bar;
```

さらに、プログラム t2.C が以下のとおりであるとしてします。

```
#include "foo.h"
main() { }
```

t.C および t2.C は以下の 2 つのステップでコンパイルします。最初は、コンパイラーを起動してオブジェクト・ファイルを作成します。

```
xlc -c t.C t2.C
```

次に、それらをリンクして実行可能ファイル a.out を作成します。

```
xlc t.o t2.o
```

a.out を起動すると、以下が得られます。

```
in foo
in ~foo
```

t.o ファイルとともに AIX の **ar** コマンドを使用して、アーカイブ・ファイル t.a を作成します。

```
ar rv t.a t.o
```

その後、以下のようにデフォルトのコンパイラー・コマンドを使用します。

```
xlc t2.o t.a
```

以下の実行可能ファイルからの出力は上記と同じです。

```
in foo
in ~foo
```

しかし、以下のように **-qtwolink** オプションを使用する場合は、

```
xlc -qtwolink t2.o t.a
```

t.C にある静的コンストラクター foo() が見つからないため、実行可能ファイル a.out からの出力はありません。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

153 ページの『funcsect』

314 ページの『tbtable』

また、Web で以下を参照してください。

プロファイルについては、「コマンド・リファレンス 第 1 巻」の『ar コマンド』セクション。

U

► C ► C++

目的

コンパイラーまたは **-Dname** オプションによって定義された ID 名 を未定義にする。

構文

►► **-Uname** ◀◀

注

-Uname オプションは、**#undef** プリプロセッサー・ディレクティブと同等ではありません。このオプションでは、**#define** プリプロセッサー・ディレクティブによってソースで定義された名前を未定義にすることはできません。このオプションで未定義にできるのは、コンパイラーまたは **-Dname** オプションによって定義された名前のみです。

#undef プリプロセッサー・ディレクティブを使用して、ソース・プログラムにおいて ID 名を未定義にすることもできます。

-Uname オプションの優先順位は、**-Dname** オプションよりも上です。

例

ご使用のオペレーティング・システムが名前 **__unix** を定義していますが、コンパイルによって定義されている名前にコード・セグメント条件を入力したくないとします。以下のように入力して、名前 **__unix** の定義が無効となるように **myprogram.c** をコンパイルします。

```
xlc myprogram.c -U__unix
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

115 ページの『D』

unique

➤ C++

目的

静的コンストラクターやデコンストラクターのファイルのコンパイル単位に固有の名前を生成する。

構文

➡ -q  ➡

注

固有の名前は、静的初期化 (sinit) 関数および静的終了 (stern) 関数の名前に乱数をエンコードすることによって、**-qunique** で生成されます。デフォルトの動きでは、sinit および stern 関数にはソース・ファイルの絶対パス名がエンコードされます。絶対パス名を複数のコンパイルについて同一にする場合 (**make** スクリプトを使用する場合など) は、**-qunique** オプションが必要です。

-qunique を使用する場合は、必ずすべての **.o** および **.a** ファイルとリンクしなければなりません。リンク・ステップでは、実行可能ファイルを組み込まないでください。

例

同じパス名使用して複数のファイルをコンパイルし、静的な構成が正常に機能するようにしたいとします。この場合、**Make** ファイルによって以下のステップが生成される場合があります。

```
sqlpreprocess file1.sql > t.C
x1C -qunique t.C -o file1.o
rm -f t.C
sqlpreprocess file2.sql > t.C
x1C -qunique t.C -o file2.o
rm -f t.C
x1C file1.o file2.o
```

以下は、上記の例のためのサンプル **Make** ファイルです。

```
# rule to get from file.sql to file.o
.SUFFIXES:      .sql
.sql.o:
    sqlpreprocess $< > t.C
    $(CCC) t.C -c $(CCFLAGS) -o $@
    rm -f t.C
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

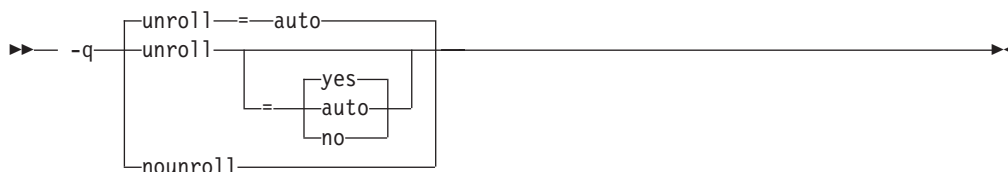
unroll

► C ► C++

目的

プログラムの内側のループをアンロールする。これにより、プログラムのパフォーマンスを向上させることができます。

構文



ここで、

<code>-qunroll=auto</code>	ループをアンロールする決定をコンパイラーに任せます。これはコンパイラーのデフォルトです。
<code>-qunroll</code> または <code>-qunroll=yes</code>	コンパイラーにループのアンロールを提案します。
<code>-qnounroll</code> または <code>-qunroll=no</code>	ループをアンロールしないようにコンパイラーに指示します。

415 ページの『`#pragma unroll`』および 394 ページの『`#pragma options`』も参照してください。

注

このオプションのコンパイラー・デフォルトは、コマンド行で特に明示的に指定されない限り、**`-qunroll=auto`** です。

サブオプションなしで **`-qunroll`** を指定するのは、**`-qunroll=yes`** を指定するのと同様です。

`-qunroll`、**`-qunroll=yes`**、または **`-qunroll=auto`** が指定されていると、最適化プログラムにより、内部ループ本体がアンロールされるか、または複製されます。最適化プログラムは、ループごとに最適なアンロール係数を判別して適用します。場合によっては、不要な分岐を避けるようにループ制御が変更される場合もあります。

`unroll` オプションによって、特定のアプリケーションのパフォーマンスが改善されるかどうかを確認するには、まず、通常オプションでプログラムをコンパイルしてから、それを代表的なワークロードで実行してください。次に、コマンド行 **`-qunroll`** オプションを指定するか、**`unroll`** プラグマを使用可能にして (あるいはその両方を行って)、プログラムを再コンパイルしてから、同じ条件下で再実行して、パフォーマンスが改善されたかどうか確認してください。

`#pragma unroll` ディレクティブを使用して、アンロールに関してさらに細かく制御することができます。このプラグマを設定すると、**`-qunroll`** コンパイラー・オプションの設定がオーバーライドされます。

例

1. 以下の例では、アンロールは使用できません。

```
x1C -qnounroll file.C
```

```
x1C -qunroll=no file.C
```

2. 以下の例では、アンロールを使用できます。

```
x1C -qunroll file.C
```

```
x1C -qunroll=yes file.C
```

```
x1C -qunroll=auto file.C
```

3. プログラム・コードがコンパイラによってどのようにアンロールされるのかの例は、415 ページの『#pragma unroll』を参照してください。

関連資料

61 ページの『コンパイラのコマンド行オプション』

394 ページの『#pragma options』

415 ページの『#pragma unroll』

unwind

► C ► C++

目的

コンパイルのルーチンがアクティブである間、スタックをアンwindできることをコンパイラーに通知する。

構文

►► — -q — unwind —
 └─┬─┘
 noundwind —————►◄

注

-qnoundwind を指定すると、不揮発性レジスターの保管と復元の最適化を改善できます。

► C++ C++ プログラムの場合、**-qnoundwind** を指定すると、**-qnoeh** も暗黙指定されます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

129 ページの『eh』

upconv



目的

整数拡張を行うときに **unsigned** の指定を保持する。

構文



394 ページの『#pragma options』も参照してください。

注

-qupconv オプションは、**int** よりも小さい任意の **unsigned** 型を、**int** ではなく **unsigned int** に拡張します。

符号なしの保持は、古い形式の C と互換性を持たせるために提供されています。ANSI C 規格では、符号なしの保持とは対照的に値を保持するように要請しています。

デフォルト

デフォルトは **-qnoupconv** ですが、**-qlanglvl=extc89** のときのデフォルトは、**-qupconv** です。コンパイラーは **unsigned** の指定を保存しません。

デフォルトのコンパイラーの処置では、整数拡張によって、**char**、**short int**、**int** ビット・フィールド、もしくはこれらの **signed** 型、**unsigned** 型、または **enumeration** 型が **int** に変換されます。それ以外の場合、型は **unsigned int** に変換されます。

例

int より小さいすべての **unsigned** 型を **unsigned int** に変換するように **myprogram.c** をコンパイルするには、以下のように入力します。

```
xlc myprogram.c -qupconv
```

次の短いリストで、**-qupconv** の効果を示します。

```
#include <stdio.h>
int main(void) {
    unsigned char zero = 0;
    if (-1 < zero)
        printf("Value-preserving rules in effect\n");
    else
        printf("Unsignedness-preserving rules in effect\n");
    return 0;
}
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

198 ページの『langlvl』

utf

► C ► C++

目的

UTF リテラル構文の認識を使用可能にする。

構文

►► — -q —  —————►►

注

コンパイラーは **iconv** を使用して、ソース・ファイルをユニコードに変換します。
ソース・ファイルを変換できなかった場合、コンパイラーは **-qutf** オプションを無視して警告を出します。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

V

► C ► C++

目的

コンパイルの進行に関する情報、およびコンパイラー内で起動中のプログラムおよび各プログラムに指定されているオプションの名前を報告するようにコンパイラーに指示する。情報は、スペースで区切られたリストに表示されます。

構文

►► -V ◀◀

注

-V オプションは、**-#** オプションによってオーバーライドされます。

例

myprogram.C をコンパイルして、コンパイルの進行を監視したり、コンパイルの進行、起動中のプログラム、および指定されているオプションを説明するメッセージを参照したりすることができるようにするには、以下を入力します。

```
xlc myprogram.C -V
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

72 ページの『# (ポンド記号)』

339 ページの『v』

V

目的

コンパイルの進行に関する情報、およびコンパイラ内で起動中のプログラムおよび各プログラムに指定されているオプションの名前を報告するようにコンパイラに指示する。情報は、コンマで区切られたリストに表示されます。

構文



注

-v オプションは、**-#** オプションによってオーバーライドされます。

例

myprogram.c をコンパイルして、コンパイルの進捗を監視したり、コンパイルの進行、起動中のプログラム、および指定されているオプションを説明するメッセージを参照したりすることができるようにするには、以下を入力します。

```
xlc myprogram.c -v
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

72 ページの『# (ポンド記号)』

338 ページの『V』

vftable

► C++

目的

仮想関数テーブルの生成を制御する。

構文

►► `-q` vftable
novftable ◀◀

デフォルト

デフォルトでは、クラス・メンバー・リストで宣言されている最初の非インライン仮想メンバー関数の本体が現行のコンパイル単位に含まれる場合に、そのクラスに対する仮想関数テーブルが定義されます。

注

-qvftable を指定すると、現行のコンパイル単位で定義されている仮想関数を持つすべてのクラスについて仮想関数テーブルが生成されます。

-qnovftable を指定した場合、現行のコンパイル単位に仮想関数テーブルは生成されません。

例

仮想関数テーブルが生成されないように `myprogram.C` ファイルをコンパイルするには、以下のように入力します。

```
xlc myprogram.C -qnovftable
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

W

➤ C ➤ C++

目的

リストしたオプションを、指定したコンパイラー・プログラムに渡す。

構文



プログラムは以下のとおりです。

プログラム	説明
a	アセンブラー
b	コンパイラーのバックエンド
c	コンパイラーのフロントエンド
I	プロシージャー間分析ツール (コンパイル・フェーズ)
L	プロシージャー間分析ツール (リンク・フェーズ)
l	リンケージ・エディター
p	コンパイラー・プリプロセッサー

注

-W オプションは、構成ファイルで使用する際には、パラメーター・ストリング内でのコンマを表すために、エスケープ・シーケンスの円記号付きコンマ (**¥**) を使用します。

例

1. **option -pg** をリンケージ・エディター (**l**) およびアセンブラー (**a**) に渡すように **myprogram.c** をコンパイルするには、以下のように入力します。

```
xlc myprogram.c -Wl,-pg -Wa,-pg
```

2. 構成ファイルでは、コンマ (,) を表すために **¥**, シーケンスを使用します。

```
-Wl¥,-pg,-Wa¥,-pg
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』



目的

警告以下のレベルのメッセージを抑制するように要求する。このオプションを指定するのは、**-qflag=e:e** を指定するのと同等です。

構文

▶▶ **-w** ◀◀

注

重大エラーに追加情報を提供する情報および警告メッセージは、このオプションでは使用不可にできません。例えば、多重定義の解決に関する問題によって発生する重大エラーでは、情報メッセージも生成されます。これらの通知メッセージは、**-w** オプションで使用不可にできません。

```
void func(int a){}
void func(int a, int b){}
int main(void)
{
    func(1,2,3);
    return 0;
}
```

```
"x.cpp", line 6.4: 1540-0218 (S) The call does not match any parameter list for "func".
"x.cpp", line 1.6: 1540-1283 (I) "func(int)" is not a viable candidate.
"x.cpp", line 6.4: 1540-0215 (I) The wrong number of arguments have been specified for "func(int)".
"x.cpp", line 2.6: 1540-1283 (I) "func(int, int)" is not a viable candidate.
"x.cpp", line 6.4: 1540-0215 (I) The wrong number of arguments have been specified for "func(int, int)".
```

例

myprogram.c をコンパイルして警告メッセージを表示させないようにするには、以下を入力します。

```
xlc myprogram.C -w
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

139 ページの『flag』

warn64

► C ► C++

目的

32 ビットと 64 ビットのコンパイラー・モード間で起こりうるデータ変換問題の検査を使用可能にします。

構文

►► `-q` `nowarn64` `warn64` ►►

注

生成されるメッセージは、すべて通知レベルになっています。

-qwarn64 オプションは、32 ビットと 64 ビットのどちらのコンパイラー・モードでも機能します。 32 ビット・モードでは、32 ビットから 64 ビットへの移行時に起こり得る問題を事前に探す補助として機能します。

データ変換によって、64 ビット・コンパイル・モードで問題が発生する可能性がある場合には、以下のような、通知メッセージが表示されます。

- **long** 型を **int** 型に明示的または暗黙的に変換したために切り捨てが行われる。
- **int** 型を **long** 型に明示的または暗黙的に変換したために予期しない結果が発生する。
- キャスト操作によって **ポインター** 型から **int** 型に明示的に変換したために無効なメモリー参照が行われる。
- キャスト操作によって **int** 型から **ポインター** 型に明示的に変換したために無効なメモリー参照が行われる。
- **定数** 型から **long** 型に明示的または暗黙的に変換したために問題が発生する。
- キャスト操作によって **定数** 型から **ポインター** 型に明示的または暗黙的に変換したために問題が発生する。
- ソース・ファイル内とコマンド行でプラグマ・オプション **arch** が矛盾している。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

73 ページの『32、64』

weaksymbol

➤ C ➤ C++

目的

弱いシンボルを生成するようにコンパイラーに指示する。

構文

➤➤ — -q  ➤➤

注

-qweaksymbol が有効な場合、コンパイラーは、以下の関数に対して弱いシンボルを生成します。

- 外部結合を持つインライン関数。

extern インライン関数を含んだ C++ プログラムをコンパイルする場合は、**-qweaksymbol** コンパイラー・オプションを使用して、重複シンボルに関するリンカー・メッセージ警告を抑制することができます。

- **#pragma weak** を使用して、弱として指定された ID、または **__attribute__((weak))** を使用して、弱として指定された関数。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

419 ページの『#pragma weak』

xcall

► C ► C++

目的

コンパイル単位内の静的ルーチンを扱うコードを、外部ルーチンであるかのように生成します。

構文

►► — -q — 

注

-qxcall は、**-qnoxcall** よりも処理が遅いコードを生成します。

例

myprogram.c をコンパイルしてすべての静的ルーチンを外部ルーチンとしてコンパイルするには、以下を入力します。

```
xlc myprogram.c -qxcall
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

xref

➤ C ➤ C++

目的

すべての ID の相互参照リストを含むコンパイラー・リストを生成する。

構文



ここで、

noxref	プログラムにある ID を報告しません。
xref	使用されている ID のみを報告します。
xref=full	プログラムにある ID をすべて報告します。

394 ページの『#pragma options』も参照してください。

注

-qnoprint オプションは、このオプションをオーバーライドします。

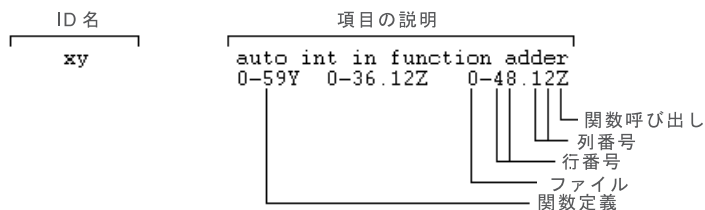
#pragma mc_func *function_name* ディレクティブで定義された関数は、いずれも **#pragma** ディレクティブの行で定義されているものとしてリストされます。

例

myprogram.C をコンパイルして、使用されているかどうかに関係なく、すべての ID の相互参照リストを生成させるには、以下を入力します。

```
xlc myprogram.C -qxref=full -qattr
```

一般的な相互参照リストの形式は、以下のとおりです。



関連資料

61 ページの『コンパイラーのコマンド行オプション』

92 ページの『attr』

265 ページの『print』

387 ページの『#pragma mc_func』

394 ページの『#pragma options』

y

► C ► C++

目的

浮動小数点定数式のコンパイル時丸めモードを指定する。

構文



サブオプションは、以下のとおりです。

n	表現可能な近似値まで丸める。これはデフォルトです。
m	負の無限大の方向に丸める。
p	正の無限大の方向に丸める。
z	ゼロの方向に丸める。

例

myprogram.c をコンパイルして浮動小数点定数式をコンパイル時にゼロの方向に丸めるには、以下を入力します。

```
xlc myprogram.c -yz
```

関連資料

61 ページの『コンパイラーのコマンド行オプション』

276 ページの『rndflt』

Z

➤ C ➤ C++

目的

このリンカー・オプションは、ライブラリー検索のパスのプレフィックスを指定します。

構文

➤— *-Z— string* —➤

注

このリンカー・オプションは、新バージョンのライブラリーを開発する場合に便利です。通常は、オペレーティング・システムの 1 つのレベル上でビルドし、それとは異なるレベル上で実行する場合に使用します。そうすることによって、ターゲット・プラットフォームではなく開発プラットフォーム上のパスを検索することができます。これが可能となるのは、プレフィックスが実行可能ファイル内に保管されないためです。

このオプションを 2 回以上使用した場合、ストリングは、指定された順序で相互に追加され、ライブラリー検索のパスの先頭に追加されます。

関連資料

61 ページの『コンパイラーのコマンド行オプション』

汎用プリグマ

以下にリストしたプリグマは、汎用プログラミングで使用可能です。特に断りのない限り、プリグマは、C プログラムと C++ プログラムの両方で使用できます。

言語アプリケーション	#pragma	説明
➤ C ➤ C++	#pragma align	構造体内のデータ項目の位置を合わせる。
➤ C	#pragma alloca	関数 alloca(size_t size) のインライン・バージョンを提供する。
➤ C ➤ C++	#pragma block_loop	ループ・ネストの特定ループ用のブロック化ループを作成するようにコンパイラーに指示を出す。
➤ C ➤ C++	#pragma chars	文字データの符号型を設定する。
➤ C ➤ C++	#pragma comment	オブジェクト・ファイルにコメントを入れる。
➤ C++	#pragma define	クラスのオブジェクトを実際に定義することなく、テンプレート・クラスの定義を強制的に行う。
➤ C ➤ C++	#pragma disjoint	その使用スコープ内で互いに別名ではない ID をリストする。
➤ C++	#pragma do_not_instantiate	指定されたテンプレート宣言のインスタンス化を抑制する。
➤ C ➤ C++	#pragma enum	後続の enum 変数のサイズを指定する。
➤ C ➤ C++	#pragma execution_frequency	実行頻度が非常に高いか非常に低いと予期されるプログラム・ソース・コードにマークを付ける。
➤ C++	#pragma hashome	指定されたクラスに IsHome プリグマで指定されるホーム・モジュールがあることをコンパイラーに通知する。
➤ C ➤ C++	#pragma ibm_snapshot	プリグマのポイントにデバッグ・ブレークポイントを設定し、プログラム実行がそのポイントに達したときに検査する変数のリストを定義する。
➤ C++	#pragma implementation	プリグマが入っているインクルード・ファイル内のテンプレート宣言に対応する関数テンプレート定義を含むファイルの名前を、コンパイラーに伝える。
➤ C ➤ C++	#pragma info	info(...) コンパイラー・オプションによって生成された診断メッセージを制御する。
➤ C++	#pragma instantiate	指定されたテンプレート宣言の即時インスタンス化が生じる。
➤ C++	#pragma ishome	指定されたクラスのホーム・モジュールが現行のコンパイル単位であることをコンパイラーに通知する。

言語アプリケーション	#pragma	説明
<div>➤ C</div> <div>➤ C++</div>	#pragma isolated_call	そのパラメーターによって暗黙指定される副次作用以外に、副次作用がない、または依存しない関数にマークを付ける。
<div>➤ C</div>	#pragma langlvl	コンパイル用の C または C++ の言語レベルを選択する。
<div>➤ C</div> <div>➤ C++</div>	#pragma leaves	関数名を取って、関数の呼び出し後に命令に戻らない関数を指定する。
<div>➤ C</div> <div>➤ C++</div>	#pragma loopid	ブロックにスコープ固有の ID でマークを付ける。
<div>➤ C</div> <div>➤ C++</div>	#pragma map	ID に対するすべてのリファレンスが新しい名前に変換されることをコンパイラーに通知する。
<div>➤ C</div> <div>➤ C++</div>	#pragma mc_func	マシン・インストラクションの短いシーケンスを含んだ関数を定義できるようにする。
<div>➤ C++</div>	#pragma namemangling	ネーム・マングリング方式およびソース・コードから生成される外部名の最大長を設定する。
<div>➤ C++</div>	#pragma nameManglingRule	関数仮パラメーターの型に応じて関数名をマングルするかどうかをコンパイラーに指示する。
<div>➤ C++</div>	#pragma object_model	オブジェクト・モデルに続く、構造体、共用体、およびクラスに使用するオブジェクト・モデルを指定する。
<div>➤ C</div> <div>➤ C++</div>	#pragma options	使用しているソース・プログラムのコンパイラーへのオプションを指定する。
<div>➤ C</div> <div>➤ C++</div>	#pragma option_override	指定された関数に対して代替最適化オプションを指定する。
<div>➤ C</div> <div>➤ C++</div>	#pragma pack	このプラグマの後に続く構造体のメンバーに対する現行の位置合わせ方式を変更する。
<div>➤ C++</div>	#pragma pass_by_value	const または参照メンバーを含むクラスを関数引き数に渡す方法を指定する。コンパイル単位のクラスはすべて、このオプションの影響を受けます。
<div>➤ C++</div>	#pragma priority	静的オブジェクトが初期設定される順序を指定する。
<div>➤ C</div> <div>➤ C++</div>	#pragma reachable	到達可能とマークされたルーチンへの呼び出し後のポイントがいくつかの認識されないロケーションからの分岐のターゲットとなれることを宣言する。
<div>➤ C</div> <div>➤ C++</div>	#pragma reg_killed_by	指定された関数によって値が破壊されるレジスターを指定する。これは #pragma mc_func と一緒に使用する必要があります。
<div>➤ C++</div>	#pragma report	特定のメッセージの生成を管理する。
<div>➤ C</div> <div>➤ C++</div>	#pragma stream_unroll	ループに含まれるストリームを複数のストリームに分割する。
<div>➤ C</div> <div>➤ C++</div>	#pragma strings	ストリングのストレージ型を設定する。

言語アプリケーション	#pragma	説明
<div>▶ C</div> <div>▶ C++</div>	#pragma unroll	プログラムの最内部および最外部のループをアンロールする。これにより、プログラムのパフォーマンスを向上させることができます。
<div>▶ C</div> <div>▶ C++</div>	#pragma unrollandfuse	ネストされた for ループ上で、アンロールおよびフューズ操作を試行するようにコンパイラーに指示する。これにより、プログラムのパフォーマンスを向上させることができます。
<div>▶ C</div> <div>▶ C++</div>	#pragma weak	リンク・エディターがシンボルの定義を見つけなかった場合、またはリンク時に複数定義されたシンボルが見つかった場合に、リンク・エディターがエラー・メッセージを出さないようにする。

関連概念

15 ページの『プログラムの並列化』

関連タスク

35 ページの『プログラム・ソース・ファイル内のコンパイラー・オプションの指定』

47 ページの『プラグマを使用した並列処理の制御』

関連資料

422 ページの『並列処理を制御するプラグマ』

#pragma align

➤ C ➤ C++

説明

#pragma align ディレクティブは、コンパイラーが構造体内でデータ項目を位置合わせする方法を指定します。

構文

```
➤➤ #pragma align ( ( full | power | mac68k | twobyte | packed | bit_packed | natural | reset ) )
```

394 ページの『#pragma options』も参照してください。

注

#pragma align=suboption ディレクティブは、プログラム・ソース・コードの指定されたセクションの **-qalign** コンパイラー・オプション設定をオーバーライドします。

コンパイラーは、位置合わせディレクティブをスタックします。このため、**#pragma align(reset)** ディレクティブを指定することによって、直前の位置合わせディレクティブの内容が不明であっても、直前の位置合わせディレクティブの使用に戻ることができます。

例えば、インクルード・ファイル内にクラス宣言があり、そのクラスに対して指定した位置合わせ規則をクラスの組み込み先に適用したくない場合に、このオプションを使用することができます。ソース・ファイルで **#pragma align(reset)** をコーディングして、最後に指定した位置合わせオプションの前の指定内容に位置合わせオプションを変更することができます。直前の位置合わせ規則がファイルにない場合は、呼び出しコマンドで指定した位置合わせ規則が使用されます。

#pragma align を指定すると、ご使用のソース・ファイルで **#pragma options align** を指定した場合と同じ影響があります。 **#pragma align** および **#pragma options align** の使用についての詳細および実例は、78 ページの『align』を参照してください。

関連資料

349 ページの『汎用プラグマ』

78 ページの『align』

「プログラミング・ガイド」の『集合体内のデータの位置合わせ』も参照してください。

#pragma alloca

▶ C

説明

#pragma alloca ディレクティブは、コンパイラーが関数 **alloca(size_tsize)** のインライン・バージョンを提供するように指定します。関数 **alloca(size_tsize)** を使用して、オブジェクトにスペースを割り振ることができます。割り振られるスペースの大きさは、*size* の値 (バイト単位) で決まります。割り振りスペースは、スタックに入ります。

構文

▶▶ #pragma alloca —————▶▶

注

コンパイラーに **alloca** のインライン・バージョンを提供させるためには、**#pragma alloca** ディレクティブ、あるいは **-qalloca** または **-ma** コンパイラー・オプションのいずれかを指定しなければなりません。

#pragma alloca は、一度指定すると、ファイル全体に適用され、オフにすることはできません。**#pragma alloca** を指定せずにコンパイルしたい関数がソース・ファイルに含まれている場合は、それらの関数を別のファイルに移してください。

関連資料

349 ページの『汎用プラグマ』

82 ページの『alloca』

228 ページの『ma』

#pragma block_loop

► C ► C++

説明

ループ・ネストの特定ループ用のブロック化ループを作成するようにコンパイラに指示を出す。

構文

```
► #pragma block_loop (—n—, —name_list—) ►
```

ここで、

- | | |
|------------------|---|
| <i>n</i> | 変数とリテラルによって構成される整数式であり、反復グループのブロック・サイズを指定します。 |
| <i>name_list</i> | #pragma loopid ディレクティブによって作成された 0 個以上のコンマ区切りの固有 ID のリストです。 <i>name_list</i> を指定しないと、 #pragma block_loop ディレクティブの後に来る最初の for ループでブロック化が行われます。 |

注

ループのブロック化に伴い、ループの反復スペースはブロックと呼ばれる部分に分割されます。外部に追加のループ（ブロック化ループと呼ばれます）が作成され、このブロック化ループにより各部分の元のループ（ブロック化対象ループと呼ばれます）が駆動されます。

ループのブロック化を発生させるには、**for** ループの前に **#pragma block_loop** ディレクティブが配置されている必要があります。

#pragma block_loop を同じ **for** ループの **nounroll**、**unroll**、**nounrollandfuse**、**unrollandfuse**、または **stream_unroll** **#pragma** ディレクティブと組み合わせて使用しないようにしてください。

例

1. 名前の指定されていない、単純なブロック化ループ。

```
#pragma block_loop(2)
for (....) { ... }
```

2. 名前の指定されたループのブロック化

```
#pragma block_loop(5, myloop)
#pragma loopid(myloop)
for (....) { ... }
```

3. ブロック化対象ループ内でのブロック化

```
#pragma block_loop(2, outerLoop)
for (i = 0; i < 100; i++) {
    #pragma loopid(outerLoop)
    #pragma block_loop(5)
```

```
        for (j = 0; j < 100; j++) {  
            A[i][j] = i + j;  
        }  
    }
```

関連資料

- 349 ページの『汎用プラグマ』
- 384 ページの『#pragma loopid』
- 415 ページの『#pragma unroll』
- 417 ページの『#pragma unrollandfuse』
- 412 ページの『#pragma stream_unroll』
- 333 ページの『unroll』

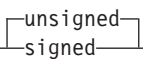
#pragma chars

► C ► C++

説明

#pragma chars ディレクティブは、char オブジェクトの符号型を **signed** または **unsigned** のいずれか設定します。

構文

```
►► #pragma chars (  ) ►►
```

注

有効にするには、このプラグマがすべてのソース・ステートメントの前になければなりません。

このプラグマは、一度指定するとファイル全体に適用され、オフにすることはできません。 **#pragma chars** を指定せずにコンパイルしたい関数がソース・ファイルに含まれている場合は、それらの関数を別のファイルに移してください。プラグマをソース・ファイルで何度も指定した場合は、最初のプラグマが優先されます。

注: デフォルトの文字型は、unsigned char と同様です。

関連資料

349 ページの『汎用プラグマ』

104 ページの『chars』

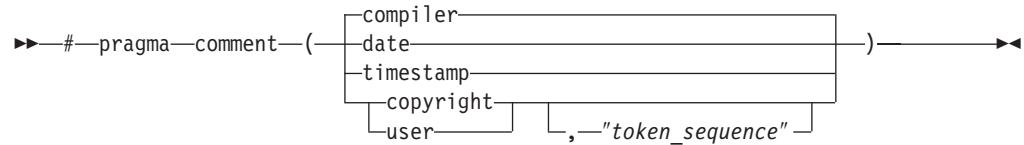
#pragma comment

➤ C ➤ C++

説明

#pragma comment ディレクティブは、ターゲットまたはオブジェクト・ファイルにコメント・ストリングを入れます。

構文



ここで、サブオプションでは以下が行われます。

compiler	コンパイラーの名前とバージョンが、生成されたオブジェクト・モジュールの最後に追加されます。
date	コンパイルの日付と時刻が、生成されたオブジェクト・モジュールの最後に追加されます。
timestamp	ソースを最後に変更した日付と時刻が、生成されたオブジェクト・モジュールの最後に追加されます。
copyright	<i>token_sequence</i> によって指定されたテキストが、生成されたオブジェクト・モジュール内にコンパイラーによって入れられ、プログラム実行時にメモリーにロードされます。
user	<i>token_sequence</i> によって指定されたテキストが、生成されたオブジェクト内にコンパイラーによって入れられますが、プログラム実行時にメモリーにロードされません。

例

以下のプログラム・コードが出力ファイル **a.out** を作成するようにコンパイルされていると想定します。

```
#pragma comment(date)
#pragma comment(compiler)
#pragma comment(timestamp)
#pragma comment(copyright,"My copyright")

int main() {

return 0;
}
```

オペレーティング・システムの **strings** コマンドを使用して、オブジェクトまたはバイナリー・ファイルで、これらのストリングおよび他のストリングを検索することができます。コマンド

```
strings a.out
```

の発行により、**a.out** で検出される可能性がある他のすべてのストリングとともに、**a.out** に組み込まれたコメント情報が表示されます。例えば、上記のプログラム・コードを想定します。

Mon Mar 1 10:28:09 2004
XL C/C++ for AIX Compiler Version 7.0
Mon Mar 1 10:28:13 2004
My copyright

関連資料

349 ページの『汎用プラグマ』

#pragma define

▶ C++

説明

#pragma define ディレクティブは、クラスのオブジェクトを実際に定義することなく、テンプレート・クラスの定義を強制的に行います。このプリAGMAは、逆方向の互換性のためだけに提供されています。

構文

▶ `#pragma define(—template_classname—)` ▶▶

ここで、*template_classname* は、定義するテンプレートの名前です。

注

ユーザーは、フォームの構成を使用して、クラス、関数、またはメンバーのテンプレート特殊化のインスタンスを明示的に生成することができます。

template declaration

例を以下に示します。

```
#pragma define(Array<char>)
```

は、以下と同等です。

```
template class Array<char>;
```

このプリAGMAは、ネーム・スペース・スコープ内で定義しなければなりません (つまり、関数/クラス本体の中に収めることはできません)。これは、テンプレート関数を効率的または自動的に生成するためにプログラムを再編成する場合に使用されます。

関連資料

349 ページの『汎用プリAGMA』

362 ページの『#pragma do_not_instantiate』

376 ページの『#pragma instantiate』

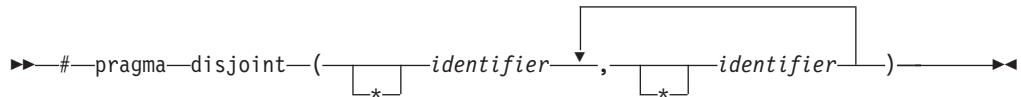
#pragma disjoint

► C ► C++

説明

#pragma disjoint ディレクティブは、その使用スコープ内で互いに別名ではない ID をリストします。

構文



注

このディレクティブは、リストされている ID はどれも、同じ物理ストレージを共用していない（これによって、さらに最適化の機会が提供されます）ことをコンパイラーに通知します。ID のいずれかが実際に物理ストレージを共用している場合は、プラグマが原因でプログラムが間違った結果を出す場合があります。

