

日立マイクロコンピュータ開発環境システム
SuperH RISC engine
C/C++コンパイラ
ユーザーズマニュアル

HS0700CLCS5S

発行年月日	平成 10 年 9 月 第 1 版
発行	株式会社 日立製作所 半導体事業本部 統括営業本部
編集	株式会社 超 L メディア 技術ドキュメントグループ

©株式会社 日立製作所 1998

はじめに

本マニュアルは、**SuperH RISC engine C/C++コンパイラ**(以下、コンパイラと称します)の使用方法を述べたものです。

本コンパイラをご使用になる前に本マニュアルを良く読んで理解してください。

本 C/C++コンパイラは、C/C++言語で記述したソースプログラムを **Super H RISC engine** ファミリ(SH-1、SH-2、SH-2E、SH-3、SH-3E、SH-4、以下 **SuperH マイコン**と称します)のリロケータブルオブジェクトプログラムまたはアセンブリソースプログラムに変換するソフトウェアシステムです。

本コンパイラシステムの特長

- (1) 機器組み込み用として **ROM** 化可能なオブジェクトプログラムを生成します。
- (2) オブジェクトプログラムの実行速度向上やサイズ縮小のための最適化機能をサポートしています。
- (3) プログラム記述言語として、**C** 言語、**C++**言語をサポートしています。
- (4) **C** 言語記述プログラムにおいては、**C** ソースレベルデバッガによる **C** ソース解析を行うためのデバッグ情報出力オプションをサポートしています。また、**C++**言語記述プログラムにおいては、**C++**ソースレベルデバッガ、ブラウザによる **C++**ソース解析を行うためのデバッグ情報、ブラウザ情報オプションをサポートしています。
- (5) アセンブリソースプログラムまたはリロケータブルオブジェクトプログラムを選択して出力することができます。
- (6) モジュール間にまたがった最適化を行うツールに必要な付加情報を出力するモジュール間最適化情報出力オプションをサポートしています。

本マニュアルは本文 8 章と付録で構成されています。各章の内容を以下に示します。

第 1 章 概要・操作

概要では、コンパイラの機能、プログラムの開発手順について説明します。操作では、コンパイラの起動方法、オプション機能の指定方法、コンパイルリストの見方について説明します。

第2章 C/C++プログラミング

C/C++プログラミングでは、コンパイラの限界値、オブジェクトプログラムの実行方式などプログラム開発時に考慮すべき事項について説明します。

第3章 システム組み込み

システム組み込みでは、本コンパイラの生成したオブジェクトプログラムをシステムに組み込むために必要なメモリ割り付け方法、ROM化の方法について説明します。また、標準入出力ライブラリ、メモリ管理ライブラリを使用する場合にユーザが作成しなければならない低水準インタフェースルーチンの仕様について説明します。

第4章 エラーメッセージ

コンパイル時に発生するエラーメッセージとエラー内容、実行時に発生するCライブラリ関数のエラーメッセージとエラー内容を説明します。

第5章 モジュール間最適化ツール

コンパイラ出力オブジェクトプログラムをリンク時にモジュール間にまたがって最適化するモジュール間最適化ツールの機能、使用方法、およびオブジェクトフォーマットの変換機能について説明します

第6章 モジュール間最適化ツールエラーメッセージ

モジュール間最適化ツール実行時に発生するエラーメッセージとエラー内容を説明します。

第7章 標準ライブラリ

C/C++プログラムから呼び出すことが可能な標準ライブラリの仕様について説明しています。なお、本コンパイラでサポートしていない標準ライブラリを、「付録 A.2(9) サポートしていないライブラリ」に挙げていますので、必ず参照してください。

第8章 DSP ライブラリ

高速フーリエ変換、デジタルフィルタ処理などを行なう DSP (デジタル信号処理) ライブラリの仕様について説明しています。

本マニュアルは UNIX^{*1}または、PC-9801^{*2}シリーズ、IBM PC^{*3}及びその互換機上で動作する Microsoft®Windows®95operating system^{*4} , Microsoft®WindowsNT®operating system^{*4}に対応するように書かれています。UNIX 上で動作するコンパイラを以下 UNIX 版と称します。PC-9801^{*2}シリーズ、IBM PC^{*3}及びその互換機上で動作するコンパイラを以下 PC 版と称します。

表記上の注意事項

本マニュアルでコマンド等の指定方法の説明で用いる記号を以下に示します。

この記号で囲まれた内容を指定することを示します。

[] 省略してもよい項目を示します。

. . . 直前の項目を 1 回以上指定することを示します。

1 個以上の空白を示します。

(RET) キャリッジリターンキー（リターンキーともいいます）を示します。

| | で区切られた項目を選択できることを示します。

(CNTL) 次の文字を、コントロールキーを押しながら入力することを示します。

*1 UNIX は、X/Open カンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標です。

*2 PC-9801 は、日本電気株式会社の商標です。

*3 IBM PC は、米国 International Business Machines Corporation の登録商標です。

*4 Microsoft ® Windows®95 operating system, Microsoft® WindowsNT® operating system は、米国 Microsoft Corporation の米国及びその他の国における登録商標です。

コンパイラバージョンアップにおける注意事項

コンパイラをバージョンアップしてプログラム開発される場合、プログラムの動作が変わることがあります。プログラムを作成する際は、以下の点に注意して、お客さまのプログラムを十分にテストしてください

(1)プログラム実行時間やタイミングに依存するプログラム

C/C++言語仕様は、プログラムの実行時間については何も規定していません。したがってコンパイラのバージョンの違いによりプログラムの実行時間と I/O 等周辺機器のタイミングのずれ、あるいは割り込み処理等非同期処理の時間の差等により、プログラムの動作が変わる場合があります。

(2)一つの式に2個以上の副作用が含まれているプログラム

一つの式に2個以上の副作用が含まれている場合、コンパイラのバージョンによって、動作が変わる可能性があります。

```
例: a[i++] = b[i++];    /* i のインクリメントの順序は不定です。 */  
     f(i++, i++);      /* インクリメントの順序でパラメタの値が変わります。 */  
                        /* i の値が 3 の時 f(3,4)または f(4,3)になります。 */
```

(3)結果がオーバーフローや不当演算に依存するプログラム

オーバーフローが生じた場合や、不当演算を実施した場合、結果の値は保証しません。したがって、コンパイラのバージョンによって動作が変わる可能性があります。

```
例: int a, b;  
     x = (a * b) / 10;  
     /* a と b の値の範囲によっては、オーバーフローする可能性があります。 */
```

(4)変数の初期化抜け、型の不一致

変数が初期化されていない場合や、パラメタやリターン値の型が呼び出し側と呼び出される側で対応していない場合、不正な値をアクセスすることになります。したがって、コンパイラのバージョンによって動作が変わる可能性があります。

例:file 1:

```
int f(double d){}
```

file 2:

```
f(1);    /*関数呼び出し側のパラメタは int 型ですが、関数定義側のパラメタは、double 型のため、値を正しく参照できません。*/
```

上記に記載された情報が全ての起こりうる状況を示したわけではありません。したがって、お客様の責任で本コンパイラを正しくご使用の上、お客様のプログラムを十分にテストしてください。

目次

第 1 章 概要・操作

1.1	概要	3
1.2	プログラムの開発手順	4
1.3	コンパイラの実行	5
1.3.1	コンパイラの起動方法	5
1.3.2	ファイル名の付け方	7
1.3.3	コンパイラオプション	7
1.3.4	オプションの組み合わせ	27
1.3.5	標準ライブラリとの対応	28
1.3.6	コンパイルリストの見方	31
1.3.7	コンパイラの環境変数	37
1.3.8	オプションによる暗黙の宣言	39

第 2 章 C/C++プログラミング

2.1	コンパイラの限界値	43
2.2	C/C++プログラムの実行方式	45
2.2.1	オブジェクトプログラムの構造	46
2.2.2	データの内部表現	49
2.2.3	C プログラムとの結合	62
2.2.4	アセンブリプログラムとの結合	62
2.3	拡張機能	73
2.3.1	割り込み関数	73
2.3.2	組み込み関数	77
2.3.3	セクション切り替え機能	90
2.3.4	単精度浮動小数点ライブラリ	91
2.3.5	文字列内の日本語記述	93
2.3.6	関数のインライン展開	93

2.3.7	アセンブラ埋め込みインライン展開	94
2.3.8	2 バイトアドレス変数の指定	96
2.3.9	GBR ベース変数の指定	97
2.3.10	レジスタ退避・回復の制御	98
2.3.11	グローバル変数のレジスタ割り付け	99
2.3.12	構造体/クラスメンバの境界調整	100
2.4	プログラム作成上の注意事項	102
2.4.1	コーディング上の注意事項	102
2.4.2	プログラム開発上のトラブル対処方法	105

第3章 システム組み込み

3.1	システム組み込みの概要	109
3.2	メモリ領域の割り付け	110
3.2.1	静的領域の割り付け	110
3.2.2	動的領域の割り付け	114
3.3	実行環境の設定	118
3.4	ライブラリ関数の実行環境の設定	124

第4章 エラーメッセージ

4.1	コンパイラのエラーメッセージ	143
4.1.1	エラーメッセージ一覧	143
4.2	標準ライブラリのエラーメッセージ	179

第5章 モジュール間最適化ツール

5.1	モジュール間最適化ツール概要	185
5.2	最適化ツールの起動方法	186
5.3	オプション/サブコマンド	189
5.3.1	最適化内容指定	190
5.3.2	最適化抑止指定	193
5.3.3	オブジェクトフォーマット指定	194
5.3.4	最適化情報	195
5.3.5	サブコマンドファイル	195

第 6 章 モジュール間最適化時のエラーメッセージ

6.1	モジュール間最適化ツールのエラーメッセージ.....	199
6.1.1	エラーメッセージ一覧.....	199

第 7 章 標準ライブラリ

7.1	ライブラリの概要	213
7.2	<stddef.h>.....	222
7.3	<assert.h>	223
7.3.1	assert マクロ	224
7.4	<ctype.h>	225
7.4.1	isalnum 関数	227
7.4.2	isalpha 関数	228
7.4.3	isctrl 関数.....	229
7.4.4	isdigit 関数	230
7.4.5	isgraph 関数.....	231
7.4.6	islower 関数	232
7.4.7	isprint 関数	233
7.4.8	ispunct 関数	234
7.4.9	isspace 関数.....	235
7.4.10	isupper 関数.....	236
7.4.11	isxdigit 関数	237
7.4.12	tolower 関数.....	238
7.4.13	toupper 関数	239
7.5	<float.h>	240
7.6	<limits.h>.....	243
7.7	<math.h>	244
7.7.1	acos 関数	247
7.7.2	asin 関数.....	248
7.7.3	atan 関数	249
7.7.4	atan2 関数	250
7.7.5	cos 関数	251
7.7.6	sin 関数.....	252
7.7.7	tan 関数.....	253
7.7.8	cosh 関数	254
7.7.9	sinh 関数	255

	7.7.10	tanh 関数.....	256
	7.7.11	exp 関数.....	257
	7.7.12	frexp 関数	258
	7.7.13	ldexp 関数	259
	7.7.14	log 関数	260
	7.7.15	log10 関数	261
	7.7.16	modf 関数.....	262
	7.7.17	pow 関数	263
	7.7.18	sqrt 関数.....	264
	7.7.19	ceil 関数.....	265
	7.7.20	fabs 関数	266
	7.7.21	floor 関数	267
	7.7.22	fmod 関数.....	268
7.8		<setjmp.h>.....	269
	7.8.1	setjmp 関数	271
	7.8.2	longjmp 関数	272
7.9		<signal.h>	273
	7.9.1	signal 関数.....	275
	7.9.2	raise 関数	277
7.10		<stdarg.h>	278
	7.10.1	va_start マクロ.....	280
	7.10.2	va_arg マクロ	281
	7.10.3	va_end マクロ	282
7.11		<stdio.h>.....	283
	7.11.1	remove 関数.....	287
	7.11.2	rename 関数.....	288
	7.11.3	tmpfile 関数	289
	7.11.4	tmpnam 関数	290
	7.11.5	fclose 関数	291
	7.11.6	fflush 関数.....	292
	7.11.7	fopen 関数	293
	7.11.8	freopen 関数	295
	7.11.9	setbuf 関数	296
	7.11.10	setvbuf 関数	297
	7.11.11	fprintf 関数	299
	7.11.12	fscanf 関数	307
	7.11.13	printf 関数.....	311
	7.11.14	scanf 関数.....	312