ディレクティブの中の ID は、プログラム中のプラグマが現れる点では可視でなくてはなりません。`disjoint` 名前リスト内の ID は、次のいずれも参照できません。

- 構造体または共用体のメンバー
- 構造体、共用体、または列挙タグ
- 列挙型定数
- 型定義名
- ラベル

このプラグマは、**-qignprag** コンパイラー・オプションによって使用不可にできます。

例

```
int a, b, *ptr_a, *ptr_b;
#pragma disjoint(*ptr_a, b) // *ptr_a never points to b
#pragma disjoint(*ptr_b, a) // *ptr_b never points to a
void one_function()
{
    b = 6;
    *ptr_a = 7; // Assignment does not alter the value of b
    another_function(b); // Argument "b" has the value 6
}
```

外部ポインター `ptr_a` は、外部変数 `b` とメモリーを共用せず、また外部変数を指し示すこともありません。したがって、`ptr_a` が指し示すオブジェクトに 7 を代入しても、`b` の値は変わりません。同様に、外部ポインター `ptr_b` は、外部変数 `a` とメモリーを共用せず、またそれを指し示すこともありません。コンパイラーは、`another_function` の引き数の値を 6 と見なすことができるので、メモリーから変数を再ロードしません。

関連資料

349 ページの『汎用プラグマ』

170 ページの『ignprag』

76 ページの『alias』

#pragma do_not_instantiate

➤ C++

説明

#pragma do_not_instantiate ディレクティブは、指定されたテンプレート宣言をインスタンス化しない ようにコンパイラーに指示します。

構文

▶▶ #pragma do_not_instantiate *template* ▶▶

ここで、*template* は、クラス・テンプレート ID です。例を以下に示します。

```
#pragma do_not_instantiate Stack < int >
```

注

定義を入力するテンプレートの暗黙的インスタンス化を抑制するには、このプラグマを使用します。

テンプレート・インスタンス化を手動で処理しており（つまり、**-qnotempinc** および **-qnotemplateregistry** が指定されている）、指定されたテンプレート・インスタンス化が別のコンパイル単位にすでに存在する場合、**#pragma do_not_instantiate** を使用すると、リンク・エディット・ステップ中に複数のシンボル定義を取得しなくなります。

関連資料

- 349 ページの『汎用プラグマ』
- 359 ページの『#pragma define』
- 376 ページの『#pragma instantiate』
- 316 ページの『tempinc』
- 318 ページの『templateregistry』

#pragma enum

▶ C ▶ C++

説明

#pragma enum ディレクティブは、後続の `enum` 変数のサイズを指定します。宣言の左方中括弧のサイズは、宣言内で `enum` ディレクティブがさらに生じるかどうかに関係なく、宣言に影響を与えます。このプラグマは、使用されるたびに値をスタックにプッシュし、リセット・オプションを使用すれば、直前にプッシュした値に戻ることができます。

構文

```
▶ #pragma enum (—suboption—)
                ==suboption
```

ここで、*suboption* は次のいずれかです。

- | | |
|---------|--|
| 1 | 列挙型は長さが 1 バイトで、列挙の値の範囲が signed char の制限内である場合、型は char で、そうでない場合は unsigned char です。 |
| 2 | 列挙型は長さが 2 バイトで、列挙の値の範囲が signed short の制限内である場合、型は short で、そうでない場合は unsigned short です。 |
| 4 | 列挙型は長さが 4 バイトで、列挙の値の範囲が signed int の制限内である場合、型は int で、そうでない場合は unsigned int です。 |
| 8 | 列挙型は長さが 8 バイトです。
32 ビット・コンパイル・モードでは、列挙値の範囲が signed long long の制限内にある場合、その列挙は long long 型で、そうでない場合は unsigned long long です。
64 ビット・コンパイル・モードでは、列挙値の範囲が signed long の制限内である場合は、その列挙は long 型で、そうでない場合は unsigned long です。 |
| int | #pragma enum=4 と同じ。 |
| intlong | 列挙の値の範囲が int の制限を越える場合、列挙がストレージの 8 バイトを占有するように指定します。 #pragma enum=8 の説明を参照してください。 |
| small | 列挙の値の範囲が int の制限を超えない場合は、列挙は 4 バイトのストレージを占有し、 int によって示されます。
列挙型は、すべての変数を含むことができる最小の整数型です。
8 バイトの <code>enum</code> が結果として生じる場合、実際に使用される列挙型はコンパイル・モードに依存しています。 #pragma enum=8 の説明を参照してください。 |
| pop | このサブオプションは、 <code>enum</code> サイズ設定を前の #pragma enum 設定にリセットします。前の設定がない場合、 -qenum のコマンド行設定が使用されます。 |
| reset | pop と同じ。このオプションは、後方互換性のために用意されています。 |

注

空のスタックをポップすると、警告メッセージが生成され、enum 値は未変更のままとなります。

#pragma enum ディレクティブは、**-qenum** コンパイラー・オプションをオーバーライドします。

ソース・ファイルに組み込んだ各 **#pragma enum** ディレクティブごとに、対応する **#pragma enum=reset** を該当するファイルの終わりの前に組み込むようにしてください。これは、**#include** する他のファイルの **enum** 設定がファイルによって変更される可能性をなくす唯一の方法です。

#pragma options enum= ディレクティブは、**#pragma enum** の代わりに使用することができます。この 2 つのプラグマは交換可能です。

#pragma enum=reset ディレクティブに対応する **-qenum=reset** オプションは存在しません。**-qenum=reset** を使用しようとする、警告メッセージが生成され、このオプションは無視されます。

例

1. **pop** および **reset** サブオプションの使用法は、以下のコード・セグメントで示されています。

```
#pragma enum(1)
#pragma enum(2)
#pragma enum(4)
#pragma enum(pop) /* will reset enum size to 2 */
#pragma enum(reset) /* will reset enum size to 1 */
#pragma enum(pop) /* will reset enum size to the -qenum setting,
                  assuming -qenum was specified on the command
                  line. If -qenum was not specified on the
                  command line, the compiler default is used. */
```

2. **reset** サブオプションは、一般に、メインファイルのデフォルトと異なる列挙型のストレージを指定するインクルード・ファイルの最後で、設定された列挙型のサイズをリセットするために使用します。例えば、以下のインクルード・ファイル `small_enum.h` では、さまざまな最小サイズの列挙を宣言し、インクルード・ファイルの最後の指定をオプション・スタックの最後の値にリセットしています。

```
#ifndef small_enum_h
#define small_enum_h 1
/*
 * File small_enum.h
 * This enum must fit within an unsigned char type
 */

#pragma options enum=small
enum e_tag {a, b=255};
enum e_tag u_char_e_var; /* occupies 1 byte of storage */

/* Reset the enumeration size to whatever it was before */
#pragma options enum=reset
#endif
```

以下のソース・ファイル `int_file.c` には、`small_enum.h` が組み込まれています。


```

/*
 * File int_file.c
 * Defines 4 byte enums
 */
#pragma options enum=int
enum testing {ONE, TWO, THREE};
enum testing test_enum;

/* various minimum-sized enums are declared */
#include "small_enum.h"

/* return to int-sized enums. small_enum.h has reset the
 * enum size
 */
enum sushi {CALIF_ROLL, SALMON_ROLL, TUNA, SQUID, UNI};
enum sushi first_order = UNI;

```

列挙 **test_enum** および **first_order** は、両方とも 4 バイトのストレージを占有し、**int** 型です。small_enum.h で定義された変数 **u_char_e_var** は、1 バイトのストレージを占有し、**unsigned char** データ型で表されます。

- 以下の C コード・フラグメントを **enum=small** オプションでコンパイルすると、

```
enum e_tag {a, b, c} e_var;
```

enum 定数の範囲は 0 から 2 までになります。この範囲は、上記の表で説明したいずれの範囲にも収まります。優先順位に基づいて、コンパイラーは、事前定義型 **unsigned char** を使用します。

- 以下の C コード・フラグメントを **enum=small** オプションでコンパイルすると、

```
enum e_tag {a=-129, b, c} e_var;
```

enum 定数の範囲は -129 から -127 までになります。この範囲は、**short (signed short)** および **int (signed int)** の範囲にしか収まりません。**short (signed short)** は小さ過ぎるため、**enum** を表すのに使用されます。

- 以下のコマンドを使用してファイル myprogram.C をコンパイルすると、

```
xlc myprogram.C -qenum=small
```

ファイル myprogram.C に **#pragma options=int** ステートメントが含まれていないとすると、ソース・ファイル内の **enum** 変数は、すべて最小量のストレージを占有します。

- 以下の行を含むファイル yourfile.C をコンパイルする場合

```

enum testing {ONE, TWO, THREE};
enum testing test_enum;

#pragma options enum=small
enum sushi {CALIF_ROLL, SALMON_ROLL, TUNA, SQUID, UNI};
enum sushi first_order = UNI;

#pragma options enum=int
enum music {ROCK, JAZZ, NEW_WAVE, CLASSICAL};
enum music listening_type;

```

以下のコマンドの使用

```
xlc yourfile.C
```

enum 変数 **first_order** のみが最小サイズになります (つまり、enum 変数 **first_order** は 1 バイトのストレージしか占有しません)。他の 2 つの enum 変数 **test_enum** および **listening_type** は **int** 型となり、4 バイトのストレージを占有します。

以下の例は、無効な列挙または **#pragma enum** の使用です。

- enum の宣言内で **#pragma enum** を使用することによって enum のストレージ割り振りを変更することはできません。以下のコード・セグメントには警告が生成され、2 番目に現れる **enum** オプションは無視されます。

```
#pragma enum=small
enum e_tag {
    a,
    b,
    #pragma enum=int /* error: cannot be within a declaration */
    c
} e_var;
#pragma enum=reset /* second reset isn't required */
```

- **enum** 定数の範囲は、選択した **enum** サブオプションで許可される範囲内になければなりません。以下のコード・セグメントにはエラーがあります。

```
#pragma enum(small)
enum e_tag {
    a = -1,
    b = 0x8000000000000000, /* error: enum value > MAX long long */
} e_var;
#pragma enum(reset)
```

- **enum** 定数の範囲が 32 ビット・モードの **unsigned long long**、または 64 ビット・モードの **unsigned long** の範囲内に収まりません。

```
#pragma enum(small)
enum e_tag {
    a=0,
    b=0xFFFFFFFFFFFFFFFFLL + 1LL, /* error, larger than max permitted range */
} e_var;
#pragma enum(reset)
```

関連資料

349 ページの『汎用プラグマ』

130 ページの『enum』

394 ページの『#pragma options』

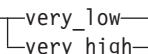
#pragma execution_frequency

► C ► C++

説明

#pragma execution_frequency ディレクティブを使用すると、実行頻度が非常に高いか非常に低いと予期されるプログラム・ソース・コードにマークを付けることができます。

構文

► #pragma execution_frequency () ►

注

このプラグマは、実行頻度が非常に高いか非常に低いと予期されるプログラム・ソース・コードにマークを付けるために使用します。プラグマは、ブロック・スコープ内に配置する必要があり、次の分岐ポイントで実行されます。

このプラグマは、最適化プログラムに対するヒントとして使用します。最適化を選択していない場合、このプラグマは有効ではありません。

例

1. このプラグマは、実行頻度の低いコードにマークを付けるために **if** ステートメント・ブロックで使用します。

```
int *array = (int *) malloc(10000);

if (array == NULL) {
    /* Block A */
    #pragma execution_frequency(very_low)
    error();
}
```

コード・ブロック "Block B" に実行頻度が低いというマークが付けられていると、分岐の際には "Block C" が選択されることになります。

```
if (Foo > 0) {
    #pragma execution_frequency(very_low)
    /* Block B */
    doSomething();
} else {
    /* Block C */
    doAnotherThing();
}
```

2. このプラグマは、実行頻度の高いコードにマークを付けるために **switch** ステートメント・ブロックで使用します。

```
while (counter > 0) {
    #pragma execution_frequency(very_high)
    doSomething();
} /* This loop is very likely to be executed. */

switch (a) {
    case 1:
        doOneThing();
        break;
    case 2:
        #pragma execution_frequency(very_high)
```

- ```
 doTwoThings();
 break;
 default:
 doNothing();
 } /* The second case is frequently chosen. */
}
```
3. このプラグマは、ファイル・スコープでは使用できません。プラグマは、ブロック・スコープ内の任意の場所に配置することができ、最も近い分岐に影響します。

```
int a;
#pragma execution_frequency(very_low)
int b;

int foo(boolean boo) {
 #pragma execution_frequency(very_low)
 char c;

 if (boo) {
 /* Block A */
 doSomething();
 {
 /* Block C */
 doSomethingAgain();
 #pragma execution_frequency(very_low)
 doAnotherThing();
 }
 } else {
 /* Block B */
 doNothing();
 }

 return 0;
}

#pragma execution_frequency(very_low)
```

1 番目と 4 番目のプラグマは無効ですが、2 番目と 3 番目は有効です。ただし、3 番目のプラグマのみが効果を有し、**"if (boo)"** 判別内で、プログラム実行が Block A または Block B のいずれに分岐するのかに影響します。2 番目のプラグマは、コンパイラーによって無視されます。

## 関連資料

349 ページの『汎用プラグマ』

## #pragma hashome

➤ C++

### 説明

**#pragma hashome** ディレクティブは、**#pragma ishome** で指定されるホーム・モジュールが指定されたクラスにあることをコンパイラーに通知します。このクラスの仮想関数テーブルであり、ある種のインライン関数とともに、静的には生成されません。代わりに、**#pragma ishome** が指定されたクラスのコンパイル単位で外部参照されます。

### 構文

➤ #pragma hashome (—*className*—  
AllInlines—)

ここで、

*className* 上記の外部参照を必要とするクラスの名前を指定します。*className* はクラスでなければならず、またこれを定義しておかなければなりません。

AllInlines *className* 内から、すべてのインライン関数を外部参照する必要があることを指定します。この引き数は大/小文字を区別しません。

### 注

**#pragma hashome** と一致しない **#pragma ishome** がある場合は、警告が出されます。

### 例

以下の例では、コード・サンプルをコンパイルすることにより、仮想関数テーブル、およびコンパイル単位 b.o ではなく、a.o のみの S::foo() の定義が生成されます。これにより、アプリケーション用に生成されるコードの量が削減されます。

```
// a.h
struct S
{
 virtual void foo() {}

 virtual void bar();
};
```

```
// a.C
#pragma ishome(S)
#pragma hashome (S)

#include "a.h"

int main()
{
 S s;
 s.foo();
 s.bar();
}
```

```
// b.C
#pragma hashome(S)
#include "a.h"

void S::bar() {}
```

## 関連資料

349 ページの『汎用プラグマ』

377 ページの『#pragma ishome』


## #pragma ibm snapshot

▶ C ▶ C++

### 説明

**#pragma ibm snapshot** ディレクティブは、プラグマのポイントにデバッグ・ブレークポイントを設定し、プログラム実行が当該ポイントに達したときに検査する変数のリストを定義します。

### 構文

▶—#—pragma—ibm snapshot—(—variable\_name—)—▶

ここで、*variable\_name* は、事前定義された、またはネーム・スペースのスコープ型です。クラス、構造体、または共用体のメンバーは指定できません。

### 注

**#pragma ibm snapshot** で指定した変数は、デバッガーで監視可能ですが、変更してはいけません。デバッガー内のこれらの変数を変更すると、予期しない動きをする場合があります。

### 例

```
#pragma ibm snapshot(a, b, c)
```

### 関連資料

349 ページの『汎用プラグマ』

## #pragma implementation

➤ C++

### 説明

**#pragma implementation** ディレクティブは、関数テンプレート定義を含むテンプレート・インスタンス化ファイルの名前をコンパイラーに通知します。これらの定義は、プラグマが入っている `#include` ファイル内のテンプレート宣言に対応します。

### 構文

▶▶ `#pragma implementation (—string_literal—)` ▶▶

### 注

このプラグマは、宣言が使用できる場所ならばどこにでも入れることができます。これは、テンプレート関数を効率的または自動的に生成するためにプログラムを再編成する場合に使用されます。

### 関連資料

349 ページの『汎用プラグマ』

320 ページの『tempmax』



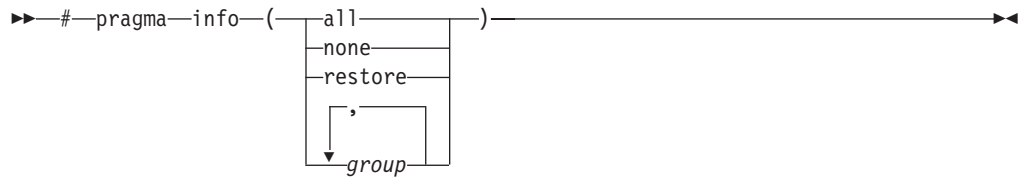
## #pragma info

▶ C ▶ C++

### 説明

**#pragma info** ディレクティブは、特定のグループのコンパイラ・メッセージを作成または抑制するようコンパイラに指示します。

### 構文



ここで、

- |         |                                                      |
|---------|------------------------------------------------------|
| all     | すべての診断検査をオンにします。                                     |
| none    | プログラムの特定の部分について、すべての診断サブオプションをオフにします。                |
| restore | 直前の <b>#pragma info</b> ディレクティブの前に有効であったオプションを復元します。 |

*group* 指定した診断 *group* に関連付けられるすべてのメッセージを生成または抑制します。以下のリストにある複数の *group* 名を指定できます。

| <b>group</b>     | <b>戻される (抑制される) メッセージのタイプ</b>                   |
|------------------|-------------------------------------------------|
| <b>c99 noc99</b> | C89 言語レベルと C99 言語レベルの間で異なる動きをする場合がある C コード。     |
| <b>cls nocls</b> | C++ クラス                                         |
| <b>cmp nocmp</b> | 無符号比較における起こりうる冗長度                               |
| <b>cnd nocnd</b> | 条件式における起こりうる冗長度または問題                            |
| <b>cns nocns</b> | 定数を含む命令                                         |
| <b>cnv nocnv</b> | 変換                                              |
| <b>dcl nodcl</b> | 宣言の整合性                                          |
| <b>eff noeff</b> | 効果のないステートメントおよびプラグマ                             |
| <b>enu noenu</b> | enum 変数の整合性                                     |
| <b>ext noext</b> | 未使用の外部定義                                        |
| <b>gen nogen</b> | 汎用診断メッセージ                                       |
| <b>gnr nognr</b> | 一時変数の生成                                         |
| <b>got nogot</b> | goto 文の使用                                       |
| <b>ini noini</b> | 初期設定で起こりうる問題                                    |
| <b>inl noinl</b> | インラインされていない関数                                   |
| <b>lan nolan</b> | 言語レベル効果                                         |
| <b>obs noobs</b> | 旧フィーチャー                                         |
| <b>ord noord</b> | 評価の未指定順序                                        |
| <b>par nopar</b> | 未使用パラメーター                                       |
| <b>por nopor</b> | 移送不能言語構造体                                       |
| <b>ppc noppc</b> | プリプロセッサの使用で起こりうる問題                              |
| <b>ppt noppt</b> | プリプロセッサ・アクションのトレース                              |
| <b>pro nopro</b> | 欠落関数プロトタイプ                                      |
| <b>rea norea</b> | 到達できないコード                                       |
| <b>ret noret</b> | 戻りステートメントの整合性                                   |
| <b>trd notrd</b> | データまたは精度の起こりうる切り捨てまたは欠落                         |
| <b>tru notru</b> | コンパイラーで切り捨てられた変数名                               |
| <b>trx notrx</b> | 16 進浮動小数点定数の丸め                                  |
| <b>uni noui</b>  | 未初期化変数                                          |
| <b>upg noupg</b> | 前のリリースと比べて、現在のコンパイラー・リリースの新しい振る舞いを示すメッセージを生成する。 |
| <b>use nouse</b> | 未使用の自動および静的変数                                   |
| <b>vft novft</b> | C++ プログラムで仮想関数テーブルの生成                           |
| <b>zea nozea</b> | ゼロ範囲の配列。                                        |

## 注

**#pragma info** ディレクティブを使用すると、コマンド行、構成ファイル、または以前の **#pragma info** ディレクティブの呼び出しで指定した、現行の **-qinfo** コンパイラー・オプション設定を一時的にオーバーライドすることができます。

## 例

例えば、下記のコード・セグメントで、MyFunction1 の前の **#pragma info(eff, nouni)** ディレクティブは、影響のないステートメントまたはプラグマを識別するメッセージを生成ことと、結合化された変数を識別するメッセージを抑制することをコンパイラーに指示します。MyFunction2 の前の **#pragma info(restore)** ディレクティブは、**#pragma info(eff, nouni)** ディレクティブが呼び出される前に有効であったメッセージ・オプションを復元するようコンパイラーに指示します。

```
#pragma info(eff, nouni)
int MyFunction1()
{
 .
 .
 .
}

#pragma info(restore)
int MyFunction2()
{
 .
 .
 .
}
```

## 関連資料

349 ページの『汎用プラグマ』

171 ページの『info』

## #pragma instantiate

➤ C++

### 説明

**#pragma instantiate** ディレクティブは、指定されたテンプレート宣言を即時にインスタンス化するようにコンパイラーに指示します。

### 構文

▶▶ #pragma instantiate *template* ▶▶

ここで、*template* は、クラス・テンプレート ID です。例を以下に示します。

```
#pragma instantiate Stack < int >
```

### 注

既存のコードをマイグレーションする場合は、このプラグマを使用します。新規のコードは、標準 C++ 明示的インスタンス化を使用しなければなりません。

テンプレート・インスタンス化を手動で処理している（つまり、**-qnotempinc** および **-qnotemplateregistry** が指定されている）場合、**#pragma instantiate** を使用すると、指定されたテンプレートのインスタンス化がコンパイル単位内に表示されます。

### 関連資料

349 ページの『汎用プラグマ』

359 ページの『#pragma define』

362 ページの『#pragma do\_not\_instantiate』

316 ページの『tempinc』

318 ページの『templateregistry』

## #pragma ishome

▶ C++

### 説明

**#pragma ishome** ディレクティブは、指定したクラスのホーム・モジュールが現行のコンパイル単位であることをコンパイラーに通知します。ホーム・モジュールは、仮想関数テーブルなどの項目が保管されている場所です。項目は、コンパイル単位の外側から参照されると、そのホームの外部には生成されません。これにより、アプリケーション用に生成されるコードの量を削減することができます。

### 構文

▶ #pragma ishome (—*className*—) ▶

ここで、

*className* ホームが現行のコンパイル単位になるクラスのリテラル名です。

### 注

**#pragma hashome** と一致しない **#pragma ishome** がある場合は、警告が出されます。

### 例

以下の例では、コード・サンプルをコンパイルすることにより、仮想関数テーブル、およびコンパイル単位 `b.o` ではなく、`a.o` のみの `S::foo()` の定義が生成されます。これにより、アプリケーション用に生成されるコードの量が削減されます。

```
// a.h
struct S
{
 virtual void foo() {}

 virtual void bar();
};
```

```
// a.C
#pragma ishome(S)
#pragma hashome (S)

#include "a.h"

int main()
{
 S s;
 s.foo();
 s.bar();
}
```

```
// b.C
#pragma hashome(S)
#include "a.h"

void S::bar() {}
```

## 関連資料

349 ページの『汎用プラグマ』

369 ページの『#pragma hashome』

## #pragma isolated\_call

▶ C ▶ C++

### 説明

**#pragma isolated\_call** ディレクティブは、パラメーターが意味する副次作用の他には、副次作用を持つことも、また副次作用に依存することもない関数にマークを付けます。

### 構文

▶ `#pragma isolated_call (—function—)` ▶

*function* は、1 次式であり、ID、演算子関数、変換関数、限定名のいずれかです。ID は、型関数または typedef 関数でなければなりません。名前により多重定義関数を参照する場合、この関数のすべての可変部は、孤立した呼び出しとしてマークされています。

### 注

**-qisolated\_call** コンパイラー・オプションには、このプラグマと同じ効果があります。

このプラグマは、リストされた関数が、そのパラメーターが意味する副次作用の他には副次作用がなく、それに依存もしないことをコンパイラーに通知します。以下のような場合、関数は副次作用を持つか、それに依存していると考えられます。

- 揮発性オブジェクトにアクセスする場合
- 外部オブジェクトを変更する場合
- 静的オブジェクトを変更する場合
- ファイルを変更する場合
- 別のプロセスまたはスレッドにより変更されるファイルにアクセスする場合
- 戻る前に解放されない限り、動的オブジェクトを割り当てる場合
- 同じ起動時に割り当てられていない限り、動的オブジェクトを解放する場合
- 丸めモードまたは例外処理などの、システム状態を変更する場合
- 上記のいずれかの行う関数を呼び出す場合

基本的には、実行時環境の状態での変更は副次作用と見なされます。ポインターまたは参照により渡された関数引き数の変更だけが許可されている副次作用です。他の副次作用のある関数は、**#pragma isolated\_call** ディレクティブにリストされている場合、誤った結果を与える可能性があります。

関数を **isolated\_call** としてマーク付けすると、呼び出し先関数によって外部変数および静的変数を変更できないこと、および、必要に応じて、ストレージへの不正な参照を呼び出し関数から削除できることが、最適化プログラムに通知されます。命令はより自由にリオーダーでき、その結果、パイプラインの遅延が少なくなり、プロセッサの実行が速くなります。同じパラメーターを指定している同じ関数に対する複数の呼び出しを結合することが可能で、結果が不要であれば、呼び出しを削除することができて、呼び出し順序を変更することができます。

指定された関数は、不揮発性外部オブジェクトを検査することが許可され、実行時環境の不揮発性状態に依存する結果を戻します。また、関数は、関数に渡されるポインター引き数、すなわち、参照体による呼び出しによって、ストレージを変更することもできます。それ自体を呼び出す関数、またはローカル静的ストレージに依存する関数は指定しないでください。このような関数を **#pragma isolated\_call** ディレクティブ内にリストすると、予想しない結果が出ます。

**-qignprag** コンパイラー・オプションによって、別名割り当てプラグマは無視されます。この **-qignprag** コンパイラー・オプションを使用して、**#pragma isolated\_call** ディレクティブを含むアプリケーションをデバッグしてください。

## 例

次の例は、**#pragma isolated\_call** ディレクティブの使用法を示します。**this\_function** には副次作用がないので、この関数を呼び出しても外部関数 **a** の値は変更されません。コンパイラーは、**other\_function** への引き数には値 6 が指定されていて、メモリーから変数が再ロードされないことを前提とすることができます。

```
int a;

// Assumed to have no side effects
int this_function(int);

#pragma isolated_call(this_function)
that_function()
{
 a = 6;
 // Call does not change the value of "a"
 this_function(7);

 // Argument "a" has the value 6
 other_function(a);
}
```

## 関連資料

349 ページの『汎用プラグマ』

170 ページの『ignprag』

191 ページの『isolated\_call』



## #pragma langlvl

▶ C

### 説明

**#pragma langlvl** ディレクティブは、コンパイルで使用する C 言語レベルを選択します。

### 構文

▶ #pragma langlvl (—*language*—) ▶▶

ここで、*language* については、以下で説明します。

▶ C

C プログラムの場合、以下のいずれかの値を *language* に指定できます。

|          |                                                                                                                                                                                                                                          |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| classic  | 非 stdc89 プログラムのコンパイルを許可し、K&R レベルのプリプロセッサに厳密に適合させます。この言語レベルは、 <code>math.h</code> などの AIX v5.1 システム・ヘッダー・ファイルではサポートされていません。AIX v5.1 システム・ヘッダー・ファイルを使用しなければならない場合は、プログラムを <b>stdc89</b> 言語レベルまたは <b>extended</b> 言語レベルへコンパイルすることを検討してください。 |
| extended | RT コンパイラーおよび <b>classic</b> との互換性を提供します。この言語レベルは C89 に基づいています。                                                                                                                                                                           |
| saa      | 現行の SAA C CPI 言語定義に適合するコンパイルです。これは現在は SAA C レベル 2 です。                                                                                                                                                                                    |
| saa12    | SAA C レベル 2 CPI 言語定義に適合するコンパイルです。これにはいくつかの例外があります。                                                                                                                                                                                       |
| stdc89   | ANSI C89 標準に適合するコンパイルで、ISO C90 としても知られています。                                                                                                                                                                                              |
| stdc99   | ISO C99 標準に適合するコンパイルです。<br>注: C99 の要求するヘッダー・ファイルと実行時ライブラリーは、すべてのオペレーティング・システム・リリースでサポートされているわけではありません。                                                                                                                                   |
| extc89   | コンパイルは、ANSI C89 標準に適合しており、インプリメンテーション固有の言語拡張を受け入れます。                                                                                                                                                                                     |
| extc99   | コンパイルは、ISO C99 標準に適合しており、インプリメンテーション固有の言語拡張を受け入れます。<br>注: C99 の要求するヘッダー・ファイルと実行時ライブラリーは、すべてのオペレーティング・システム・リリースでサポートされているわけではありません。                                                                                                       |

### デフォルト

デフォルトの言語レベルは、コンパイラーの呼び出しに使用するコマンドに応じてそれぞれ異なります。

| 呼び出し       | デフォルトの言語レベル |
|------------|-------------|
| <b>xlc</b> | extc89      |
| <b>cc</b>  | extended    |
| <b>c89</b> | stdc89      |

## 注

この `pragma` は、1 つのソース・ファイル内に 1 回だけ指定することができます。また、必ずソース・ファイル内のどの非コメント・ステートメントよりも前に置かなければなりません。

コンパイラーは、ヘッダー・ファイル内の事前定義マクロを使用して、指定された言語レベルを定義する宣言および定義を使用可能にします。

このディレクティブは、プリプロセッサーの動きを動的に変更することができます。結果として、**-E** コンパイラー・オプションを指定してコンパイルすると、**-E** オプションを指定しないでコンパイルしたときに生成される結果とは異なる結果を生成する場合があります。

## 関連資料

349 ページの『汎用プラグマ』

126 ページの『E』

198 ページの『`__langlvl`』

「C/C++ ランゲージ・リファレンス」の『*IBM C* 言語拡張機能』および『*IBM C++* 言語拡張機能』セクションも参照してください。

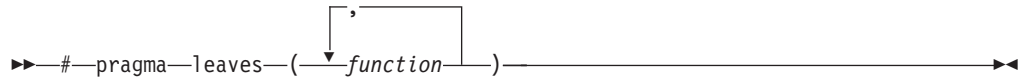
## #pragma leaves

▶ C ▶ C++

### 説明

**#pragma leaves** は関数名を取得し、呼び出し後に関数がディレクティブに戻ることはないように指定します。

### 構文



```
#pragma leaves (function)
```

### 注

このプラグマは、*function* が呼び出し元に戻らないようにコンパイラーに指示します。

このプラグマの利点は、これを使用するとコンパイラーは *function* の後にあるコードをすべて無視することができるので、最適化プログラムはより効率的なコードを生成することができるという点です。一般的に、このプラグマはカスタムのエラー処理関数に使用されますが、この場合特定のエラーが発生すると、プログラムを終了することができます。これと同様の関数には、**exit**、**longjmp**、**terminate** があります。

### 例

```
#pragma leaves(handle_error_and_quit)
void test_value(int value)
{
 if (value == ERROR_VALUE)
 {
 handle_error_and_quit(value);
 TryAgain(); // optimizer ignores this because
 // never returns to execute it
 }
}
```

### 関連資料

349 ページの『汎用プラグマ』

## #pragma loopid

► C ► C++

### 説明

関数ブロックに、スコープ内で固有の ID でマークを付けます。

### 構文

► `#pragma loopid (—name—)` ◀

ここで、*name* は、関数のスコープ単位内で固有の ID です。

### 注

**#pragma loopid** ディレクティブは、**#pragma block\_loop** ディレクティブまたは **for** ループのすぐ前になければなりません。指定した名前を **#pragma block\_loop** で使用して、そのループ上での変換を制御することができます。これを使用して、**-qreport** コンパイラー・オプションの使用を通じてのループ変換に関する情報を提供することもできます。

**#pragma loopid** は、指定されたループに対して複数回指定しないでください。

### 関連資料

- 349 ページの『汎用プラグマ』
- 275 ページの『report』
- 333 ページの『unroll』
- 354 ページの『#pragma block\_loop』
- 412 ページの『#pragma stream\_unroll』
- 415 ページの『#pragma unroll』
- 417 ページの『#pragma unrollandfuse』

## #pragma map

▶ C ▶ C++

### 説明

**#pragma map** ディレクティブは、ID に対するすべての参照を “*name*” に変換するようにコンパイラーに指示します。

### 構文

```
▶▶ #pragma map ([identifier] , "name") ▶▶
 [function_signature]
```

ここで、

*identifier* 外部結合を持つデータ・オブジェクトまたは多重定義されていない関数の名前。

*function\_signature* 内部リンクを持つ関数または演算子の名前。この名前は修飾できません。

*name* 指定したオブジェクト、関数、または演算子にバインドされる外部名。

▶ C++ ID が多重定義関数またはメンバー関数の名前である場合、プラグマはコンパイラーにより生成された名前をオーバーライドするおそれがあります。これにより、リンク時に問題が生じます。C++ 名 (C++ のデフォルト・シグニチャーである C++ リンケージ・シグニチャーを持つ名前) にリンクする場合は、マングルされた名前を指定します。下の例の、例 4 を参照してください。

### 注

**#pragma map** を使用して、以下のものをマップしないでください。

- C++ メンバー関数
- 多重定義関数
- テンプレートから生成されたオブジェクト
- リンケージにビルドされた関数

このディレクティブは、プログラムのどこに記述してもかまいません。ディレクティブ内に現れる ID は、プロトタイプ引き数リストで使用されるすべての型名を含めて、実際の発生箇所とは無関係に、ディレクティブがファイル・スコープで現れた場合と同様に解決されます。