7.11.15	sprintf 関数	313
7.11.16	sscanf 関数	314
7.11.17	vfprintf 関数	315
7.11.18	vprintf 関数.....	317
7.11.19	vsprintf 関数	318
7.11.20	fgetc 関数	319
7.11.21	fgets 関数.....	320
7.11.22	fputc 関数.....	321
7.11.23	fputs 関数.....	322
7.11.24	getc 関数.....	323
7.11.25	getchar 関数.....	324
7.11.26	gets 関数	325
7.11.27	putc 関数	326
7.11.28	putchar 関数	327
7.11.29	puts 関数	328
7.11.30	ungetc 関数	329
7.11.31	fread 関数	330
7.11.32	fwrite 関数	332
7.11.33	fseek 関数.....	334
7.11.34	ftell 関数.....	336
7.11.35	rewind 関数.....	337
7.11.36	clearerr 関数.....	338
7.11.37	feof 関数	339
7.11.38	ferror 関数	340
7.11.39	perror 関数	341
7.12	<stdlib.h>.....	342
7.12.1	atof 関数.....	344
7.12.2	atoi 関数	345
7.12.3	atol 関数	346
7.12.4	strtod 関数	347
7.12.5	strtol 関数	349
7.12.6	rand 関数	351
7.12.7	srand 関数.....	352
7.12.8	calloc 関数.....	353
7.12.9	free 関数.....	354
7.12.10	malloc 関数	355
7.12.11	realloc 関数	356
7.12.12	abort 関数	357

7.12.13	exit 関数	358
7.12.14	getenv 関数	359
7.12.15	onexit 関数	360
7.12.16	system 関数	361
7.12.17	bsearch 関数	362
7.12.18	qsort 関数	364
7.12.19	abs 関数	365
7.12.20	div 関数	366
7.12.21	labs 関数	367
7.12.22	ldiv 関数	368
7.13	<string.h>	369
7.13.1	memcpy 関数	372
7.13.2	strcpy 関数	373
7.13.3	strncpy 関数	374
7.13.4	strcat 関数	375
7.13.5	strncat 関数	376
7.13.6	memcmp 関数	377
7.13.7	strcmp 関数	378
7.13.8	strncmp 関数	379
7.13.9	memchr 関数	380
7.13.10	strchr 関数	381
7.13.11	strcspn 関数	382
7.13.12	strpbrk 関数	383
7.13.13	strrchr 関数	384
7.13.14	strspn 関数	385
7.13.15	strstr 関数	386
7.13.16	strtok 関数	387
7.13.17	memset 関数	389
7.13.18	strerror 関数	390
7.13.19	strlen 関数	391
7.14	<time.h>	392
7.14.1	clock 関数	393
7.14.2	difftime 関数	394
7.14.3	time 関数	395
7.14.4	asctime 関数	396
7.14.5	ctime 関数	397
7.14.6	gmtime 関数	398
7.14.7	localtime 関数	399

第 8 章 DSP ライブラリ

8.1	概要	403
8.2	データフォーマット	404
8.3	効率	406
8.4	高速フーリエ変換	407
8.4.1	概要	407
8.4.1.1	関数一覧	407
8.4.1.2	複素数データ配列フォーマット	407
8.4.1.3	実数データ配列フォーマット	408
8.4.1.4	スケーリング	408
8.4.1.5	FFT 構造	409
8.4.2	各関数の説明	410
8.4.2.1	FftComplex 関数	410
8.4.2.2	FftReal 関数	412
8.4.2.3	IfftComplex 関数	414
8.4.2.4	IfftReal 関数	415
8.4.2.5	FftInComplex 関数	417
8.4.2.6	FftInReal 関数	418
8.4.2.7	IfftInComplex 関数	420
8.4.2.8	IfftInReal 関数	421
8.4.2.9	LogMagnitude 関数	423
8.4.2.10	InitFft 関数	424
8.4.2.11	FreeFft 関数	425
8.5	窓関数	426
8.5.1	概要	426
8.5.1.1	関数一覧	426
8.5.2	各関数の説明	427
8.5.2.1	GenBlackman 関数	427
8.5.2.2	GenHamming 関数	428
8.5.2.3	GenHanning 関数	429
8.5.2.4	GenTriangle 関数	430
8.6	フィルタ	431
8.6.1	概要	431
8.6.1.1	関数一覧	431
8.6.1.2	係数のスケーリング	431
8.6.1.3	作業領域	432

8.6.1.4	メモリの使用.....	432
8.6.2	各関数の説明.....	433
8.6.2.1	Fir 関数	433
8.6.2.2	Fir1 関数	435
8.6.2.3	Iir 関数	437
8.6.2.4	Iir1 関数	439
8.6.2.5	DIir 関数	441
8.6.2.6	DIir1 関数	443
8.6.2.7	Lms 関数	445
8.6.2.8	Lms1 関数	447
8.6.2.9	InitFir 関数	449
8.6.2.10	InitIir 関数	450
8.6.2.11	InirDIir 関数.....	451
8.6.2.12	InitLms 関数.....	452
8.6.2.13	FreeFir 関数.....	453
8.6.2.14	FreeIir 関数	454
8.6.2.15	FreeDIir 関数.....	455
8.6.2.16	FreeLms 関数	456
8.7	畳み込みと相関.....	457
8.7.1	概要	457
8.7.1.1	関数一覧.....	457
8.7.2	各関数の説明.....	458
8.7.2.1	ConvComplete 関数	458
8.7.2.2	ConvCyclic 関数	459
8.7.2.3	ConvPartial 関数.....	460
8.7.2.4	Correlate 関数	461
8.7.2.5	CorrCyclic 関数.....	463
8.8	その他	464
8.8.1	概要	464
8.8.1.1	関数一覧.....	464
8.8.2	各関数の説明.....	465
8.8.2.1	Limit 関数	465
8.8.2.2	CopyXtoY 関数	466
8.8.2.3	CopyYtoX 関数	467
8.8.2.4	CopyToX 関数	468
8.8.2.5	CopyToY 関数	469
8.8.2.6	CopyFromX 関数	470
8.8.2.7	CopyFromY 関数	471

8.8.2.8	GenGWnoise 関数.....	472
8.8.2.9	MatrixMult 関数	473
8.8.2.10	VectorMult 関数.....	475
8.8.2.11	MsPower 関数	476
8.8.2.12	Mean 関数.....	477
8.8.2.13	Variance 関数	478
8.8.2.14	MaxI 関数	479
8.8.2.15	MinI 関数.....	480
8.8.2.16	PeakI 関数.....	481

《付録》

A.	コンパイラが規定する言語仕様とライブラリ関数仕様.....	485
	A.1 言語仕様.....	485
	A.2 ライブラリ関数仕様.....	491
	A.3 浮動小数点数の仕様.....	496
B.	引数割り付けの具体例	504
C.	レジスタとスタック領域の使用法	506
D.	終了処理関数の作成例	507
	D.1 終了処理の登録と実行(onexit)ルーチンの作成例	507
	D.2 プログラムの終了(exit)ルーチンの作成例	508
	D.3 異常終了(abort)ルーチンの作成例	509
E.	低水準インタフェースルーチンの作成例	510
F.	ASCII コード一覧表	515
G.	エンコード規則	516
H.	実行時ルーチン命名規則	519
I.	割り込みハンドラ	521
J.	リエントラントライブラリ	524

索引

K.1	日本語索引	527
K.2	英語索引	534

図目次

< 概要・操作 >

図 1-1	本コンパイラの機能.....	3
図 1-2	プログラムの開発手順.....	4
図 1-3	show = noinclude, noexpansion のソースリスト情報.....	32
図 1-4	show = include, expansion のソースリスト情報.....	33
図 1-5	show = source, object のオブジェクト情報.....	34
図 1-6	show = nosource, object のオブジェクト情報.....	35
図 1-7	統計情報.....	36
図 1-8	コマンド指定情報.....	37

< C/C++プログラミング >

図 2-1	スタックフレームの割り付け、解放に関する規則.....	65
図 2-2	引数の割り付け領域.....	69
図 2-3	引数格納用レジスタの割り付け例.....	71
図 2-4	リターン値をメモリに設定する場合のリターン値の設定領域.....	72
図 2-5	割り込み関数によるスタック使用例.....	75

< システム組み込み >

図 3-1	統計情報例.....	110
図 3-2	静的な領域の割り付け例.....	113
図 3-3	関数呼び出しの関係とスタック使用量の例.....	116
図 3-4	プログラムの構成例（ライブラリ関数を使用しない場合）.....	118
図 3-5	プログラムの構成例（ライブラリ関数を使用する場合）.....	124
図 3-6	FILE 型データ.....	129

< 標準ライブラリ >

図 7-1	標準インクルードファイル説明の凡例.....	212
図 7-2	関数説明の凡例.....	213
図 7-3	atan2 関数の意味.....	248

< DSP ライブラリ >

図 8-1	データフォーマット.....	403
-------	----------------	-----

< 付録 >

図 A-1	浮動小数点数の内部表現の構成.....	494
図 C-1	レジスタとスタック領域の使用法.....	504

表目次

< 概要・操作 >

表 1-1	本コンパイラおよび関連ソフトウェアで使用する標準のファイル拡張子	7
表 1-2	コンパイラオプション一覧.....	8
表 1-3	define オプションで指定できるマクロ名、名前、定数	17
表 1-4	オプションの組み合わせ	27
表 1-5	標準ライブラリとコンパイルオプションの関係.....	28
表 1-6	コンパイルリストの構成と内容.....	31
表 1-7	環境変数	37
表 1-8	暗黙の宣言	39

< C/C++プログラミング >

表 2-1	コンパイラの限界値	43
表 2-2	メモリ領域の種類とその性質の概要.....	47
表 2-3	スカラ型、基本型の内部表現.....	50
表 2-4	複合型、クラス型の内部表現.....	51
表 2-5	ビットフィールドメンバの仕様.....	57
表 2-6	関数呼び出し前後のレジスタ保証規則.....	65
表 2-7	引数割り付け領域の一般規則.....	70
表 2-8	リターン値の型と設定場所.....	72
表 2-9	割り込み仕様一覧	74
表 2-10	組み込み関数一覧	77
表 2-11	単精度浮動小数点ライブラリ関数一覧.....	92
表 2-12	日本語コードのデフォルト設定.....	93
表 2-13	トラブル発生時の対処方法.....	105

< システム組み込み >

表 3-1	スタックサイズの計算例	116
表 3-2	低水準インタフェースルーチンの一覧.....	131

< モジュール間最適化ツール >

表 5-1	オプション一覧	189
-------	---------------	-----

< 標準ライブラリ >

表 7-1	ライブラリの種類と対応する標準インクルードファイル.....	211
表 7-2	マクロ名定義からなる標準インクルードファイル.....	212
表 7-3	ファイルアクセスモードの種類.....	218
表 7-4	文字の種類	224
表 7-5	フラグの種類と意味	298
表 7-6	パラメタのサイズ指定の種類とその意味.....	300

表 7-7	変換文字と変換の方式	301
表 7-8	変換後のデータのサイズ指定の種類とその意味.....	306
表 7-9	変換文字と変換の内容	307
表 7-10	オフセットの種類	333

< 付録 >

表 A-1	翻訳の仕様	483
表 A-2	環境の仕様	483
表 A-3	識別子の仕様	483
表 A-4	文字の仕様	484
表 A-5	整数の仕様	484
表 A-6	整数型とその値の範囲	485
表 A-7	浮動小数点数の仕様	485
表 A-8	浮動小数点数の限界値	486
表 A-9	配列とポインタの仕様	486
表 A-10	レジスタの仕様	486
表 A-11	構造体、共用体、列挙型、ビットフィールドの仕様.....	487
表 A-12	修飾子の仕様.....	487
表 A-13	宣言の仕様.....	487
表 A-14	文の仕様.....	488
表 A-15	プリプロセッサの仕様.....	488
表 A-16	stddef.h の仕様	489
表 A-17	assert.h の仕様.....	489
表 A-18	ctype.h の仕様.....	489
表 A-19	真となる文字の集合	490
表 A-20	math.h の仕様.....	490
表 A-21	setjmp.h の仕様	490
表 A-22	stdio.h の仕様	491
表 A-23	無限大および非数の表示形式	491
表 A-24	string.h の仕様.....	492
表 A-25	errno.h の仕様	492
表 A-26	サポートしていないライブラリ	493
表 A-27	浮動小数点数の表現する値の種類	496
表 F-1	ASCII コード一覧表.....	513
表 G-1	演算子のエンコード	514
表 J-1	リエントラントライブラリー一覧 (1)	522
表 J-1	リエントラントライブラリー一覧 (2)	523
表 J-1	リエントラントライブラリー一覧 (3)	524

1. 概要・操作

1.1 概要

本コンパイラは、C/C++言語で記述したソースプログラムを、SuperH マイコン用リロケータブルオブジェクトプログラムまたはアセンブリソースプログラムに変換するソフトウェアシステムです。本コンパイラがサポートする SuperH マイコンは SH-1、SH-2、SH-2E、SH-3、SH-3E、SH-4 です。