### 例

#### 例 1 ▶ C

```
int funcname1()
{
 return 1;
}

#pragma map(func , "funcname1") /* maps func to funcname1 */
```

```
int main()
{
 return func(); /* no function prototype needed in C */
}
```

#### 例 2

```
extern "C" int funcname1()
{
 return 0;
}

extern "C" int func(); //function prototypes needed in C++

#pragma map(func, "funcname1") // maps ::func to funcname1

int main()
{
 return func();
}
```

#### 例 3

```
#pragma map(foo, "bar")

int foo(); //function prototypes needed in C++

int main()
{
 return foo();
}

extern "C" int bar() {return 7;}
```

#### 例 4

```
#pragma map(foo, "bar__Fv")

int foo(); //function prototypes needed in C++

int main()
{
 return foo();
}

int bar() {return 7;}
```

**注:** **bar** が C リンケージを持つことを宣言することによって、マングルされた名前 **bar\_FV** の使用を避けることができます。上の例 3 を参照してください。

## 関連資料

349 ページの『汎用プラグマ』

## #pragma mc\_func

► C ► C++

### 説明

**#pragma mc\_func** ディレクティブは、マシン・インストラクションの短いシーケンスを含んだ関数を定義できるようにします。

### 構文

```
►—#pragma mc_func function—{—instruction_seq—}—►
```

ここで、

*function*

C または C++ プログラムにおいて事前定義された関数を指定する必要があります。関数が事前定義されていない場合、コンパイラーはプラグマを関数定義として処理します。

*instruction\_seq*

ゼロ個以上の 16 進数字のシーケンスを含んだストリングです。数字の数は、32 ビットの整数倍で構成される必要があります。

### 注

**mc\_func** プラグマによって、マシン・インストラクション "inline" の短いシーケンスをプログラムのソース・コード内に埋め込みます。このプラグマは、通常のリンケージ・コードの代わりに、指定された命令を生成するようにコンパイラーに指示します。このプラグマを使用すると、アセンブラーでコーディングした外部関数への呼び出しに関連するパフォーマンス負荷を回避できます。このプラグマの機能は、このコンパイラーおよび他のコンパイラーにある **asm** キーワードに類似しています。

**mc\_func** プラグマは、関数を定義し、関数の定義が通常行われるプログラム・ソース内にのみ出現しなければなりません。 **#pragma mc\_func** によって定義される関数名は、事前に宣言またはプロトタイプ化される必要があります。

コンパイラーは、他の関数と同様にパラメーターを関数に渡します。例えば、整数型の引き数を取る関数の場合、1 番目のパラメーターは GPR3 に、2 番目は GPR4 に、というように渡されます。関数によって戻される値は、integer 値の場合は GPR3 に、float または double 値の場合は FPR1 となります。ご使用のシステムで使用可能な揮発性レジスターのリストについては、 **#pragma reg\_killed\_by** を参照してください。

**#pragma reg\_killed\_by** を使用して、関数によって使用される特定のレジスター・セットをリストしていない限り、 *instruction\_seq* から生成されるコードは、ご使用のシステムで使用可能な揮発性レジスターをどれでも使用できます。

インライン化オプションは、 **#pragma mc\_func** によって定義された関数に影響を与えません。しかし、 **#pragma isolated\_call** を使用して、そのような関数のランタイム・パフォーマンスを改善できます。

## 例

次の例では、**add\_logical** と呼ばれる関数を定義するために、**#pragma mc\_func** が使用されています。関数は、マシン・インストラクションから構成されて、いわゆる循環桁上げ を使用して 2 つの `int` を加算しています。つまり、加算の結果、桁上げが発生する場合、桁上げが合計に加算されます。これは、チェックサム計算において頻繁に使用されます。

例では、**#pragma mc\_func** によって定義された関数による変更が可能な揮発性レジスターの特定のセットをリストする **#pragma reg\_killed\_by** の使用法も示しています。

```
int add_logical(int, int);
#pragma mc_func add_logical {"7c632014" "7c630194"}
/* addc r3 <- r3, r4 */
/* addze r3 <- r3, carry bit */

#pragma reg_killed_by add_logical gr3, xer
/* only gpr3 and the xer are altered by this function */

main() {

 int i,j,k;

 i = 4;
 k = -4;
 j = add_logical(i,k);
 printf("%n%result = %d%n",j);
}
```

## 関連資料

349 ページの『汎用プラグマ』

379 ページの『**#pragma isolated\_call**』

408 ページの『**#pragma reg\_killed\_by**』

89 ページの『**asm**』



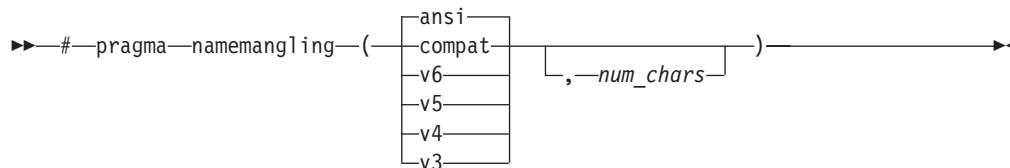
## #pragma namemangling

➤ C++

### 説明

**#pragma namemangling** ディレクティブは、ネーム・マングリング方式と、C++ ソース・コードから生成される外部シンボル名の最大長を設定します。

### 構文



ここで、

- |        |                                                                                                                                                                                                                                                                                                                       |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ansi   | ネーム・マングリング方式は、標準 C++ の各種の言語フィーチャーを、関数テンプレートの多重定義も含めて、完全にサポートしています。 <b>ANSI</b> を指定していても、 <i>num_chars</i> でサイズを指定していない場合には、デフォルトの最大値は 64000 文字になります。                                                                                                                                                                  |
| v6     | ネーム・マングリング方式は、VisualAge C++ バージョン 6.0 との互換性があります。 <b>v6</b> を指定していても、 <i>num_chars</i> でサイズを指定していない場合には、デフォルトの最大値は 64000 文字になります。                                                                                                                                                                                     |
| v5     | ネーム・マングリング方式は、VisualAge C++ バージョン 5.0 との互換性があります。 <b>v5</b> を指定していても、 <i>num_chars</i> でサイズを指定していない場合には、デフォルトの最大値は 64000 文字になります。                                                                                                                                                                                     |
| v4     | ネーム・マングリング方式は、VisualAge C++ バージョン 4.0 との互換性があります。 <b>v4</b> を指定していても、 <i>num_chars</i> でサイズを指定していない場合には、デフォルトの最大値は 64000 文字になります。                                                                                                                                                                                     |
| v3     | ネーム・マングリング方式は、VisualAge C++ バージョン 4.0 より前のバージョンとの互換性があります。 <b>v3</b> を指定していても、 <i>num_chars</i> でサイズを指定していない場合には、デフォルトの最大値は 255 文字になります。この方式は、XL C/C++ のバージョン 4.0 より前にリリースされたバージョンで作成したリンク・モジュール、あるいは <b>#pragma namemangling</b> または <b>-qnamemangling=compat</b> コンパイラー・オプションを指定して作成されたリンク・モジュールとの互換性のために、使用してください。 |
| compat | 上記の <b>v3</b> サブオプションと同様。                                                                                                                                                                                                                                                                                             |

### 注

**ansi** ネーム・マングリング方式に対する変更点の詳細については、**-qnamemangling** を参照してください。

**#pragma namemangling** ディレクティブを使用して、ネーム・マングリング方式に対する変更点からヘッダー・ファイルを保護することができます。例えば、指定したヘッダー・ファイルが常に **v5** ネーム・マングリング方式を使用するには、ヘッダー・ファイルの先頭に **#pragma namemangling(v5)** を挿入し、そのファイルの最後に **#pragma namemangling(pop)** を付加します。

## 関連資料

349 ページの『汎用プラグマ』

241 ページの『namemangling』

391 ページの『#pragma nameManglingRule』


## #pragma nameManglingRule

➤ C++

### 説明

**#pragma nameManglingRule** ディレクティブは、関数仮パラメーターの型に応じて関数名をマングルするかどうかをコンパイラーに指示します。

### 構文

```
➤ #pragma nameManglingRule (—fnparmtype—, )
```

ここで、

- |     |                                                                                                                                                                          |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| on  | 関数引き数は関数仮パラメーターの型に応じてマングルされます。例えば、関数引き数の cv 修飾子はマングルされません。                                                                                                               |
| off | ネーム・マングリングは、VisualAge C++ バージョン 5.0 と互換性があり、関数引き数の cv 修飾子はマングルされます。                                                                                                      |
| pop | 現在の <b>#pragma nameManglingRule</b> 設定を廃棄し、スタックにあるその直前の <b>#pragma nameManglingRule</b> 設定に置き換えます。直前の設定がスタックに残っていない場合は、デフォルトの <b>#pragma nameManglingRule</b> 設定が使用されます。 |

### デフォルト

**-qnamemangling=ansi** または **#pragma namemangling(ansi)** コンパイラー・オプションが有効なとき、デフォルトは、**#pragma nameManglingRule(fnparmtype, on)** です。

**-qnamemangling** または **#pragma namemangling** の他のすべての設定に関しては、デフォルトは、**#pragma nameManglingRule(fnparmtype, off)** です。

### 注

このディレクティブは、現行レベルの XL C/C++ コンパイラーの現在のレベルと、以前のバージョンとの間で、ネーム・マングリング方式の互換性を提供します。

**#pragma nameManglingRule** は、外部結合を持つブロック・スコープ関数宣言に影響を与えません。それは、グローバル、クラス、および関数スコープ内で許可されています。関数宣言および定義の前に、別のプラグマ設定を指定することができます。後続の宣言および定義内で **#pragma nameManglingRule** 設定が矛盾する場合、コンパイラーはこれらの設定を無視し、警告メッセージを出します。

指定した **#pragma nameManglingRule** 設定は、別の **#pragma nameManglingRule** 設定でオーバーライドされるまで有効なままとなります。

それぞれの新規 **#pragma nameManglingRule** 設定は、スタック上の、前に指定された設定の一番上にプッシュされます。現在有効な設定は、**pop** サブオプションでスタックの一番上から除去することができます。スタックに保管されている前の設定が残っている場合は、その設定に置き換えられます。設定が残っていない場合は、デフォルトの **#pragma nameManglingRule** 設定が使用されます。

## 関連資料

349 ページの『汎用プラグマ』

241 ページの『namemangling』

389 ページの『#pragma namemangling』

## #pragma object\_model

➤ C++

### 説明

**#pragma object\_model** ディレクティブは、このディレクティブの後に続く構造体、共用体、およびクラスに使用するオブジェクト・モデルを指定します。

### 構文

```
➤ #pragma object_model (

compat
ibm
gccv3
pop

) ➤
```

ここで、オブジェクト・モデルの選択項目は以下のとおりです。

|        |                                                                             |
|--------|-----------------------------------------------------------------------------|
| compat | 以前のバージョンのコンパイラと互換性のあるオブジェクト・モデルを使用する。                                       |
| ibm    | 新しいオブジェクト・モデルを使用する。                                                         |
| pop    | 以前に有効であったオブジェクト・モデルの設定に戻す。以前のオブジェクト・モデルの設定が存在しない場合は、オブジェクト・モデルをデフォルトに設定します。 |

### 注

このプラグマは、別の **#pragma object\_model** ステートメントに到達するまで、このプラグマの後に続く構造体、共用体、およびクラスに対して有効となります。

このプラグマを使用すると、現行のオブジェクト・モデルの設定はスタックに配置されます。このプラグマを次に使用すると、スタックの一番上に最新の設定が配置されます。**#pragma object\_model(pop)** を指定すると、現行のオブジェクト・モデルの設定がスタックから除去され、オブジェクト・モデルはスタック内の次の設定に設定されます。

### 関連概念

6 ページの『オブジェクト・モデル』

### 関連資料

349 ページの『汎用プラグマ』

250 ページの『objmodel』

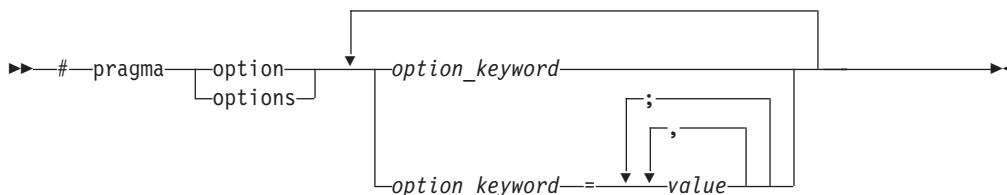
## #pragma options

► C ► C++

### 説明

**#pragma options** ディレクティブは、ソース・プログラムのコンパイラー・オプションを指定します。

### 構文



### 注

デフォルトでは、プラグマ・オプションは通常、コンパイル単位全体に適用されます。

**#pragma options** ディレクティブで複数のコンパイラー・オプションを指定するには、ブランク・スペースを使ってオプションを分割します。例を以下に示します。

```
#pragma options langlvl=stdc89 halt=s spill=1024 source
```

ほとんどの **#pragma options** ディレクティブは、ソース・プログラムのいずれのステートメントよりも前に指定しなければなりません。ただし、コメント、ブランク行、および他の **#pragma** の指定に限り、これらのディレクティブの前に置けます。例えば、以下のように、プログラムの最初の数行をコメントにして、その後に **#pragma options** ディレクティブを続けることができます。

```
/* The following is an example of a #pragma options directive: */
#pragma options langlvl=stdc89 halt=s spill=1024 source
/* The rest of the source follows ... */
```

ソース・プログラムのすべてのコードの前に指定されたオプションは、コンパイル単位全体に適用されます。プログラムの中で、他の **#pragma** ディレクティブを使用すると、ソース・コードの選択済みブロックに対してオプションをオンにすることができます。例えば、以下のとおり、ソース・コードの一部をコンパイラー・リストに組み込むように要求することができます。

```
#pragma options source

/* Source code between the source and nosource #pragma
 options is included in the compiler listing */

#pragma options nosource
```

下記の表の設定は、**#pragma options** で有効なオプションです。詳しくは、同等のコンパイラー・オプションのページを参照してください。

| 言語アプリケーション |       | #pragma options に有効な設定 <i>option_keyword</i> | 同等のコンパイラー・オプション                  | 説明                                                                                                                                                                                                                                               |
|------------|-------|----------------------------------------------|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ➤ C        | ➤ C++ | align= <i>option</i>                         | -qalign                          | コンパイラーがファイルのコンパイルに使用する集合体の位置合わせ方式を指定する。                                                                                                                                                                                                          |
| ➤ C        | ➤ C++ | [no]ansialias                                | -qansialias                      | 型ベースの別名割り当てを最適化中に使用するかどうかを指定する。                                                                                                                                                                                                                  |
| ➤ C        | ➤ C++ | assert= <i>option</i>                        | -qassert                         | 別名割り当てアサーションをコンパイル単位に適用するようにコンパイラーに要求する。                                                                                                                                                                                                         |
| ➤ C        | ➤ C++ | [no]attr<br><br>attr=full                    | -qattr                           | すべての名前を含む属性のリストを作成する。                                                                                                                                                                                                                            |
| ➤ C        | ➤ C++ | chars= <i>option</i>                         | -qchars<br><br>#pragma chars も参照 | コンパイラーに、char 型の変数をすべて signed または unsigned のいずれかとして処理するように指示する。                                                                                                                                                                                   |
| ➤ C        | ➤ C++ | [no]check                                    | -qcheck                          | 特定のタイプの実行時検査を行うコードを生成する。                                                                                                                                                                                                                         |
| ➤ C        | ➤ C++ | [no]compact                                  | -qcompact                        | 最適化とともに使用すると、可能な場合に、実行速度を犠牲にしてコード・サイズが削減される。                                                                                                                                                                                                     |
| ➤ C        | ➤ C++ | [no]dbcs                                     | -qmbcs、dbcs                      | ストリング・リテラルとコメントには、DBCS 文字を含むことができる。                                                                                                                                                                                                              |
| ➤ C        |       | [no]dbxextra                                 | -qdbxextra                       | 参照されない変数のためのシンボル・テーブル情報を生成する。                                                                                                                                                                                                                    |
| ➤ C        | ➤ C++ | [no]digraph                                  | -qdigraph                        | 特定の連字とキーワード演算子を使用することができる。                                                                                                                                                                                                                       |
| ➤ C        | ➤ C++ | [no]dollar                                   | -qdollar                         | \$ シンボルを ID の名前でできるようにする。                                                                                                                                                                                                                        |
| ➤ C        | ➤ C++ | enum= <i>option</i>                          | -qenum<br><br>#pragma enum も参照   | 列挙の占めるストレージの量を指定する。                                                                                                                                                                                                                              |
| ➤ C        | ➤ C++ | [no]extchk                                   | -qextchk                         | 外部名の型検査と関数呼び出しの検査を実行する。                                                                                                                                                                                                                          |
| ➤ C        | ➤ C++ | flag= <i>option</i>                          | -qflag                           | 報告させる診断メッセージの最低の重大度レベルを指定する。<br><br>重大度レベルは、以下のように指定することもできます。<br><br>#pragma options flag=i => #pragma report (level,I)<br><br>#pragma options flag=w => #pragma report (level,W)<br><br>#pragma options flag=e,s,u => #pragma report (level,E) |

| 言語アプリケーション |       | #pragma options に有効な設定 <i>option_keyword</i> | 同等のコンパイラー・オプション                              | 説明                                                                                                                                                                 |
|------------|-------|----------------------------------------------|----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ➤ C        | ➤ C++ | float=[no] <i>option</i>                     | -qfloat                                      | 浮動小数点演算を高速化するか正確度を向上させるために、各種浮動小数点オプションを指定する。                                                                                                                      |
| ➤ C        | ➤ C++ | [no]flttrap= <i>option</i>                   | -qflttrap                                    | 追加の命令を生成し、浮動小数点例外を検出してトラップする。                                                                                                                                      |
| ➤ C        | ➤ C++ | [no]fold                                     | -qfold                                       | 浮動小数点定数式をコンパイル時に評価するように指定する。                                                                                                                                       |
| ➤ C        | ➤ C++ | [no]fullpath                                 | -qfullpath                                   | ファイルに保管されているパス情報を dbx スタブ・ストリングに指定する。                                                                                                                              |
| ➤ C        | ➤ C++ | [no]funcsect                                 | -qfuncsect                                   | 各関数ごとの命令を別個の cset に入れる。                                                                                                                                            |
| ➤ C        | ➤ C++ | halt                                         | -qhalt                                       | 指定された重大度のエラーが検出されると、コンパイラーを停止する。                                                                                                                                   |
| ➤ C        | ➤ C++ | [no]idirfirst                                | -qidirfirst                                  | ユーザー・インクルード・ファイルの検索順序を指定する。                                                                                                                                        |
| ➤ C        | ➤ C++ | [no]ignerrno                                 | -qignerrno                                   | コンパイラーが、システム呼び出しによって <b>errno</b> が変更されないと想定して最適化を行うことを許可する。                                                                                                       |
| ➤ C        | ➤ C++ | ignprag= <i>option</i>                       | -qignprag                                    | 特定のプラグマ・ステートメントを無視するようにコンパイラーに指示する。                                                                                                                                |
| ➤ C        | ➤ C++ | [no]info= <i>option</i>                      | -qinfo<br>#pragma info も参照                   | 通知メッセージを作成する。                                                                                                                                                      |
| ➤ C        | ➤ C++ | initauto= <i>value</i>                       | -qinitauto                                   | 指定された 16 進バイト値に自動ストレージを初期化する。                                                                                                                                      |
| ➤ C        | ➤ C++ | [no]inlglue                                  | -qinlglue                                    | 外部関数の呼び出しまたは関数ポインターを介した呼び出しを行うために必要なポインター・グルー・コードをインライン化することによって、高速な外部結合を生成する。                                                                                     |
| ➤ C        | ➤ C++ | isolated_call= <i>names</i>                  | -qisolated_call<br>#pragma isolated_call も参照 | ソース・ファイル内の副次作用がない関数を指定する。                                                                                                                                          |
| ➤ C        |       | langlvl                                      | -qlanglvl                                    | 異なる言語レベルを指定する。<br><br>このディレクティブは、プリプロセッサの動きを動的に変更することができます。結果として、 <b>-E</b> コンパイラー・オプションを指定してコンパイルすると、 <b>-E</b> オプションを指定しないでコンパイルしたときに生成される結果とは異なる結果を生成する場合があります。 |
| ➤ C        | ➤ C++ | [no]ldbl128                                  | -qldbl128、longdouble                         | long double 型のサイズを 64 ビットから 128 ビットに増加させる。                                                                                                                         |



| 言語アプリケーション |       | #pragma options に有効な設定 <i>option_keyword</i>          | 同等のコンパイラー・オプション                                 | 説明                                                                                                                                                                                                                                                                                         |
|------------|-------|-------------------------------------------------------|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ➤ C        | ➤ C++ | [no]libansi                                           | -qlibansi                                       | ANSI C ライブラリー関数の名前が付いたすべての関数が実際はシステム関数であると見なす。                                                                                                                                                                                                                                             |
| ➤ C        | ➤ C++ | [no]list                                              | -qlist                                          | オブジェクト・リストを含むコンパイラー・リストを生成する。                                                                                                                                                                                                                                                              |
| ➤ C        | ➤ C++ | [no]longlong                                          | -qlonglong                                      | プログラムで long long 型を許可する。                                                                                                                                                                                                                                                                   |
| ➤ C        |       | [no]macpstr                                           | -qmacpstr                                       | Pascal スtring・リテラルを、先頭バイトにStringの長さが含まれるヌル終了Stringに変換する。                                                                                                                                                                                                                                   |
| ➤ C        | ➤ C++ | [no]maf                                               | -qmaf                                           | 浮動小数点乗算・加算命令を生成するかどうかを指定する。                                                                                                                                                                                                                                                                |
| ➤ C        | ➤ C++ | [no]maxmem= <i>number</i>                             | -qmaxmem                                        | 指定した重大度以上のエラーの件数が指定した数に達した場合に、コンパイルを停止するようにコンパイラーに指示する。                                                                                                                                                                                                                                    |
| ➤ C        | ➤ C++ | [no]mbcs                                              | -qmbcs、dbcs                                     | String・リテラルとコメントには、DBCS 文字を含むことができる。                                                                                                                                                                                                                                                       |
| ➤ C        | ➤ C++ | [no]optimize<br>optimize= <i>number</i>               | -O、optimize                                     | <p>プログラム・コードのセクションに適用される最適化レベルを指定する。</p> <p>コンパイラーは <i>number</i> の値として以下を受け入れます。</p> <ul style="list-style-type: none"> <li>• 0 - レベル 0 の最適化を設定します</li> <li>• 2 - レベル 2 の最適化を設定します</li> <li>• 3 - レベル 3 の最適化を設定します</li> </ul> <p><i>number</i> に値が指定されなかった場合、コンパイラーはレベル 2 の最適化を想定します。</p> |
|            | ➤ C++ | priority= <i>number</i>                               | -qpriority<br><br>406 ページの『#pragma priority』も参照 | 静的コンストラクターを初期化する場合の優先順位を指定する。                                                                                                                                                                                                                                                              |
| ➤ C        | ➤ C++ | [no]proclcal、<br>[no]procimported、<br>[no]procunknown | -qproclcal、procimported、<br>procunknown         | ローカル、インポートされるもの、または不明として、関数にマークを付ける。                                                                                                                                                                                                                                                       |
| ➤ C        |       | [no]proto                                             | -qproto                                         | このオプションが設定されると、コンパイラーは、すべての関数がプロトタイプ化されているものと想定する。                                                                                                                                                                                                                                         |
| ➤ C        | ➤ C++ | [no]ro                                                | -qro                                            | String・リテラルの保管型を指定する。                                                                                                                                                                                                                                                                      |
| ➤ C        | ➤ C++ | [no]roconst                                           | -qroconst                                       | 定数値の保管場所を指定する。                                                                                                                                                                                                                                                                             |
| ➤ C        | ➤ C++ | [no]roptr                                             | -qroptr                                         | 定数ポインターの保管場所を指定する。                                                                                                                                                                                                                                                                         |

| 言語アプリケーション |       | #pragma options に有効な設定 <i>option_keyword</i> | 同等のコンパイラー・オプション | 説明                                                                   |
|------------|-------|----------------------------------------------|-----------------|----------------------------------------------------------------------|
| ➤ C        | ➤ C++ | [no]rrm                                      | -qrrm           | 正および負の無限大への実行時丸めモードと互換性がない浮動小数点の最適化を回避する。                            |
| ➤ C        | ➤ C++ | [no]showinc                                  | -qshowinc       | <b>-qsource</b> で使用された場合は、すべてのインクルード・ファイルをソース・リストに組み込む。              |
| ➤ C        | ➤ C++ | [no]source                                   | -qsource        | ソース・リストを作成する。                                                        |
| ➤ C        | ➤ C++ | spill= <i>number</i>                         | -qspill         | レジスター割り振り予備域のサイズを指定する。                                               |
| ➤ C        |       | [no]srcmsg                                   | -qsrcmsg        | <b>stderr</b> ファイル内の診断メッセージに、対応するソース・コード行を追加する。                      |
| ➤ C        | ➤ C++ | [no]stdinc                                   | -qstdinc        | #include <file_name> および #include "file_name" ディレクティブで組み込むファイルを指定する。 |
| ➤ C        | ➤ C++ | [no]strict                                   | -qstrict        | プログラムのセマンティクスを変更する可能性がある <b>-O3</b> コンパイラー・オプションの積極的な最適化をオフにする。      |
| ➤ C        | ➤ C++ | tbtable= <i>option</i>                       | -qtbtable       | コンパイラーによって認識されるタブの長さを変更する。                                           |
| ➤ C        | ➤ C++ | tune= <i>option</i>                          | -qtune          | 実行可能プログラムの最適化対象とするアーキテクチャーを指定する。                                     |
| ➤ C        | ➤ C++ | [no]unroll<br>unroll= <i>number</i>          | -qunroll        | 指定した係数によってプログラムの内部ループをアンロールする。                                       |
| ➤ C        |       | [no]upconv                                   | -qupconv        | 整数拡張を行うときに <b>unsigned</b> の指定を保持する。                                 |
|            | ➤ C++ | [no]vftable                                  | -qvftable       | 仮想関数テーブルの生成を制御する。                                                    |
| ➤ C        | ➤ C++ | [no]xref                                     | -qxref          | すべての ID の相互参照リストを含むコンパイラー・リストを生成する。                                  |

## 関連資料

349 ページの『汎用プラグマ』

126 ページの『E』

## #pragma option\_override

► C ► C++

### 説明

**#pragma option\_override** ディレクティブにより、特定の関数に対する代替最適化オプションを指定します。

### 構文

► `#pragma option_override ( fname , option )` ►

*option* の有効な設定と構文、およびそれらに対応するコマンド行オプションを下記に示します。

| #pragma option_override <i>option</i> の設定および構文 | コマンド行オプション                 | 例                                                                                                |
|------------------------------------------------|----------------------------|--------------------------------------------------------------------------------------------------|
| opt(level,number)                              | -O、-O2、<br>-O3、-O4、<br>-O5 | #pragma option_override (fname, "opt(level, 3)")                                                 |
| opt(registerSpillSize,num)                     | -qspill=num                | #pragma option_override (fname, "opt(registerSpillSize,512)")                                    |
| opt(size[,yes])                                | -qcompact                  | #pragma option_override (fname, "opt(size)")<br>#pragma option_override (fname, "opt(size,yes)") |
| opt(size,no)                                   | -qnocompact                | #pragma option_override (fname, "opt(size,no)")                                                  |
| opt(strict)                                    | -qstrict                   | #pragma option_override (fname, "opt(strict)")                                                   |
| opt(strict,no)                                 | -qnostrict                 | #pragma option_override (fname, "opt(strict,no)")                                                |

### 注

デフォルトでは、コマンド行で指定した最適化オプションは、ソース・プログラム全体に適用されます。ただし、最適化がオンになっている場合のみ、特定のタイプのランタイム・エラーが発生する可能性があります。このプラグマにより、プログラム内の特定の関数 (*fname*) に対するコマンド行最適化設定をオーバーライドします。これは、これらの関数のプログラミング・エラーを識別したり修正したりする場合に役立ちます。

関数単位の最適化は、コンパイル・オプションによって最適化がすでに使用可能になっている場合にのみ有効です。コンパイル中の残りのプログラムに適用されたレベルより下のレベルでの、関数単位の最適化を要求することができます。このプラグマを介してオプションを選択すると、選択された特定の最適化オプションにのみ有効になり、関連オプションの暗黙設定には無効です。

オプションを指定するときは、二重引用符で囲むため、マクロ展開には影響されません。引用符で囲んで指定したオプションは、ビルド・オプションの構文に従わなければなりません。

このプラグマは、多重定義されたメンバー関数とともに使用することはできません。

このプラグマは、コンパイル単位で定義された関数にのみ有効で、例えば、以下の  
ように、コンパイル単位の任意の場所に置くことができます。

- コンパイル単位の前または後ろ
- 関数定義の前または後ろ
- 関数宣言の前または後ろ
- 参照された関数の前または後ろ
- 関数定義の内側または外側

## 関連資料

349 ページの『汎用プラグマ』

108 ページの『compact』

244 ページの『O、optimize』

299 ページの『spill』

305 ページの『strict』

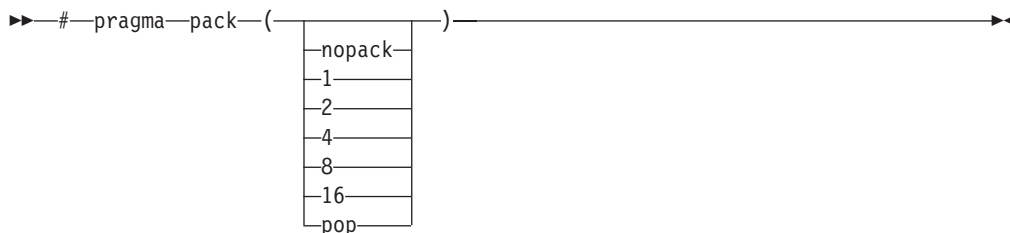
## #pragma pack

► C ► C++

### 説明

**#pragma pack** ディレクティブは、このディレクティブの後に続く構造体のメンバーに対する現行の位置合わせ方式を変更します。

### 構文



ここで、

|                    |                                                                             |
|--------------------|-----------------------------------------------------------------------------|
| 1   2   4   8   16 | 構造体のメンバーは、指定したバイト位置合わせ、または自然な位置合わせ境界のいずれか少ない方で位置合わせされ、指定された値はスタックにプッシュされます。 |
| nopack             | パッキングは適用されず、パック・スタックには "nopack" がプッシュされます。                                  |
| pop                | プラグマ・パック・スタックの一番上のエレメントがポップされます。                                            |
| (引き数の指定なし)         | #pragma pack() の指定は、 #pragma pack(pop) の指定と同じ効果があります。                       |

### 注

**#pragma pack** ディレクティブは、このディレクティブの後に続けて宣言される構造体のメンバーのみに対して現行の位置合わせ方式を変更します。これにより、構造体の位置合わせに直接、影響することはありませんが、構造体のメンバーの位置合わせに影響することで、位置合わせ方式に従って、構造体全体の位置合わせに影響する場合があります。

**#pragma pack** ディレクティブでは、メンバーの位置合わせを強化することはできず、むしろ位置合わせを低下させる可能性があります。例えば、整数データ型 (int) のメンバーの場合、**#pragma pack(2)** ディレクティブでは、当該メンバーは構造体内で 2 バイト境界でパックされますが、**#pragma pack(4)** ディレクティブでは有効ではありません。

**#pragma pack** ディレクティブは、スタックをベースにしています。すべてのパック値は、ソース・コードの構文解析時にスタックにプッシュされます。現行のプラグマ・パック・スタックの一番上にある値が、現行の位置合わせ方式の範囲内の後続のすべての構造体のメンバーをパックするために使用されます。

**#pragma pack** スタックは、位置合わせ方式スタック内の現行エレメントに関連付けられます。位置合わせ方式は、**-qalign** コンパイラー・オプションまたは

**#pragma options align** ディレクティブで指定します。新規の位置合わせ方式が作成される場合、新規 **#pragma pack** スタックが作成されます。現行の位置合わせ方式がポップされて、位置合わせ方式スタックから外された場合、現行の **#pragma pack** スタックは空になり、直前の **#pragma pack** スタックが復元されます。スタック操作 (パック設定のプッシュおよびポップ) は、現行の **#pragma pack** スタックにのみ影響します。

**#pragma pack** ディレクティブは、ビット・フィールド内のビットの位置合わせには影響しません。

## 例

- 以下に示すコードでは、構造体 `s_t2` のメンバーは 1 バイトにパックされますが、構造体 `s_t1` はその影響を受けません。これは、`s_t1` の宣言がプラグマ・ディレクティブよりも前に開始されているためです。ただし、`s_t2` の宣言はプラグマ・ディレクティブより後に始まるため、`s_t2` は影響を受けます。

```
struct s_t1 {
 char a;
 int b;
 #pragma pack(1)
 struct s_t2 {
 char x;
 int y;
 } S2;
 char c;
 int d;
} S1;
```

- 以下のコード・セグメントは、次のようになっています。

- `s_t1` のメンバーは 2 バイト位置合わせ方式で位置合わせされ、`s_t2` のメンバーは 1 バイトでパックされ、`s_t3` のメンバーは 2 バイトでパックされ、`s_t4` は 4 バイトでパックされます。
- #pragma options align=reset** ディレクティブは、現行の位置合わせ方式 (上記の場合では 2 バイト位置合わせ方式) をポップします。 **#pragma options align=twobyte** ディレクティブが有効である間に発行された **#pragma pack** ディレクティブもすべてポップされます。
- #pragma options align=twobyte** ディレクティブがポップされるまで有効であった位置合わせ方式と同様に、検出された **#pragma pack(4)** ディレクティブも復元されます。

```
#pragma pack(4)
#pragma options align=twobyte
struct s_t1 {
 char a;
 int b;
} S1;

#pragma pack(1)
struct s_t2 {
 char a;
 short b;
} S2;

#pragma pack(2)
struct s_t3 {
 char a;
 double b;
} S3;
```

```
#pragma options align=reset
struct s_t4 {
 char a;
 int b;
} S4;
```

3. この例では、**#pragma pack** ディレクティブが構造体のサイズとマッピングにどのように影響を与える可能性があるのかを示します。

```
struct s_t {
 char a;
 int b;
 short c;
 int d;
} S;
```

デフォルト・マッピング:

```
sizeof s_t = 16
offsetof a = 0
offsetof b = 4
offsetof c = 8
offsetof d = 12
align of a = 1
align of b = 4
align of c = 2
align of d = 4
```

**#pragma pack(1):**

```
sizeof s_t = 11
offsetof a = 0
offsetof b = 1
offsetof c = 5
offsetof d = 7
align of a = 1
align of b = 1
align of c = 1
align of d = 1
```

## 関連資料

349 ページの『汎用プラグマ』

78 ページの『align』

394 ページの『#pragma options』

## #pragma pass\_by\_value

➤ C++

### 説明

**#pragma pass\_by\_value** ディレクティブは、const または参照メンバーが入ったクラスを関数引き数で受け渡す方法を指定します。コンパイル単位のクラスはすべて、このオプションの影響を受けます。

### 構文

```
➤ #pragma pass_by_value (compat)
```

|         |
|---------|
| ansi    |
| default |
| source  |
| pop     |
| reset   |

ここで、

|         |                                                                                                                    |
|---------|--------------------------------------------------------------------------------------------------------------------|
| compat  | <b>-qoldpassbyvalue</b> と同等のものをスタックにプッシュします。                                                                       |
| ansi    | <b>-qnooldpassbyvalue</b> オプションと同等のものをスタックにプッシュします。                                                                |
| default | オプション <b>-qnooldpassbyvalue</b> のコンパイラーのデフォルト設定をスタックにプッシュします。                                                      |
| source  | 元のコマンド行オプション ( <b>-qoldpassbyvalue</b> または <b>-qnooldpassbyvalue</b> ) の値をスタックにプッシュします。                            |
| pop     | スタックをポップして、現行の <b>#pragma pass_by_value</b> 設定を廃棄し、これから有効なオプションとして、直前の <b>#pragma pass_by_value</b> 設定をスタックに復元します。 |
| reset   | <b>pop</b> と同じ。                                                                                                    |

### 注

**#pragma pass\_by\_value** の現行設定は、プラグマの後に続けられるプログラム・ソース・コードに関数引き数としてクラスを受け渡す方法を指定します。この設定は、次の当該プラグマの使用で新規設定を呼び出すまで有効となります。

**#pragma pass\_by\_value** の設定は、**-qoldpassbyvalue** コンパイラー・オプションをオーバーライドします。

空のスタックに対して **pop** または **reset** が呼び出されると、コンパイラーは、警告メッセージを発行し、コマンド行に本来、設定されていた **-qoldpassbyvalue** 設定を想定します。 **-qoldpassbyvalue** がコマンド行に設定されていない場合、コンパイラーは、コンパイラー・デフォルト構成ファイルに設定されているデフォルト設定を想定します。

**#pragma pass\_by\_value(compat)** は、クラスを関数引き数として渡す際に、以前のバージョンの IBM C/C++ コンパイラー (v3.6 以前) の動きを模倣するようにコンパイラーに指示するために使用します。 **const** または参照メンバーを含むクラスは、値では受け渡されません。



## 関連資料

349 ページの『汎用プラグマ』

251 ページの『oldpassbyvalue』

## #pragma priority

➤ C++

### 説明

**#pragma priority** ディレクティブは、静的オブジェクトを初期化する順序を指定します。

### 構文

➤ #pragma priority(—*n*—) ➤

### 注

*n* の値は、INT\_MIN から INT\_MAX までの範囲の整数リテラルでなければなりません。デフォルト値は 0 です。負の値は、優先順位が高いことを示します。正の値は、優先順位が低いことを示します。最初から 1024 までの優先順位 (INT\_MIN から INT\_MIN + 1023) は、コンパイラーとコンパイラーのライブラリーが使用するため予約されています。

明示的な値が **init\_priority** 属性を経由して指定されるか、または別の **#pragma priority** ディレクティブが検出されない限り、優先度の値は **#pragma priority** ディレクティブに続くすべてのグローバルおよび静的オブジェクトに適用されます。

同じ優先度の値を持つオブジェクトは、宣言の順序で構成されます。 **#pragma priority** を使用して、複数のファイルにまたがったオブジェクトの作成順序を指定します。ただし、ソース・ファイルから、実行可能な、または共用のライブラリー・ターゲットを作成する場合、コンパイラーは、 **#pragma priority** をオーバーライドする可能性のある、依存性の順序付けを検査します。

例えば、オブジェクト A のコピーがオブジェクト B コンストラクターにパラメーターとして渡されると、コンパイラーは、上から下、または **#pragma priority** の順序に違反することになっても、最初に構成される A に対する調整を行います。このことは、コンパイラーが許可する、オーダーレス・プログラミングに不可欠です。ターゲットが .obj.lib の場合、依存性を検出するための情報が十分ではない可能性があるため、この処理は行われません。

### 例

```
#pragma priority(1001)
```

### 関連資料

349 ページの『汎用プラグマ』

171 ページの『info』

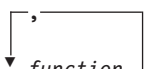
## #pragma reachable

▶ C ▶ C++

### 説明

**#pragma reachable** ディレクティブは、ルーチンである関数 を呼び出した後のポイントが、いくつかの不明のロケーションからの分岐の対象となる可能性があることを宣言します。このプラグマは、`setjmp` とともに使用してください。

### 構文

▶▶ #pragma reachable (  ) ▶▶

### 関連資料

349 ページの『汎用プラグマ』

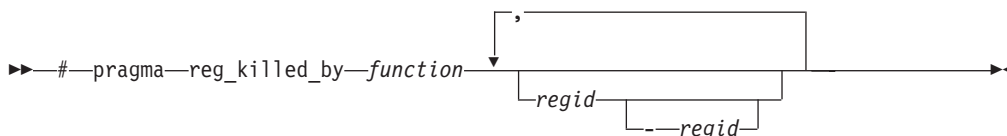
## #pragma reg\_killed\_by

➤ C ➤ C++

### 説明

**#pragma reg\_killed\_by** ディレクティブは、指定の関数によって変更（強制終了）可能な一連の揮発性レジスターを指定する。この pragma は、**#pragma mc\_func** を使って定義された関数でしか使用できません。

### 構文



ここで、

*function*  
*regid*

以前に **#pragma mc\_func** を使って定義した関数。  
指定された *function* によって変更される、単一レジスターまたは一連のレジスターのシンボル名。一連のレジスターは、ダッシュで区切られた開始および終了の両レジスターのシンボル名を使用して識別されます。レジスターが指定されていない場合、どのレジスターも、指定された *function* によって変更されません。

シンボル名は、2 つの部分から構成されています。1 つ目の部分は、レジスター・クラス名であり、“a” から “z” および/または “A” から “Z” の範囲にある 1 つ以上の文字のシーケンスを使用して指定されます。

2 つ目の部分は、符号なし int の範囲にある整数です。この数値は、レジスター・クラス内の特定のレジスター番号を表します。一部のレジスター・クラスには、レジスター番号の指定が不要であり、レジスター番号を指定しようとすると、エラーが起こります。

*regid* が指定されていない場合、どの揮発性レジスターも、指定された *function* によって強制終了されません。

| レジスター               |                                                                                                                                                                                                                             |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| クラスおよび<br>[レジスター番号] | 説明および使用法                                                                                                                                                                                                                    |
| ctr                 | カウント・レジスター (CTR)                                                                                                                                                                                                            |
| cr[0-7]             | 条件レジスター (CR) <ul style="list-style-type: none"><li>このクラス内の各レジスターは、条件レジスター内の 4 ビット・フィールドの 1 つです。</li><li>8 CR フィールドのうち、<b>cr0</b>、<b>cr1</b>、および <b>cr5-cr7</b> のみが <b>#pragma reg_killed_by</b> によって指定されることができます。</li></ul> |
| fp[0-31]            | 浮動小数点レジスター (FPR) <ul style="list-style-type: none"><li>32 個のマシン・レジスターのうち、<b>fp0</b> ~ <b>fp13</b> のみが <b>#pragma reg_killed_by</b> によって指定されることができます。</li></ul>                                                              |
| fs                  | 浮動小数点状況および制御レジスター (FPSCR)                                                                                                                                                                                                   |

|          |                                                                                                                                                                               |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| lr       | リンク・レジスター (LR)                                                                                                                                                                |
| mq       | MQ レジスター (MQ)                                                                                                                                                                 |
| gr[0-31] | 汎用レジスター (GPR) <ul style="list-style-type: none"> <li>32 個のマシン・レジスターのうち、 <b>gr0</b>、および <b>gr3</b> ～ <b>gr12</b> のみが <b>#pragma reg_killed_by</b> によって指定されることができます。</li> </ul> |
| vr[0-31] | ベクトル・レジスター (Altivec プロセッサのみ)                                                                                                                                                  |
| xer      | 固定小数点例外 (XER)                                                                                                                                                                 |

## 注

通常、**#pragma mc\_func** によって指定された関数用に生成されたコードは、ご使用のシステムで使用可能な揮発性レジスターをどれでも変更できます。 **#pragma reg\_killed\_by** を使用して、そのような関数によって変更される揮発性レジスターの特定のセットを明示的にリストすることができます。このリストにないレジスターは、変更されません。

*regid* によって指定されたレジスターは、以下の要件を満たさなければなりません。

- 登録名のクラス名部分は有効でなければならない
- レジスター番号は必須または禁止である
- レジスター番号が必須であるときは、有効範囲内になければならない

これらの要件のいずれかが満たされない場合は、エラーが発行され、プラグマは無視されます。

## 例

次の例では、 **#pragma mc\_func** によって定義された関数が使用する揮発性レジスターの特定のセットをリストする **#pragma reg\_killed\_by** の使用法を示しています。

```
int add_logical(int, int);
#pragma mc_func add_logical {"7c632014" "7c630194"}
/* addc r3 <- r3, r4 */
/* addze r3 <- r3, carry bit */

#pragma reg_killed_by add_logical gr3, xer
/* only gpr3 and the xer are altered by this function */

main() {

 int i,j,k;

 i = 4;
 k = -4;
 j = add_logical(i,k);
 printf("%n%nresult = %d%n%n",j);
}
```

## 関連資料

349 ページの『汎用プラグマ』

387 ページの『#pragma mc\_func』

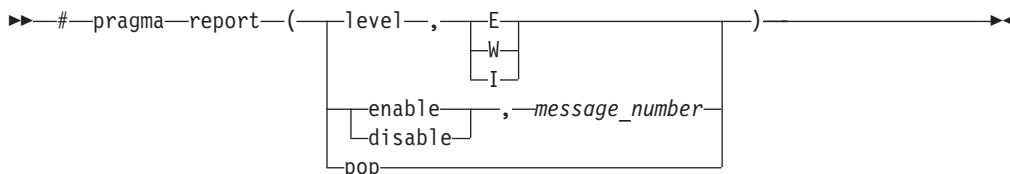
## #pragma report

➤ C++

### 説明

**#pragma report** ディレクティブは、特定のメッセージの生成を制御します。プラグマは、**#pragma info** に優先します。**#pragma report(pop)** を指定すると、レポート・レベルは直前のレベルに戻されます。前のレポート・レベルが指定されていないと、警告を出して、そのレポート・レベルがそのまま変更されません。

### 構文



ここで、

|                  |                                                                                                                                                                                                              |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| レベル              | 表示する診断メッセージの最小の重大度レベルを示します。                                                                                                                                                                                  |
| E   W   I        | 表示する診断メッセージの型を決定するためのレベルの論理積で使<br>用します。                                                                                                                                                                      |
| E                | 'error' の最小のメッセージ重大度を示します。これは診断メッ<br>セージの最も厳密な型であると見なされます。 'E' のレポート・レ<br>ベルは、'error' メッセージだけ表示します。別の方法で 'E' にレ<br>ポート・レベルを設定するには、 <b>-qflag=e:e</b> コンパイラー・オプ<br>ションを指定します。                                 |
| W                | 'warning' の最小のメッセージ重大度を示します。'W' のレポ<br>ート・レベルは、すべての通知メッセージを遮り、警告メッセージ<br>またはエラーのメッセージとして分類されたメッセージだけを表<br>示します。別の方法で 'W' にレポート・レベルを設定するに<br>は、 <b>-qflag=w:w</b> コンパイラー・オプションを指定します。                         |
| I                | 'information' の最小のメッセージ重大度を示します。通知メッ<br>セージは、診断メッセージの最も厳密でない型と見なします。 'I'<br>のレベルはすべてのメッセージ型を表示します。コンパイラーは<br>これをデフォルト・オプションとして設定します。別の方法で 'I'<br>にレポート・レベルを設定するには、 <b>-qflag=i:i</b> コンパイラー・<br>オプションを指定します。 |
| enable   disable | 指定したメッセージ番号を使用可能または使用不可にします。                                                                                                                                                                                 |
| message_number   | メッセージ番号が続く、メッセージ番号のプレフィックスを含む ID で<br>す。メッセージ番号の例: CPPC1004                                                                                                                                                  |
| pop              | 直前のレポート・レベルにレポート・レベルがリセットされます。ポップ<br>操作が空のスタックで実行されている場合、レポート・レベルは未変更の<br>まま有効であり、メッセージも生成されません。                                                                                                             |

## 例

1. **#pragma info** を指定すると、すべての情報診断を印刷するようにコンパイラーに指示します。プラグマのレポートは、**'W'** の重大度または警告メッセージを持つメッセージのみを表示するようにコンパイラーに指示します。この場合、情報診断は何も表示されません。

```
1 #pragma info(all)
2 #pragma report(level, W)
```

2. **CPPC1000** がエラー・メッセージの場合、表示されます。別の診断メッセージ型の場合は、表示されません。

```
1 #pragma report(enable, CPPC1000) // enables message number CPPC1000
2 #pragma report(level, E) // display only error messages.
```

次のようにコードの順序を変更すると、

```
1 #pragma report(level, E)
2 #pragma report(enable, CPPC1000)
```

同じ結果になります。コードの 2 つの行が表示される順序は、結果に影響しません。ただし、メッセージが **'disabled'** の場合、設定されているレポート・レベルおよび表示されるコードの行の順序に関係なく、診断メッセージは表示されません。

3. 下記の例の行 1 で、初期レポート・レベルが **'I'** に設定されているので、分類される診断メッセージの型に関係なくメッセージ **CPPC1000** が表示されます。行 3 は、新しいレポート・レベル **'E'** が設定され、重大度レベルが **'E'** であるメッセージのみが表示されることを示します。行 3 の直後に、現行レベル **'E'** が **'ポップ'** であり、**'I'** にリセットされます。

```
1 #pragma report(level, I)
2 #pragma report(enable, CPPC1000)
3 #pragma report(level, E)
4 #pragma report(pop)
```

## 関連資料

349 ページの『汎用プラグマ』

139 ページの『flag』

## #pragma stream\_unroll

➤ C ➤ C++

### 説明

**for** ループに含まれるストリームを複数のストリームに分割します。

### 構文

➤ `#pragma stream_unroll (  $\underbrace{\hspace{1cm}}$  )` ➤

ここで、 $n$  はループのアンロール係数です。C プログラムでは、値  $n$  は正の整数定数式です。C++ プログラムでは、値  $n$  は正のスカラ整数またはコンパイル時定数の初期設定式です。アンロール係数 1 はアンロールを使用不可にします。 $n$  を指定しないとき、および最適化が **-O3** またはそれ以上に設定されているときは、最適化プログラムにより、ネストされた各ループごとに該当するアンロール係数が決まります。

### 注

このプラグマは、POWER4、POWER5、および PowerPC 970 アーキテクチャーでのみ効果があります。

ストリーム・アンロールを使用可能にするには、**-qhot**、**-qipa=level=2**、または **-qsm** コンパイラ・オプションを指定する必要があります。最適化レベル **-O4** 以上により、コンパイラはストリーム・アンロールを実行することもできます。

ストリーム・アンロールを発生させるには、**#pragma stream\_unroll** ディレクティブが **for** ループより優先されなければなりません。

このディレクティブは、指定された **for** ループに対して複数回指定しないでください。このディレクティブを同じ **for** ループの **block\_loop**、**unroll**、**nounroll**、**unrollandfuse**、または **nounrollandfuse** ディレクティブと組み合わせて使用しないでください。警告メッセージは出されませんが、コンパイラは **stream\_unroll** 最適化を適用しない可能性があります。

### 例

以下に、**#pragma stream\_unroll** を使用してパフォーマンスを向上させる方法の例を示します。

```
int i, m, n;
int a[1000][1000];
int b[1000][1000];
int c[1000][1000];

....

#pragma stream_unroll(4)
for (i=1; i<n; i++) {
 a[i] = b[i] * c[i];
}
```

アンロール係数 4 は、以下のように、反復の数を  $n$  から  $n/4$  まで削減します。



```

for (i=1; i<n/4; i++) { /* consider m = n/4 */
 a[i] = b[i] + c[i];
 a[i+m] = b[i+m] + c[i+m];
 a[i+2*m] = b[i+2*m] + c[i+2*m];
 a[i+3*m] = b[i+3*m] + c[i+3*m];
}

```

読み取りおよび保管操作は数が増えると、コンパイラーによって決定された多数のストリーム間で分散され、計算時間が削減され、パフォーマンスが向上します。

## 関連資料

- 349 ページの『汎用プラグマ』
- 333 ページの『unroll』
- 354 ページの『#pragma block\_loop』
- 415 ページの『#pragma unroll』
- 417 ページの『#pragma unrollandfuse』

## #pragma strings

➤ C ➤ C++

### 説明

**#pragma strings** ディレクティブは、ストリング・リテラルのストレージ型を設定し、コンパイラーが読み取り専用メモリーにストリングを配置できるように指定します。それ以外の場合、コンパイラーは読み取り/書き込みメモリーにストリングを配置します。

### 構文

➤ #pragma strings ( writeable  
readonly ) ➤

### 注

ストリングは、デフォルトでは読み取り専用です。

このプラグマは、すべてのソース・ステートメントの前になければ、有効にはなりません。

### 例

```
#pragma strings(writeable)
```

### 関連資料

349 ページの『汎用プラグマ』

## #pragma unroll

▶ C ▶ C++

### 説明

**#pragma unroll** ディレクティブは、プログラム内の最内部または最外部のループをアンロールするために使用し、プログラムのパフォーマンスを向上させるのに役立つものです。

### 構文

```
▶ #pragma [nounroll | unroll (n)]
```

ここで、 $n$  は、ループのアンロール係数です。C プログラムでは、値  $n$  は正の整数定数式です。C++ プログラムでは、値  $n$  は正のスカラー整数またはコンパイル時定数の初期設定式です。アンロール係数 1 はアンロールを使用不可にします。 $n$  を指定しないとき、および最適化が **-O3** またはそれ以上に設定されているときは、最適化プログラムにより、各ループごとに該当するアンロール係数が決まります。

### 注

**#pragma unroll** または **#pragma nounroll** ディレクティブは対象の **for** ループの直前にあるか、対象の **for** ループに適用されるすべての **#pragma block\_loop** または **SMP** および **OMP** プラグマより前にある必要があります。

このディレクティブを複数回指定したり、ディレクティブを同じ **for** ループの **block\_loop**、**nounrollandfuse**、**unrollandfuse**、または **stream\_unroll** ディレクティブと組み合わせて使用しないでください。

ループ構造体は、以下の条件を満たしていなければなりません。

- ループ・カウンター変数は 1 つのみで、その変数に対する増分ポイントは 1 つ、終了変数は 1 つでなければなりません。これらは、ループ・ネストの任意のポイントで変更することはできません。
- ループは、複数のエントリーおよびエグジット・ポイントを持つことはできません。ループ終了は、ループを終了するための唯一の手段である必要があります。
- ループの依存関係は、「後方重視」であってはなりません。例えば、 $A[i][j] = A[i-1][j+1] + 4$  などのステートメントは、ループ内に表示されてはなりません。

ループに **#pragma nounroll** を指定すると、コンパイラーにそのループをアンロールしないように指示します。 **#pragma unroll(1)** を指定しても同じ結果になります。

**unroll** オプションによって、特定のアプリケーションのパフォーマンスが改善されるかどうかを確認するには、まず、通常オプションでプログラムをコンパイルしてから、それを代表的なワークロードで実行してください。次に、コマンド行 **-qunroll** オプションを指定するか、**unroll** プラグマを使用可能にして (あるいはその両方を行って)、プログラムを再コンパイルしてから、同じ条件下で再実行して、パフォーマンスが改善されたかどうか確認してください。

## 例

1. 以下の例では、ループ制御は変更されません。

```
#pragma unroll(2)
while (*s != 0)
{
 *p++ = *s++;
}
```

これを係数 2 でアンロールすると、以下が生成されます。

```
while (*s)
{
 *p++ = *s++;
 if (*s == 0) break;
 *p++ = *s++;
}
```

2. この例では、ループ制御が変更されます。

```
#pragma unroll(3)
for (i=0; i<n; i++) {
 a[i]=b[i] * c[i];
}
```

3 でアンロールすると、以下が生成されます。

```
i=0;
if (i>n-2) goto remainder;
for (; i<n-2; i+=3) {
 a[i]=b[i] * c[i];
 a[i+1]=b[i+1] * c[i+1];
 a[i+2]=b[i+2] * c[i+2];
}
if (i<n) {
 remainder:
 for (; i<n; i++) {
 a[i]=b[i] * c[i];
 }
}
```

## 関連資料

- 349 ページの『汎用プラグマ』
- 333 ページの『unroll』
- 354 ページの『#pragma block\_loop』
- 384 ページの『#pragma loopid』
- 412 ページの『#pragma stream\_unroll』
- 417 ページの『#pragma unrollandfuse』

## #pragma unrollandfuse

► C ► C++

### 説明

このプラグマは、ネストされた **for** ループ上で、アンロールおよびフューズ操作を試行するようにコンパイラーに指示します。

### 構文

```
► #pragma [nounrollandfuse | unrollandfuse(n)]
```

ここで、*n* はループのアンロール係数です。C プログラムでは、値 *n* は正の整数定数式です。C++ プログラムでは、値 *n* は正のスカラ整数またはコンパイル時定数の初期設定式です。アンロール係数 1 はアンロールを使用不可にします。*n* を指定しないとき、および最適化が **-O3** またはそれ以上に設定されているときは、最適化プログラムにより、ネストされた各ループごとに該当するアンロール係数が決まります。

### 注

**#pragma unrollandfuse** ディレクティブは、以下の条件を満たす、ネストされた **for** ループ構造体の外部ループにのみ適用されます。

- ループ・カウンター変数は 1 つのみで、その変数に対する増分ポイントは 1 つ、終了変数は 1 つでなければなりません。これらは、ループ・ネストの任意のポイントで変更することはできません。
- ループは、複数のエントリーおよびエグジット・ポイントを持つことはできません。ループ終了は、ループを終了するための唯一の手段である必要があります。
- ループの依存関係は、「後方重視」であってはなりません。例えば、 $A[i][j] = A[i-1][j+1] + 4$  などのステートメントは、ループ内に表示されてはなりません。

ループのアンロールを行うには、**#pragma unrollandfuse** ディレクティブが対象の **for** ループの直前にあるか、対象の **for** ループに適用されるすべての **#pragma block\_loop** または **SMP** および **OMP** プラグマより前にある必要があります。**#pragma unrollandfuse** は、最内部の **for** ループに対して指定してはなりません。

このディレクティブを複数回指定したり、ディレクティブを同じ **for** ループの **block\_loop**、**nounrollandfuse**、**nounroll**、**unroll**、または **stream\_unroll** ディレクティブと組み合わせて使用しないでください。

**#pragma nounrollandfuse** を指定すると、コンパイラーにそのループをアンロールしないように指示します。

### 例

- 以下の例では、**#pragma unrollandfuse** ディレクティブがループの本体を複製し、フューズします。これにより、配列 *b* のキャッシュ・ミス数が削減されます。

```

int i, j;
int a[1000][1000];
int b[1000][1000];
int c[1000][1000];

....

#pragma unrollandfuse(2)
for (i=1; i<1000; i++) {
 for (j=1; j<1000; j++) {
 a[j][i] = b[i][j] * c[j][i];
 }
}

```

次の **for** ループは、**#pragma unrollandfuse(2)** ディレクティブを上記のループ構造体に適用した場合に考えられる結果を示します。

```

for (i=1; i<1000; i=i+2) {
 for (j=1; j<1000; j++) {
 a[j][i] = b[i][j] * c[j][i];
 a[j][i+1] = b[i+1][j] * c[j][i+1];
 }
}

```

2. ネストされたループ構造体で、複数の **#pragma unrollandfuse** ディレクティブを指定することもできます。

```

int i, j, k;
int a[1000][1000];
int b[1000][1000];
int c[1000][1000];
int d[1000][1000];
int e[1000][1000];

....

#pragma unrollandfuse(4)
for (i=1; i<1000; i++) {
 #pragma unrollandfuse(2)
 for (j=1; j<1000; j++) {
 for (k=1; k<1000; k++) {
 a[j][i] = b[i][j] * c[j][i] + d[j][k] * e[i][k];
 }
 }
}

```

## 関連資料

- 349 ページの『汎用プラグマ』
- 333 ページの『unroll』
- 354 ページの『#pragma block\_loop』
- 384 ページの『#pragma loopid』
- 412 ページの『#pragma stream\_unroll』
- 415 ページの『#pragma unroll』

## #pragma weak

▶ C ▶ C++

### 説明

**#pragma weak** ディレクティブを実行すると、シンボルの定義が検索されないときや、リンク中に複数のシンボルが定義されていることが発見されたときに、リンケージ・エディターはエラー・メッセージを発行できません。

### 構文

▶ `#pragma weak identifier` = identifier2

### 注

このプラグマは関数とともに使用することを基本条件として設計されていますが、ほとんどのデータ・オブジェクトに対しても有効に機能します。

このプラグマは、未初期化状態のグローバル・データと共に、あるいは実行可能ファイルにエクスポートされる共用ライブラリー・データ・オブジェクトと共に使用しないようにしてください。

プログラム・ソースには、2 つの形式の **#pragma weak** を指定できます。

#### **#pragma weak ID**

この形式のプラグマは、*identifier* を弱いグローバル・シンボルとして定義します。

*Identifier* が **#pragma weak identifier** と同じコンパイル単位で定義されている場合、*identifier* は weak 定義として処理されます。*identifier* を使用も宣言もしないコンパイル単位に **#pragma weak** が存在する場合、プラグマは受け入れられても無視されます。

*identifier* が C++ リンケージを持つ関数を指示するときは、*identifier* はその関数の C++ マングル名を使用して指定しなければなりません。また、C++ 関数がテンプレート関数である場合、このテンプレート関数を明示的にインスタンス化する必要があります。

#### **#pragma weak identifier=identifier2**

この形式のプラグマは、*identifier* を弱いグローバル・シンボルとして定義します。*identifier* を参照するときは、*identifier2* の値を使用します。

*identifier2* はメンバー関数にできません。

*identifier* は **#pragma weak** と同じコンパイル単位に宣言できるとできないことがあります、このコンパイル単位には定義しないでください。

*identifier* がこのコンパイル単位に宣言されると、*identifier* の宣言は *identifier2* の宣言と互換性がなければなりません。例えば *identifier2* が関数のときは、*identifier* は *identifier2* と同じ戻りまたは引き数の型がなければなりません。

*identifier2* は **#pragma weak** と同じコンパイル単位で宣言されていなければなりません。

*identifier2* が C++ リンケージを持つ関数を指示するときは、*identifier* と *identifier2* はその関数のマングル名を使用して指定しなければなりません。C++ 関数がテンプレート関数である場合、このテンプレート関数を明示的にインスタンス化する必要があります。

以下の場合、コンパイラーは **#pragma weak** を無視して警告メッセージを出します。

- 指定された *identifier2* がコンパイル単位に定義されていない。
- 指定された *identifier2* がメンバー関数である。
- *identifier* は宣言されているが、その型が指定された *identifier2* の型と互換性がない。

コンパイラーは **#pragma weak** を無視し、弱い *identifier* が定義されているときには重大エラー・メッセージを出します。

## 例

1. 以下はプラグマの **#pragma weak identifier** 形式の例です。

```
// Begin Compilation Unit 1
#include <stdio.h>
extern int foo;
#pragma weak foo

int main()
{
 int *ptr;
 ptr = &foo;
 if (ptr == 0)
 printf("foo was not defined\n");
 else
 printf("foo was already defined\n");
}
//End Compilation Unit 1

// Begin Compilation Unit 2
int foo = 1;
// End Compilation Unit 2
```

コンパイル単位 1 が、実行可能ファイルを作成するために単独でコンパイルされる場合、リンカーは、ID `foo` が定義されていないことを示すエラー・メッセージを発行します。コンパイル単位 1 およびコンパイル単位 2 は、実行可能ファイルを作成するために結合されなければなりません。

2. 以下はプラグマの **#pragma weak identifier=identifier2** の形式の例です。

```
//Begin Compilation Unit
extern "C" void printf(char *,...);

void fool(void)
{
 printf("Just in function fool()\n");
}

#pragma weak foo__Fv = fool__Fv
```



```
int main()
{
 foo();
}
//End Compilation Unit
```

## 関連資料

349 ページの『汎用プラグマ』

344 ページの『weaksymbol』

## 並列処理を制御するプラグマ

このページの `#pragma` ディレクティブによって、ユーザー・プログラムでの並列処理をコンパイラーが処理する方法を制御することができます。これらのプラグマは、IBM 固有のディレクティブ、および OpenMP アプリケーション・プログラム・インターフェース仕様に準拠したディレクティブの 2 つのグループに分けられます。

**-qsmp** コンパイラー・オプションを使用して、プログラム内での並列処理の方法を指定します。**-qignprag=ibm:omp** コンパイラー・オプションを指定して、並列処理関連のすべての `#pragma` ディレクティブを無視するよう、コンパイラーに指示することもできます。

ディレクティブは、このディレクティブの直後に続くステートメントまたはステートメント・ブロックに対してのみ適用されます。

| IBM プラグマ・ディレクティブ<br>▶ C                    | 説明                                                                |
|--------------------------------------------|-------------------------------------------------------------------|
| <code>#pragma ibm critical</code>          | このプラグマの直後に続くステートメントまたはステートメント・ブロックがクリティカル・セクションであることをコンパイラーに指示する。 |
| <code>#pragma ibm independent_calls</code> | 選択したループ内に指定された関数呼び出しが、ループの実行に依存しないことを明示する。                        |
| <code>#pragma ibm independent_loop</code>  | 選択したループの反復が独立しているため、そのループが並列化できることを明示する。                          |
| <code>#pragma ibm iterations</code>        | 選択したループについて、おおよそのループの反復回数を指定する。                                   |
| <code>#pragma ibm parallel_loop</code>     | 選択したループを並列化するようにコンパイラーに明示的に指示する。                                  |
| <code>#pragma ibm permutation</code>       | 選択したループ内の指定した配列に、繰り返される値が入っていないことを明示する。                           |
| <code>#pragma ibm schedule</code>          | 並列にループを実行するためのスケジューリング・アルゴリズムを指定する。                               |
| <code>#pragma ibm sequential_loop</code>   | 選択したループを順次実行するようにコンパイラーに明示的に指示する。                                 |

| OpenMP プラグマ・ディレクティブ<br>▶ C ▶ C++  | 説明                                                                                              |
|-----------------------------------|-------------------------------------------------------------------------------------------------|
| <code>#pragma omp atomic</code>   | アトミックに更新されなければならない、しかも、複数の同時書き込みスレッドに公開してはならない特定のメモリー位置を識別する。                                   |
| <code>#pragma omp parallel</code> | 複数のスレッドによって並列で実行されるように並列領域を定義する。特定の例外はありますが、他のすべての OpenMP ディレクティブは、このディレクティブで定義された並列領域内で実行されます。 |
| <code>#pragma omp for</code>      | 反復が並列で実行される反復 <code>for</code> ループを識別する作業共有構成。                                                  |

| OpenMP プラグマ・ディレクティブ<br>C C++             | 説明                                                                                                               |
|------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| #pragma omp parallel for                 | <b>omp parallel</b> と <b>omp for</b> プラグマ・ディレクティブのショートカット組み合わせで単一の <b>for</b> ディレクティブを含む並列領域の定義に使用される。           |
| #pragma omp ordered                      | 順番に実行しなければならないコードの構造化ブロックを識別する作業共有構成。                                                                            |
| #pragma omp section、#pragma omp sections | 並列で実行する必要があるコードの 1 つ以上のサブセクションを含む、コードの非反復セクションを識別する作業共有構成。                                                       |
| #pragma omp parallel sections            | <b>omp parallel</b> と <b>omp sections</b> プラグマ・ディレクティブのショートカット組み合わせで単一の <b>sections</b> ディレクティブを含む並列領域の定義に使用される。 |
| #pragma omp single                       | 単一の使用可能スレッドによって実行しなければならないコードのセクションを識別する作業共有構成。                                                                  |
| #pragma omp master                       | マスター・スレッドでしか実行してはならないコードのセクションを識別する同期構成。                                                                         |
| #pragma omp critical                     | 単一スレッドによって一度に実行しなければならないステートメント・ブロックを識別する同期構成。                                                                   |
| #pragma omp barrier                      | 並列領域内のすべてのスレッドを同期化する。                                                                                            |
| #pragma omp flush                        | 並列領域内のすべてのスレッドがメモリー内の同じビューの指定したオブジェクトを持っていることをコンパイラーが保証するポイントを識別する同期構成。                                          |
| #pragma omp threadprivate                | 選択したファイル・スコープ・データ変数のスコープがスレッドに対して <b>private</b> であるが、そのスレッド内でファイル・スコープが可視であると定義する。                              |