本コンパイラの機能を図 1-1 に示します。

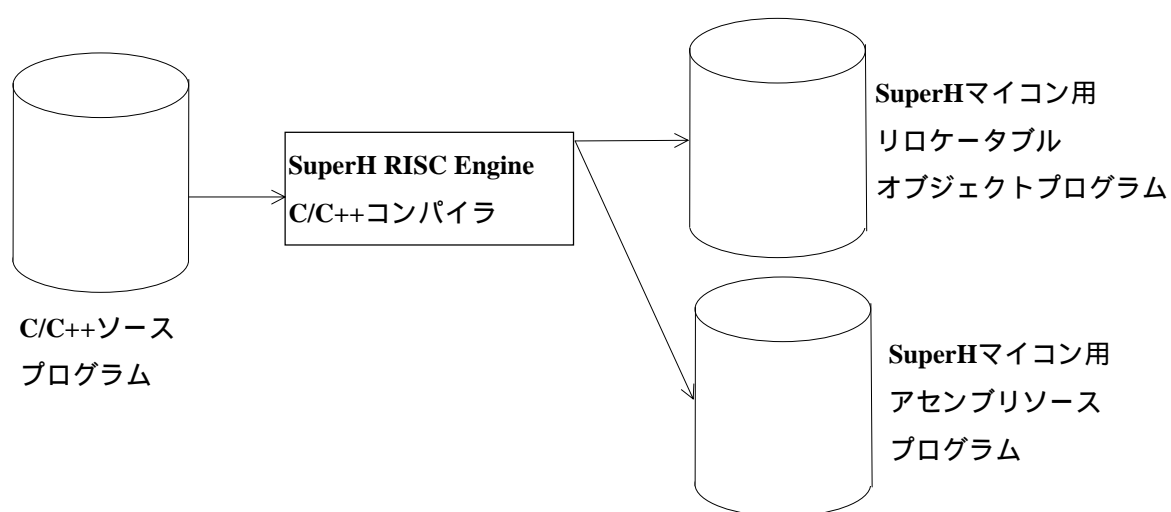


図 1-1 本コンパイラの機能

本マニュアルでは、コンパイラのほかに、標準ライブラリ(C 言語で記述されているプログラム内で標準的に利用する C 言語レベル関数群、実行時ルーチン群)、組み込み向け C++ライブラリ(Embedded C++クラスライブラリ)、DSP ライブラリ(デジタル信号処理ライブラリ関数群)およびモジュール間最適化ツールについて説明します。ただし、組み込み向け C++ライブラリは、Ver.5.0 ではサポートしておりません。

1.2 プログラムの開発手順

本システムを用いたプログラムの開発手順を図 1-2 に示します。

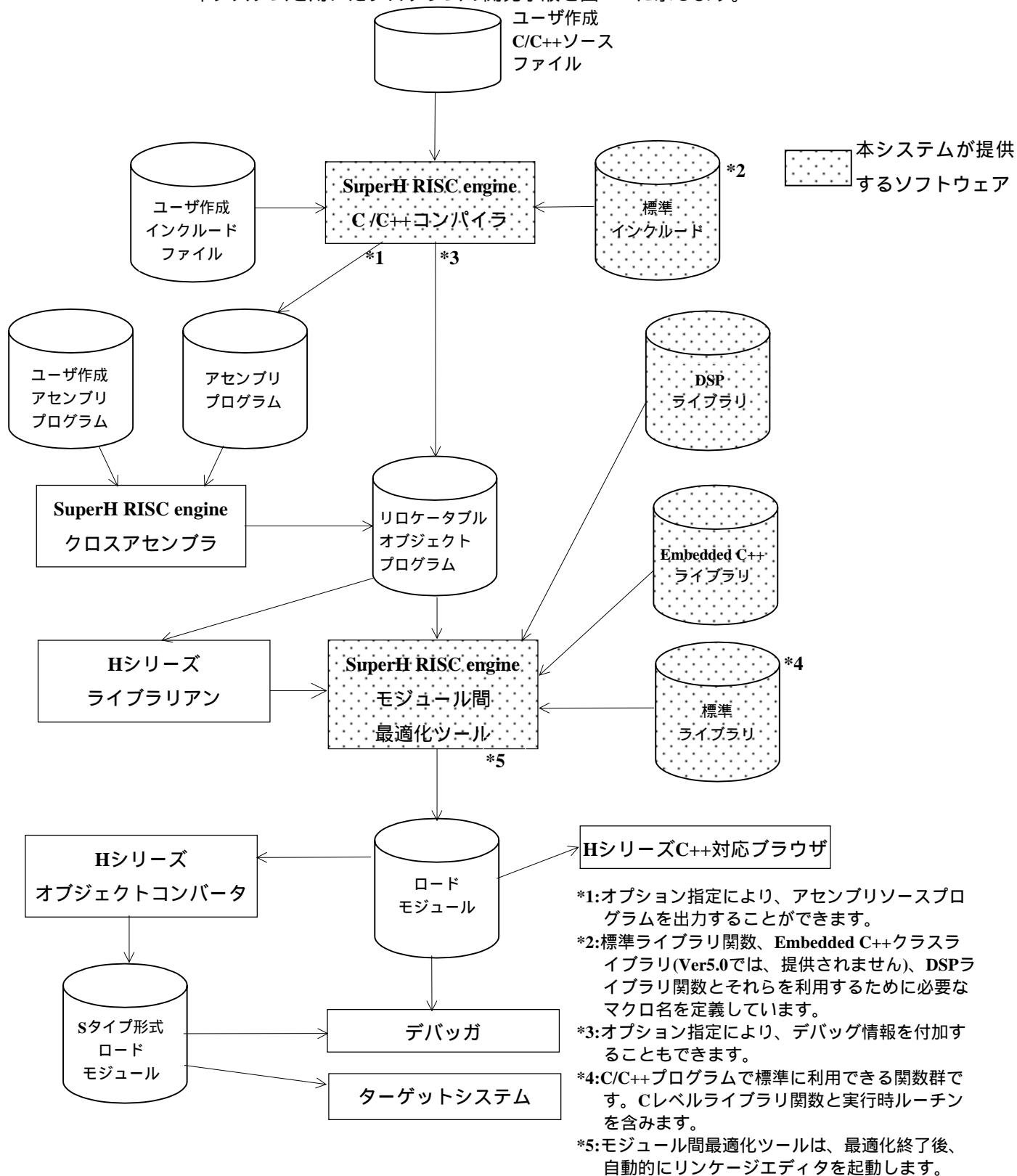


図 1-2 プログラムの開発手順

1.3 コンパイラの実行

本節ではコンパイラの起動方法、オプションの指定方法、コンパイルリストの見方について解説します。

1.3.1 コンパイラの起動方法

コンパイラを起動するコマンドラインの形式は次のとおりです。

```
shc[ <オプション>...][ <ファイル名>[ <オプション>...]....]
```

```
shcpp[ <オプション>...][ <ファイル名>[ <オプション>...]....]
```

コマンド **shc** は、C プログラム、C++プログラムをそれぞれ、**lang** オプションまたは、プログラムファイル名の拡張子に従い、C コンパイル^{*1}、C++コンパイル^{*1}します。コマンド **shcpp** は、C プログラム、C++プログラムに関係なく、C++コンパイルします。

^{*1}:C コンパイルとは、プログラムを C 言語の文法に基づいてコンパイル、C++コンパイルとは、C++言語の文法に基づいてコンパイルすることを意味しています。

以下、コンパイラの基本的な操作方法を説明します。

(1) プログラムのコンパイル

C ソースプログラム「**test.c**」をコンパイルします。

```
shc test.c (RET)
```

C++ソースプログラム「**test.cpp**」をコンパイルします。

```
shc test.cpp(RET)
```

```
shcpp test.cpp(RET)
```

(2) コマンド入力形式、コンパイラオプションの表示

標準出力画面上にコマンドの入力形式、コンパイラオプションの一覧を表示します。

```
shc (RET)
```

```
shcpp(RET)
```

(3) オプション指定方法

オプション(debug、listfile、show 等)の前に - を付加し、複数のオプションを指定するときはスペース()で区切ります。PC 版では、DOS プロンプトで - のかわりに / を使用することもできます。

複数のサブオプションを指定するときはコンマ(,)で区切って指定します。

```
shc -debug -listfile -show=noobject,expansion test.c (RET)
```

PC 版では、さらに括弧() でくくって指定することもできます。

```
shc /debug /listfile /show=(noobject,expansion) test.c (RET)
```

(4) 複数の C/C++プログラムのコンパイル

複数の C/C++ プログラムを一度にコンパイルできます。以下に、C ソースプログラムをコンパイルする例を示します。

例 1 複数プログラムの指定方法

```
shc test1.c test2.c (RET)
```

例 2 オプションの指定(C ソースプログラムすべてに有効なオプション指定例)

```
shc -listfile test1.c test2.c (RET)
```

「test1.c」、 「test2.c」とも listfile オプションが有効となります。

例 3 オプションの指定(プログラムごとに有効なオプション指定例)

```
shc test1.c test2.c -listfile (RET)
```

listfile オプションは「test2.c」だけに対して有効になります。プログラムごとのオプション指定は、ソースプログラム全体に対するオプション指定よりも優先されます。

1.3.2 ファイル名の付け方

本コンパイラは、ファイル名指定時に拡張子を省略した場合、標準のファイル拡張子を付加したファイル名のファイルをコンパイルします。本コンパイラおよび関連ソフトウェアで使用する標準のファイル拡張子を表 1-1 に示します。なお、ファイル名の付け方の一般的な規則は各ホストマシンに準じています。ご使用になるホストマシンのマニュアルを参照してください。

表 1-1 本コンパイラおよび関連ソフトウェアで使用する標準のファイル拡張子

No.	拡張子	意味
1	c	C 言語で記述されたソースプログラムファイル
2	cpp、cc、cp、CC	C++言語で記述されたソースプログラムファイル
3	h	インクルードファイル
4	lis、lst、lpp	リストファイル ^{*1*2}
5	p、pp	プリプロセッサ展開後のファイル ^{*2*3}
6	obj	リロケータブルオブジェクトプログラムファイル ^{*2}
7	src	アセンブリソースプログラムファイル ^{*2}
8	dtb	デバッグ情報ファイル ^{*2}
9	iop	モジュール間最適化用情報ファイル ^{*2}
10	dwi	DWARF フォーマット変換用情報ファイル ^{*2}
11	lib	ライブラリファイル
12	abs	アブソリュートロードモジュールファイル
13	rel	リロケータブルロードモジュールファイル
14	map	リンケージマップリストファイル

【注】 *1:UNIX 版では、lis、PC 版の C コンパイル時は lst、C++コンパイル時は lpp を使用します。

*2:コンパイラがオプションに従い、自動生成します。

*3:拡張子は、C コンパイルのときは p、C++コンパイルのときは pp になります。

1.3.3 コンパイラオプション

コンパイラオプションの形式と短縮形および省略時解釈の一覧を表 1-2 に示します。下線部 () は短縮形指定時の文字を示します。また、斜体字は省略時解釈を示します。

項目中の括弧内 () の C、C++は、それぞれ C コンパイル、C++コンパイルに有効なオプションを示します。また、項目中の括弧内 [] の SH1、SH2、SH2E、SH3、SH3E、SH4 は、どのオプション CPU 種別に有効かを示します。

【注】 " * " のないものは、Ver.5.0 で有効なオプションです。 " * " のあるものは、Ver.5.1 以降でサポートいたします。

表 1-2 コンパイラオプション一覧

No.	項目	形式	指定内容
1	CPU 種別 (C,C++) [SH1 ~ SH4]	<code>cpu = sh1</code> <code>sh2</code> <code>sh2e</code> <code>sh3</code> <code>sh3e</code> <code>sh4</code>	SH-1 のオブジェクトを生成 SH-2 のオブジェクトを生成 SH-2E のオブジェクトを生成 SH-3 のオブジェクトを生成 SH-3E のオブジェクトを生成 SH-4 のオブジェクトを生成
2	最適化レベル (C,C++) [SH1 ~ SH4]	<code>optimize = 0</code> <code>1</code>	最適化なしのオブジェクトを出力 最適化ありのオブジェクトを出力
3	最適化方法の選択 (C,C++) [SH1 ~ SH4]	<code>speed</code> <code>speed = shift*</code> <code>loop*</code> <code>switch*</code> <code>struct*</code> <code>nospeed</code> <code>size</code>	実行速度優先のコードを生成 シフト演算をより高速なオブジェクトコードで展開 ループ文をより高速なオブジェクトコードで展開 switch 文をより高速なオブジェクトコードで展開 構造体、クラス型や double 型の代入文をコードに直接 インライン展開 実行速度、サイズのバランスのとれたコードを生成 サイズ優先のコードを生成
4	デバッグ情報 (C,C++) [SH1 ~ SH4]	<code>debug</code> <code>nodebug</code>	出力あり 出力なし
5	リスト内容と形式 (C,C++) [SH1 ~ SH4]	<code>show = source</code> <code>nosource</code> <code>object</code> <code>noobject</code> <code>statistics</code> <code>nostatistics</code> <code>include</code> <code>noinclude</code> <code>expansion</code> <code>noexpansion</code> <code>allocation*</code> <code>noallocation*</code> <code>width = <数値></code> <code>length = <数値></code>	ソースリストの有無 オブジェクトリストの有無 統計情報の有無 インクルード展開後リストの有無 マクロ展開後リストの有無 シンボル割り付け情報出力の有無 1 行の最大文字数 数値: 0, 80 ~ 132(省略時:132) ページ内の最大行数 数値: 0, 40 ~ 255(省略時:80)
6	リストファイル (C,C++) [SH1 ~ SH4]	<code>listfile [= <ファイル名>]</code> <code>nolistfile</code>	出力あり 出力なし
7	オブジェクトファイル (C,C++) [SH1 ~ SH4]	<code>objectfile = <ファイル名></code>	出力あり