## 関連概念

15 ページの『プログラムの並列化』

## 関連タスク

28 ページの『並列処理ランタイム・オプションの設定』

47 ページの『プラグマを使用した並列処理の制御』

## 関連資料

293 ページの『smp』

461 ページの『IBM SMP 並列処理のためのランタイム・オプション』

465 ページの『並列処理のための OpenMP ランタイム・オプション』

467 ページの『並列処理に使用する組み込み関数』

OpenMP 仕様の完全な情報については、以下を参照してください。

- OpenMP Web サイト
- OpenMP 仕様

## #pragma ibm critical



### 説明

**critical** プラグマは、一度に 1 つのプロセスでしか実行してはならない、プログラム・コードのクリティカル・セクションを識別します。

### 構文

```
#pragma ibm critical [(name)]
<statement>
```

ここで、*name* を使用すると、オプションでクリティカル領域を識別することができます。クリティカル領域を指定している ID には、外部結合があります。

### 注

クリティカル・セクションに出入りする分岐を行おうとすると、コンパイラーによってエラーが報告されます。エラーの原因となる状態を、以下にいくつか示します。

- クリティカル・セクションに **return** ステートメントが含まれている。
- クリティカル・セクションに、クリティカル・セクションの外側にプログラム・フローを受け渡す、**goto**、**continue**、または **break** ステートメントが含まれている。
- クリティカル・セクション内に定義されているラベルにプログラム・フローを受け渡す **goto** 文がクリティカル・セクションの外側にある。

### 関連資料

422 ページの『並列処理を制御するプラグマ』

## #pragma ibm independent\_calls



### 説明

**independent\_calls** プラグマは、選択したループ内の指定した関数呼び出しが、ループの実行に依存しないことを明示します。この情報は、コンパイラーによる依存性の分析に役立ちます。

### 構文

```
#pragma ibm independent_calls [(identifier [,identifier] ...)]
<countable for/while/do loop>
```

ここで、*identifier* は関数名を表します。

### 注

*identifier* を関数へのポインターの名前にすることはできません。

関数 ID が指定されていない場合、コンパイラーは、ループ内のすべての関数について実行に対する依存性がないと見なします。

### 関連資料

422 ページの『並列処理を制御するプラグマ』

## #pragma ibm independent\_loop



### 説明

**independent\_loop** プラグマは、選択したループの反復が独立していて、かつそのループが並列化できることを明示します。

### 構文

```
#pragma ibm independent_loop [if (exp)]
<countable for/while/do loop>
```

ここで、*exp* はスカラー式を表します。

### 注

**if** 引き数を指定したときは、*exp* が実行時に TRUE と評価される場合に限り、ループの反復が独立していると見なされます。

このプラグマを **schedule** プラグマと組み合わせて、特定の並列処理スケジューリング・アルゴリズムを選択することができます。詳しくは、**schedule** プラグマについての説明を参照してください。

### 関連資料

422 ページの『並列処理を制御するプラグマ』

430 ページの『#pragma ibm schedule』

## #pragma ibm iterations



### 説明

**iterations** プラグマは、選択したループについて、おおよそのループの反復回数を指定します。

### 構文

```
#pragma ibm iterations (iteration-count)
<countable for/while/do loop>
```

ここで、*iteration-count* は正の整数定数式を表します。

### 注

コンパイラーは、*iteration-count* 変数の情報を使用して、ループの並列化が効率的であるかどうかを判別します。

### 関連資料

422 ページの『並列処理を制御するプラグマ』

## #pragma ibm parallel\_loop



### 説明

**parallel\_loop** プラグマは、選択したループを並列化するようにコンパイラーに明示的に指示します。

### 構文

```
#pragma ibm parallel_loop [if (exp)] [schedule (sched-type)]
<countable for/while/do loop>
```

ここで、*exp* はスカラー式を、*sched-type* は *schedule* ディレクティブに有効な任意のスケジューリング・アルゴリズムを表します。

### 注

**if** 引き数を指定したときは、*exp* が実行時に TRUE と評価された場合に限り、ループは並列に実行されます。それ以外の場合は、ループは順次実行されます。ループは、クリティカル・セクションにある場合にも順次実行されます。

このプラグマは、多種多様な C ループに適用することができ、コンパイラーは、ループがカウント可能であるかどうかを判別しようとします。

**parallel\_loop** プラグマを使用するプログラム・セクションでは、順次モードでも並列モードでも正しい結果が生成できなければなりません。例えば、ループの反復が独立していなければ、ループを並列化することはできません。条件付き同期化に関連する明示的な並列プログラミング技法は許されていません。

コンパイラーは、このプラグマでマークを付けられたループに対する縮小を自動的には検出しません。縮小を伴うループを正しく並列化するには、以下を行います。

- **#pragma omp parallel for** を使用して縮小を明示的に指定する。
- **#pragma ibm independent\_loop** を使用して、コンパイラーに縮小を発見させる。

このプラグマを **schedule** プラグマと組み合わせて、特定の並列処理スケジューリング・アルゴリズムを選択することができます。詳しくは、**schedule** プラグマについての説明を参照してください。

このプラグマの後にカウント可能なループを指定していない場合には、警告が生成されます。

### 関連資料

422 ページの『並列処理を制御するプラグマ』

430 ページの『#pragma ibm schedule』



## #pragma ibm permutation



### 説明

**permutation** プラグマは、選択したループ内の指定した配列に、繰り返される値が入っていないことを明示します。

### 構文

```
#pragma ibm permutation (identifier [,identifier] ...)
<countable for/while/do loop>
```

ここで、*identifier* は配列の名前を表します。

### 注

*identifier* は、関数仮パラメーターまたはポインターの名前にすることはできません。

### 関連資料

422 ページの『並列処理を制御するプラグマ』

## #pragma ibm schedule



### 説明

**schedule** プラグマは、並列処理に使用するスケジューリング・アルゴリズムを指定します。

### 構文

```
#pragma ibm schedule (sched-type)
<countable for/while/do loop>
```

ここで、*sched-type* は以下のオプションのいずれかを表します。

|                    |                                                                                                                                                                                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| affinity           | ループの反復が、最初に、サイズが <b>ceiling</b> ( <i>number_of_iterations/number_of_threads</i> ) のローカル区画に分割されます。その後、各ローカル区画は、サイズが <b>ceiling</b> ( <i>number_of_iterations_remaining_in_partition/2</i> ) のチャンクにさらに細分化されます。      |
| affinity, <i>n</i> | スレッドは、使用可能になると、そのローカル区画から次のチャンクを受け入れます。ローカル区画にチャンクがなくなった場合は、スレッドは、別のスレッドの区画から使用可能なチャンクを受け入れます。ローカル区画がそれぞれサイズ <i>n</i> のチャンクに再分割されることを除いて、上記と同様です。 <i>n</i> は 1 以上の値の整数代入式でなければなりません。                              |
| dynamic            | ループの反復が、サイズが 1 のチャンクに分割されます。<br><br>チャンクは、スレッドが使用可能になると、最初に提供されたものから順に処理されるようにスレッドに割り当てられます。これは、すべての作業が完了するまで続けられます。                                                                                              |
| dynamic, <i>n</i>  | すべてのチャンクのサイズが <i>n</i> に設定されることを除いて、上記と同様です。 <i>n</i> は、1 以上の値の整数代入式でなければなりません。                                                                                                                                   |
| guided             | チャンクは、チャンク・サイズが 1 に達するまで順次小さくされます。最初のチャンクのサイズは <b>ceiling</b> ( <i>number_of_iterations/number_of_threads</i> ) です。それ以外のチャンクのサイズは、 <b>ceiling</b> ( <i>number_of_iterations_remaining/number_of_threads</i> ) です。 |
| guided, <i>n</i>   | チャンクは、スレッドが使用可能になると、最初に提供されたものから順に処理されるようにスレッドに割り当てられます。これは、すべての作業が完了するまで続けられます。最小チャンク・サイズが <i>n</i> に設定されることを除いて、上記と同様です。 <i>n</i> は、1 以上の値の整数代入式でなければなりません。                                                     |
| runtime            | 実行時にスケジューリングの方針が判別されます。                                                                                                                                                                                           |
| static             | ループの反復が、サイズが <b>ceiling</b> ( <i>number_of_iterations/number_of_threads</i> ) のチャンクに分割されます。スレッドにはそれぞれ別個のチャンクが割り当てられます。<br><br>このスケジューリング方針は、ブロック・スケジューリングとしても知られています。                                             |

`static,n` ループの反復が、サイズが  $n$  のチャンクに分割されます。チャンクはそれぞれラウンドロビン 方式でスレッドに割り当てられます。

$n$  は、1 以上の値の整数代入式でなければなりません。

このスケジューリング方針は、ブロック巡回スケジューリング としても知られています。

`static,1` ループの反復が、サイズが 1 のチャンクに分割されます。チャンクはそれぞれラウンドロビン 方式でスレッドに割り当てられます。

このスケジューリング方針は、巡回スケジューリング としても知られています。

## 注

並列処理のためのスケジューリング・アルゴリズムは、以下に示す任意の方法を使用して指定することができます。リストの上位の方式を使用すると、その方式によってリストの下位の項目がオーバーライドされます。

- プラグマ・ステートメント
- コンパイラーのコマンド行オプション
- 実行時のコマンド行オプション
- 実行時のデフォルトのオプション

スケジューリング・アルゴリズムは、**parallel\_loop** および **independent\_loop** プラグマ・ステートメントの **schedule** 引き数を使用して指定することもできます。例えば、以下のステートメントの設定は同等です。

```
#pragma ibm parallel_loop
#pragma ibm schedule (sched_type)
<countable for|while|do loop>
および
#pragma ibm parallel_loop (sched_type)
<countable for|while|do loop>
```

あるループに異なるスケジューリング型を指定した場合は、最後に指定したものが適用されます。

## 関連資料

422 ページの『並列処理を制御するプラグマ』

426 ページの『`#pragma ibm independent_loop`』

428 ページの『`#pragma ibm parallel_loop`』

## #pragma ibm sequential\_loop



### 説明

**sequential\_loop** プラグマは、選択したループを順次実行するようにコンパイラーに明示的に指示します。

### 構文

```
#pragma ibm sequential_loop
<countable for/while/do loop>
```

### 注

このプラグマは、選択したループの自動的な並列化を使用不可にし、コンパイラーは常にこのプラグマに従います。

### 関連資料

422 ページの『並列処理を制御するプラグマ』

## #pragma omp atomic

➤ C ➤ C++

### 説明

**omp atomic** ディレクティブは、アトミックに更新しなければならない、また複数の同時書き込みスレッドに公開してはならない、特定のメモリー・ロケーションを識別します。

### 構文

```
#pragma omp atomic
<statement_block>
```

ここで、*statement* は、以下に続く形式のいずれかを採用するスカラー型の式ステートメントです。

| <i>statement</i>                  | 条件                                                                                                                                                                 |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $x \text{ bin\_op} = \text{expr}$ | ここで、<br><br><i>bin_op</i> は、以下のいずれかです。<br>$+ \quad * \quad - \quad / \quad \& \quad ^ \quad   \quad \ll \quad \gg$<br><br><i>expr</i> は、 <i>x</i> を参照しないスカラー型の式です。 |
| $x++$                             |                                                                                                                                                                    |
| $++x$                             |                                                                                                                                                                    |
| $x--$                             |                                                                                                                                                                    |
| $--x$                             |                                                                                                                                                                    |
|                                   |                                                                                                                                                                    |

### 注

ロードおよび保管の操作は、オブジェクト *x* に対してのみ **atomic** です。 *expr* の評価は、**atomic** ではありません。

プログラム内の指定オブジェクトに対するすべてのアトミック参照は、互換タイプを持っていない限りなりません。

並列で更新できる、また、競合状態の対象となり得るオブジェクトは、**omp atomic** ディレクティブで保護されているはずです。

### 例

```
extern float x[], *p = x, y;

/* Protect against race conditions among multiple updates. */
#pragma omp atomic
x[index[i]] += y;

/* Protect against races with updates through x. */
#pragma omp atomic
p[i] -= 1.0f;
```

### 関連資料

422 ページの『並列処理を制御するプラグマ』

## #pragma omp parallel

### 説明

**omp parallel** ディレクティブは、選択したコードのセグメントを並列化するようにコンパイラーに明示的に指示します。

### 構文

```
#pragma omp parallel [clause[[, clause] ...]
<statement_block>
```

ここで、*clause* は、次のいずれかです。

|                                                |                                                                                                                                                                                    |
|------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>if</b> ( <i>exp</i> )                       | <b>if</b> 引き数を指定したときは、 <i>exp</i> が表したスカラー式が実行時に非ゼロ値と評価された場合に限り、プログラム・コードが並列に実行されます。 <b>if</b> 文節は 1 つのみ指定することができます。                                                               |
| <b>private</b> ( <i>list</i> )                 | <i>list</i> 内のデータ変数のスコープが各スレッドに対して <b>private</b> であることを宣言します。 <i>list</i> 内のデータ変数は、コンマで区切られています。                                                                                  |
| <b>firstprivate</b> ( <i>list</i> )            | <i>list</i> 内のデータ変数のスコープが各スレッドに対して <b>private</b> であることを宣言します。それぞれの新規の <b>private</b> オブジェクトは、あたかもステートメント・ブロック内に暗黙の宣言があるように、元の変数の値を使用して初期化されます。 <i>list</i> 内のデータ変数は、コンマで区切られています。 |
| <b>num_threads</b><br>( <i>int_exp</i> )       | <i>int_exp</i> の値は、並列領域用に使用するスレッドの数を指定する整数式です。スレッドの数の動的調整も使用可能である場合、 <i>int_exp</i> は使用されるスレッドの最大数を指定します。                                                                          |
| <b>shared</b> ( <i>list</i> )                  | <i>list</i> 内のデータ変数のスコープがすべてのスレッドに渡って共用されることを宣言します。                                                                                                                                |
| <b>default</b> ( <b>shared</b>   <b>none</b> ) | 各スレッド内の変数のデフォルトのデータ・スコープを定義します。 <b>default</b> 文節は、1 つの <b>omp parallel</b> ディレクティブ上に 1 つのみ指定することができます。                                                                            |

**default(shared)** の指定は、**shared(list)** 文節内の各変数を指定するのと同じです。

**default(none)** の指定には、並列化されたステートメント・ブロックに対して可視である各データ変数が、データ・スコープ文節に明示的にリストされている必要があります。ただし、次のような変数の例外があります。

- **const** によって限定されている
- 囲まれたデータ・スコープ属性の文節内に指定されている
- 対応する **omp for** または **omp parallel for** ディレクティブによってのみ参照されるループ制御変数として使用されている

|                               |                                                                                                                                 |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>copyin</b> ( <i>list</i> ) | <i>list</i> 内に指定されているデータ変数ごとに、マスター・スレッド内のデータ変数の値は、並列領域の開始地点のスレッド <b>private</b> コピーにコピーされます。 <i>list</i> 内のデータ変数は、コンマで区切られています。 |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------|

**copyin** 文節内で指定するデータ変数は、それぞれ、**threadprivate** 変数でなければなりません。

**reduction** (operator: list) 指定された *operator* を使用して、*list* 内のすべてのスカラー変数の縮約を実行します。*list* 内の縮約変数は、コンマで区切られています。

*list* 内の各変数の **private** コピーは、スレッドごとに作成されます。ステートメント・ブロックの最後で、縮約変数のすべての **private** コピーの最終値は、その演算子に適切な方法で結合され、その結果は、共用の縮約変数の元の値に戻されます。

**reduction** 文節で指定する変数は、以下のとおりでなければなりません。

- 演算子に適切な型でなければならない。
- 囲んでいるコンテキスト内で共用されていないなければならない。
- **const** によって修飾された変数であってはならない。
- ポインター型があってはならない。

## 注

並列領域が検出されると、スレッドの論理チームが形成されます。チーム内の各スレッドは、作業共有構成を除いて、並列領域内のすべてのステートメントを実行します。作業共有構成内の作業は、チーム内のスレッド間で配布されます。

ループの反復が独立していなければ、ループを並列化することはできません。暗黙のバリアが、並列化されたステートメント・ブロックの終了地点にあります。

ネストされた並列領域は、常に直列化されています。

## 関連資料

422 ページの『並列処理を制御するプラグマ』

436 ページの『**#pragma omp for**』

441 ページの『**#pragma omp parallel for**』

444 ページの『**#pragma omp parallel sections**』

## #pragma omp for

### 説明

**omp for** ディレクティブは、この作業共有構成を検出するスレッドのチーム内でループ反復を配布するようコンパイラーに指示します。

### 構文

```
#pragma omp for [clause[:,] clause] ...]
<for_loop>
```

ここで、*clause* は、次のいずれかです。

|                                           |                                                                                                                                                                                          |
|-------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>private</b> ( <i>list</i> )            | <i>list</i> 内のデータ変数のスコープが各スレッドに対して <b>private</b> であることを宣言します。 <i>list</i> 内のデータ変数は、コンマで区切られています。                                                                                        |
| <b>firstprivate</b> ( <i>list</i> )       | <i>list</i> 内のデータ変数のスコープが各スレッドに対して <b>private</b> であることを宣言します。それぞれの新規の <b>private</b> オブジェクトは、ステートメント・ブロック内に暗黙の宣言がある場合のように初期化されます。 <i>list</i> 内のデータ変数は、コンマで区切られています。                    |
| <b>lastprivate</b> ( <i>list</i> )        | <i>list</i> 内のデータ変数のスコープが各スレッドに対して <b>private</b> であることを宣言します。 <i>list</i> 内の各変数の最終値は、割り当てられる場合、最後の反復でその変数に割り当てられる値となります。値が割り当てられていない変数は、不確定値を持っています。 <i>list</i> 内のデータ変数は、コンマで区切られています。 |
| <b>reduction</b> ( <i>operator:list</i> ) | 指定された <i>operator</i> を使用して、 <i>list</i> 内のすべてのスカラー変数の縮約を実行します。 <i>list</i> 内の縮約変数は、コンマで区切られています。                                                                                        |

*list* 内の各変数の **private** コピーは、スレッドごとに作成されます。ステートメント・ブロックの最後で、縮約変数のすべての **private** コピーの最終値は、その演算子に適切な方法で結合され、その結果は、共用の縮約変数の元の値に戻されます。

**reduction** 文節で指定する変数は、以下のとおりでなければなりません。

- 演算子に適切な型でなければならない。
- 囲んでいるコンテキスト内で共用されていなければならない。
- **const** によって修飾された変数であってはならない。
- ポインター型があってはならない。

**ordered** *ordered* 構成が **omp for** ディレクティブの動的範囲内に存在する場合、この文節を指定します。



schedule  
(type)

**for** ループの反復を使用可能なスレッド間で分割する方法を指定します。  
*type* の許容値は、以下のとおりです。

**dynamic**

ループの反復が、サイズが **ceiling**

( $\text{number\_of\_iterations}/\text{number\_of\_threads}$ ) のチャンクに分割されます。

チャンクは、スレッドが使用可能になると、最初に提供されたものから順に処理されるようにスレッドに動的に割り当てられます。これは、すべての作業が完了するまで続けられます。

**dynamic,*n***

チャンクのサイズが *n* に設定されることを除いて、上記と同様です。*n* は、1 以上の値の整数代入式でなければなりません。

**guided** チャンクは、デフォルトの最小チャンク・サイズに達するまで順次小さくされます。最初のチャンクのサイズは **ceiling**

( $\text{number\_of\_iterations}/\text{number\_of\_threads}$ ) です。それ以外のチャンクのサイズは、**ceiling**

( $\text{number\_of\_iterations\_left}/\text{number\_of\_threads}$ ) です。

チャンクの最小サイズは 1 です。

チャンクは、スレッドが使用可能になると、最初に提供されたものから順に処理されるようにスレッドに割り当てられます。これは、すべての作業が完了するまで続けられます。

**guided,*n***

最小チャンク・サイズが *n* に設定されることを除いて、上記と同様です。*n* は、1 以上の値の整数代入式でなければなりません。

**runtime**

実行時にスケジューリングの方針が判別されます。

OMP\_SCHEDULE 環境変数を使用して、スケジューリング型およびチャンク・サイズを設定します。

**static** ループの反復が、サイズが **ceiling**

( $\text{number\_of\_iterations}/\text{number\_of\_threads}$ ) のチャンクに分割されます。スレッドにはそれぞれ別個のチャンクが割り当てられます。

このスケジューリング方針は、ブロック・スケジューリングとしても知られています。

**static,*n*** ループの反復が、サイズが *n* のチャンクに分割されます。チャンクはそれぞれラウンドロビン 方式でスレッドに割り当てられます。

*n* は、1 以上の値の整数代入式でなければなりません。

このスケジューリング方針は、ブロック巡回スケジューリングとしても知られています。

**static,1**

ループの反復が、サイズが 1 のチャンクに分割されます。チャンクは、それぞれラウンドロビン 方式でスレッドに割り当てられます。

このスケジューリング方針は、巡回スケジューリングとしても知られています。

**nowait** この文節は、**for** ディレクティブ終了時の暗黙の **barrier** を回避するために使用します。これは、指定した並列領域内に複数の独立した作業共有セクションまたは反復ループがある場合に有効です。 **nowait** 文節が、所定の **for** ディレクティブ上に現れるのは、1 回のみです。

また、*for\_loop* の個所は、以下の規範的形状を持つ **for** ループ構成体です。

```
for (init_expr; exit_cond; incr_expr)
 statement
```

ここで、

|                  |              |                                                                                                                                                                                                                                           |
|------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>init_expr</i> | は、次の形式を取ります。 | <i>iv</i> = <i>b</i><br><i>integer-type iv</i> = <i>b</i>                                                                                                                                                                                 |
| <i>exit_cond</i> | は、次の形式を取ります。 | <i>iv</i> <= <i>ub</i><br><i>iv</i> < <i>ub</i><br><i>iv</i> >= <i>ub</i><br><i>iv</i> > <i>ub</i>                                                                                                                                        |
| <i>incr_expr</i> | は、次の形式を取ります。 | ++ <i>iv</i><br><i>iv</i> ++<br>-- <i>iv</i><br><i>iv</i> --<br><i>iv</i> += <i>incr</i><br><i>iv</i> -= <i>incr</i><br><i>iv</i> = <i>iv</i> + <i>incr</i><br><i>iv</i> = <i>incr</i> + <i>iv</i><br><i>iv</i> = <i>iv</i> - <i>incr</i> |

また、ここでは以下のとおりです。

|                                    |                                                                                                                                                                                  |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>iv</i>                          | 反復変数。反復変数は、 <b>for</b> ループ内のいずれの個所でも変更されていない符号付き整数でなければなりません。反復変数は、 <b>for</b> 演算の間は、暗黙的に <b>private</b> にされます。<br><b>lastprivate</b> として指定されていない場合、反復変数は、演算の完了後に不確定値を持つことになります。 |
| <i>b</i> , <i>ub</i> , <i>incr</i> | ループ・インバリアント符号付き整数式。これらの式の評価中は、同期は実行されません。また、評価済みの副次作用は、結果として、不確定値になる場合があります。                                                                                                     |

## 注

**omp for** プラグマを使用するプログラム・セクションでは、いずれのスレッドが特定の反復を実行するかにかかわらず、正しい結果を生成できなければなりません。同様に、プログラムの正確さは、特定のスケジューリング・アルゴリズムの使用に依存するものであってはなりません。

**for** ループの反復変数は、ループの実行の間、スコープ内で暗黙的に **private** にされます。この変数は、**for** ループの本体内で変更してはなりません。増分変数の値は、その変数がデータ・スコープの **lastprivate** を持つよう指定されていない限り、不確定です。

**nowait** 文節が指定されていない限り、**for** ループの終了時に暗黙の **barrier** が存在します。

制限は、以下のとおりです。

- **for** ループは、構造化ブロックでなければなりません。また、**break** ステートメントで終了してはなりません。
- ループ制御式の値は、ループのすべての反復について同一でなければなりません。
- **omp for** ディレクティブが受け入れることができる **schedule** 文節は、1 つのみです。
- $n$  の値 (チャンク・サイズ) は、並列領域のすべてのスレッドについて同一でなければなりません。

## 関連資料

422 ページの『並列処理を制御するプリAGMA』

441 ページの『`#pragma omp parallel for`』

## #pragma omp ordered

### 説明

**omp ordered** ディレクティブは、順次配列で実行されなければならないコードの構造化ブロックを識別します。

### 構文

```
#pragma omp ordered
 statement_block
```

### 注

**omp ordered** ディレクティブは、以下のように使用しなければなりません。

- **ordered** 文節を含んでいる **omp for** または **omp parallel for** 構成の範囲内に表示されなければなりません。
- すぐ後に続くステートメント・ブロックに適用します。そのブロックのステートメントは、反復が順次ループ内で実行されるのと同じ順序で実行されます。
- ループの反復は、同一の **omp ordered** ディレクティブを 1 回以上実行してはなりません。
- ループの反復では、複数の特殊 **omp ordered** ディレクティブを実行してはなりません。

### 関連資料

422 ページの『並列処理を制御するプラグマ』

436 ページの『#pragma omp for』

441 ページの『#pragma omp parallel for』

## #pragma omp parallel for

### 説明

**omp parallel for** ディレクティブは、**omp parallel** ディレクティブと **omp for** ディレクティブを効果的に結合します。このディレクティブを使用すると、単一の **for** ディレクティブを含んでいる並列領域をワンステップで定義することができます。

### 構文

```
#pragma omp parallel for [clause[[, clause] ...]
<for_loop>
```

### 注

**nowait** 文節を除き、**omp parallel** および **omp for** ディレクティブに記載されているすべての文節および制限は、**omp parallel for** ディレクティブに適用されます。

### 関連資料

422 ページの『並列処理を制御するプラグマ』

436 ページの『#pragma omp for』

434 ページの『#pragma omp parallel』

## #pragma omp section、#pragma omp sections

### 説明

**omp sections** ディレクティブは、定義済みの並列領域にバインドされたスレッド間で作業を配布します。

### 構文

```
#pragma omp sections [clause [clause] ...]
{
 [#pragma omp section]
 statement-block
 [#pragma omp section]
 statement-block
 .
 .
 .
}
```

ここで、*clause* は、次のいずれかです。

|                                                       |                                                                                                                                                                                   |
|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>private (<i>list</i>)</code>                    | <i>list</i> 内のデータ変数のスコープが各スレッドに対して <code>private</code> であることを宣言します。 <i>list</i> 内のデータ変数は、コンマで区切られています。                                                                           |
| <code>firstprivate (<i>list</i>)</code>               | <i>list</i> 内のデータ変数のスコープが各スレッドに対して <code>private</code> であることを宣言します。それぞれの新規の <code>private</code> オブジェクトは、ステートメント・ブロック内に暗黙の宣言がある場合のように初期化されます。 <i>list</i> 内のデータ変数は、コンマで区切られています。 |
| <code>lastprivate (<i>list</i>)</code>                | <i>list</i> 内のデータ変数のスコープが各スレッドに対して <code>private</code> であることを宣言します。 <i>list</i> 内の各変数の最終値は、割り当てられる場合、最後の <b>section</b> でその変数に割り当てられる値となります。値が割り当てられていない変数は、不確定値を持っています。         |
| <code>num_threads (<i>int_exp</i>)</code>             | <i>int_exp</i> の値は、並列領域用に使用するスレッドの数を指定する整数式です。スレッドの数の動的調整も使用可能である場合、 <i>int_exp</i> は使用されるスレッドの最大数を指定します。                                                                         |
| <code>reduction (<i>operator</i>: <i>list</i>)</code> | 指定された <i>operator</i> を使用して、 <i>list</i> 内のすべてのスカラー変数の縮約を実行します。 <i>list</i> 内の縮約変数は、コンマで区切られています。                                                                                 |

*list* 内の各変数の `private` コピーは、スレッドごとに作成されます。ステートメント・ブロックの最後で、縮約変数のすべての `private` コピーの最終値は、その演算子に適切な方法で結合され、その結果は、共用の縮約変数の元の値に戻されます。

**reduction** 文節で指定する変数は、以下のとおりでなければなりません。

- 演算子に適切な型でなければならない。
- 囲んでいるコンテキスト内で共用されていなければならない。
- `const` によって修飾された変数であってはならない。
- ポインター型があってはならない。

`nowait`

この文節は、**sections** ディレクティブ終了時の暗黙の **barrier** を回避するために使用します。これは、指定した並列領域内の複数の独立した作業共有セクションがある場合に有効です。**nowait** 文節が、所定の **sections** ディレクティブ上に現れるのは、1 回のみです。

## 注

**omp section** ディレクティブは、**omp sections** ディレクティブの中にある最初のプログラム・コードのセグメントのためのオプションです。後に続くセグメントは、その前に **omp section** ディレクティブがなければなりません。すべての **omp section** ディレクティブは、**omp sections** ディレクティブに関連したプログラム・ソース・コードのセグメントの字句構成内になければなりません。

プログラム実行が **omp sections** ディレクティブに到達すると、後続の **omp section** ディレクティブによって定義されたプログラム・セグメントは、並列実行のために使用可能なスレッド間で配布されます。**nowait** 文節が指定されていない限り、**barrier** は **omp sections** ディレクティブに関連した、より広いプログラム領域の終了地点に暗黙的に定義されます。

## 関連資料

422 ページの『並列処理を制御するプラグマ』

444 ページの『`#pragma omp parallel sections`』

## #pragma omp parallel sections

### 説明

**omp parallel sections** ディレクティブは、 **omp parallel** ディレクティブと **omp sections** ディレクティブを効果的に結合します。このディレクティブを使用すると、単一の **sections** ディレクティブを含んでいる並列領域をワンステップで定義することができます。

### 構文

```
#pragma omp parallel sections [clause[[,] clause] ...]
{
 [#pragma omp section]
 statement-block
 [#pragma omp section]
 statement-block
 .
 .
 .
}
]
```

### 注

**omp parallel** および **omp sections** ディレクティブに記載されているすべての文節および制限は、 **omp parallel sections** ディレクティブに適用されます。

### 関連資料

422 ページの『並列処理を制御するプラグマ』

434 ページの『#pragma omp parallel』

442 ページの『#pragma omp section、#pragma omp sections』



## #pragma omp single

### 説明

**omp single** ディレクティブは、単一の使用可能スレッドで実行しなければならないコードのセクションを識別します。

### 構文

```
#pragma omp single [clause[[,] clause] ...]
statement_block
```

ここで、*clause* は、次のいずれかです。

**private** (*list*) *list* 内のデータ変数のスコープが各スレッドに対して **private** であることを宣言します。 *list* 内のデータ変数は、コンマで区切られています。

また、**private** 文節内の変数は、同じ **omp single** ディレクティブ用の **copyprivate** 文節内に出現することはできません。

**copyprivate** (*list*) *list* 内で指定された変数の値を、チーム内のあるメンバーから他のメンバーにブロードキャストします。これは、**omp single** ディレクティブに関連付けられた構造化ブロックの実行の後、かつ、構成の終了時にすべてのスレッドがバリアから離れる前に行われます。チーム内の他のすべてのスレッドの場合、*list* 内の各変数は、構造化ブロックを実行したスレッド内の対応する変数の値を使用して定義されるようになります。 *list* 内のデータ変数は、コンマで区切られています。この文節に対する使用制限は、以下のとおりです。

- **copyprivate** 文節内の変数は、同じ **omp single** ディレクティブ用の **private** または **firstprivate** 文節内に出現することはできません。
- **copyprivate** 文節を持つ **omp single** ディレクティブが並列領域の動的範囲内で検出された場合、**copyprivate** 文節内で指定された変数はすべて、囲んでいるコンテキスト内で **private** でなければなりません。
- 並列領域の動的範囲内の **copyprivate** 文節内で指定された変数は、囲んでいるコンテキスト内で **private** でなければなりません。
- **copyprivate** 文節内で指定された変数には、アクセス可能かつ、あいまいさのないコピー割り当て演算子がなければなりません。

**firstprivate** (*list*) *list* 内のデータ変数のスコープが各スレッドに対して **private** であることを宣言します。それぞれの新規の **private** オブジェクトは、ステートメント・ブロック内に暗黙の宣言がある場合のように初期化されます。 *list* 内のデータ変数は、コンマで区切られています。

**firstprivate** 文節内の変数は、同じ **omp single** ディレクティブ用の **copyprivate** 文節内に出現することはできません。

**nowait** この文節は、**single** ディレクティブ終了時の暗黙の **barrier** を回避するために使用します。**nowait** 文節が、所定の **single** ディレクティブ上に現れるのは、1 回のみです。**nowait** 文節は、**copyprivate** 文節と共に使用してはいけません。

### 注

**nowait** 文節が指定されていない限り、暗黙の **barrier** が並列化されたステートメント・ブロックの最後にあります。

## 関連資料

422 ページの『並列処理を制御するプリAGMA』

## #pragma omp master

### 説明

**omp master** ディレクティブは、マスター・スレッドによってのみ実行されなければならないコードのセクションを識別します。

### 構文

```
#pragma omp master
statement_block
```

### 注

マスター・スレッド以外のスレッドは、この構成に関連したステートメント・ブロックを実行しません。

暗黙のバリアは、マスター・セクションの出入り口には存在しません。

### 関連資料

422 ページの『並列処理を制御するプリAGMA』

## #pragma omp critical

### 説明

**omp critical** ディレクティブは、単一スレッドによって一度に実行されなければならないコードのセクションを識別します。

### 構文

```
#pragma omp critical [(name)]
 statement_block
```

ここで、*name* は、オプションでクリティカル領域を識別するのに使用することができます。クリティカル領域を命名している ID には、外部結合があり、通常 ID が使用しているネーム・スペースとは異なるネーム・スペースを占めます。

### 注

スレッドは、プログラム内の他のスレッドが同じ名前でもクリティカル領域を実行しなくなるまで、指定された名前でも識別されたクリティカル領域の開始時点で待機します。**omp critical** ディレクティブ呼び出しによって特に命名されていないクリティカル・セクションは、未指定の同じ名前にマップされます。

### 関連資料

422 ページの『並列処理を制御するプリAGMA』

## #pragma omp barrier

### 説明

**omp barrier** ディレクティブは、そのセクション内の他のすべてのスレッドが同じポイントに達するまで並列領域のスレッドが待機する同期点を識別します。**omp barrier** ポイントを過ぎたステートメントの実行は、その後、並列で続行します。

### 構文

```
#pragma omp barrier
```

### 注

**omp barrier** ディレクティブは、1つのブロック内、または複合ステートメント内に現れなければなりません。例を以下に示します。

```
if (x!=0) {
 #pragma omp barrier /* valid usage */
}
if (x!=0)
 #pragma omp barrier /* invalid usage */
```

### 関連資料

422 ページの『並列処理を制御するプラグマ』

## #pragma omp flush

### 説明

**omp flush** ディレクティブは、並列領域内のすべてのスレッドがメモリー内の同じビューの指定したオブジェクトを持っていることをコンパイラーが保証するポイントを識別します。

### 構文

```
#pragma omp flush [(list)]
```

ここで、*list* は、同期化される変数のコンマ区切りのリストです。

### 注

*list* にポインターが含まれる場合、ポインターに参照されているオブジェクトではなく、ポインターがフラッシュされます。*list* が指定されていない場合は、自動ストレージ期間にアクセス不能なオブジェクトを除くすべての共用オブジェクトが同期化されます。

暗黙の **flush** ディレクティブは、以下のディレクティブとともに表示されます。

- **omp barrier**
- **omp critical** の出入り口。
- **omp parallel** からの出口。
- **omp for** からの出口。
- **omp sections** からの出口。
- **omp single** からの出口。

**omp flush** ディレクティブは、1 つのブロック内、または複合ステートメント内に現れなければなりません。例を以下に示します。

```
if (x!=0) {
 #pragma omp flush /* valid usage */
}
if (x!=0)
 #pragma omp flush /* invalid usage */
```

### 関連資料

- 422 ページの『並列処理を制御するプラグマ』
- 449 ページの『#pragma omp barrier』
- 448 ページの『#pragma omp critical』
- 436 ページの『#pragma omp for』
- 434 ページの『#pragma omp parallel』
- 441 ページの『#pragma omp parallel for』
- 444 ページの『#pragma omp parallel sections』
- 442 ページの『#pragma omp section、#pragma omp sections』
- 445 ページの『#pragma omp single』

## #pragma omp threadprivate

### 説明

**omp threadprivate** ディレクティブは、スレッドに **private** になる名前付きのファイル・スコープ、名前スペース・スコープ、または静的ブロック・スコープ変数を作成します。

### 構文

```
#pragma omp threadprivate (list)
```

ここで、*list* は、変数のコンマ区切りのリストです。

### 注

**omp threadprivate** データ変数の各コピーは、そのコピーを最初に使用する前に一度初期化されます。 **threadprivate** データ変数を初期化するために、オブジェクトが使用前に変更された場合、振る舞いは指定解除されます。

スレッドは、**omp threadprivate** データ変数の別のスレッドのコピーを参照してはなりません。プログラムの直列領域およびマスター領域の実行時に、参照は常にデータ変数のマスター・スレッドのコピーに対して行われます。

**omp threadprivate** ディレクティブの使用は、以下の点で管理されています。

- **omp threadprivate** ディレクティブは、すべての定義および宣言外のファイル・スコープになければならない。
- **omp threadprivate** ディレクティブは、静的ブロック・スコープ変数に対して適用でき、字句ブロック内に出現してそれらのブロック・スコープ変数を参照することができる。このディレクティブは、ネストされたスコープ内ではなく、変数のスコープ内にあり、かつそのリスト内の変数に対するすべての参照の前には出現しなければなりません。
- データ変数は、**omp threadprivate** ディレクティブの *list* に組み込む前に、ファイル・スコープで宣言しなければならない。
- **omp threadprivate** ディレクティブとその *list* の字句は、その *list* 内にあるデータ変数への参照の前になければならない。
- ある変換単位で **omp threadprivate** ディレクティブに指定しているデータ変数は、その変数が宣言されている他のすべての変換単位でも同様に指定しておかなければならない。
- **omp threadprivate** *list* 内で指定されたデータ変数は、**copyin**、**copyprivate**、**if**、**num\_threads**、および **schedule** 文節以外の文節に出現してはならない。
- **omp threadprivate** *list* 内のデータ変数のアドレスは、アドレス定数ではない。
- **omp threadprivate** *list* で指定しているデータ変数には、不完全型または参照型があってはならない。

### 関連資料

422 ページの『並列処理を制御するプリAGMA』

## コンパイラー・モードおよびプロセッサのアーキテクチャーの有効な組み合わせ

**-q32**、**-q64**、**-qarch**、および **-qtune** コンパイラー・オプションを使用して、コンパイラーの出力を以下に適合するように最適化することができます。

- ターゲット・プロセッサの非常に広範囲に渡る可能な選択
- 指定されたプロセッサ・アーキテクチャー・ファミリー内のプロセッサの範囲
- 単一の特定プロセッサ

一般的に、オプションでは以下が行われます。

- **-q32** は、32 ビット実行モードを選択します。
- **-q64** は、64 ビット実行モードを選択します。
- **-qarch** は、命令コードの生成対象として、一般的なファミリーのプロセッサ・アーキテクチャーを選択します。特定の **-qarch** 設定は、選択した **-qarch** 設定に応じてコンパイラーが生成するすべての 命令をサポートする RS/6000 システム上でのみ 稼働するコードを生成します。
- **-qtune** は、コンパイラー出力の最適化対象とする特定のプロセッサを選択します。 **-qtune** の設定には、 **-qarch** オプションとして指定できるものもあり、その場合は **-qtune** オプションとして再度指定する必要はありません。 **-qtune** オプションは、特定のシステム上で動作している場合にコードのパフォーマンスにのみ影響しますが、コードがどこで動作するかの判別はしません。

RS/6000 マシンには、以下の 3 つのメイン・ファミリーがあります。

- POWER
- POWER2
- PowerPC

すべての RS/6000 マシンは、共通の命令セットを共有しますが、所定のプロセッサまたはプロセッサ・ファミリーに固有の命令を追加して組み込むこともできます。

例えば、POWER2 の命令セットは、POWER の命令セットのスーパーセットです。PowerPC の命令セットには、POWER システムでは使用できない命令の一部が含まれていますが、POWER の命令セットのすべてをサポートしているわけではありません。また、これには、POWER の命令セットでは使用できない POWER2 の命令がいくつか含まれています。さらに、POWER2 の命令セットのいくつかのフィーチャーが、特定の PowerPC プロセッサ上にインプリメントされている場合もあれば、インプリメントされていない場合もあります。これらのオプションのフィーチャー・グループは、以下のとおりです。

- グラフィックス命令グループのサポート
- sqrt 命令グループのサポート
- 64 ビット・モード (**-q64** コンパイラー・オプション) のサポート

以下のテーブルに、選択されたいくつかのプロセッサ、およびさまざまなフィーチャーを、サポートされるもの、またはサポートされないものも含めて示します。



| プロセッサ | グラフィックス・サポート | sqr3 サポート | 64 ビット・サポート | 大規模ページ・サポート                    |
|-------|--------------|-----------|-------------|--------------------------------|
| 601   | なし           | なし        | なし          | なし                             |
| 603   | あり           | なし        | なし          | なし                             |
| 604   | あり           | なし        | なし          | なし                             |
| rs64a | なし           | なし        | あり          | なし                             |
| rs64b | あり           | あり        | あり          | なし                             |
| rs64c | あり           | あり        | あり          | なし                             |
| pwr3  | あり           | あり        | あり          | なし                             |
| pwr4  | あり           | あり        | あり          | あり (AIX v5.1D<br>以降を使用の場<br>合) |
| pwr5  | あり           | あり        | あり          | あり (AIX v5.1D<br>以降を使用の場<br>合) |

多様な種類のプロセッサ上で稼働するコードを生成したい場合は、以下のガイドラインを使用して、適切な **-qarch** および/または **-qtune** コンパイラ・オプションを選択します。以下をコンパイルに使用したコードの場合:

- **-qarch=com** を使用したコードは、いずれのシステム上でも稼働する。
- **-qarch=pwr** を使用したコードは、いずれの POWER または POWER2 マシン上でも稼働する。
- **-qarch=pwr2** (または **pwr2s**、**pwrx**、**p2sc**) を使用したコードは、POWER2 マシン上でのみ稼働する。
- **-qarch=pwr4** を使用したコードは、POWER4 マシン上でのみ稼働する。
- **-qarch=pwr5** を使用したコードは、POWER5 マシン上でのみ稼働する。
- **-qarch=ppc** を使用したコードは、いずれの PowerPC システム上でも稼働する。
- **-q64** を使用したコードは、64 ビットがサポートされている PowerPC マシン上でのみ稼働する。
- 特定のプロセッサを参照するその他の **-qarch** オプションは、機能的に同等の PowerPC マシンであればいずれにおいても稼働する。以下の表にある例では、**-qarch=pwr3** を使用してコンパイルされたコードも **rs64c** 上で稼働しますが、**rs64a** 上では稼働しません。同様に、**-qarch=603** を使用してコンパイルされたコードは、**pwr3** 上では稼働しますが、**rs64a** 上では稼働しません。

特定のプロセッサ用に特に最適化されたコードを生成したい場合の、**-q32**、**-q64**、**-qarch**、および **-qtune** コンパイラ・オプションの有効な組み合わせを以下の表に示します。

デフォルトの実行モードは、OBJECT\_MODE 環境変数が 64 に設定されているか、コマンド行で **-q64** が指定されていない限り、32 ビットです。

## 関連タスク

39 ページの『アーキテクチャー固有の (32 ビットまたは 64 ビットの) コンパイル用コンパイラ・オプションの指定』

28 ページの『64 ビットまたは 32 ビット・モードへの環境変数の設定』

## 関連資料

- 61 ページの『コンパイラーのコマンド行オプション』
- 73 ページの『32、64』
- 84 ページの『arch』
- 327 ページの『tune』

32 ビット実行モードの -qarch/-qtune の有効な組み合わせ

| -qarch オプション |           | 事前定義マクロ                                                                                                      | -qtune default | 使用可能<br>-qtune 設定                                                                       |
|--------------|-----------|--------------------------------------------------------------------------------------------------------------|----------------|-----------------------------------------------------------------------------------------|
| com          |           | _ARCH_COM                                                                                                    | pwr3           | auto pwr pwr2 pwr2s pwr3 pwr4 pwr5 ppc970 p2sc<br>601 602 603 604 403 rs64a rs64b rs64c |
|              | pwr       | _ARCH_COM _ARCH_PWR                                                                                          | pwr3           | auto pwr pwr2 pwr2s p2sc 601                                                            |
|              |           | _ARCH_COM _ARCH_PWR _ARCH_PWR2                                                                               | pwr2           | auto pwr2 pwr2s p2sc                                                                    |
|              |           | _ARCH_COM _ARCH_PWR _ARCH_PWR2 _ARCH_PWR2S                                                                   | pwr2s          | auto pwr2s                                                                              |
|              | ppc       | _ARCH_COM _ARCH_PWR _ARCH_PWR2 _ARCH_P2SC                                                                    | p2sc           | auto p2sc                                                                               |
|              |           | _ARCH_COM _ARCH_PPC                                                                                          | pwr3           | auto 601 602 603 604 403 rs64a rs64b rs64c pwr3<br>pwr4 pwr5 ppc970                     |
|              | 601       | _ARCH_COM _ARCH_PWR _ARCH_PPC _ARCH_PPC _ARCH_601                                                            | 601            | auto 601                                                                                |
|              | 602       | _ARCH_COM _ARCH_PPC _ARCH_602                                                                                | 602            | auto 602                                                                                |
|              | 403       | _ARCH_COM _ARCH_PPC _ARCH_403                                                                                | 403            | auto 403                                                                                |
|              | ppcgr     | _ARCH_COM _ARCH_PPC _ARCH_PPCGR                                                                              | 604            | auto 603 604 rs64b rs64c pwr3 pwr4 pwr5 ppc970                                          |
|              | 603       | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_603                                                                    | 603            | auto 603                                                                                |
|              | 604       | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_604                                                                    | 604            | auto 604                                                                                |
|              | ppc64     | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64                                                                  | pwr3           | auto rs64a rs64b rs64c pwr3 pwr4 pwr5 ppc970                                            |
|              | rs64a     | ARCH_COM _ARCH_PPC _ARCH_PPC64 _ARCH_RS64A                                                                   | rs64a          | auto rs64a                                                                              |
|              | ppc64gr   | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR                                                    | pwr3           | auto rs64b rs64c pwr3 pwr4 pwr5 ppc970                                                  |
|              | ppc64grsq | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_PPC64 _ARCH_PPC64GRSQ     | pwr3           | auto rs64b rs64c pwr3 pwr4 pwr5 ppc970                                                  |
|              | ppc64grsq | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_PPC64GRSQ _ARCH_PPC64GRSQ | pwr3           | auto pwr3 pwr4 pwr5 ppc970                                                              |
|              |           | ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_PPC64GRSQ _ARCH_PPC64GRSQ  | pwr4           | auto pwr4 pwr5 ppc970                                                                   |
|              |           | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_PPC64GRSQ _ARCH_PPC64GRSQ | pwr5           | auto pwr5                                                                               |
|              |           | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_PPC64GRSQ _ARCH_PPC64GRSQ | ppc970         | auto ppc970                                                                             |
|              |           | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_PPC64GRSQ _ARCH_PPC64GRSQ | rs64b          | auto rs64b                                                                              |
|              |           | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_PPC64GRSQ _ARCH_PPC64GRSQ | rs64c          | auto rs64c                                                                              |

64 ビット実行モードの -qarch/-qtune の有効な組み合わせ

| -qarch オプション |       | 事前定義マクロ                                                                                                         | -qtune default | 使用可能<br>-qtune 設定                                                                       |
|--------------|-------|-----------------------------------------------------------------------------------------------------------------|----------------|-----------------------------------------------------------------------------------------|
| com          |       | _ARCH_COM                                                                                                       | pwr3           | auto pwr pwr2 pwr2s pwr3 pwr4 pwr5 ppc970 p2sc<br>601 602 603 604 403 rs64a rs64b rs64c |
|              | ppc   | _ARCH_COM _ARCH_PPC                                                                                             | pwr3           | auto 601 602 604 403 rs64a rs64b rs64c pwr3<br>pwr4 pwr5 ppc970                         |
|              | ppc64 | _ARCH_COM _ARCH_PPC _ARCH_PPC64                                                                                 | pwr3           | auto rs64a rs64b rs64c pwr3 pwr4 pwr5 ppc970                                            |
|              |       | _ARCH_COM _ARCH_PPC _ARCH_PPC64 _ARCH_RS64A                                                                     | rs64a          | auto rs64a                                                                              |
|              |       | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR                                                       | pwr3           | auto rs64b rs64c pwr3 pwr4 pwr5 ppc970                                                  |
|              |       | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ                                    | pwr3           | auto rs64b rs64c pwr3 pwr4 pwr5 ppc970                                                  |
|              |       | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_PWR3                         | pwr3           | auto pwr3 pwr4 pwr5 ppc970                                                              |
|              |       | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_PWR3 _ARCH_PWR4              | pwr4           | auto pwr4 pwr5 ppc970                                                                   |
|              |       | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_PWR3 _ARCH_PWR4 _ARCH_PWR5   | pwr5           | auto pwr5                                                                               |
|              |       | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_PWR3 _ARCH_PWR4 _ARCH_PPC970 | ppc970         | auto ppc970                                                                             |
|              |       | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_RS64B                        | rs64b          | auto rs64b                                                                              |
|              |       | _ARCH_COM _ARCH_PPC _ARCH_PPCGR _ARCH_PPC64 _ARCH_PPC64GR<br>_ARCH_PPC64GRSQ _ARCH_RS64C                        | rs64c          | auto rs64c                                                                              |



## コンパイラー・メッセージ

このセクションでは、コンパイラーがコンパイル・エラーを記述するのに使用するいくつかの基本的レポート・メカニズムについて概説します。

- 『メッセージ重大度レベルとコンパイラー応答』
- 458 ページの『コンパイラー戻りコード』
- 459 ページの『コンパイラー・メッセージ・フォーマット』

### メッセージ重大度レベルとコンパイラー応答

以下の表では、各レベルのメッセージ重大度に関連したコンパイラー応答を示します。

| 文字 | 重大度     | コンパイラーの応答                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| I  | 通知      | コンパイルは継続します。このメッセージは、コンパイルと途中で検出された条件を報告します。                                                                                                                                                                                                                                                                                                                                                                                         |
| W  | 警告      | コンパイルは継続します。このメッセージは、疑わしい条件および意図したものではない条件を報告します。プログラムは、書かれたとおりに実行します。                                                                                                                                                                                                                                                                                                                                                               |
| E  | エラー     |  コンパイルは継続し、オブジェクト・コードが生成されます。コンパイラーが訂正することのできるエラー条件が存在しますが、プログラムは正しく実行できない可能性があります。                                                                                                                                                                                                                                                               |
| S  | 重大エラー   | コンパイルは継続しますが、オブジェクト・コードは生成されません。コンパイラーが訂正できないエラー条件が存在します。                                                                                                                                                                                                                                                                                                                                                                            |
| U  | 回復不能エラー |  コンパイラーは停止します。内部コンパイラー・エラーが発生しました。 <ul style="list-style-type: none"><li>• メッセージがリソースの限界 (例えばファイル・システムまたはページング・スペースがいっぱいになった) を示している場合は、リソースを追加して再コンパイルしてください。</li><li>• メッセージが別のコンパイラー・オプションの必要性を示している場合は、それらを使用して再コンパイルしてください。</li><li>• 回復不能エラーの前に報告されたその他のエラーを検査して訂正してください。</li><li>• 回復不能エラーが解決されない場合は、IBM サービス技術員にメッセージを報告してください。</li></ul> |

### 関連概念

12 ページの『コンパイラー・メッセージおよびリスト情報』

## 関連資料

『コンパイラー戻りコード』

459 ページの『コンパイラー・メッセージ・フォーマット』

158 ページの『halt』

234 ページの『maxerr』

160 ページの『haltonmsg』

---

## コンパイラー戻りコード

コンパイルの終了時に、コンパイラーは、以下の任意の条件のもとで戻りコードをゼロに設定します。

- メッセージが発行されない。
- 診断されたすべてのエラーの最高重大度レベルが、**-qhalt** コンパイラー・オプションの設定値よりも小さい、さらに、エラーの数が **-qmaxerr** コンパイラー・オプションで設定した限界値に達していない。
- **-qhaltonmsg** コンパイラー・オプションで指定されたメッセージが発行されない。

それ以外の場合、コンパイラーは以下の値の 1 つに戻りコードを設定します。

| 戻りコード | エラー・タイプ                                                           |
|-------|-------------------------------------------------------------------|
| 1     | halt コンパイラー・オプションの設定値よりも高い重大度レベルのエラーが検出されました。                     |
| 40    | オプション・エラーまたは回復不能エラーが検出されました。                                      |
| 41    | 構成ファイル・エラーが検出されました。                                               |
| 250   | メモリー不足のエラーが検出されました。 <b>xlc</b> コマンドで、使用するためのメモリーをさらに割り振ることはできません。 |
| 251   | シグナルで受信されたエラーが検出されました。つまり、回復不能エラーまたは割り込みシグナルが発生しました。              |
| 252   | ファイルを見つけられないエラーが検出されました。                                          |
| 253   | 入出力エラーが検出されました。ファイルを読み取ったり、ファイルに書き込むことができません。                     |
| 254   | fork エラーが検出されました。新規プロセスを作成することができません。                             |
| 255   | プロセスの実行中にエラーが検出されました。                                             |

**注:** 実行時エラーの戻りコードが表示される可能性があります。例えば、実行時戻りコード **99** は、静的な初期化が失敗したことを示します。

## 関連概念

12 ページの『コンパイラー・メッセージおよびリスト情報』

## 関連資料

457 ページの『メッセージ重大度レベルとコンパイラー応答』

459 ページの『コンパイラー・メッセージ・フォーマット』

158 ページの『halt』  
234 ページの『maxerr』  
160 ページの『haltonmsg』

---

## コンパイラー・メッセージ・フォーマット

**-qnosrcmsg** オプションがアクティブ (これがデフォルト) のとき、診断メッセージのフォーマットは次のとおりです。

`"file", line line_number.column_number: 15dd-nnn (severity) text.`

ここで、

|                      |                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------|
| <i>file</i>          | エラーのある C または C++ ソース・ファイル名です。                                                              |
| <i>line_number</i>   | エラーの行番号です。                                                                                 |
| <i>column_number</i> | エラーの列番号です。                                                                                 |
| 15                   | コンパイラー製品 ID です。                                                                            |
| <i>dd</i>            | このメッセージを発行した XL C/C++ Enterprise Edition コンポーネントを示す 2 桁のコードです。 <i>dd</i> は、以下の値のいずれかになります。 |
| 00                   | - メッセージを生成または最適化するコード                                                                      |
| 01                   | - コンパイラー・サービス・メッセージ                                                                        |
| 05                   | - C コンパイラーに固有のメッセージ                                                                        |
| 06                   | - C コンパイラーに固有のメッセージ                                                                        |
| 40                   | - C++ コンパイラーに固有のメッセージ                                                                      |
| 47                   | - munch ユーティリティーに固有のメッセージ                                                                  |
| 86                   | - プロシージャーク間分析 (IPA) に固有のメッセージ                                                              |
| <i>nnn</i>           | メッセージ番号です。                                                                                 |
| <i>severity</i>      | エラーの重大度を表す文字です。                                                                            |
| <i>text</i>          | エラーの内容を説明するメッセージです。                                                                        |

**-qsrcmsg** オプションが指定されているとき、診断メッセージのフォーマットは次のとおりです。

`x - 15dd-nnn(severity) text.`

ここで、**x** はフィンガー行のフィンガーを示す文字です。

### 関連概念

12 ページの『コンパイラー・メッセージおよびリスト情報』

### 関連資料

457 ページの『メッセージ重大度レベルとコンパイラー応答』  
458 ページの『コンパイラー戻りコード』  
158 ページの『halt』  
234 ページの『maxerr』  
160 ページの『haltonmsg』





## 並列処理のサポート

本節では、並列処理の制御に使用する環境変数と組み込み関数についての情報を記載します。本節には以下のトピックがあります。

- 『IBM SMP 並列処理のためのランタイム・オプション』
- 465 ページの『並列処理のための OpenMP ランタイム・オプション』
- 467 ページの『並列処理に使用する組み込み関数』

## IBM SMP 並列処理のためのランタイム・オプション

SMP 並列処理に影響を及ぼすランタイム・オプションは、XLSMPOPTS 環境変数を使用して指定できます。この環境変数は、アプリケーションを実行する前に設定しなければならず、以下の形式の基本構文を使用します。



並列化ランタイム・オプションは、OMP 環境変数を使用して指定することもできます。OMP および XLSMPOPTS に固有の環境変数で指定されるランタイム・オプションが競合する場合は、OMP オプションが優先されます。

**注:** 並列化されたプログラム・コードをコンパイルする場合は、スレッド・セーフ・コンパイラ・モードを使用して起動しなければなりません。

XLSPMPOPTS 環境変数に応じた SMP 実行時オプション設定を、カテゴリー別に分類して以下に示します。

## スケジューリング・アルゴリズム・オプション

### XLSMPOPTS

#### 環境変数

#### オプション

`schedule=algorithm=[n]`

#### 説明

このオプションは、**ibm schedule** プラグマでスケジューリング・アルゴリズムを明示的に割り当てられていないループ用に使用するスケジューリング・アルゴリズムを指定します。

*algorithm* の有効なオプションは、以下のとおりです。

- `guided`
- `affinity`
- `dynamic`
- `static`

指定する場合、チャンク・サイズ *n* は 1 以上の整数値でなければなりません。

デフォルトのスケジューリング・アルゴリズムは **static** です。

これらのアルゴリズムに関する説明は、『**#pragma ibm schedule**』を参照してください。

## 並列環境オプション

### XLSMPOPTS

#### 環境変数

#### オプション

`parthds=num`

#### 説明

*num* は、必要な並列スレッドの数を示します。この数は、システムで使用可能なプロセッサの数と通常同じです。

アプリケーションによっては、使用可能なプロセッサの最大数を超えてスレッドを使用することはできません。その他のアプリケーションは、存在するプロセッサ数より多くのスレッドを使用するとパフォーマンスがかなり向上します。このオプションでは、プログラムを実行するために使用するユーザー・スレッドの数を完全に制御することができます。

*num* のデフォルト値は、システムで使用可能なプロセッサの数です。

`usrthds=num`

*num* は、予期されるユーザー・スレッドの数を示します。

プログラム・コードによって明示的にスレッドが作成される場合は、このオプションを使用してください。この場合は、作成するスレッドの数に *num* を設定してください。

*num* のデフォルト値は 0 です。

`stack=num`

*num* は、スレッドのスタックに必要なスペースの最大量を指定します。

*num* のデフォルト値は 4194304 です。

## パフォーマンス調整オプション

### XLSMPOPTS

#### 環境変数

#### オプション

`spins=num`

#### 説明

*num* は、譲歩するまでのループ・スピンまたは反復の数を示します。

スレッドは、処理を完了すると、新しい処理を探してタイトなループの実行を続行します。各作業待ち状態中に、作業待ち行列の完全スキャンが 1 回実行されます。拡張作業待ち状態によって特定のアプリケーションの応答性を高くすることができますが、定期的なスキャンして他のアプリケーションからの要求に譲歩するようにスレッドに指示しない限り、システム全体としての応答が低下する場合があります。

**spins** および **yields** の両方を 0 に設定することによって、ベンチマークを目的として完全な作業待ち状態にすることができます。

*num* のデフォルト値は 100 です。

`yields=num`

*num* は、スリープするまでの譲歩の数を示します。

スレッドがスリープすると、実行する処理があることが別のスレッドによって示されるまで完全に実行が中断されます。これによって、システムの使用効率は向上しますが、アプリケーションに余分なシステム・オーバーヘッドが加わります。

*num* のデフォルト値は 100 です。

`delays=num`

*num* は、作業キューの各スキャンの間の、処理なしの遅延時間の長さを示します。遅延の各単位は、メモリーへのアクセスがない遅延ループを一度実行することによって行われます。

*num* のデフォルト値は 500 です。

## 動的プロファイル・オプション

### XLSPMPOPTS

#### 環境変数

#### オプション

profilefreq=num

#### 説明

*num* は、並列処理が適正であるかどうかを判別するために各ループを再調査するサンプリング率を示します。

ランタイム・ライブラリーは、動的プロファイルを使用して、自動的に並列化されるループのパフォーマンスを動的に調整します。動的プロファイルは、ループの実行時間に関する情報を収集して、次回はループを順次実行すべきか並列に実行すべきかを判別します。実行時間のしきい値は、以下で説明する **parthreshold** および **seqthreshold** 動的プロファイル・オプションによって設定します。

*num* が 0 の場合は、プロファイルはすべてオフになり、プロファイルのために起こるオーバーヘッドはなくなります。 *num* が 0 より大きい場合は、ループを *num* 回実行するごとに 1 回、ループの実行時間がモニターされます。

*num* のデフォルトは 16 です。最大のサンプリング率は 32 です。32 を超える *num* の値は 32 に変更されます。

parthreshold=mSec

*mSec* は、ループを順次実行しなければならないとする実行時間の上限をミリ秒単位で指定します。 *mSec* は、小数部を使用して指定することができます。

**parthreshold** を 0 に設定すると、並列化されたループは動的プロファイラーによって直列化されなくなります。

*mSec* のデフォルト値は 0.2 ミリ秒です。

seqthreshold=mSec

*mSec* は、動的プロファイラーによって直列化されたループを再び並列モードで実行するように戻す実行時間の下限をミリ秒単位で指定します。 *mSec* は、小数部を使用して指定することができます。

*mSec* のデフォルト値は 5 ミリ秒です。

### 関連概念

15 ページの『プログラムの並列化』

15 ページの『IBM SMP ディレクティブ』

17 ページの『OpenMP ディレクティブ』

### 関連資料

465 ページの『並列処理のための OpenMP ランタイム・オプション』

293 ページの『smp』

422 ページの『並列処理を制御するプラグマ』

467 ページの『並列処理に使用する組み込み関数』

OpenMP 仕様の完全な情報については、以下を参照してください。

OpenMP Web サイト [www.openmp.org](http://www.openmp.org)

OpenMP Specification [www.openmp.org/specs](http://www.openmp.org/specs)

---

## 並列処理のための OpenMP ランタイム・オプション

並列処理に影響を及ぼす OpenMP ランタイム・オプションは、OMP 環境変数を指定して設定します。これらの環境変数は、以下の形式の構文を使用します。

▶—*env\_variable*—=*option\_and\_args*—▶

OMP 環境変数が明示的に設定されない場合、デフォルト設定が使用されます。

並列化ランタイム・オプションは、XLSMPOPTS 環境変数で指定することもできます。OMP ランタイム・オプションおよび XLSMPOPTS ランタイム・オプションが競合するときは、OMP オプションが優先されます。

**注:** 並列化されたプログラム・コードをコンパイルする場合は、スレッド・セーフ・コンパイラー・モードを使用して起動しなければなりません。

OpenMP ランタイム・オプションは、以下で説明する別々のカテゴリーに分類されます。

### スケジューリング・アルゴリズム環境変数

OMP\_SCHEDULE=*algorithm*

このオプションは、**omp schedule** ディレクティブでスケジューリング・アルゴリズムを明示的に割り当てられないループに対して使用するスケジューリング・アルゴリズムを指定します。例を以下に示します。

```
OMP_SCHEDULE="guided, 4"
```

*algorithm* の有効なオプションは、以下のとおりです。

- **dynamic**[, *n*]
- **guided**[, *n*]
- **runtime**
- **static**[, *n*]

チャンク・サイズを *n* で指定する場合、*n* の値は 1 以上の整数値でなければなりません。

デフォルトのスケジューリング・アルゴリズムは **static** です。

これらのアルゴリズムに関する説明は、462 ページの『スケジューリング・アルゴリズム・オプション』を参照してください。

## 並列環境環境変数

OMP\_NUM\_THREADS=*num*

*num* は、必要な並列スレッドの数を示します。この数は、システムで使用可能なプロセッサの数と通常同じです。

この数は、**omp\_set\_num\_threads( )** ランタイム・ライブラリー関数を呼び出すことによって、プログラム実行中にオーバーライドすることができます。

アプリケーションによっては、使用可能なプロセッサの最大数を超えてスレッドを使用することはできません。その他のアプリケーションは、存在するプロセッサ数より多くのスレッドを使用するとパフォーマンスがかなり向上します。このオプションでは、プログラムを実行するために使用するユーザー・スレッドの数を完全に制御することができます。

*num* のデフォルト値は、システムで使用可能なプロセッサの数です。

いくつかの **#pragma omp** ディレクティブで使用可能な **num\_threads** 文節を使用して、指定された **parallel section** の OMP\_NUM\_THREADS の設定をオーバーライドすることができます。

OMP\_NESTED=TRUEIFALSE

この環境変数は、ネストされた並列性を使用可能または使用不可にします。この環境変数の設定は、**omp\_set\_nested( )** ランタイム・ライブラリー関数を呼び出してオーバーライドすることができます。

ネストされた並列性が使用不可の場合、ネストされた並列領域は直列化され、現在のスレッドで稼働します。

現在のインプリメンテーションでは、ネストされた並列領域は、常に直列化されています。その結果、OMP\_SET\_NESTED は何の影響も及ぼさず、**omp\_get\_nested()** は常に 0 を戻します。**-qsmp=nested\_par** オプションがオンの場合 (非精密 OMP モードでのみ) は、ネストされた並列領域は、追加のスレッドを使用可能として使用場合があります。ただし、ネストされた並列領域を実行するために、新規のチームが作成されることはありません。

OMP\_NESTED のデフォルト値は FALSE です。

## 動的プロファイル環境変数

OMP\_DYNAMIC=TRUEIFALSE

この環境変数は、並列領域の実行に使用可能なスレッドの数の動的調整を使用可能または使用不可にします。

TRUE に設定されている場合、並列領域の実行に使用可能なスレッドの数は、システム・リソースを最も有効に使用するために実行時に調整されます。詳しくは、464 ページの『動的プロファイル・オプション』の **profilefreq=num** の説明を参照してください。

FALSE に設定されている場合、動的調整は使用不可になります。

デフォルト設定は TRUE です。

### 関連概念

- 15 ページの『プログラムの並列化』
- 15 ページの『IBM SMP ディレクティブ』
- 17 ページの『OpenMP ディレクティブ』

### 関連資料

- 461 ページの『IBM SMP 並列処理のためのランタイム・オプション』
- 293 ページの『smp』
- 422 ページの『並列処理を制御するプラグマ』
- 『並列処理に使用する組み込み関数』

OpenMP 仕様の完全な情報については、以下を参照してください。

- OpenMP Web サイト ( [www.openmp.org](http://www.openmp.org) )
- OpenMP Specification ( [www.openmp.org/specs](http://www.openmp.org/specs) )