No.	項目	形式	指定内容
8	オブジェクト形式 (C,C++) [SH1 ~ SH4]	<code>code = machinecode</code> <code>asmcode</code>	機械語プログラムを出力 アセンブリソースプログラムを出力
9	マクロ名の定義 (C,C++) [SH1 ~ SH4]	<code>define = <マクロ名>=<名前></code> <code><マクロ名>=<定数></code> <code><マクロ名></code>	<名前>を<マクロ名>として定義 <定数>を<マクロ名>として定義 <マクロ名>を定義したものと仮定
10	インクルードファイル (C,C++) [SH1 ~ SH4]	<code>include = <パス名></code>	インクルードファイルの取り込み先パス名を指定 (複数指定可)
11	セクション名 (C,C++) [SH1 ~ SH4]	<code>section =</code> <code>program = <セクション名></code> <code>const = <セクション名></code> <code>data = <セクション名></code> <code>bss = <セクション名></code> 省略時: (p=P, c=C, d=D, b=B)	プログラム領域のセクション名を指定 定数領域のセクション名を指定 初期化データ領域のセクション名を指定 未初期化データ領域のセクション名を指定
12	ヘルプメッセージ (C,C++) [SH1 ~ SH4]	<code>help</code>	出力あり
13	プログラムセクションポ ジションインディペンデ ント (C,C++) [SH2 ~ SH4]	<code>pic = 0</code> <code>1</code>	プログラムセクションのポジションインディペンデ ントコードを生成しない プログラムセクションのポジションインディペンデ ントコードを生成する
14	文字列出力領域 (C,C++) [SH1 ~ SH4]	<code>string = const</code> <code>data</code>	文字列を定数領域セクション(C)へ出力 初期化データ領域セクション(D)へ出力
15	コメントのネスト (C,C++) [SH1 ~ SH4]	<code>comment = nest</code> <code>nonest</code>	コメント(/ * /)のネストを許す コメント(/ * /)のネストを許さない
16	文字列内の日本語コード の選択 (C,C++) [SH1 ~ SH4]	<code>euc</code> <code>sjis</code>	euc コードを選択 sjis コードを選択

1. 概要・操作

No.	項目	形式	指定内容
17	サブコマンドファイルの選択 (C,C++) [SH1 ~ SH4]	<code>subcommand = <ファイル名></code>	<ファイル名>で指定したファイルからコマンドオプションを取り込む
18	除算の方式 (C,C++) [SH2]	<code>division = <i>cpu</i></code> <code>peripheral</code> <code>nomask</code>	<code>cpu</code> の除算命令を使用 除算器を使用(割り込みマスクあり) 除算器を使用(割り込みマスクなし)
19	メモリのバイト並び順の指定 (C,C++) [SH3 ~ SH4]	<code>endian = <i>big</i></code> <code>little</code>	Big Endian Little Endian
20	インライン展開の仕様 (C,C++) [SH1 ~ SH4]	<code>inline</code> <code>inline = <数値></code> <code>noinline</code>	自動インライン展開を行う 自動インライン展開する関数のサイズの限界を指定 自動インライン展開を行わない
21	デフォルトのヘッダファイルの指定 (C,C++) [SH1 ~ SH4]	<code>preinclude = <ファイル名></code>	指定したファイルの内容をコンパイル単位の先頭に取り込む
22	MACH、MACL レジスタの保証 (C,C++) [SH1 ~ SH4]	<code>macsave = 0</code> <code>1</code>	関数呼び出しで MACH、MACL レジスタを保証しない 関数呼び出しで MACH、MACL レジスタを保証する
23	インフォメーションメッセージ出力 (C,C++) [SH1 ~ SH4]	<code>message</code> <code>nomessage</code>	インフォメーションメッセージを出力する インフォメーションメッセージを出力しない
24	ラベルの 16 バイト整合 (C,C++) [SH1 ~ SH4]	<code>align16</code> <code>noalign16</code>	プログラムセクション内のラベルで、サブルーチンコール以外の無条件分岐命令直後のラベルをすべて 16 バイト整合する ラベルを 16 バイト整合しない。
25	double 型の単精度化 (C,C++) [SH1 ~ SH3E]	<code>double = float</code>	double(倍精度浮動小数点数)型の数値を float(単精度浮動小数点数)型としてオブジェクト生成する

No.	項目	形式	指定内容
26	漢字変換 (C,C++) [SH1 ~ SH4]	<u>o</u> tcode = <u>e</u> uc <u>s</u> jis	漢字コードを euc コードにする 漢字コードを sjis コードにする
27	ABS16 宣言 (C,C++) [SH1 ~ SH4]	<u>a</u> bs16 = <u>r</u> un <u>a</u> ll	実行時ルーチンをすべて #pragma abs16 宣言されたものとみなす すべてのラベルアドレスを 16 ビットで生成する
28	ループ展開最適化 (C,C++) [SH1 ~ SH4]	<u>l</u> oop <u>n</u> o <u>l</u> oop	ループ展開の最適化を行う ループ展開の最適化を行わない
29	インライン展開 (C,C++) [SH1 ~ SH4]	<u>n</u> estinline = <数値>	ネストしたインライン関数を展開する回数の指定
30	リターン値の拡張 (C) [SH1 ~ SH4]	<u>r</u> tnext <u>n</u> o <u>r</u> tnext	返却値の符号/ゼロ拡張する 返却値の符号/ゼロ拡張しない
31	プリプロセッサ展開出力 (C,C++) [SH1 ~ SH4]	<u>p</u> reprocessor[= <ファイル名>]	プリプロセッサ展開後のソースプログラムを出力
32	ブラウザ情報 (C++) [SH1 ~ SH4]	<u>b</u> rowser	出力あり
33	組み込み向け C++ 言語 (C++) [SH1 ~ SH4]	<u>e</u> c <u>pp</u> *	Embedded C++ 言語仕様に基づいてコンパイルチェック
34	ISO-Latin1 コードサポート (C,C++) [SH1 ~ SH4]	<u>l</u> atin1*	文字列リテラル、コメント部、および文字定数を ISO-Latin1 コードとしてコンパイル
35	FPU (C,C++) [SH4]	<u>f</u> pu = <u>s</u> ingle <u>d</u> ouble	浮動小数点演算をすべて単精度浮動小数点で演算 浮動小数点演算をすべて倍精度浮動小数点で演算
36	非正規化数の扱い (C,C++) [SH4]	<u>d</u> enormalization = <u>o</u> ff <u>o</u> n	非正規化数を 0 として扱います 非正規化数を非正規化数として扱います

No.	項目	形式	指定内容
37	丸め方向 (C,C++) [SH4]	<code>round = <u>zero</u></code> <code> nearest</code>	Round to Zero で丸めます Round to Nearest で丸めます
38	switch 文展開方式 (C,C++) [SH1 ~ SH4]	<code>case = <u>ifthen</u>*</code> <code> table*</code>	<code>if_then</code> 方式で switch 文展開 テーブル方式で switch 文展開
39	外部変数最適化 (C,C++) [SH1 ~ SH4]	<code>volatile*</code> <code>novolatile*</code>	全ての外部変数を <code>volatile</code> 変数として最適化抑止 <code>volatile</code> 宣言のない外部変数を最適化
40	packed 構造体 (C,C++) [SH1 ~ SH4]	<code>pack*</code>	構造体型、クラス型の境界調整数を 1 バイト
41	モジュール間最適化 (C,C++) [SH1 ~ SH4]	<code>goptimize*</code>	モジュール間最適化情報を出力
42	言語の選択 (C,C++) [SH1 ~ SH4]	<code>lang = c</code> <code> cpp</code>	C 文法に基づいてコンパイル C++文法に基づいてコンパイル

(1)CPU 種別(C,C++)

`-cpu = sh1 | sh2 | sh2e | sh3 | sh3e | sh4`

ターゲット CPU を指定します。

`cpu=sh1` は、SH-1 用のオブジェクトを生成します。

`cpu=sh2` は、SH-2 用のオブジェクトを生成します。

`cpu=sh2e` は、SH-2E 用のオブジェクトを生成します。

`cpu=sh3` は、SH-3 用のオブジェクトを生成します。

`cpu=sh3e` は、SH-3E 用のオブジェクトを生成します。

`cpu=sh4` は、SH-4 用のオブジェクトを生成します。

また、選択する CPU により、リンクするライブラリが異なります。詳しくは、「第 1 章 概要・操作 1.3.5 標準ライブラリとの対応」を参照してください。

本オプションの省略時解釈は、`cpu=sh1` です。

(2)最適化レベル(C,C++)

`-optimize = 0 | 1`

コンパイラの最適化を行うかどうかを指定します。

optimize = 0 は、コンパイラがオブジェクトプログラムに対し最適化を行いません。

optimize = 1 は、コンパイラがオブジェクトプログラムに対し最適化を行います。

本オプションの省略時解釈は、**optimize = 1** です。

(3)最適化方法の選択(C,C++)

`-speed`

speed 優先の最適化を行います。**speed** オプションを指定するとプログラムの実行速度は向上しますが、プログラムサイズが増大する場合があります。

サブオプションの指定省略時解釈は、**speed = shift,loop,switch,struct** です。

`-speed = shift`

シフト演算をより高速なオブジェクトコードで展開します。

`-speed = loop`

loop 文をより高速なオブジェクトコードで展開します。

`-speed = switch`

switch 文をより高速なオブジェクトコードで展開します。

`-speed = struct`

構造体、クラス型や **double** 型の代入文をオブジェクトコードにインラインに展開します。

`-nospeed`

nospeed が指定されており、**size** が指定されていない場合は、実行速度、サイズのバランスをとった最適化を行います。

`-size`

オブジェクトサイズ優先の最適化を行います。

本オプションの省略時解釈は、**nospeed** です。

(4)デバッグ情報(C,C++)

-debug

ソースレベルデバッグに必要なデバッグ情報をオブジェクトファイルに出力します。オブジェクト形式が、機械語プログラムの時は直接デバッグ情報が出力されます。またアセンブリソースプログラム出力時には、**.LINE** 制御命令がアセンブリソースプログラム中に組み込まれます。このため、コンパイラが出力したアセンブリソースプログラムでも C ソースレベルのステップ実行が可能です。C++コンパイル時には、オブジェクトファイルの下にディレクトリ **cppdtb** を作成し、拡張子 **dtb** のブラウザ情報ファイルを生成します。ただし、宣言/定義情報のみのブラウザ情報ファイルのため、宣言/定義に関するブラウジングのみ可能です。C++ソースプログラム全体をブラウジングするときは、**browser** オプションを指定してください。

-nodebug

デバッグ情報をオブジェクトファイル中に出力しません。

本オプションの省略時解釈は、**nodebug** です。

(5) リスト内容と形式 (C,C++)

`-show = source | nosource | object | noobject | statistics | nostatistics | include | noinclude | expansion | noexpansion | allocation | noallocation | width = <数値> | length = <数値>`

`show` オプションはリストファイルの出力形式を指定します。`show` オプションは `listfile` オプション指定時に有効です。

`show = source` は、ソースプログラムのリストを出力します。

`show = nosource` は、ソースプログラムのリストを出力しません。

`show = object` は、オブジェクトプログラムのリストを出力します。

`show = noobject` は、オブジェクトプログラムのリストを出力しません。

`show = statistics` は、統計情報のリストを出力します。

`show = nostatistics` は、統計情報のリストを出力しません。

`show = include` は、インクルード展開後のリストを出力します。

`show = noinclude` は、インクルード展開後のリストを出力しません。

`show = expansion` は、マクロ展開後のリストを出力します。

`show = noexpansion` は、マクロ展開後のリストを出力しません。

`show = allocation` は、シンボル割り付け情報のリストを出力します。

`show = noallocation` は、シンボル割り付け情報のリストを出力しません。

`show = width = <数値>` は、<数値>で指定された文字数を 1 行の最大文字数とします。

<数値>は、0、または 80 から 132 の整数を指定することができます。

`show=length=<数値>` は、<数値>で指定された行数を 1 ページの最大行数とします。

<数値>は、0、または 40 から 255 の整数を指定することができます。

`show = width = 0`、または `show = length = 0` を指定した場合、次のように解釈します。

`show = width = 0` を指定した場合、改行コードが出力されるまでを 1 行とします。

`show = length = 0` を指定した場合、最大行数は設定せず、改行は行いません。

(6) リストファイル (C,C++)

`-listfile [= <リストファイル名>]`

リストファイルを出力します。ファイル名の指定を省略した場合、ソースファイル名と同じファイル名に標準の拡張子(`.lis`/`.lst`/`.lpp`)を付加したファイルを生成します。UNIX 版: 拡張子(`.lis`)、PC 版:C コンパイル時は拡張子(`.lst`)、C++コンパイル時は拡張子(`.lpp`)です。