---

## 並列処理に使用する組み込み関数

以下の組み込み関数を使用して、並列環境に関する情報を取得します。 **omp\_** 関数の関数定義は、**omp.h** ヘッダー・ファイルにあります。

| 関数プロトタイプ             | 説明                                                                                                                                                                                                                               |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int __parthds(void); | <p>この関数は、<b>parthds</b> ランタイム・オプションの値を返します。 <b>parthds</b> オプションがユーザーによって明示的に設定されていない場合、関数は、ランタイム・ライブラリーによって設定されたデフォルト値を返します。</p> <p>プログラムのコンパイル時に <b>-qsmp</b> コンパイラー・オプションが指定されなかった場合、この関数は、選択されたランタイム・オプションに関係なく 1 を返します。</p> |

| 関数プロトタイプ                                                | 説明                                                                                                                                                                                                                                                                                                              |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>int __usrthds(void);</code>                       | <p>この関数は、<b>usrthds</b> ランタイム・オプションの値を返します。</p> <p><b>usrthds</b> オプションがユーザーによって明示的に設定されていないか、プログラムのコンパイル時に <b>-qsmp</b> コンパイラー・オプションが指定されなかった場合、この関数は、選択されたランタイム・オプションに関係なく 0 を返します。</p>                                                                                                                       |
| <code>int omp_get_num_threads(void);</code>             | <p>この関数は、この関数が呼び出されている並列領域を実行しているチームに現在あるスレッドの数を返します。</p>                                                                                                                                                                                                                                                       |
| <code>void omp_set_num_threads(int num_threads);</code> | <p>この関数は <b>OMP_NUM_THREADS</b> 環境変数の設定をオーバーライドし、このディレクティブに続く並列領域で使用するスレッド数を指定します。値 <code>num_threads</code> は正の整数でなければなりません。</p> <p><code>num_threads</code> 文節が存在し、並列領域のために適用される場合、これは、<code>omp_set_num_threads</code> ライブラリー関数または <b>OMP_NUM_THREADS</b> 環境変数によって要求されるスレッド数を置き換えます。後続の並列領域は、この影響を受けません。</p> |
| <code>int omp_get_max_threads(void);</code>             | <p>この関数は、<code>omp_get_num_threads</code> に対する呼び出しで戻ることができる最大値を返します。</p>                                                                                                                                                                                                                                        |
| <code>int omp_get_thread_num(void);</code>              | <p>この関数は、そのチームの範囲内で、この関数を実行しているスレッドのスレッド番号を返します。スレッド番号は、0 以上 <code>omp_get_num_threads()-1</code> 以下となります。チームのマスター・スレッドは、スレッド 0 です。</p>                                                                                                                                                                         |
| <code>int omp_get_num_procs(void);</code>               | <p>この関数は、プログラムに割り当てることができるプロセッサの最大数を返します。</p>                                                                                                                                                                                                                                                                   |
| <code>int omp_in_parallel(void);</code>                 | <p>この関数は、並列で実行している並列領域の動的範囲内で呼び出された場合、非ゼロを返します。それ以外は 0 を返します。</p>                                                                                                                                                                                                                                               |
| <code>void omp_set_dynamic(int dynamic_threads);</code> | <p>この関数は、並列領域の実行に使用可能なスレッドの数の動的調整を使用可能または使用不可にします。</p>                                                                                                                                                                                                                                                          |
| <code>int omp_get_dynamic(void);</code>                 | <p>この関数は、動的スレッドの調整が使用可能な場合に非ゼロを返し、それ以外は 0 を返します。</p>                                                                                                                                                                                                                                                            |
| <code>void omp_set_nested(int nested);</code>           | <p>この関数は、ネストされた並列性を使用可能または使用不可にします。</p>                                                                                                                                                                                                                                                                         |
| <code>int omp_get_nested(void);</code>                  | <p>この関数は、ネストされた並列性が使用可能な場合に非ゼロを返し、使用不可の場合は 0 を返します。</p>                                                                                                                                                                                                                                                         |



| 関数プロトタイプ                                                                                               | 説明                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>void omp_init_lock(omp_lock_t *lock); void omp_init_nest_lock(omp_nest_lock_t *lock);</pre>       | これらの関数は、ロックを初期化する手段のみを提供します。それぞれの関数は、以降の呼び出しで使用するために、パラメーター <i>lock</i> に関連したロックを初期化します。                                                                  |
| <pre>void omp_destroy_lock(omp_lock_t *lock); void omp_destroy_nest_lock(omp_nest_lock_t *lock);</pre> | これらの関数では、指定されたロック変数 <i>lock</i> が初期化されません。                                                                                                                |
| <pre>void omp_set_lock(omp_lock_t *lock); void omp_set_nest_lock(omp_nest_lock_t *lock);</pre>         | これらの各関数は、指定されたロックが使用可能になりロックを設定するまで、その関数を実行しているスレッドをブロックします。アンロックされている場合は、単純ロックが使用可能です。ネスト可能なロックは、アンロックされている場合、またはこの関数を実行しているスレッドによってすでに所有されている場合は使用可能です。 |
| <pre>void omp_unset_lock(omp_lock_t *lock); void omp_unset_nest_lock(omp_nest_lock_t *lock);</pre>     | これらの関数は、ロックの所有権を解放する手段を提供します。                                                                                                                             |
| <pre>int omp_test_lock(omp_lock_t *lock); int omp_test_nest_lock(omp_nest_lock_t *lock);</pre>         | これらの関数は、ロックを設定しようとしませんが、スレッドの実行はブロックしません。                                                                                                                 |
| <pre>double omp_get_wtime(void);</pre>                                                                 | 固定された開始時刻から経過した時間を返します。固定された開始時刻の値は、現行プログラムの開始時に決定され、プログラム実行中は一定に保たれます。                                                                                   |
| <pre>double omp_get_wtick(void);</pre>                                                                 | 時計が音を刻む間の秒数を返します。                                                                                                                                         |

**注:** 現在のインプリメンテーションでは、ネストされた並列領域は、常に直列化されています。その結果、**omp\_set\_nested** は、何の影響力も持たず、**omp\_get\_nested** は、常に、0 を返します。

OpenMP ランタイム・ライブラリー関数についての完全な情報は、OpenMP C/C++ アプリケーション・プログラム・インターフェースの仕様を参照してください。

## 関連概念

15 ページの『プログラムの並列化』

## 関連タスク

28 ページの『並列処理ランタイム・オプションの設定』

47 ページの『プラグマを使用した並列処理の制御』

## 関連資料

422 ページの『並列処理を制御するプラグマ』

293 ページの『smp』

461 ページの『IBM SMP 並列処理のためのランタイム・オプション』

465 ページの『並列処理のための OpenMP ランタイム・オプション』  
475 ページの『付録 B. 組み込み関数』

---

## 第 4 部 付録



## 付録 A. 事前定義マクロ

事前定義マクロは、AIX プラットフォームに関連するマクロと、言語フィーチャーに関連するマクロの、2 つの大きなカテゴリに分かれます。ここでは、プラットフォーム固有のマクロについて説明します。

言語固有のマクロの詳細については、「*C/C++ ランゲージ・リファレンス*」の『プリプロセッサ・ディレクティブ』セクションを参照してください。

### プラットフォームに関連するマクロ

プラットフォーム間でのアプリケーションの移植を容易にするために、以下の事前定義マクロが提供されています。

| 事前定義マクロ名                       | 説明                                                                                                                                                                                                                                                                                     |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>__BASE_FILE__</code>     | 基本ソース・ファイルの完全修飾ファイル名に対して定義されます。                                                                                                                                                                                                                                                        |
| <code>__BIG_ENDIAN</code>      | 1 に定義されます。                                                                                                                                                                                                                                                                             |
| <code>__BIG_ENDIAN__</code>    | 1 に定義されます。                                                                                                                                                                                                                                                                             |
| <code>__CALL_SYSV</code>       | 1 に定義されます。                                                                                                                                                                                                                                                                             |
| <code>__CHAR_UNSIGNED__</code> | ▶ <b>C++</b> <code>-qchars=unsigned</code> または <code>#pragma chars(unsigned)</code> が有効な場合、1 に定義されます。 <code>-qchars=signed</code> または <code>#pragma chars(signed)</code> が有効な場合、このマクロは未定義です。                                                                                           |
| <code>__EXCEPTIONS</code>      | ▶ <b>C++</b> <code>-qeh</code> オプションが有効な場合、1 に定義されます。それ以外の場合、これは定義されません。                                                                                                                                                                                                               |
| <code>__OPTIMIZE__</code>      | 最適化レベル <code>-O</code> または <code>-O2</code> では 2 に定義され、最適化レベル <code>-O3</code> またはそれ以上では 3 に定義されます。                                                                                                                                                                                    |
| <code>__OPTIMIZE_SIZE__</code> | オプション <code>-qcompact</code> および <code>-O</code> が設定されている場合、1 に定義されます。それ以外の場合、これは定義されません。                                                                                                                                                                                              |
| <code>__powerpc</code>         | 1 に定義されます。                                                                                                                                                                                                                                                                             |
| <code>__powerpc__</code>       | 1 に定義されます。                                                                                                                                                                                                                                                                             |
| <code>__powerpc64__</code>     | 64 ビット・モードでコンパイルするときは、1 に定義されます。それ以外の場合、これは定義されません。                                                                                                                                                                                                                                    |
| <code>__PPC</code>             | 1 に定義されます。                                                                                                                                                                                                                                                                             |
| <code>__PPC__</code>           | 1 に定義されます。                                                                                                                                                                                                                                                                             |
| <code>__ppc64</code>           | 64 ビット・モードでコンパイルするときは、1 に定義されます。それ以外の場合、これは定義されません。                                                                                                                                                                                                                                    |
| <code>__PPC64__</code>         | 64 ビット・モードでコンパイルするときは、1 に定義されます。それ以外の場合、これは定義されません。                                                                                                                                                                                                                                    |
| <code>__PTRDIFF_TYPE__</code>  | このプラットフォームでの基礎となる型 <code>ptrdiff_t</code> に定義されます。32 ビット・モードでは、値は <code>int</code> です。64 ビット・モードでは、値は <code>long int</code> です。                                                                                                                                                        |
| <code>__SIZE_TYPE__</code>     | このプラットフォームでの基礎となる型 <code>size_t</code> に定義されます。このプラットフォームでは、マクロは <code>long unsigned int</code> として定義されます。32 ビット・モードでは、マクロは <code>unsigned int</code> として定義されます。64 ビット・モードでは、マクロは <code>long int</code> として定義されます。コンパイル・モードは、 <code>-q32</code> および <code>-q64</code> オプションによって制御されます。 |
| <code>__unix</code>            | すべての UNIX 型のプラットフォームで 1 に定義されます。それ以外の場合、これは定義されません。                                                                                                                                                                                                                                    |
| <code>__unix__</code>          | すべての UNIX 型のプラットフォームで 1 に定義されます。それ以外の場合、これは定義されません。                                                                                                                                                                                                                                    |



## 付録 B. 組み込み関数

コンパイラーは、より効率的なプログラムの作成を援助するため、ユーザーが組み込み関数を選択できるようにします。本節では、使用可能な各種の組み込み関数を要約します。

467 ページの『並列処理に使用する組み込み関数』で説明している追加の組み込み関数を検索することもできます。

| 名前        | プロトタイプ                                                              | 説明                                                                                                               |
|-----------|---------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| __alignx  | void __alignx(int <i>alignment</i> , const void * <i>address</i> ); | 指定された <i>address</i> が既知のコンパイル時オフセットで位置合わせされることをコンパイラーに知らせます。 <i>位置合わせ</i> は、ゼロより大きい値を持つ正の定数整数で、2 の累乗でなければなりません。 |
| __assert1 | int __assert1(int, int, int);                                       | カーネルのデバッグ用の TRAP 命令を生成します。<br><i>/usr/include/sys/syspest.h</i> を参照してください。                                       |
| __assert2 | void assert2(int);                                                  | カーネルのデバッグ用の TRAP 命令を生成します。<br><i>/usr/include/sys/syspest.h</i> を参照してください。                                       |
| __bcopy   | void __bcopy(char *, char *, int);                                  | ブロック・コピー                                                                                                         |
| __bzero   | void __bzero(char *, int);                                          | ブロック・ゼロ                                                                                                          |

| 名前                           | プロトタイプ                                                                                         | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------|------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>__check_lock_mp</code> | <code>unsigned int __check_lock_mp (const int*<br/>addr, int old_value, int new_value);</code> | <p>マルチプロセッサ・システムでのロックを検査します。</p> <p>条件付きで、シングル・ワード変数をアトミックに更新します。<i>addr</i> は、シングル・ワード変数のアドレスを指定します。<i>old_value</i> は、シングル・ワード変数の値に照らし合わせて検査される元の値を指定します。<i>new_value</i> は、シングル・ワード変数に条件付きで割り当てられる新規の値を指定します。ワード変数は、フルワード境界で位置合わせしておかなければなりません。</p> <p>戻り値:</p> <ol style="list-style-type: none"> <li>1. <code>false</code> (偽) が戻された場合は、シングル・ワード変数が元の値と同じであったこと、さらにこの変数が新規の値に設定されたことを示す。</li> <li>2. <code>true</code> (真) が戻された場合は、シングル・ワード変数が元の値と同じでなかったこと、さらにこの変数が変更されないままになっていることを示す。</li> </ol> |



| 名前                            | プロトタイプ                                                                                                                                       | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>__check_lockd_mp</code> | <pre>unsigned int __check_lockd_mp (const long long int* <i>addr</i>, long long int <i>old_value</i>, long long int <i>new_value</i>);</pre> | <p>マルチプロセッサ・システムでのロック・ダブルワードを検査します。</p> <p>条件付きで、ダブルワード変数をアトミックに更新します。<i>addr</i> は、ダブルワード変数のアドレスを指定します。<i>old_value</i> は、ダブルワード変数の値に照らし合わせて検査される元の値を指定します。<i>new_value</i> は、ダブルワード変数に条件付きで割り当てられる新規の値を指定します。ダブルワード変数は、ダブルワード境界で位置合わせしておかなければなりません。</p> <p>戻り値:</p> <ol style="list-style-type: none"> <li>1. <code>false</code> (偽) が戻された場合は、ダブルワード変数が元の値と同じであったこと、さらにこの変数が新規の値に設定されたことを示す。</li> <li>2. <code>true</code> (真) が戻された場合は、ダブルワード変数が元の値と同じでなかったこと、さらにこの変数が変更されないままになっていることを示す。</li> </ol> |

| 名前                           | プロトタイプ                                                                                         | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------------|------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>__check_lock_up</code> | <code>unsigned int __check_lock_up (const int*<br/>addr, int old_value, int new_value);</code> | <p>単一プロセッサ・システムでのロックを検査します。</p> <p>条件付きで、シングル・ワード変数をアトミックに更新します。<i>addr</i> は、シングル・ワード変数のアドレスを指定します。<i>old_value</i> は、シングル・ワード変数の値に照らし合わせて検査される元の値を指定します。<i>new_value</i> は、シングル・ワード変数に条件付きで割り当てられる新規の値を指定します。ワード変数は、フルワード境界で位置合わせしておかなければなりません。</p> <p>戻り値:</p> <ul style="list-style-type: none"> <li>• <code>false</code> (偽) が戻された場合は、シングル・ワード変数が元の値と同じであったこと、さらにこの変数が新規の値に設定されたことを示す。</li> <li>• <code>true</code> (真) が戻された場合は、シングル・ワード変数が元の値と同じでなかったこと、さらにこの変数が変更されないままになっていることを示す。</li> </ul> |

| 名前                            | プロトタイプ                                                                                                                                         | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>__check_lockd_up</code> | <code>unsigned int __check_lockd_up (const long long int* <i>addr</i>, long long int <i>old_value</i>, int long long <i>new_value</i>);</code> | <p>単一プロセッサ・システムでのロック・ダブルワードを検査します。</p> <p>条件付きで、ダブルワード変数をアトミックに更新します。<i>addr</i> は、ダブルワード変数のアドレスを指定します。<i>old_value</i> は、ダブルワード変数の値に照らし合わせて検査される元の値を指定します。<i>new_value</i> は、ダブルワード変数に条件付きで割り当てられる新規の値を指定します。ダブルワード変数は、ダブルワード境界で位置合わせしておかなければなりません。</p> <p>戻り値:</p> <ul style="list-style-type: none"> <li>• <code>false</code> (偽) が戻された場合は、ダブルワード変数が元の値と同じであったこと、さらにこの変数が新規の値に設定されたことを示す。</li> <li>• <code>true</code> (真) が戻された場合は、ダブルワード変数が元の値と同じでなかったこと、さらにこの変数が変更されないままになっていることを示す。</li> </ul> |
| <code>__cimag (C99)</code>    | <code>double __cimag(__complex__ double);</code>                                                                                               | 複素数の虚数部を戻します。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>__cimagf (C99)</code>   | <code>float __cimagf(__complex__ float);</code>                                                                                                | 複素数の虚数部を戻します。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>__cimagl (C99)</code>   | <code>long double __cimagl(__complex__ long double);</code>                                                                                    | 複素数の虚数部を戻します。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>__clear_lock_mp</code>  | <code>void __clear_lock_mp (const int* <i>addr</i>, int <i>value</i>);</code>                                                                  | <p>マルチプロセッサ・システムでのロックをクリアします。</p> <p>アドレス <i>addr</i> にあるシングル・ワード変数に、<i>value</i> をアトミックに保管します。ワード変数は、フルワード境界で位置合わせしておかなければなりません。</p>                                                                                                                                                                                                                                                                                                                                                                 |
| <code>__clear_lockd_mp</code> | <code>void __clear_lockd_mp (const long long int* <i>addr</i>, long long int <i>value</i>);</code>                                             | <p>マルチプロセッサ・システムでのロック・ダブルワードをクリアします。</p> <p>アドレス <i>addr</i> にあるダブルワード変数に、<i>value</i> をアトミックに保管します。ダブルワード変数は、ダブルワード境界で位置合わせしておかなければなりません。</p>                                                                                                                                                                                                                                                                                                                                                        |

| 名前                            | プロトタイプ                                                                               | 説明                                                                                                                                      |
|-------------------------------|--------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>__clear_lock_up</code>  | <code>void __clear_lock_up (const int* addr, int value);</code>                      | 単一プロセッサ・システムでのロックをクリアします。<br><br>アドレス <i>addr</i> にあるシングル・ワード変数に、 <i>value</i> をアトミックに保管します。ワード変数は、フルワード境界で位置合わせしておかなければなりません。          |
| <code>__clear_lockd_up</code> | <code>void __clear_lockd_up (const long long int* addr, long long int value);</code> | ユニプロセッサ・システムでのロック・ダブルワードをクリアします。<br><br>アドレス <i>addr</i> にあるダブルワード変数に、 <i>value</i> をアトミックに保管します。ダブルワード変数は、ダブルワード境界で位置合わせしておかなければなりません。 |
| <code>__cntlz4</code>         | <code>unsigned int __cntlz4(unsigned int);</code>                                    | 先行ゼロ・カウント。4 バイト整数。                                                                                                                      |
| <code>__cntlz8</code>         | <code>unsigned int __cntlz8(unsigned long long);</code>                              | 先行ゼロ・カウント。8 バイト整数。                                                                                                                      |
| <code>__cnttz4</code>         | <code>unsigned int __cnttz4(unsigned int);</code>                                    | 後続ゼロ・カウント。4 バイト整数。                                                                                                                      |
| <code>__cnttz8</code>         | <code>unsigned int __cnttz8(unsigned long long);</code>                              | 後続ゼロ・カウント。8 バイト整数。                                                                                                                      |
| <code>__conj (C99)</code>     | <code>__complex__ double conj(__complex__ double);</code>                            | 複素共役を生成します。                                                                                                                             |
| <code>__conjf (C99)</code>    | <code>__complex__ float conjf(__complex__ float);</code>                             | 複素共役を生成します。                                                                                                                             |
| <code>__conjl(C99)</code>     | <code>__complex__ long double conjl(__complex__ long double);</code>                 | 複素共役を生成します。                                                                                                                             |
| <code>__cos</code>            | <code>double __cos(double);</code>                                                   | コサイン値を戻します。                                                                                                                             |
| <code>__cosf (C99)</code>     | <code>float __cosf(float);</code>                                                    | コサイン値を戻します。                                                                                                                             |
| <code>__cosl (C99)</code>     | <code>long double __cosl(long double);</code>                                        | コサイン値を戻します。                                                                                                                             |
| <code>__creal (C99)</code>    | <code>double __creal(__complex__ double);</code>                                     | 複素数の実数部を戻します。                                                                                                                           |
| <code>__crealf (C99)</code>   | <code>float __crealf(__complex__ float);</code>                                      | 複素数の実数部を戻します。                                                                                                                           |
| <code>__creall (C99)</code>   | <code>long double __creall(__complex__ long double);</code>                          | 複素数の実数部を戻します。                                                                                                                           |
| <code>__dcbt( )</code>        | <code>void __dcbt (void *);</code>                                                   | データ・キャッシュ・ブロック・タッチ。<br><br>指定のアドレスを含むメモリのブロックをデータ・キャッシュ内にロードします。                                                                        |

| 名前                     | プロトタイプ                                 | 説明                                                                                                                                                                                                                            |
|------------------------|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>__dcbz( )</code> | <code>void __dcbz (void *);</code>     | <p>ゼロに設定されたデータ・キャッシュ・ブロック。</p> <p>データ・キャッシュ内の指定のアドレスをゼロ (0) に設定します。</p>                                                                                                                                                       |
| <code>__eieio</code>   | <code>void __eieio();</code>           | <p>すでに組み込まれている <code>__iospace_eieio</code> のもう 1 つの名前です。</p> <p>コンパイラーは、<code>__eieio</code> を組み込まれているものとして認識します。名前以外は、すべてが <code>__iospace_eieio</code> とまったく同じです。 <code>__eieio</code> には、対応する PowerPC の命令名との一貫性があります。</p> |
| <code>__exp</code>     | <code>double __exp(double);</code>     | 指数値を戻します。                                                                                                                                                                                                                     |
| <code>__fabs</code>    | <code>double __fabs(double);</code>    | 絶対値を戻します。                                                                                                                                                                                                                     |
| <code>__fabss</code>   | <code>float __fabss(float);</code>     | 短精度浮動小数点の絶対値を戻します。                                                                                                                                                                                                            |
| <code>__fcfid</code>   | <code>double __fcfid (double);</code>  | <p>整数ダブルワードからの浮動変換です。</p> <p>64 ビット符号付き固定小数点オペランドが、倍精度浮動小数点に変換されます。</p>                                                                                                                                                       |
| <code>__fctid</code>   | <code>double __fctid (double);</code>  | <p>整数ダブルワードへの浮動変換です。</p> <p>FPSCR<sub>RN</sub> (浮動小数点状況および制御レジスターの浮動小数点丸め制御フィールド) で指定された丸めモードを使用して、浮動小数点オペランドが、64 ビット符号付き固定小数点整数に変換されます。</p>                                                                                  |
| <code>__fctidz</code>  | <code>double __fctidz (double);</code> | <p>「ゼロの方向に丸める」による、整数ダブルワードへの浮動変換です。</p> <p>浮動小数点オペランドが、丸めモード「ゼロの方向に丸める」を使用して、64 ビット符号付き固定小数点整数に変換されます。</p>                                                                                                                    |

| 名前         | プロトタイプ                                   | 説明                                                                                                                                 |
|------------|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| __fctiw    | double __fctiw (double);                 | 整数ワードへの浮動変換です。<br><br>FPSCR <sub>RN</sub> (浮動小数点状況および制御レジスタの浮動小数点丸め制御フィールド) で指定された丸めモードを使用して、浮動小数点オペランドが、32 ビット符号付き固定小数点整数に変換されます。 |
| __fctiwz   | double __fctiwz (double);                | 「ゼロの方向に丸める」による、整数ワードへの浮動変換です。<br><br>浮動小数点オペランドが、丸めモード「ゼロの方向に丸める」を使用して、32 ビット符号付き固定小数点整数に変換されます。                                   |
| __fmadd    | double __fmadd (double, double, double); | 浮動小数点の乗算と加算                                                                                                                        |
| __fmadds   | float __fmadds (float, float, float);    | 浮動小数点の乗算と加算 short                                                                                                                  |
| __fmsub    | double __fmsub(double, double, double);  | 浮動小数点の乗算と減算                                                                                                                        |
| __fmsubs   | float __fmsubs (float, float, float);    | 浮動小数点の乗算と減算                                                                                                                        |
| __fnabs    | double __fnabs(double);                  | 負の浮動小数点の絶対値                                                                                                                        |
| __fnabss   | float __fnabss(float);                   | 負の浮動小数点の絶対値                                                                                                                        |
| __fnmadd   | double __fnmadd(double, double, double); | 負の浮動小数点の乗算と加算                                                                                                                      |
| __fnmadds  | float __fnmadds (float, float, float);   | 負の浮動小数点の乗算と加算                                                                                                                      |
| __fnmsub   | double __fnmsub(double, double, double); | 負の浮動小数点の乗算と減算                                                                                                                      |
| __fnmsubs  | float __fnmsubs (float, float, float);   | __fnmsubs (a, x, y) = [- (a * x - y)]                                                                                              |
| __fre      | double __fre (double);                   | 浮動小数点の逆数<br>__fre (x) = [(estimate of) 1.0/x]                                                                                      |
| __fres     | float __fres (float);                    | 浮動小数点の逆数<br>__fres (x) = [1.0/x (の概算)]                                                                                             |
| __frsqrt   | double __frsqrt (double);                | 浮動小数点の逆数平方根<br>__frsqrt (x) = [1.0/sqrt(x) (の概算)]                                                                                  |
| __frsqrtes | float __frsqrtes (float);                | 浮動小数点の逆数平方根<br>__frsqrtes (x) = [(estimate of) 1.0/sqrt(x)]                                                                        |
| __fsel     | double __fsel (double, double, double);  | 浮動小数点の選択<br>if (a >= 0.0) then __fsel (a, x, y) = x;<br>それ以外の場合 __fsel (a, x, y) = y                                               |
| __fsels    | float __fsels (float, float, float);     | 浮動小数点の選択<br>if (a >= 0.0) then __fsels (a, x, y) = x;<br>それ以外の場合 __fsels (a, x, y) = y                                             |

| 名前               | プロトタイプ                                           | 説明                                                                                                                                                                                                                                                                          |
|------------------|--------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| __fsqrt          | double __fsqrt (double);                         | 浮動小数点の平方根<br>__fsqrt (x) = x の平方根                                                                                                                                                                                                                                           |
| __fsqrts         | float __fsqrts (float);                          | 浮動小数点の平方根<br>__fsqrts (x) = x の平方根                                                                                                                                                                                                                                          |
| __iospace_eieio  | void __iospace_eieio(void);                      | EIEIO 命令を生成します。                                                                                                                                                                                                                                                             |
| __iospace_lwsync | (equivalent to: void<br>__iospace_lwsync(void);) | lwsync 命令を生成します。                                                                                                                                                                                                                                                            |
| __iospace_sync   | (equivalent to: void<br>__iospace_sync(void);)   | 同期命令を生成します。                                                                                                                                                                                                                                                                 |
| __isync          | void __isync(void);                              | 前のすべての命令が完了するのを待機した後、プリフェッチされたすべての命令を廃棄して、以降の命令が、前の命令によって確立されたコンテキストの中でフェッチ (または再フェッチ) および実行されるようにします。                                                                                                                                                                      |
| __load2r         | unsigned short __load2r(unsigned short*);        | 逆のハーフワード・バイトをロードします。                                                                                                                                                                                                                                                        |
| __load4r         | unsigned int __load4r(unsigned int*);            | 逆のワード・バイトをロードします。                                                                                                                                                                                                                                                           |
| __lwsync         | void __lwsync();                                 | すでに組み込まれている<br><b>__iospace_lwsync</b> のもう 1 つの名前です。<br><br>コンパイラーは、 <b>__lwsync</b> を組み込まれているものとして認識します。名前以外は、すべてが <b>__iospace_lwsync</b> とまったく同じです。 <b>__lwsync</b> には、対応する PowerPC の命令名との一貫性があります。この関数は、PowerPC 970 プロセッサによってのみサポートされます。<br><br>値は、コンパイル時に既知でなければなりません。 |
| __mfdcr          | unsigned __mfdcr(const int);                     | 装置制御レジスタの値を戻します。この命令は特権命令で、PowerPC 440 に対してのみ有効です。                                                                                                                                                                                                                          |
| __mtdcr          | void __mtdcr(const int, unsigned long);          | unsigned long 値を使用して、装置制御レジスタの値を設定します。値は、コンパイル時に既知でなければなりません。この命令は特権命令で、PowerPC 440 に対してのみ有効です。                                                                                                                                                                             |

| 名前       | プロトタイプ                                                                     | 説明                                                                                                                                                                                 |
|----------|----------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| __mfspr  | unsigned __mfspr(const int);                                               | 特殊目的レジスタの値を戻します。"const int" の値は、コンパイル時に既知でなければなりません。                                                                                                                               |
| __mtspr  | void __mtspr(const int, unsigned long);                                    | unsigned long 値を使用して、特殊目的レジスタの値を設定します。値は、コンパイル時に既知でなければなりません。                                                                                                                      |
| __mfmsr  | unsigned __mfmsr( );                                                       | マシン状態レジスタの値を戻します。この命令は特権命令で、PowerPC システムに対してのみ有効です。                                                                                                                                |
| __mtmsr  | void (unsigned long);                                                      | unsigned long 値を使用して、マシン状態レジスタの値を設定します。この命令は特権命令です。                                                                                                                                |
| __mtfsb0 | void __mtfsb0(unsigned int <i>bt</i> );                                    | FPSCR のビット 0 に移動します。<br><br>FPSCR のビット <i>bt</i> は 0 に設定されます。 <i>bt</i> は定数で、 $0 \leq bt \leq 31$ でなければなりません。                                                                      |
| __mtfsb1 | void __mtfsb1(unsigned int <i>bt</i> );                                    | FPSCR Bit 1 に移動します。<br><br>FPSCR のビット <i>bt</i> は 1 に設定されます。 <i>bt</i> は定数で、 $0 \leq bt \leq 31$ でなければなりません。                                                                       |
| __mtfsf  | void __mtfsf(unsigned int <i>flm</i> , unsigned int <i>frb</i> );          | FPSCR フィールドに移動します。<br><br><i>frb</i> の内容が、 <i>flm</i> で指定するフィールド・マスクの制御下の FPSCR に入れられます。フィールド・マスク <i>flm</i> は、対象の FPSCR の 4 ビットのフィールドを識別します。 <i>flm</i> は定数の 8 ビットのマスクでなければなりません。 |
| __mtfsfi | void __mtfsfi(unsigned int <i>bf</i> , unsigned int <i>u</i> );            | FPSCR フィールドに即時移動します。<br><br><i>u</i> の値が、 <i>bf</i> で指定する FPSCR フィールドに入れられます。 <i>bf</i> と <i>u</i> は、 $0 \leq bf \leq 7$ および $0 \leq u \leq 15$ の定数でなければなりません。                     |
| __mulhd  | long long int __mulhd(long long int <i>ra</i> , long long int <i>rb</i> ); | 高位の符号付きダブルワードの乗算です。<br><br>オペランド <i>ra</i> と <i>rb</i> の 128 ビットの積から、高位の 64 ビットを戻します。                                                                                              |



| 名前        | プロトタイプ                                                                                                       | 説明                                                                                                                                                                                                         |
|-----------|--------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| __mulhdu  | unsigned long long int<br>__mulhdu(unsigned long long int <i>ra</i> ,<br>unsigned long long int <i>rb</i> ); | 高位の符号なしダブルワードの乗算です。<br><br>オペランド <i>ra</i> と <i>rb</i> の 128 ビットの積から、高位の 64 ビットを返します。                                                                                                                      |
| __mulhw   | int __mulhw(int <i>ra</i> , int <i>rb</i> );                                                                 | 高位の符号付きワードの乗算です。<br><br>オペランド <i>ra</i> と <i>rb</i> の 64 ビットの積から、高位の 32 ビットを返します。                                                                                                                          |
| __mulhwu  | unsigned int __mulhwu(unsigned int <i>ra</i> ,<br>unsigned int <i>rb</i> );                                  | 高位の符号なしワードの乗算です。<br><br>オペランド <i>ra</i> と <i>rb</i> の 64 ビットの積から、高位の 32 ビットを返します。                                                                                                                          |
| __parthds | int __parthds(void);                                                                                         | parthds ランタイム・オプションの値を返します。<br><br>parthds オプションがユーザーによって明示的に設定されていない場合、関数は、ランタイム・ライブラリーによって設定されたデフォルト値を返します。プログラムのコンパイル時に <code>-qsmp</code> コンパイラー・オプションが指定されなかった場合、この関数は、選択されたランタイム・オプションに関係なく 1 を返します。 |
| __popcnt4 | int __popcnt4(unsigned int);                                                                                 | 32 ビット整数用に設定されたビットの数を返します。                                                                                                                                                                                 |
| __popcnt8 | int __popcnt8(unsigned long long);                                                                           | 64 ビット整数用に設定されたビットの数を返します。                                                                                                                                                                                 |
| __popcntb | unsigned long __popcntb(unsigned long);                                                                      | ソース・オペランドの各バイトに含まれる 1 ビットの数を返し、その数を結果の対応するバイトに配置します。これは、POWER5 アーキテクチャーに対してのみ有効です。                                                                                                                         |
| __poppar4 | int __poppar4(unsigned int);                                                                                 | 32 ビット整数内に奇数のビットが設定されている場合、1 を返します。それ以外の場合は、0 を返します。                                                                                                                                                       |
| __poppar8 | int __poppar8(unsigned long long);                                                                           | 64 ビット整数内に奇数のビットが設定されている場合、1 を返します。それ以外の場合は、0 を返します。                                                                                                                                                       |