`-nolistfile`

リストファイルは、出力されません。

本オプションの省略時解釈は、`nolistfile` です。

(7)オブジェクトファイル (C,C++)

`-objectfile = <ファイル名>`

出力するオブジェクトファイル名を指定します。<ファイル名>を指定しない場合には、ソースファイルと同じファイル名で拡張子が「**obj**」(オブジェクト形式が機械語プログラムの時)、または、「**src**」(オブジェクト形式がアセンブリソースプログラムの時)のオブジェクトファイルが出力されます。

(8)オブジェクト形式 (C,C++)

`-code = machinecode | asmcode`

コンパイラが直接機械語のオブジェクトファイルを出力するか、アセンブリソースファイルを出力するかを指定します。

`code = machinecode` は、モジュール間最適化ツールに入力可能な機械語プログラムを出力します。

`code = asmcode` は、アセンブラに入力可能なアセンブリソースプログラムを出力します。

`code = asmcode` と `debug` を指定すると、アセンブリソースプログラム中に `.LINE` 制御命令を出力します。ただし、そのファイルはブラウジング対象外になります。

本オプションの省略時解釈は、`code=machinecode` です。

(9)マクロ名の定義 (C,C++)

`-define = <マクロ名>=<名前> | <マクロ名>=<定数> | <マクロ名>`

本オプションで指定されたマクロ定義をソースプログラムの先頭で有効にします。マクロ名、名前および定数の長さはそれぞれ先頭から 31 文字まで有効となります。

本オプションを用いれば、ソースプログラム中と同様のマクロ定義がコマンドラインオプションで指定できます。

オプションで指定できるマクロ名の仕様を表 1-3 に示します。

表 1-3 define オプションで指定できるマクロ名、名前、定数

No.	項目	説明
1	マクロ名	英字またはアンダラインで始まり、そのあとに 0 個以上の英字、アンダラインまたは数字が続く文字列です。
2	名前	英字またはアンダラインで始まり、そのあとに 0 個以上の英字、アンダラインまたは数字が続く文字列です。
3	定数	<p>10 進定数：1 個以上の数字(0～9)の繰り返し、または 1 個以上の数字の繰り返しの後にピリオドが続き、そのあとに 0 個以上の数字が続く文字列です。</p> <p>8 進定数：数字 0 で始まり、そのあとに 1 個以上の 0～7 の数字が続く文字列です。</p> <p>16 進定数：数字 0 に x が続き、そのあとに 1 個以上の数字または A～F の英字が続く文字列です。</p>

(10)インクルードファイル(C,C++)

`-include = <パス名>`

インクルードファイルを検索するディレクトリを指定します。検索方法の詳細は、「付録 A.1 (13) プリプロセッサ」を参照してください。

(11)セクション名(C,C++)

`-section = | program=<セクション名> | _const=<セクション名> | _data=<セクション名> |
_bss=<セクション名>`

オブジェクトプログラムのセクション名を変更します。<セクション名>は、英字、数字または、アンダライン(_)で先頭が数字以外のものです。セクション名は、31 文字目まで有効です。

本オプション省略時のセクション名は、プログラム領域セクション P、定数領域セクション C、初期化データ領域セクション D、未初期化データ領域セクション B です。セクション名が変更できないセクションとして、初期処理データ領域セクション D_INIT_、後処理データ領域セクション D_END_、仮想関数表領域 C_\$VTBL があります。

(12)ヘルプメッセージ(C,C++)

`-help`

コンパイラのオプション一覧を表示します。本オプションが指定された場合、他のオプションは無効になります。

(13)ポジションインディペンデント(C,C++)

`-pic = 0 | 1`

pic = 1 指定時は、リンク後のプログラムセクションを任意のアドレスに配置して実行できます。データセクションはリンク時に決定したアドレス以外には配置できません。ポジションインディペンデントコードとして実行する場合は、関数のアドレスを初期値として指定することはできません。C++コンパイルでは、仮想関数、関数メンバへのポインタも関数のアドレスを初期値として必要とするため、仮想関数やメンバ関数へのポインタを含んだ C++プログラムは、ポジションインディペンデントコードとして実行できません。**cpu = sh1** 指定時は、**pic = 1** 指定を無視します。**cpu**、**pic**、**endian**、**double** オプションにより、リンクするライブラリが異なります。詳しくは、「第 1 章 概要・操作 1.3.5 標準ライブラリとの対応」を参照して下さい。

例 1

```
extern int f();  
int (*fp)() = f;      指定不可
```

例 2

```
struct A{virtual void f();};  指定不可  
void (A::*ap)() = &A::f;    指定不可  
本オプションの省略時解釈は、pic=0 です。
```

(14)文字列出力領域(C,C++)

`-string = const | data`

文字列を定数領域セクション(C)、または初期化データ領域セクション(D)のどちらのセクションに出力するかを指定します。

string = const は、文字列を定数領域セクション (C) へ出力します。

string = data は、文字列を初期化データ領域セクション (D) へ出力します。

本オプションの省略時解釈は、**string = const** です。

(15)コメントのネスト(C,C++)

`-comment = nest | nonest`

コメント `/* */` のネストを許可するかどうかを指定します。

`comment = nest` は、以下の例では、下線部がネストしたコメントと解釈され、一番外側のコメントが有効になります。

`comment = nonest` は、以下の例では、`nest2 */`でコメントが終了したと判断し、以降の`*/`がエラーになります。

例

```
/* comment
   int a;  /* nest1 /* nest2 */ */
*/
```

本オプションの省略時解釈は、`comment = nonest` です。

(16)文字列内の日本語コードの選択(C,C++)

`-euc`

プログラム中の文字列、文字定数およびコメント内の日本語コードを `euc` コードと解釈します。

`-sjis`

プログラム中の文字列、文字定数およびコメント内の日本語コードを `sjis` コードと解釈します。

本オプションの省略時はホストマシンによって日本語コードの解釈が異なります。

「第2章 C/C++プログラミング 2.3.5 文字列内の日本語記述」を参照してください。

(17)サブコマンドファイルの選択 (C,C++)

-subcommand = <ファイル名>

指定されたファイル名の内容をオプションと解釈します。

subcommand オプションはコマンドラインの中に複数回指定できます。サブコマンドファイル内にはコマンドラインの引数を空白、改行またはタブで区切ってオプションを指定してください。サブコマンドファイルの内容がコマンドライン引数の **subcommand** 指定位置に展開されます。サブコマンドファイル内に **subcommand** オプションを指定することはできません。

例

下記の例は、コマンドライン **shc -debug -cpu = sh2 test.c** と等価になります。

コマンドライン

```
shc -sub = test.sub test.c
```

test.sub の内容

```
-debug
```

```
-cpu = sh2
```

(18)除算の方式 (C,C++)

-division= *cpu* | *peripheral* | *nomask*

プログラム中の整数型除算、剰余算に対する実行時ルーチンを選択します。

division = cpu は、DIV1 命令による実行時ルーチンを選択します。

division = peripheral は、除算器を用いた実行時ルーチンを選択(割り込みマスクに 15 を設定)します。CPU 種別が、SH2 の時のみ実行可能です。

division = nomask は、除算器を用いた実行時ルーチンを選択(割り込みマスクは変更なし)します。CPU 種別が、SH2 の時のみ実行可能です。

peripheral、**nomask** 指定時は以下の点に注意してください。

- (1) ゼロ除算のチェックおよび **errno** の設定は行いません。
- (2) **nomask** 指定時には、除算器動作中に割り込みがかかり、割り込み処理ルーチンで除算器を用いた場合、動作は保証しません。
- (3) オーバフロー割り込みはサポートしていません。
- (4) ゼロ除算、オーバフローなどの演算結果は除算器の仕様に従います。cpu サブオプション指定時と異なる場合があります。

本オプションの省略時解釈は、**division = cpu** です。

(19)メモリのバイト並び順の指定(C,C++)

`-endian = big | little`

本オプションは `cpu` オプションの任意のサブオプションと組み合わせが可能です。

`endian = big` は、データのバイト並びが **Big Endian** になります。

`endian = little` は、データのバイト並びが **Little Endian** になります。Little Endian のオブジェクトプログラムは、SH1、SH2、SH2E では実行できません。`cpu`、`pic`、`endian`、`double` オプションにより、リンクするライブラリが異なります。詳しくは、「第 1 章 概要・操作 1.3.5 標準ライブラリとの対応」を参照してください。

本オプションの省略時解釈は、`endian=big` です。

(20)インライン展開の仕様(C,C++)

`-inline`, `-inline = <数値>`, `-noinline`

関数の自動インライン展開をするかしないかを指定します。

`inline` は、自動インライン展開を行います。

`inline = <数値>` は、自動インライン展開対象とする最大サイズを関数のノード数(宣言部を除く変数、演算子等の語句の総数)で示すものです。

`noinline` は、自動インライン展開を行いません。

`speed` オプション指定時のデフォルト値は、`inline = 20` です。`nospeed`、`size` オプション指定時、または `optimize = 0` オプション指定時のデフォルトは `noinline` です。

(21) デフォルトのヘッダファイルの指定 (C,C++)

`-preinclude = <ファイル名>`

指定したファイルの内容をコンパイル単位の先頭に取り込みます。ファイル名には、コンパイラを起動したディレクトリを基点に相対パスで指定するか、絶対パスで指定してください。

(22)MACH、MACL レジスタの保証(C,C++)

`-macsave = 0 | 1`

MACH、MACL レジスタを関数の呼び出し前後で保証するかどうかを指定します。

`macsave = 0` は、関数の呼び出し前後で MACH、MACL レジスタを保証しません。

`macsave = 1` は、関数の呼び出し前後で MACH、MACL レジスタを保証します。

`macsave = 1` でコンパイルした関数から `macsave = 0` でコンパイルした関数を呼び出すことはできません。逆に `macsave = 0` でコンパイルした関数から `macsave = 1` でコンパイルした関数を呼び出すことは可能です。

本オプションの省略時解釈は、`macsave = 1` です。

(23)インフォメーションメッセージ出力(C,C++)

`-message`

インフォメーションメッセージを出力することを指定します。

`-nomessage`

インフォメーションメッセージを出力しないことを指定します。

本オプションの省略時解釈は、`nomessage` です。

(24)ラベルの 16 バイト整合(C,C++)

`-align16`

プログラムセクション内のラベルで、サブルーチンコール以外の無条件分岐命令直後のラベルを、すべて 16 バイト整合することを指定します。

`-noalign16`

サブルーチンコール以外の無条件分岐命令直後のラベルを 16 バイト整合しません。

(25)double 型の単精度化(C,C++)

`-double = float`

`double`(倍精度浮動小数点数)型の数値を `float`(単精度浮動小数点数)型としてオブジェクト生成することを指定します。

(26)漢字変換(C,C++)

`-outcode = euc | sjis`

文字列、文字定数内に日本語を記述したときに、オブジェクトプログラムに出力する漢字コードを指定することができます。

`outcode = euc` は、漢字コードを `euc` コードにすることを指定します。

`outcode = sjis` は、漢字コードを `sjis` コードにすることを指定します。

(27)ABS16 宣言(C,C++)

`-abs16 = run | all`

`abs16 = run` は、実行時ルーチンをすべて `#pragma abs16` 宣言されたものとみなすことを指定します。

`abs16 = all` は、すべてのラベルアドレスを 16 ビットで生成することを指定します。

(28)ループ展開最適化(C,C++)

`-loop, -noloop`

ループ展開の最適化をするかどうかを指定します。

`loop` は、ループ文(`for`,`while`,`do-while`)をスピード優先で展開します。

`noloop` は、ループ文をスピード優先で展開しません。

本オプションの省略時解釈は、`noloop` です。

(29)インライン展開(C,C++)

`-noinline = <数値>`

ネストしたインライン関数を展開する回数を指定します。

指定できる最大値は **16** です。また、オプション省略時には **1** を指定したものとして処理します。

例

ソースプログラム

```
#pragma inline(fun1,fun2,fun3)
extern int dat;
void fun1(){a++;}
void fun2(){fun1();}
void fun3(){fun2();}
```

(1) `noinline=1` の時のインライン展開イメージ

```
#pragma inline(fun1,fun2,fun3)
extern int dat;
void fun1(){a++;}
void fun2(){a++;}
void fun3(){f1();}
```

(2) `noinline=2` の時インライン展開イメージ

```
#pragma inline(fun1,fun2,fun3)
extern int dat;
void fun1(){a++;}
void fun2(){a++;}
void fun3(){a++;}
```

(30)リターン値の拡張(C)

`-rtnext, -nortnext`

(unsigned) char 型、(unsigned) short 型を返す return 文において、関数返却値レジスタ **R0** の符号拡張あるいは、ゼロ拡張を行うか否かを指定します。関数プロトタイプ宣言がある場合は、本オプションを指定する必要はありません。

rtnext は、関数返却値の符号/ゼロ拡張を行います。

nortnext は、関数返却値の符号/ゼロ拡張を行いません。

本オプションの省略時解釈は、**nortnext** です。

(31)プリプロセッサ展開出力(C,C++)