| 名前                                    | プロトタイプ                                                                                               | 説明                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------------------|------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>__pow</code>                    | <code>double __pow(double, double);</code>                                                           | 指数計算の組み込み関数です。                                                                                                                                                                                                                                                                                                                                         |
| <code>__prefetch_by_load</code>       | <code>void __prefetch_by_load(const void*);</code>                                                   | 明示的ロードによりメモリー・ロケーションをタッチします。                                                                                                                                                                                                                                                                                                                           |
| <code>__prefetch_by_stream</code>     | <code>void __prefetch_by_stream(const int, const void*);</code>                                      | 明示的ストリームによりメモリー・ロケーションをタッチします。                                                                                                                                                                                                                                                                                                                         |
| <code>__protected_stream_count</code> | <code>void __protected_stream_count(unsigned int unit_cnt, unsigned int ID);</code>                  | <p><i>ID</i> という ID を持つ、限られた長さの <i>protected</i> ストリームに対して、<i>unit_cnt</i> 本のキャッシュ・ラインを設定します。<i>Unit_cnt</i> は、0 から 1023 までの値を持つ整数でなければなりません。ストリーム <i>ID</i> には、0 から 15 までの整数値がなければなりません。</p> <p>(POWER5 プロセッサのみ)</p>                                                                                                                                  |
| <code>__protected_stream_go</code>    | <code>void __protected_stream_go();</code>                                                           | <p>すべての限られた長さの <i>protected</i> ストリームの事前取り出しを開始します。</p> <p>(POWER5 プロセッサのみ)</p>                                                                                                                                                                                                                                                                        |
| <code>__protected_stream_set</code>   | <code>void __protected_stream_set(unsigned int direction, const void* addr, unsigned int ID);</code> | <p><i>ID</i> という ID を使用する限られた長さの <i>protected</i> ストリームを確立します。これは、<i>addr</i> にあるキャッシュ・ラインから始まり、その後 <i>direction</i> の値に応じて、インクリメンタル (前方) またはデクリメンタル (後方) にメモリー・アドレスから取り出します。ストリームは、ハードウェアで検出されるストリームに置き換えられないように保護されます。</p> <p><i>Direction</i> には、値 1 (前方) または 3 (後方) が必要です。ストリーム <i>ID</i> には、0 から 15 までの整数値がなければなりません。</p> <p>(POWER5 プロセッサのみ)</p> |

| 名前                                  | プロトタイプ                                                                                                                                                  | 説明                                                                                                                                                                                                                                                                                                                                                              |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| __protected_unlimited_stream_set_go | void __protected_unlimited_stream_set_go(unsigned int <i>direction</i> , const void* <i>addr</i> , unsigned int <i>ID</i> );                            | <p><i>ID</i> という ID を使用して無制限の長さの protected ストリームを確立します。これは、<i>addr</i> にあるキャッシュ・ラインから始まり、その後 <i>direction</i> の値に応じて、インクリメンタル (前方) またはデクリメンタル (後方) にメモリー・アドレスから取り出します。ストリームは、ハードウェアで検出されるストリームに置き換えられないように保護されます。</p> <p><i>Direction</i> には、値 1 (前方) または 3 (後方) が必要です。ストリーム <i>ID</i> には、0 から 15 までの整数値がなければなりません。</p> <p>(PowerPC 970 および POWER5 プロセッサのみ)</p> |
| __protected_stream_stop             | void __protected_stream_stop(unsigned int <i>ID</i> );                                                                                                  | <p>ID <i>ID</i> を持つ protected ストリームの事前取り出しを停止します。</p> <p>(POWER5 プロセッサのみ)</p>                                                                                                                                                                                                                                                                                   |
| __protected_stream_stop_all         | void __protected_stream_stop_all();                                                                                                                     | <p>すべての protected ストリームの事前取り出しを停止します。</p> <p>(POWER5 プロセッサのみ)</p>                                                                                                                                                                                                                                                                                               |
| __rdlam                             | unsigned long long __rdlam(unsigned long long <i>rs</i> , unsigned int <i>shift</i> , unsigned long long <i>mask</i> );                                 | <p>ダブルワードを左方に回転し、マスクに AND します。</p> <p><i>rs</i> の内容を、左方に <i>shift</i> ビットだけ回転します。回転したデータは、マスクに AND され、結果として戻されます。 <i>mask</i> は定数で、隣接するビット・フィールドを示していなければなりません。</p>                                                                                                                                                                                             |
| __readflm                           | double __readflm();                                                                                                                                     | 浮動小数点状況/制御レジスターを読み取ります。                                                                                                                                                                                                                                                                                                                                         |
| __rldimi                            | unsigned long long __rldimi(unsigned long long <i>rs</i> , unsigned long long <i>is</i> , unsigned int <i>shift</i> , unsigned long long <i>mask</i> ); | <p>ダブルワードを左方に即時回転してから、挿入をマスクします。</p> <p><i>rs</i> を左方に <i>shift</i> ビットだけ回転してから、<i>rs</i> をビット・マスク <i>mask</i> の下の <i>is</i> に挿入します。 <i>shift</i> は定数で、<math>0 \leq \text{shift} \leq 63</math> でなければなりません。 <i>mask</i> は定数で、隣接するビット・フィールドを示していなければなりません。</p>                                                                                                   |

| 名前                      | プロトタイプ                                                                                                               | 説明                                                                                                                                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>__rlwimi</code>   | <code>unsigned int __rlwimi(unsigned int rs,<br/>unsigned int is, unsigned int shift,<br/>unsigned int mask);</code> | ワードを左方に即時回転してから、挿入をマスクします。<br><br><i>rs</i> を左方に <i>shift</i> ビットだけ回転してから、 <i>rs</i> をビット・マスク <i>mask</i> の下の <i>is</i> に挿入します。<br><i>shift</i> は定数で、 $0 \leq \text{shift} \leq 31$ でなければなりません。 <i>mask</i> は定数で、隣接するビット・フィールドを示していなければなりません。 |
| <code>__rlwnm</code>    | <code>unsigned int __rlwnm(unsigned int rs,<br/>unsigned int shift, unsigned int mask);</code>                       | ワードを左方に回転し、マスクに AND します。<br><br><i>rs</i> を左方に <i>shift</i> ビットだけ回転し、ビット・マスク <i>mask</i> に <i>rs</i> を AND します。 <i>mask</i> は定数で、隣接するビット・フィールドを示していなければなりません。                                                                                 |
| <code>__rotatel4</code> | <code>unsigned int __rotatel4(unsigned int rs,<br/>unsigned int shift);</code>                                       | ワードを左方に回転します。<br><br><i>rs</i> を左方に <i>shift</i> ビットだけ回転します。                                                                                                                                                                                  |
| <code>__setflm</code>   | <code>double __setflm(double);</code>                                                                                | 浮動小数点状況/制御レジスターを設定します。                                                                                                                                                                                                                        |
| <code>__setrnd</code>   | <code>double __setrnd(int);</code>                                                                                   | 丸めモードを設定します。                                                                                                                                                                                                                                  |
| <code>__stfiw</code>    | <code>void __stfiw( const int* addr, double<br/>value);</code>                                                       | 浮動小数点を整数ワードとして保管します。<br><br><i>value</i> の下位 32 ビットの内容が、 <i>addr</i> でアドレッシングするストレージのワードに、変換されずに保管されます。                                                                                                                                       |
| <code>__store2r</code>  | <code>void __store2r(unsigned short, unsigned<br/>short *);</code>                                                   | 2 バイトを逆順 (逆のエンディアン・モード) で保管します。                                                                                                                                                                                                               |
| <code>__store4r</code>  | <code>void __store4r(unsigned int, unsigned int<br/>*);</code>                                                       | 4 バイトを逆順 (逆のエンディアン・モード) で保管します。                                                                                                                                                                                                               |
| <code>__swdiv</code>    | <code>double __swdiv(double numerator, double<br/>denominator);</code>                                               | <code>double</code> 型の、浮動小数点除法です。引き数の制限はありません。<br><b>-qstrict</b> が有効である場合は、結果は IEEE 除法とビット単位で同等となります。 <b>-qnostrict</b> が有効である場合は、結果は IEEE 除法の結果と若干異なる可能性があります。<br><br>この関数は、ループ内で繰り返し除法をする場合に、通常の除法演算子よりも高いパフォーマンスを得られることがあります。             |

| 名前                          | プロトタイプ                                                                   | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------------|--------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>__swdiv_nochk</code>  | <code>double __swdiv_nochk(double numerator, double denominator);</code> | <p><code>double</code> 型の、浮動小数点除法です。範囲チェックは行いません。引き数として、値が無限大である分子、および値が無限大、ゼロ、または非正規化数である分母は使用できません。<b>-qstrict</b> が有効であると、結果は IEEE の除法とビット単位で同等となります。ただし例外として、分子の数値の大きさがおよそ <math>2^{-1000}</math> よりも小さい値である場合は、結果が IEEE の除法の結果と若干異なることがあります。<b>-qnostrict</b> が有効である場合は、結果は IEEE 除法の結果と若干異なる可能性があります。</p> <p>この関数は、ループ内で繰り返し除法をする場合に、通常の除法演算子や <code>__swdiv</code> 組み込み関数よりも高いパフォーマンスを得られることがあり、引き数が許容範囲内にあると分かっている場合に使用できます。</p> |
| <code>__swdivs</code>       | <code>float __swdivs(float numerator, float denominator);</code>         | <p><code>float</code> 型の、浮動小数点除法です。引き数の制限はありません。結果は IEEE 除法とビット単位で同等となります。</p> <p>この関数は、ループ内で繰り返し除法をする場合に、通常の除法演算子よりも高いパフォーマンスを得られることがあります。</p>                                                                                                                                                                                                                                                                                         |
| <code>__swdivs_nochk</code> | <code>float __swdivs_nochk(float numerator, float denominator);</code>   | <p><code>float</code> 型の、浮動小数点除法です。範囲チェックは行いません。引き数として、値が無限大である分子、および値が無限大、ゼロ、または非正規化数である分母は使用できません。結果は IEEE 除法とビット単位で同等となります。</p> <p>この関数は、ループ内で繰り返し除法をする場合に、通常の除法演算子や <code>__swdivs</code> 組み込み関数よりも高いパフォーマンスを得られることがあり、引き数が許容範囲内にあると分かっている場合に使用できます。</p>                                                                                                                                                                            |

| 名前      | プロトタイプ                                                 | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------|--------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| __sync  | void __sync();                                         | <p>すでに組み込まれている<br/>__iospace_sync のもう 1 つの名前です。</p> <p>コンパイラーは、__sync を組み込まれているものとして認識します。名前以外は、すべてが__iospace_sync とまったく同じです。__sync には、対応する PowerPC の命令名との一貫性があります。</p>                                                                                                                                                                                                                                                                                                                                                                    |
| __tdw   | void __tdw(long long a, long long b, unsigned int TO); | <p>ダブルワードをトラップします。</p> <p>オペランド <i>a</i> が、オペランド <i>b</i> と比較されます。この比較の結果、5 ビットの定数 <i>TO</i> (0 から 31 の範囲内の値を含む) に AND される条件は 5 つです。</p> <p>結果が 0 でない場合、システム・トラップ・ハンドラーが呼び出されます。各ビットの位置 (設定されていれば) は、以下の起こりうる条件を 1 つ以上示します。</p> <p><b>0 (高位ビット)</b><br/> <i>a</i> は <i>b</i> より小 (符号付きの比較を使用)。</p> <p><b>1</b>      <i>a</i> は <i>b</i> より大 (符号付きの比較を使用)。</p> <p><b>2</b>      <i>a</i> は <i>b</i> と等しい。</p> <p><b>3</b>      <i>a</i> は <i>b</i> より小 (符号なしの比較を使用)。</p> <p><b>4 (下位ビット)</b><br/> <i>a</i> は <i>b</i> より大 (符号なしの比較を使用)。</p> |
| __trap  | void __trap(int);                                      | トラップ                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| __trapd | void __trapd (longlong);                               | パラメーターがゼロでない場合にトラップします。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

| 名前        | プロトタイプ                                                            | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------|-------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| __tw      | void __tw(int <i>a</i> , int <i>b</i> , unsigned int <i>TO</i> ); | <p>ワードをトラップします。</p> <p>オペランド <b><i>a</i></b> が、オペランド <b><i>b</i></b> と比較されます。この比較の結果、5 ビットの定数 <b><i>TO</i></b> (<b>0</b> から <b>31</b> の範囲内の値を含む) に AND される条件は 5 つです。</p> <p>結果が <b>0</b> でない場合、システム・トラップ・ハンドラーが呼び出されます。各ビットの位置 (設定されていれば) は、以下の起こりうる条件を 1 つ以上示します。</p> <p><b>0 (高位ビット)</b><br/> <i>a</i> は <i>b</i> より小 (符号付きの比較を使用)。</p> <p><b>1</b>      <i>a</i> は <i>b</i> より大 (符号付きの比較を使用)。</p> <p><b>2</b>      <i>a</i> は <i>b</i> と等しい。</p> <p><b>3</b>      <i>a</i> は <i>b</i> より小 (符号なしの比較を使用)。</p> <p><b>4 (下位ビット)</b><br/> <i>a</i> は <i>b</i> より大 (符号なしの比較を使用)。</p> |
| __usrthds | int __usrthds(void);                                              | <p><b>usrthds</b> ランタイム・オプションの値を戻します。</p> <p><b>usrthds</b> オプションがユーザーによって明示的に設定されていないか、プログラムのコンパイル時に <b>-qsmp</b> コンパイラー・オプションが指定されなかった場合、この関数は、選択されたランタイム・オプションに関係なく <b>0</b> を戻します。</p>                                                                                                                                                                                                                                                                                                                                                                                        |





---

## 付録 C. XL C/C++ Enterprise Edition における各国語サポート

当ページおよび関連ページでは、IBM XL C/C++ Enterprise Edition に特定の各国語サポート (NLS) を要約します。

詳しくは、本節の下記のトピックを参照してください。

- 『マルチバイト・データを含むファイルの新規コード・ページへの変換』
- 『マルチバイト文字サポート』

「General Programming Concepts: Writing and Debugging Programs」の『National Language Support』も参照してください。

---

### マルチバイト・データを含むファイルの新規コード・ページへの変換

ご使用のシステムで新規コード・ページをすでにインストール済みであれば、AIX **iconv** マイグレーション・ユーティリティーを使用して、マルチバイト・データを含むファイルを新規コード・ページを使うように変換できます。このコマンドは、マルチバイト・データを含むファイルを、**IBM-932** コード・セットから **IBM-euc** コード・セットに変換します。

**iconv** コマンドの説明は「AIX コマンド・リファレンス」にあります。**iconv** コマンドと一緒に NLS コード・セット・コンバーターを使用する方法は、「AIX *General Programming Concepts*」の『Converters Overview for Programming』に説明があります。

#### 関連資料

『付録 C. XL C/C++ Enterprise Edition における各国語サポート』  
『マルチバイト文字サポート』

以下も参照してください。

「コマンド・リファレンス 第 3 巻」の **iconv** コマンド

「AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs」の『Converters Overview for Programming』セクション

---

### マルチバイト文字サポート

マルチバイト文字のサポートは、ワイド文字のサポートも含みます。一般的に、ワイド文字はマルチバイト文字のあるところであればどこでも許可されますが、ワイド文字とマルチバイト文字は、同じストリング内では互換性がありません。これは、ビット・パターンが異なっているためです。許可されるところであればどこでも、単一バイト文字とマルチバイト文字を同じストリングで混在させることができます。

**注:** プログラムの任意の場所でマルチバイト文字を使用するには、**-qmbcs** オプションを指定しなければなりません。

以下の例では、`multibyte_char` は 1 つまたは複数のマルチバイト文字のストリングを表します。

## ストリング・リテラルと文字定数

マルチバイト文字は、ストリング・リテラルと文字定数でサポートされます。マルチバイト文字を含むストリングは、基本的にはマルチバイト文字の入っていないストリングと同様に扱われます。マルチバイト文字は、以下のコンテキストで指定することができます。

- プリプロセッサ・ディレクティブ
- マクロ定義
- `#` および `##` 演算子
- **-D** コンパイラー・オプションにおけるマクロ名の定義

ワイド文字ストリングは、1 バイト文字ストリングと同様に扱うことができます。システムは、等価のワイド文字ストリング関数と単一バイト・ストリング関数を提供します。

**cc** およびその派生物を除く、すべてのコンパイラー呼び出しについて、ストリング・リテラルのデフォルト・ストレージ・タイプは読み取り専用です。**-qro** オプションは、ストリング・リテラルのストレージ・タイプを読み取り専用に設定し、**-qnoro** オプションは、ストリング・リテラルを書き込み可能にします。

**注：** 文字定数が保管できるのは 1 バイトだけであるため、マルチバイト文字を文字定数に割り当てないようにしてください。マルチバイト文字定数の最終バイトだけが保管されます。その代わりに、ワイド文字表示を使用してください。ワイド文字のストリング・リテラルおよび定数には、`L` のプレフィックスを付けなければなりません。例えば、次のとおりです。

```
wchar_t *a = L"wide_char_string";
wchar_t b = L'c';
```

## プリプロセッサ・ディレクティブ

以下のプリプロセッサ・ディレクティブは、マルチバイト文字定数およびストリング・リテラルを許可します。

- **#define**
- **#pragma comment**
- **#include**

## マクロ定義

ストリング・リテラルと文字定数は **#define** ステートメントの一部となることがあるため、マルチバイト文字も、オブジェクト類似および関数類似の両方のマクロ定義で許可されます。

## コンパイラー・オプション

マルチバイト文字は、引き数としてファイル名を扱うコンパイラー・サブオプションで指定することができます。

- **-B prefix**

- **-F** *config\_file:stanza*
- **-I** *directory*
- **-L** *directory*
- **-I** *key*
- **-o** *file\_name*
- **-qexpfile=***file\_name*
- **-qipa=***file\_name*
- **-qtempinc=***directory*
- **-qtemplateregistry=***registry\_file\_name*
- **-qpath** *path\_name*

**-Dname=definition** オプションは、マクロ名の定義でマルチバイト文字を許可します。次の例では、最初の定義がストリング・リテラルで、2 番目の定義が文字定数です。

```
-DMYMACRO="kpsmultibyte_chardcs"
-DMYMACRO='multibyte_char'
```

**-qmbcs** コンパイラー・オプションは、2 バイト文字とマルチバイト文字の両方を許可します。このオプションは、他の点では **-qdbcs** オプションと等価ですが、マルチバイト文字がプログラムにあるときには、このオプションを使用してください。

**-qlist** および **-qsource** オプションで作成されたリストは、該当する国際言語の日時を表示します。C または C++ ソース・ファイルのファイル名の中のマルチバイト文字もまた、対応するリスト・ファイルの名前に表示されます。例えば、以下の名前の C ソース・ファイルがあるとして。

```
multibyte_char.c
```

この C ソース・ファイルは、以下の名前のリスト・ファイルを作成します。

```
multibyte_char.lst
```

## ファイル名とコメント

どのファイル名にもマルチバイト文字を含めることができます。ファイル名は、相対または絶対パス名にすることができます。例を以下に示します。

```
#include<multibyte_char/mydir/mysource/multibyte_char.h>
#include "multibyte_char.h"

xlc /u/myhome/c_programs/kanji_files/multibyte_char.c -omultibyte_char
```

**-qmbcs** コンパイラー・オプションを指定した場合、マルチバイト文字はコメントでも許可されます。

## 制約事項

- マルチバイト文字は、ID では許可されません。
- マルチバイト文字用の 16 進値は、使用中のコード・ページの範囲内でなければなりません。

- マクロ定義で、ワイド文字とマルチバイト文字を混在させることはできません。  
例えば、ワイド・ストリングとマルチバイト・ストリングを連結するマクロ展開は許可されません。
- ワイド文字とマルチバイト文字間の割り当ては許可されません。
- ワイド文字ストリングとマルチバイト文字ストリングの連結は許可されません。

## 関連資料

493 ページの『付録 C. XL C/C++ Enterprise Edition における各国語サポート』

493 ページの『マルチバイト・データを含むファイルの新規コード・ページへの変換』

71 ページの『+ (正符号)』

238 ページの『mbcs、dbcs』

---

## 付録 D. 問題解決

本節には以下のトピックがあります。

- 『メッセージ・カタログ・エラー』
- 498 ページの『コンパイル中のページ・スペース・エラーの訂正』

---

### メッセージ・カタログ・エラー

コンパイラーでユーザー・プログラムをコンパイルする前に、あらかじめメッセージ・カタログをインストールし、メッセージ・カタログのインストール言語に環境変数の **LANG** と **NLSPATH** を設定しておかなければなりません。

以下のメッセージをコンパイルの途中で検出した場合、該当するメッセージ・カタログをオープンすることはできません。

```
Error occurred while initializing the message system in
file: message_file
```

ここで、*message\_file* は、コンパイラーが開くことのできないメッセージ・カタログ名です。このメッセージは英語でのみ発行されます。

それから、メッセージ・カタログと環境変数が、適切な場所にあり、正しいことを検証してください。メッセージ・カタログまたは環境変数が正しくない場合、コンパイルは継続できますが、診断メッセージは抑止され、代わりに以下のメッセージが発行されます。

```
No message text for message_number.
```

ここで、*message\_number* は、IBM XL C/C++ Enterprise Edition の内部メッセージ番号です。このメッセージは英語でのみ発行されます。

ご使用のシステムにインストールされているメッセージ・カタログを判別するには、コンパイラーをデフォルトのインストール・ロケーションにインストールしたと想定して、以下のコマンドを使用して、カタログ用のファイル名すべてをリストすることができます。

```
ls /usr/lib/nls/msg/%L/*.cat
```

ここで、**%L** は、現行の 1 次言語環境 (ロケール) 設定です。デフォルト・ロケールは、**C** です。米国英語の場合、ロケールは **en\_US** です。

**/usr/vacpp/exe/default\_msg** のデフォルトのメッセージ・カタログは、

- コンパイラーが **%L** で指定されたロケール用のメッセージ・カタログを検出できないとき。
- ロケールがデフォルトの **C** から変更されたことがないとき。

**NLSPATH** および **LANG** 環境変数についての詳細は、ご使用のオペレーティング・システムの資料を参照してください。

## 関連タスク

27 ページの『環境変数の設定』

29 ページの『他の環境変数の設定』

---

## コンパイル中のページ・スペース・エラーの訂正

コンパイル中にオペレーティング・システムのページ・スペースが不足すると、コンパイラーは以下のメッセージの 1 つを発行します。

```
1501-229 Compilation ended due to lack of space.
1501-224 fatal error in ../exe/xlCcode: signal 9 received.
```

ページ・スペースの不足が原因で、ほかのコンパイラー・プログラムが失敗すると、以下のメッセージが表示されます。

強制終了されました。

ページ・スペースの問題を最小限に抑えるには、以下のいずれかを行って、プログラムを再コンパイルしてください。

- プログラムを複数のソース・ファイルに分割して、プログラム・サイズを減らす。
- 最適化を使用しないでプログラムをコンパイルする。
- システム・ページ・スペースを競合するプロセスの数を減らす。
- システム・ページ・スペースを増やす。

現行のページ・スペース設定を検査するには、**lsps -a** コマンドを入力するか、AIX システム管理インターフェース・ツール (SMIT) コマンド **smit pgsp** を使用します。

ページング・スペースおよびその割り振り方法についての詳細は、ご使用のオペレーティング・システムの資料を参照してください。

## 付録 E. ASCII 文字セット

XL C/C++ Enterprise Edition は、情報交換用米国標準コード (ASCII) 文字セットを使用します。

以下の表は、標準 ASCII 文字を昇順数値順序で、対応する 10 進値、8 進値、および 16 進値とともにリストしたものです。また、制御文字も **Ctrl-** 表記とともに示しています。例えば、復帰 (ASCII シンボルの **CR**) は **Ctrl-M** として示され、**Ctrl** キーと **M** キーを同時に押すことによって入力できます。

| 10 進値 | 8 進値 | 16 進値 | 制御文字   | ASCII<br>シンボル | 意味       |
|-------|------|-------|--------|---------------|----------|
| 0     | 0    | 00    | Ctrl-@ | NUL           | ヌル       |
| 1     | 1    | 01    | Ctrl-A | SOH           | ヘッディング開始 |
| 2     | 2    | 02    | Ctrl-B | STX           | テキスト開始   |
| 3     | 3    | 03    | Ctrl-C | ETX           | テキスト終結   |
| 4     | 4    | 04    | Ctrl-D | EOT           | 伝送終了     |
| 5     | 5    | 05    | Ctrl-E | ENQ           | 照会       |
| 6     | 6    | 06    | Ctrl-F | ACK           | 確認       |
| 7     | 7    | 07    | Ctrl-G | BEL           | ベル       |
| 8     | 10   | 08    | Ctrl-H | BS            | バックスペース  |
| 9     | 11   | 09    | Ctrl-I | HT            | 水平タブ     |
| 10    | 12   | 0A    | Ctrl-J | LF            | 改行       |
| 11    | 13   | 0B    | Ctrl-K | VT            | 垂直タブ     |
| 12    | 14   | 0C    | Ctrl-L | FF            | 改ページ     |
| 13    | 15   | 0D    | Ctrl-M | CR            | 復帰       |
| 14    | 16   | 0E    | Ctrl-N | SO            | シフトアウト   |
| 15    | 17   | 0F    | Ctrl-O | SI            | シフトイン    |
| 16    | 20   | 10    | Ctrl-P | DLE           | 伝送制御拡張   |
| 17    | 21   | 11    | Ctrl-Q | DC1           | 装置制御 1   |
| 18    | 22   | 12    | Ctrl-R | DC2           | 装置制御 2   |
| 19    | 23   | 13    | Ctrl-S | DC3           | 装置制御 3   |
| 20    | 24   | 14    | Ctrl-T | DC4           | 装置制御 4   |
| 21    | 25   | 15    | Ctrl-U | NAK           | 否定応答     |
| 22    | 26   | 16    | Ctrl-V | SYN           | 同期信号     |
| 23    | 27   | 17    | Ctrl-W | ETB           | 伝送ブロック終結 |
| 24    | 30   | 18    | Ctrl-X | CAN           | 取り消し     |
| 25    | 31   | 19    | Ctrl-Y | EM            | メディア終端   |
| 26    | 32   | 1A    | Ctrl-Z | SUB           | 置換       |
| 27    | 33   | 1B    | Ctrl-[ | ESC           | エスケープ    |
| 28    | 34   | 1C    | Ctrl-¥ | FS            | ファイル分離   |

| 10 進値 | 8 進値 | 16 進値 | 制御文字   | ASCII<br>シンボル | 意味         |
|-------|------|-------|--------|---------------|------------|
| 29    | 35   | 1D    | Ctrl-] | GS            | グループ分離     |
| 30    | 36   | 1E    | Ctrl-^ | RS            | レコード分離     |
| 31    | 37   | 1F    | Ctrl-_ | US            | ユニット分離     |
| 32    | 40   | 20    |        | SP            | ディジット選択    |
| 33    | 41   | 21    |        | !             | 感嘆符        |
| 34    | 42   | 22    |        | “             | 二重引用符      |
| 35    | 43   | 23    |        | #             | ポンド記号、番号記号 |
| 36    | 44   | 24    |        | \$            | ドル記号       |
| 37    | 45   | 25    |        | %             | パーセント記号    |
| 38    | 46   | 26    |        | &             | アンパーサンド    |
| 39    | 47   | 27    |        | ,             | アポストロフィ    |
| 40    | 50   | 28    |        | (             | 左括弧        |
| 41    | 51   | 29    |        | )             | 右括弧        |
| 42    | 52   | 2A    |        | *             | アスタリスク     |
| 43    | 53   | 2B    |        | +             | 加算記号       |
| 44    | 54   | 2C    |        | ,             | コンマ        |
| 45    | 55   | 2D    |        | -             | 減算記号       |
| 46    | 56   | 2E    |        | .             | ピリオド       |
| 47    | 57   | 2F    |        | /             | 右スラッシュ     |
| 48    | 60   | 30    |        | 0             |            |
| 49    | 61   | 31    |        | 1             |            |
| 50    | 62   | 32    |        | 2             |            |
| 51    | 63   | 33    |        | 3             |            |
| 52    | 64   | 34    |        | 4             |            |
| 53    | 65   | 35    |        | 5             |            |
| 54    | 66   | 36    |        | 6             |            |
| 55    | 67   | 37    |        | 7             |            |
| 56    | 70   | 38    |        | 8             |            |
| 57    | 71   | 39    |        | 9             |            |
| 58    | 72   | 3A    |        | :             | コロン        |
| 59    | 73   | 3B    |        | ;             | セミコロン      |
| 60    | 74   | 3C    |        | <             | より小        |
| 61    | 75   | 3D    |        | =             | 等号         |
| 62    | 76   | 3E    |        | >             | より大        |
| 63    | 77   | 3F    |        | ?             | 疑問符        |
| 64    | 100  | 40    |        | @             | アットマーク     |
| 65    | 101  | 41    |        | A             |            |
| 66    | 102  | 42    |        | B             |            |
| 67    | 103  | 43    |        | C             |            |
| 68    | 104  | 44    |        | D             |            |



| 10 進値 | 8 進値 | 16 進値 | 制御文字 | ASCII<br>シンボル | 意味                                          |
|-------|------|-------|------|---------------|---------------------------------------------|
| 69    | 105  | 45    |      | E             |                                             |
| 70    | 106  | 46    |      | F             |                                             |
| 71    | 107  | 47    |      | G             |                                             |
| 72    | 110  | 48    |      | H             |                                             |
| 73    | 111  | 49    |      | I             |                                             |
| 74    | 112  | 4A    |      | J             |                                             |
| 75    | 113  | 4B    |      | K             |                                             |
| 76    | 114  | 4C    |      | L             |                                             |
| 77    | 115  | 4D    |      | M             |                                             |
| 78    | 116  | 4E    |      | N             |                                             |
| 79    | 117  | 4F    |      | O             |                                             |
| 80    | 120  | 50    |      | P             |                                             |
| 81    | 121  | 51    |      | Q             |                                             |
| 82    | 122  | 52    |      | R             |                                             |
| 83    | 123  | 53    |      | S             |                                             |
| 84    | 124  | 54    |      | T             |                                             |
| 85    | 125  | 55    |      | U             |                                             |
| 86    | 126  | 56    |      | V             |                                             |
| 87    | 127  | 57    |      | W             |                                             |
| 88    | 130  | 58    |      | X             |                                             |
| 89    | 131  | 59    |      | Y             |                                             |
| 90    | 132  | 5A    |      | Z             |                                             |
| 91    | 133  | 5B    |      | [             | 左大括弧                                        |
| 92    | 134  | 5C    |      | ¥             | 円記号                                         |
| 93    | 135  | 5D    |      | ]             | 右大括弧                                        |
| 94    | 136  | 5E    |      | ^             | ハット、曲折アクセント記号、挿入記号 (hat, circumflex, caret) |
| 95    | 137  | 5F    |      | _             | 下線                                          |
| 96    | 140  | 60    |      | `             | 抑音符号                                        |
| 97    | 141  | 61    |      | a             |                                             |
| 98    | 142  | 62    |      | b             |                                             |
| 99    | 143  | 63    |      | c             |                                             |
| 100   | 144  | 64    |      | d             |                                             |
| 101   | 145  | 65    |      | e             |                                             |
| 102   | 146  | 66    |      | f             |                                             |
| 103   | 147  | 67    |      | g             |                                             |
| 104   | 150  | 68    |      | h             |                                             |
| 105   | 151  | 69    |      | i             |                                             |
| 106   | 152  | 6A    |      | j             |                                             |

| 10 進値 | 8 進値 | 16 進値 | 制御文字 | ASCII<br>シンボル | 意味         |
|-------|------|-------|------|---------------|------------|
| 107   | 153  | 6B    |      | k             |            |
| 108   | 154  | 6C    |      | l             |            |
| 109   | 155  | 6D    |      | m             |            |
| 110   | 156  | 6E    |      | n             |            |
| 111   | 157  | 6F    |      | o             |            |
| 112   | 160  | 70    |      | p             |            |
| 113   | 161  | 71    |      | q             |            |
| 114   | 162  | 72    |      | r             |            |
| 115   | 163  | 73    |      | s             |            |
| 116   | 164  | 74    |      | t             |            |
| 117   | 165  | 75    |      | u             |            |
| 118   | 166  | 76    |      | v             |            |
| 119   | 167  | 77    |      | w             |            |
| 120   | 170  | 78    |      | x             |            |
| 121   | 171  | 79    |      | y             |            |
| 122   | 172  | 7A    |      | z             |            |
| 123   | 173  | 7B    |      | {             | 左中括弧       |
| 124   | 174  | 7C    |      |               | 論理 OR、垂直バー |
| 125   | 175  | 7D    |      | }             | 右中括弧       |
| 126   | 176  | 7E    |      | ~             | 同様、波形記号    |
| 127   | 177  | 7F    |      | DEL           | 削除         |

---

## 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032  
東京都港区六本木 3-2-31  
IBM World Trade Asia Corporation  
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

Lab Director  
IBM Canada Ltd. Laboratory  
B3/KB7/8200/MKM  
8200 Warden Avenue  
Markham, Ontario L6G 1C7  
Canada

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確認できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者をお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

#### 著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. 1998, 2002. All rights reserved.

---

## プログラミング・インターフェース情報

プログラミング・インターフェース情報は、プログラムを使用してアプリケーション・ソフトウェアを作成する際に役立ちます。

一般使用プログラミング・インターフェースにより、お客様はこのプログラム・ツール・サービスを含むアプリケーション・ソフトウェアを書くことができます。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

**警告:** 診断、修正、調整情報は、変更される場合がありますので、プログラミング・インターフェースとしては使用しないでください。

---

## 商標

以下は、IBM Corporation の商標です。

|         |                |
|---------|----------------|
| AIX     | POWER3         |
| AIX 5L  | POWER4         |
| IBM     | POWER5         |
| IBM(ロゴ) | PowerPCRS/6000 |
| OS/2    | VisualAge      |
| POWER   |                |
| POWER2  |                |

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

---

## 業界標準

以下の規格をサポートしています。

- C 言語は、International Standard for Information Systems-Programming Language C (ISO/IEC 9899-1999 (E)) に準拠しています。
- C++ 言語は、International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:1998) に準拠しています。
- C++ 言語は、International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:2002) に準拠しています。
- C および C++ 言語は、OpenMP C and C++ Application Programming Interface バージョン 2.0 に準拠しています。







Printed in Japan

SC88-9955-00



日本アイ・ビー・エム株式会社  
〒106-8711 東京都港区六本木3-2-12