`-preprocessor[= <ファイル名>]`

プリプロセッサ展開後のソースプログラムを出力します。<ファイル名>を指定しない場合には、ソースファイルと同じファイル名になります。拡張子は、C コンパイルは、「**p**」、C++コンパイルは「**pp**」のファイルが出力されます。

(32)ブラウザ情報(C++)

`-browser`

ブラウザツールに必要なブラウザ情報とデバッグ情報を出力します。C++コンパイル時にオブジェクトファイルの出力ディレクトリに `cppdtb` というディレクトリがなければ作成し、その下にブラウザ情報ファイルを生成します。

(33)組み込み向け C++言語(C++)

`-ecpp`

Embedded C++言語仕様に基づいて、C++プログラムをシンタックスチェックします。

(34)ISO-Latin1 コードサポート(C,C++)

`-latin1`

文字列リテラル、コメント部、および文字定数を ISO-Latin1 コードと解釈します。

(35)FPU(C,C++)

`-fpu = single | double`

`fpu = single` は、すべての浮動小数点演算を単精度浮動小数点で演算します。

`fpu = double` は、すべての浮動小数点演算を倍精度浮動小数点で演算します。

プログラム中に浮動小数点演算がない場合には、`fpu = single` を指定してください。

本オプションは、`cpu = sh4` のときのみ有効です。

(36)非正規化数の扱い(C,C++)

`-denormalization = off | on`

`denormalization = off` は、非正規化数を 0 として扱います。

`denormalization = on` は、非正規化数を非正規化数として扱います。

本オプションは、`cpu = sh4` のときのみ有効です。

本オプションの省略時解釈は、`denormalization = off` です。

(37)丸め方向(C,C++)

`-round = zero | nearest`

`round = zero` は、Round to Zero で丸めます。

`round = nearest` は、Round to Nearest で丸めます。

本オプションは、`cpu = sh4` のときのみ有効です。

本オプションの省略時解釈は、`round = zero` です。

(38)switch 文展開方式(C,C++)

`-case = ifthen | table`

`case = ifthen` は、switch 文を if_then 方式で展開します。if_then 方式は、switch 文の評価式の値と case ラベルの値を比較し、一致すれば case ラベルの文へ飛び処理を case ラベルの回数繰り返す展開方式です。この展開方式は、switch 文に含まれる case ラベル数に比例してオブジェクトコードのサイズが増大します。

`case = table` は、switch 文をテーブル方式で展開します。テーブル方式は、case ラベルの飛び先をジャンプテーブルに確保し、1 回のジャンプテーブル参照で switch 文の評価式と一致する case ラベルの文へ飛び越す展開方式です。この方式は、switch 文に含まれる case ラベルの数に比例して定数領域に確保されるジャンプテーブルのサイズが増えますが、実行速度は常に一定です。

本オプション省略時は、`speed`、`size` オプションが指定されていれば、それぞれスピード、サイズを優先した展開方式をコンパイラが選択します。

(39)外部変数最適化(C,C++)

`-volatile, -novolatile`

`volatile` は、外部変数に対して最適化を行いません。

`novolatile` は、外部変数に対して最適化を行います。

本オプションの省略時解釈は、`novolatile` です。

(40)packed 構造体(C,C++)

`-pack`

`pack` は、構造体、共用体、クラス型の境界調整数を 1 バイトとします。

(41)モジュール間最適化(C,C++)

`-goptimize`

モジュール間最適化付加情報の出力を指定します。

`goptimize` は、オブジェクトプログラムの出力ディレクトリに `shiop` というディレクトリがなければ作成し、その下にモジュール間最適化付加情報ファイルを生成します。

(42)言語の選択(C,C++)

`-lang=c | cpp`

言語を選択します。

`lang = c` は、C 文法に基づいてコンパイルします。

`lang = cpp` は、C++文法に基づいてコンパイルします。

本オプション省略時解釈は、拡張子によって C または C++言語文法に基づいてコンパイルします。起動コマンド「`shc`」のみ有効です。

1.3.4 オプションの組み合わせ

コンパイラオプションの組み合わせで、意味上矛盾するオプションやサブオプションを同時に指定した場合、どちらか一方が無効になります。表 1-4 にオプションの組み合わせを示します。

表 1-4 オプションの組み合わせ

No.	オプションの組み合わせ	
	有効となるオプション	無効となるオプション
1	<code>nolist</code>	<code>show</code>
2	<code>code = asmcode^{**}</code>	<code>debug^{**}</code>
3		<code>show = object</code>
4	<code>help</code>	すべてのオプション
5	<code>cpu = sh1</code>	<code>pic = 1</code>
6	<code>optimize = 0</code>	<code>loop</code>
7	<code>cpu = sh4</code>	<code>double = float</code> (<code>fpu = single</code> が指定されたものとしてコンパイルします)

No	有効となるオプション	無効となるオプション
8	cpu = sh4 以外	fpu = single double
9	cpu = sh4 以外	denormalization = on off
10	cpu = sh4 以外	round = nearest zero

*1:アセンブリソース出力時に debug オプションを指定すると、出力コード内に.LINE 制御命令を埋め込みます。.LINE 制御命令は、C/C++言語ソース行情報をデバッガに与えます。これによって、デバッグ時に対応する C/C++言語ソース行を表示することができます。ただし変数の値に関して C/C++言語レベルのデバッグはできません。

1.3.5 標準ライブラリとの対応

標準ライブラリには、74 種類があります。cpu オプション、pic オプション、endian オプション、double オプション、fpu オプション、denormalization オプションおよび round オプションの組み合わせにより表 1-5 に示すライブラリをリンクしてください。

表 1-5 標準ライブラリとコンパイルオプションの関係

ライブラリ名	コンパイラオプション						
	cpu	pic	endian	denormalization	round	fpu	double=float
shclib.lib	sh1	-	big	-	-	-	なし
shclibf.lib	sh1	-	big	-	-	-	あり
shcnpic.lib	sh2	0	big	-	-	-	なし
shcpic.lib	sh2	1	big	-	-	-	なし
shcnpicf.lib	sh2	0	big	-	-	-	あり
shcpicf.lib	sh2	1	big	-	-	-	あり
shc2enp.lib	sh2e	0	big	-	-	-	なし
shc2ep.lib	sh2e	1	big	-	-	-	なし
shc2enpf.lib	sh2e	0	big	-	-	-	あり
shc2epf.lib	sh2e	1	big	-	-	-	あり
shc3npb.lib	sh3	0	big	-	-	-	なし
shc3pb.lib	sh3	1	big	-	-	-	なし
shc3npl.lib	sh3	0	little	-	-	-	なし
shc3pl.lib	sh3	1	little	-	-	-	なし
shc3npbf.lib	sh3	0	big	-	-	-	あり
shc3pbf.lib	sh3	1	big	-	-	-	あり
shc3npplf.lib	sh3	0	little	-	-	-	あり
shc3plf.lib	sh3	1	little	-	-	-	あり

ライブラリ名	コンパイラオプション						
	cpu	pic	endian	denormalization	round	fpu	double=float
shcenpb.lib	sh3e	0	big	-	-	-	なし
shcepb.lib	sh3e	1	big	-	-	-	なし
shcenpl.lib	sh3e	0	little	-	-	-	なし
shcepl.lib	sh3e	1	little	-	-	-	なし
shcenpbf.lib	sh3e	0	big	-	-	-	あり
shcepbf.lib	sh3e	1	big	-	-	-	あり
shcenplf.lib	sh3e	0	little	-	-	-	あり
shceplf.lib	sh3e	1	little	-	-	-	あり
sh4nbmzz.lib	sh4	0	big	off	zero	なし	-
sh4pbmzz.lib	sh4	1	big	off	zero	なし	-
sh4nlmzz.lib	sh4	0	little	off	zero	なし	-
sh4plmzz.lib	sh4	1	little	off	zero	なし	-
sh4nbmdz.lib	sh4	0	big	on	zero	なし	-
sh4pbmdz.lib	sh4	1	big	on	zero	なし	-
sh4nlmdz.lib	sh4	0	little	on	zero	なし	-
sh4plmdz.lib	sh4	1	little	on	zero	なし	-
sh4nbmzn.lib	sh4	0	big	off	nearest	なし	-
sh4pbmzn.lib	sh4	1	big	off	nearest	なし	-
sh4nlmzn.lib	sh4	0	little	off	nearest	なし	-
sh4plmzn.lib	sh4	1	little	off	nearest	なし	-
sh4nbmdn.lib	sh4	0	big	on	nearest	なし	-
sh4pbmdn.lib	sh4	1	big	on	nearest	なし	-
sh4nlmdn.lib	sh4	0	little	on	nearest	なし	-
sh4plmdn.lib	sh4	1	little	on	nearest	なし	-
sh4nbfzz.lib	sh4	0	big	off	zero	single	-
sh4pbfzz.lib	sh4	1	big	off	zero	single	-
sh4nlfzz.lib	sh4	0	little	off	zero	single	-
sh4plfzz.lib	sh4	1	little	off	zero	single	-
sh4nbfzd.lib	sh4	0	big	on	zero	single	-
sh4pbfzd.lib	sh4	1	big	on	zero	single	-
sh4nlfdz.lib	sh4	0	little	on	zero	single	-
sh4plfdz.lib	sh4	1	little	on	zero	single	-
sh4nbfzn.lib	sh4	0	big	off	nearest	single	-
sh4pbfzn.lib	sh4	1	big	off	nearest	single	-

1. 概要・操作

ライブラリ名	コンパイラオプション						
	cpu	pic	endian	denormalization	round	fpu	double=float
sh4nlfzn.lib	sh4	0	little	off	nearest	single	-
sh4plfzn.lib	sh4	1	little	off	nearest	single	-
sh4nbfzn.lib	sh4	0	big	on	nearest	single	-
sh4pbfzn.lib	sh4	1	big	on	nearest	single	-
sh4nlfzn.lib	sh4	0	little	on	nearest	single	-
sh4plfzn.lib	sh4	1	little	on	nearest	single	-
sh4nbfzn.lib	sh4	0	big	off	zero	double	-
sh4pbfzn.lib	sh4	1	big	off	zero	double	-
sh4nldzn.lib	sh4	0	little	off	zero	double	-
sh4pldzn.lib	sh4	1	little	off	zero	double	-
sh4nbfzn.lib	sh4	0	big	on	zero	double	-
sh4pbfzn.lib	sh4	1	big	on	zero	double	-
sh4nldzn.lib	sh4	0	little	on	zero	double	-
sh4pldzn.lib	sh4	1	little	on	zero	double	-
sh4nbfzn.lib	sh4	0	big	off	nearest	double	-
sh4pbfzn.lib	sh4	1	big	off	nearest	double	-
sh4nldzn.lib	sh4	0	little	off	nearest	double	-
sh4pldzn.lib	sh4	1	little	off	nearest	double	-
sh4nbfzn.lib	sh4	0	big	on	nearest	double	-
sh4pbfzn.lib	sh4	1	big	on	nearest	double	-
sh4nldzn.lib	sh4	0	little	on	nearest	double	-
sh4pldzn.lib	sh4	1	little	on	nearest	double	-

1.3.6 コンパイルリストの見方

本項では、コンパイルリストの内容と形式について説明します。

(1) コンパイルリストの構成

コンパイルリストの構成と内容を表 1-6 に示します。

表 1-6 コンパイルリストの構成と内容

NO.	リストの構成	内容	オプション指定方法 ^{*1}	オプション省略時
1	ソースリスト情報	ソースプログラムリスト	show = [no]source	出力しない
		インクルードファイル、マクロ展開後のソースプログラムのリスト	(show = [no]include) ^{*2} (show = [no]expansion)	出力しない
2	オブジェクト情報	オブジェクトプログラムの機械語、アセンブリコード	show = [no]object	出力する
3	統計情報	エラーの総数、ソースプログラムの行数、セクションサイズ、シンボル数	show = [no]statistics	出力する
4	コマンド指定情報	コマンドで指定されたファイル名とオプション表示		出力する

*1: すべてのオプションは listfile 指定時に有効です。

*2: ()内は show = source 指定時に有効になります。

(2) ソースリスト情報

ソースリスト情報の出力形式には、プリプロセッサ展開前のソースプログラムを出力する形式(show = **noinclude**, **noexpansion** を指定する場合)とプリプロセッサ展開後のソースプログラムを出力する形式(show = **include**, **expansion** を指定する場合)があります。それぞれの出力形式を図 1-3、図 1-4 に示します。また、図 1 に相違点を網掛けで示します。

***** SOURCELISTING*****

FILENAME:m0260.c

Seq	File	Line	0---+---1---+---2---+---3---+---4---+---5---
1	m0260.c	1	#include"header.h"
4	m0260.c	2	
5	m0260.c	3	int sum2(void)
6	m0260.c	4	{ int j;
7	m0260.c	5	
8	m0260.c	6	#ifdef SMALL
9	m0260.c	7	j=SML_INT;
10	m0260.c	8	#else
11	m0260.c	9	j=LRG_INT;
12	m0260.c	10	#endif
13	m0260.c	11	
14	<u>m0260.c</u>	<u>12</u>	return j; /* continue123456789012345678901234567
(1)	(2)	(3)	±2345678901234567890 */
15	m0260.c	13	(7)}

図 1-3 show = noinclude,noexpansion のソースリスト情報

***** SOURCE LISTING *****

FILE NAME: m0260.c

Seq	File	Line	0-----1-----2-----3-----4-----5---
1	m0260.c	1	#include "header.h"
2	header.h	1	#define SML_INT 1 (4)
3	header.h	2	#define LRG_INT 100
4	m0260.c	2	
5	m0260.c	3	int sum2(void)
6	m0260.c	4	{ int j;
7	m0260.c	5	
8	m0260.c	6	#ifdef SMALL
9	m0260.c	7	X j=SML_INT;
10	m0260.c	8	(5) #else
11	m0260.c	9	E j=100;
12	m0260.c	10	(6) #endif
13	m0260.c	11	
14	m0260.c	12	return j; /* continue123456789012345678901234567
(1)	(2)	(3)	+2345678901234567890 */
		(7)	
15	m0260.c	13	}

図 1-4 show = include, expansion のソースリスト情報

- (1) リスト上の行番号
- (2) ソースプログラムファイル名またはインクルードファイル名
- (3) ソースプログラムまたはインクルードファイル内の行番号
- (4) show = include 指定時、インクルードファイルの展開のあったソース行
- (5) show = expansion 指定時、#ifdef 文、#elif 文等の条件コンパイル文でコンパイル対象とならないソース行
- (6) show = expansion 指定時、#define 文によるマクロ置換のあったソース行
- (7) ソースプログラムの 1 行がコンパイルリストの 1 行に入りきらず、複数行にまたがって表示されたソース行

(3) オブジェクト情報

オブジェクト情報の出力形式には、ソースプログラムを出力する形式(`show = source, object` を指定する場合)とソースプログラムを出力しない形式(`show = nosource, object` を指定する場合)があります。それぞれのリスト例を図 1-5 および図 1-6 に示します。

***** OBJECT LISTING *****

FILE NAME: m0251.c

SCT (1)	OFFSET (2)	CODE (3)	C LABEL	INSTRUCTION (4)	OPERAND (4)	COMMENT (5)
	m0251.c	1	extern int multipli(int);			
	m0251.c	2				
	m0251.c	3	int multipli(int x)			
P	00000000		_multipli:			; function: multipli ; frame size=16 (7) ; used runtime library name: ; __multi (8)
	00000000	4F22		STS.L	PR,R15	
	00000002	7FF4		ADD	#-12,R15	
	00000004	1F42		MOV.L	R4,@(8,R15)	
	m0251.c	4	{			
	m0251.c	5	int i;			
	m0251.c	6	int j;			
	m0251.c	7				
	m0251.c	8	j=1;			
	00000006	E201		MOV	#1,R2	
	00000008	2F22		MOV.L	R2,@R15	
	m0251.c	9	for(i=1; i<=x; i++){			
	0000000A	E301		MOV	#1,R3	
	0000000C	1F31		MOV.L	R3,@(4,R15)	
	0000000E	A009		BRA	L213	
	00000010	0009		NOP		
	00000012		L214:			
	m0251.c	10	j*=i;			
	00000012	50F1		MOV.L	@(4,R15),R0	
	00000014	61F2		MOV	@R15,R1	
	00000016	D30A		MOV.L	L216+2,R3	; __multi
	00000018	430B		JSR	@R3	
	.		.			
	.		.			

図 1-5 show = source,object のオブジェクト情報

***** OBJECT LISTING *****

FILE NAME: m0251.c

SCT (1)	OFFSET (2)	CODE (3)	C LABEL	INSTRUCTION (4)	OPERAND (4)	COMMENT (5)
P			: File m0251.c , Line 3			; block
	00000000		_multipli:			; function: multipli
						; <u>frame size=16</u> (7)
						; <u>used runtime library name:</u>
						; <u>_mul</u> (8)
	00000000	4F22		STS.L	PR,@R15	
	00000002	7FF4		ADD	#-12,R15	
	00000004	1F42		MOV.L	R4,@(8,R15)	
			; File m0251.c , Line 4			; block
			; File m0251.c , Line 8			; expression statement
	00000006	E201		MOV	#1,R2	
	00000008	2F22		MOV.L	R2,@R15	
			; File m0251.c , Line 9			; for
	0000000A	E301		MOV	#1,R3	
	0000000C	1F31		MOV.L	R3,@(4,R15)	
	0000000E	A009		BRA	L213	
	00000010	0009		NOP		
	00000012		L214:			
			; File m0251.c , Line 9			; block
			; File m0251.c , Line 10			; expression statement
	00000012	50F1		MOV.L	@(4,R15),R0	
	00000014	61F2		MOV.L	@R15,R1	
	00000016	D30A		MOV.L	L216+2,R3	; __mul
	00000018	430B		JSR	@R3	
	.		.			
	.		.			

図 1-6 show = nosource,object のオブジェクト情報

- (1) 各セクションのセクション属性(P、C、D、B、D_INIT_、D_END_、C_\$VTBL)
- (2) 各セクションの先頭からのオフセット
- (3) 各セクションのオフセットアドレスの内容
- (4) 機械語に対応するアセンブリコード
- (5) プログラムに対応するコメント(非最適化時だけ出力、ラベルだけ最適化時も出力)
- (6) プログラムの行情報(非最適化時だけ出力)
- (7) スタックフレームサイズ(バイト数)(最適化時も出力)
- (8) 使用している実行時ルーチン名の一覧

(4) 統計情報

統計情報の出力例を図 1-7 に示します。

```

***** STATISTICS INFORMATION *****

***** ERROR INFORMATION ***** ( 1 )

NUMBER OF ERRORS:          0
NUMBER OF WARNINGS:        0
NUMBER OF INFORMATIONS:    0

***** SOURCE LINE INFORMATION ***** ( 2 )

COMPILED SOURCE LINE:      13

***** SECTION SIZE INFORMATION ***** ( 3 )

PROGRAM   SECTION(P):      0x000044 Byte(s)
CONSTANT  SECTION(C):      0x000000 Byte(s)
DATA      SECTION(D):      0x000000 Byte(s)
BSS       SECTION(B):      0x000000 Byte(s)

TOTAL PROGRAM   SECTION(P): 0x000044 Byte(s)
TOTAL CONSTANT  SECTION(C): 0x000000 Byte(s)
TOTAL DATA     SECTION(D): 0x000000 Byte(s)
TOTAL BSS       SECTION(B): 0x000000 Byte(s)

TOTAL PROGRAM SIZE:      0x000044 Byte(s)

***** LABEL INFORMATION ***** ( 4 )

NUMBER OF EXTERNAL REFERENCE SYMBOLS:  1
NUMBER OF EXTERNAL DEFINITION SYMBOLS:  1
NUMBER OF INTERNAL/EXTERNAL SYMBOLS:    6

```

図 1-7 統計情報

- (1) レベル別メッセージの総数
- (2) ソースファイルのコンパイルした行数
- (3) 各セクションのサイズとその合計
- (4) オブジェクトプログラムの外部参照シンボルの数、外部定義シンボルの数、
内部ラベルと外部ラベルの合計数

【注】オプション `message` が指定されていない場合には、レベル別メッセージ(1)の `NUMBER OF INFORMATIONS` は出力されません。オプション `noobject` 指定時およびエラーレベル、フェータルレベルのエラーが発生した場合には、セクションサイズ情報(3)とラベル情報(4)を出力しません。また、オプション `code = asmcode` 指定時には、セクションサイズ情報(3)は当該セクションの有無を 0 と 1 で示すようになります。各セクションごとの領域のトータルサイズは、Ver.5.0 では出力されません。

(5) コマンド指定情報

コンパイラを起動したときのコマンドで指定されたファイル名とオプションを表示します。コマンド指定情報の出力例を図 1-8 に示します。

```
*** COMMAND PARAMETER ***
-listfile test.c
```

図 1-8 コマンド指定情報

1.3.7 コンパイラの環境変数

コンパイラで使用する環境変数の使用方法を表 1-7 に示します。

表 1-7 環境変数

No.	環境変数	説明
1	SHC_LIB	コンパイラのロードモジュールおよび、システムインクルードファイルを格納したディレクトリを指定します。PC 版で、DOS プロンプトからコマンド入力する場合には、コンパイラ本体をインストールしているディレクトリを設定してください。この環境変数の指定は必須です。
2	SHC_INC	システムインクルードファイル格納ディレクトリを指定します。ディレクトリはコンマで区切ることによって複数指定可能です。システムインクルードファイルの検索順序は include オプション指定ディレクトリ、SHC_INC 指定ディレクトリ、システムディレクトリ(SHC_LIB)となります。
3	SHC_TMP	コンパイラがテンポラリファイルを作成するディレクトリを指定します。PC 版で、DOS プロンプトよりコマンド入力する場合はこの環境変数の指定は必須です。UNIX 版ではこの環境変数の指定がない場合、環境変数 TMPDIR が指定されていれば、TMPDIR が示すディレクトリ。SHC_TMP、TMPDIR が指定されていなければ、/usr/tmp にテンポラリファイルを作成します。

No.	環境変数	説明
4	SHCPU	<p>コンパイラの <code>cpu</code> オプションによる CPU 種別の指定を、環境変数によって指定します。以下の指定が可能です。</p> <p>SHCPU = SH1 (cpu = sh1 オプションと同義) SHCPU = SH2 (cpu = sh2 オプションと同義) SHCPU = SH2E (cpu = sh2e オプションと同義) SHCPU = SHDSP (cpu = sh2 オプションと同義) SHCPU = SH3 (cpu = sh3 オプションと同義) SHCPU = SH3E (cpu = sh3e オプションと同義) SHCPU = SH4 (cpu = sh4 オプションと同義)</p> <p>上記以外の指定はエラーとなります。また、小文字もエラーとなります。 SHCPU 環境変数による CPU の指定と、<code>cpu</code> オプションによる CPU の指定が相反する場合は、ウォーニングメッセージを出力し、<code>cpu</code> オプションの指定を優先します。</p>

1.3.8 オプションによる暗黙の宣言

`cpu`, `pic`, `endian`, `double`, `fpu`, `denormalize`, `round` の各オプションを使用すると、以下のような暗黙の `#define` 宣言が行われます。

表 1-8 暗黙の宣言

項番	オプション	暗黙の宣言
1	<code>cpu = sh1</code>	<code>#define _SH1</code> (デフォルト時を含む)
2	<code>cpu = sh2</code>	<code>#define _SH2</code>
3	<code>cpu = sh2e</code>	<code>#define _SH2E</code>
4	<code>cpu = sh3</code>	<code>#define _SH3</code>
5	<code>cpu = sh3e</code>	<code>#define _SH3E</code>
6	<code>cpu = sh4</code>	<code>#define _SH4</code>
7	<code>pic = 1</code>	<code>#define _PIC</code>
8	<code>endian = big</code>	<code>#define _BIG</code> (デフォルト時を含む)
9	<code>endian = little</code>	<code>#define _LIT</code>
10	<code>double = float</code>	<code>#define _FLT</code>
11	<code>fpu = single</code>	<code>#define _FPS</code>
12	<code>fpu = double</code>	<code>#define _FPD</code>
13	<code>denormalize = on</code>	<code>#define _DON</code>
14	<code>round = nearest</code>	<code>#define _RON</code>

指定例および指定規則を以下に示します。

指定例：

```

#ifdef _BIG
#ifdef _SH1
.....      “cpu = sh1, endian = big “オプション指定時に有効
.....      (cpu、endianオプション指定のない場合にも有効)
#endif
#endif
#ifdef _SH2
.....      “cpu = sh2 “オプション指定時に有効
#endif
#ifdef _SH3
#ifdef _BIG
.....      “cpu = sh3,endian = big “オプション指定時に有効
#endif
#endif
#ifdef _LIT
.....      “cpu = sh3, endian = little “オプション指定時に有効
#endif
#endif

```

指定規則：

- (1) 各オプションの指定がない場合は、**#define _SH1**、**#define _BIG** を設定します。
- (2) 各暗黙の**#define** 宣言は、ソースファイル中で**#undef** 指定できます。

2. C/C++プログラミング

2.1 コンパイラの限界値

コンパイラがコンパイルできるソースプログラムの限界値を表 2-1 に示します。ソースプログラムを作成する場合は、この限界値の範囲内で作成してください。ソースプログラムの編集やコンパイル処理の効率を上げるためには、最大 2000 行程度までのプログラムに分割して、分割コンパイルをすることをお勧めします。

表 2-1 コンパイラの限界値

NO.	分類	項目	限界値
1	コンパイラの起動	一度にコンパイルできるソースプログラムの数	制限なし ^{*1}
2		define オプションで指定できるマクロ名の総数	制限なし
3		ファイル名の長さ	128 文字
4	ソースプログラムの行数	1 行の長さ	32768 文字
5		1 ファイルあたりのソースプログラムの行数	65535 行
6		コンパイル可能なソースプログラムの行数	制限なし
7	プリプロセッサ	#include 文によるファイルのネストの深さ	30 レベル
8		#define 文によるマクロ名の総数	制限なし
9		マクロ定義、マクロ呼び出しで指定できる引数の数	63 個
10		マクロ名の置き換えの数	32 回
11		#if、#ifdef、#ifndef、#else、#elif 文のネストの深さ	32 レベル
12		#if、#elif 文で指定できる演算子と被演算子の合計数	512 個
13	宣言	関数定義の数	制限なし
14		内部ラベルの数 ^{*2}	32767 個
15		シンボルテーブルエントリ数 ^{*3}	32767 個
16		基本型を修飾するポインタ型、配列型、関数型の合計数	16 個
17		配列の次元数	6 次元
18	文	構文のネストの深さ	32 レベル
19		繰り返し文(while 文、do 文、for 文)、選択文(if 文、switch 文)の組み合わせによる文のネストの深さ	32 レベル
20		一つの関数内で指定できる goto ラベルの数	511 個
21		switch 文の数	256 個
22		switch 文のネストの深さ	16 レベル
23		case ラベルの数	511 個
24		for 文のネストの深さ	16 レベル
25	式	関数定義、関数呼び出しで指定できる引数の数	63 個 ^{*4}
26		一つの式で指定できる演算子と被演算子の合計数	約 500 個
27	標準ライブラリ	open 関数で一度にオープンできるファイルの数	20 個

*1:ただし、PC 版の DOS プロンプトではコマンドラインの制約により 127 文字までの入力となります。

*2:内部ラベルとは、コンパイラが内部で生成するラベルであり、静的変数の領域を指すアドレス、繰り返し文や選択文で処理の流れが分岐する先のアドレス、case ラベルや goto ラベルのアドレスなどのことです。

*3:シンボルテーブルエントリ数を概算する式を以下に示します。

C コンパイルの場合：

外部名の数+関数ごとの内部名の数+文字列の数+複文内の構造体・配列の初期値
+複文の数+case ラベルの数+goto ラベルの数+typedef 名の数
+構造体/共用体/enum タグの数+構造体/共用体/enum メンバの数
+関数原形宣言のパラメタ数

C++コンパイルの場合：

外部名の数+関数ごとの内部名の数+文字列の数+複文内の構造体・配列の初期値
+複文の数+case ラベルの数+goto ラベルの数
+(デフォルトコンストラクタ/デストラクタの数)
+(コピーコンストラクタ/代入演算子の数)+(仮想関数表の数)
+クラス名/enum タグの数+クラス/enum メンバの数
+関数のプロトタイプ宣言のパラメタ数

() で表しているものは C++コンパイル時に、コンパイラで自動生成する外部名です。

仮想関数表が生成される場合については、「第 2 章 C/C++プログラミング 2.2.2 データの内部表現」を参照してください。

*4:非静的関数メンバのときは、62 個になります。

2.2 C/C++プログラムの実行方式

本節では、コンパイラが生成するオブジェクトプログラムについて説明します。特に、C/C++プログラムとアセンブリプログラムを結合する場合、C++プログラムと C プログラムを結合する場合や、SuperH マイコンを用いたシステムにプログラムを組み込む場合に必要となる事項について説明しています。

本節で述べる項目は以下のとおりです。

2.2.1 オブジェクトプログラムの構造

C/C++プログラム、標準ライブラリが使用するメモリ領域の性質について述べます。各セクションをメモリ領域に割り付けるときに必要です。

2.2.2 データの内部表現

C/C++プログラムが用いるデータ型のメモリ上での表現について述べます。C/C++プログラムとハードウェア、アセンブリプログラムの間でデータを相互参照するときが必要です。

2.2.3 C プログラムとの結合

C++プログラムから C の関数を呼び出したり、C のデータを参照する機能について述べます。

2.2.4 アセンブリプログラムとの結合

C/C++プログラムで使用する変数名や関数名のうち、他のオブジェクトプログラムとの間で相互に参照できる名前の規則について述べます。また、C/C++プログラムの関数呼び出しでの引数やリターン値の受け渡し方法、レジスタの使用法に関する規則について述べます。これらの規則は、C/C++プログラムの関数とアセンブリプログラムのルーチン間で相互に呼び出しや参照を行うときに必要です。

本節では、SuperH マイコンのハードウェアの知識を必要としますので、ハードウェアマニュアルをあわせてお読みください。

2.2.1 オブジェクトプログラムの構造

C/C++プログラム、標準ライブラリが使用するメモリ領域の性質には、以下のものがあります。

セクション

メモリ領域のうち、本コンパイラが静的に割り付ける領域は、セクションを構成します。セクションにはセクション名とセクション種別があります。セクション名はコンパイラオプション `section` や拡張言語仕様 `#pragma section` で変更することができます。

書き込み操作

プログラム実行時における書き込み操作の可/不可を示します。

初期値の有無

プログラム実行開始時の初期値の有無です。

境界調整数

データを割り付けるアドレスに関する制約です。

C/C++プログラム、標準ライブラリが使用するメモリ領域の種類とその性質の概要を表 2-2 に示します。

表 2-2 メモリ領域の種類とその性質の概要

No.	名称	セクション名 ^{*1}	種別	書き込み	初期値	境界調整数	内容
1	プログラム領域 (C,C++)	P	code	不可	有	4byte ^{*2}	機械語を格納する。
2	定数領域(C,C++)	C	data	不可	有	4/8byte ^{*4}	const 型のデータを格納する。 ^{*3}
3	初期化データ領域 (C,C++)	D	data	可	有	4/8byte ^{*4}	初期値データを格納する。
4	未初期化データ領域 (C,C++)	B	data	可	無	4/8byte ^{*4}	初期値のないデータを格納する。
5	スタック領域 (C,C++)	-	-	可	無	4byte	プログラムの実行に必要な領域。「第3章 システム組み込み 3.2.2 動的領域の割り付け」参照。
6	ヒープ領域(C,C++)	-	-	可	無	-	ライブラリ関数(malloc、realloc、calloc、new)で使用する領域。「第3章 システム組み込み 3.2.2 動的領域の割り付け」参照。
7	初期処理データ領域 (C++)	D_INIT_	data	不可	有	4/8byte ^{*4}	グローバルクラスオブジェクトに対して呼び出されるコンストラクタのアドレスを格納する。
8	後処理データ領域 (C++)	D_END_	data	不可	有	4/8byte ^{*4}	グローバルクラスオブジェクトに対して呼び出されるデストラクタのアドレスを格納する。
9	仮想関数表領域 (C++)	C_\$VTBL	data	不可	有	4/8byte ^{*4}	クラス宣言中に仮想関数があるときに仮想関数をコールするためのデータを格納する。

*1:セクション名はコンパイラオプション -section で特定の名前を指定しないときに本コンパイラがデフォルトで作成する名前を示します。

*2:-align16 オプションを指定している場合、16byte になります。

*3:C++コンパイルで、const 指定をしても定数領域に割り付かないものとして、キャストを含む初期値、仮想基底クラスや仮想関数を含むクラスオブジェクトがあります。初期値があるときは初期化データ領域、初期値がないときは未初期化データ領域に割り付きます。

*4:-cpu=sh4 オプションが指定されたとき、境界調整数は 8byte になります。

例1 Cプログラムとコンパイラが生成する領域との対応についてプログラム例を用いて示します。

```
int a=1;
char b;
const int c=0;
main(){
    ...
}
```

file.c

Cプログラム

プログラム領域(main(){...})

定数領域(c)

初期化データ領域(a)

未初期化データ領域(b)

コンパイラが生成する領域と
格納されるデータ

例2 C++プログラムとコンパイラが生成する領域との対応についてプログラム例を用いて示します。

```
class A{
    int m;
    A(int p);
    ~A();
};
A a(1);
int b;
extern const char c=`a`;
int d=1;
void f(){...}
```

file.cpp

C++プログラム

プログラム領域(f(){...})

定数領域(c)

初期化データ領域(a,d)

未初期化データ領域(b)

初期処理データ領域(&A::A)

後処理データ領域(&A::~A)

コンパイラが生成する領域と
格納されるデータ

2.2.2 データの内部表現

本項では、C/C++言語のデータ型とデータの内部表現の対応について述べます。データの内部表現は以下の項目から成り立っています。

(a) データのサイズ

データが占有する領域のサイズです。

(b) データの境界調整数

データを割り付けるアドレスに関する制約です。任意のアドレスに割り付ける 1 バイト境界調整、偶数バイトに割り付ける 2 バイト境界調整、4 の倍数バイトに割り付ける 4 バイト境界調整があります。

(c) データの範囲

スカラ型(C 言語), 基本型(C++言語)の値がとり得る範囲を示します。

(d) データの割り付け例

複合型(C 言語)、クラス型(C++言語)の要素となるデータの割り付け例を示します。

(1) スカラ型(C 言語)、基本型(C++言語)

C 言語におけるスカラ型および、C++言語における基本型の内部表現を表 2-3 に示します。

表 2-3 スカラ型、基本型の内部表現

No.	データ型	サイズ (バイト)	境界整合数 (バイト)	符号の 有無	最小値	最大値
1	char (signed char)	1	1	有	$-2^7(-128)$	$2^7-1(127)$
2	unsigned char	1	1	無	0	$2^8-1(255)$
3	short	2	2	有	$-2^{15}(-32768)$	$2^{15}-1(32767)$
4	unsigned short	2	2	無	0	$2^{16}-1(65535)$
5	int	4	4	有	$-2^{31}(-2147483648)$	$2^{31}-1(2147483647)$
6	unsigned int	4	4	無	0	$2^{32}-1(4294967295)$
7	long	4	4	有	$-2^{31}(-2147483648)$	$2^{31}-1(2147483647)$
8	unsigned long	4	4	無	0	$2^{32}-1(4294967295)$
9	enum	4	4	有	$-2^{31}(-2147483648)$	$2^{31}-1(2147483647)$
10	float	4^{*3}	4	有	-	+
11	double long double	$8^{*1,*3}$	4	有	-	+
12	ポインタ	4	4	無	0	$2^{32}-1(4294967295)$
13	bool ^{*2}	1	1	有	$-2^7(-128)$	$2^7-1(127)$
14	リファレンス ^{*2}	4	4	無	0	$2^{32}-1(4294967295)$
15	データメンバへのポ インタ ^{*2}	4	4	有	0	$2^{32}-1(4294967295)$
16	関数メンバへのポイ ンタ ^{*2,*4}	12	4	-	-	-

*1:double = float オプションを指定している場合、double 型のサイズは 4 バイトになります。

*2:C++コンパイルのみ有効です。Ver.5.0 ではサポートしておりません。

*3:cpu=sh4 かつ fpu = single を指定している場合、double,long double 型を 4 バイト(float 型)として扱いました、cpu = sh4 かつ fpu = double を指定している場合、float 型を 8 バイト(double 型)として扱います。

*4:関数メンバへのポインタは、以下のクラスで表現しています。

```

class _PMF{
public:
size_t delta;//オブジェクトのオフセット値
int index;//対象メンバ関数が仮想関数のときの仮想関数表中でのインデックス
union{
    int (*_deffun());//対象メンバ関数が非仮想関数のときの関数のアドレス
    size_t vt_offset;//対象メンバ関数が仮想関数のときの仮想関数表のオブジェクト
        //    中のオフセット
};
};

```



(2) 複合型(C 言語)、クラス型(C++言語)

本項では、C 言語における配列型、構造体型、共用体型および、C++言語におけるクラス型の内部表現について説明します。

表 2-4 に複合型、クラス型の内部表現を示します。

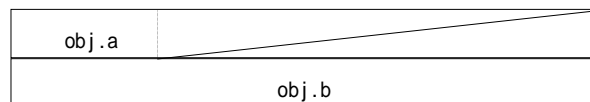
表 2-4 複合型、クラス型の内部表現

No.	データ型	境界調整数 (byte)	サイズ (byte)	データの割り付け例
1	配列型	配列要素の 境界調整数	配列要素の数 × 要素サイズ	int a[10]; : 境界調整数 4byte サイズ 40byte
2	構造体型 ¹⁾	構造体メンバの 境界調整数の うち最大値	メンバのサイズの 和 ¹⁾	struct{ int a,b; : 境界調整数 4byte サイズ 8byte };
3	クラス型	データメンバと 仮想関数へのポ インタ、仮想基 底クラスへのポ インタの境界調 整数(4byte)のう ちの最大値	データメンバのサイ ズの和+仮想関数表 へのポインタ数 × 4byte+仮想基底クラ スへのポインタ数 × 4byte	class A{ int a,b; :境界調整数 4byte virtual void f(); サイズ 12byte };
4	共用体型	共用体メンバの 境界調整数の うち最大値	メンバのサイズの 最大値 ²⁾	union { : 境界調整数 4byte int a,b; サイズ 4byte };

以下の例で  は、4 バイトを表わしています。  は境界調整領域を表しています。

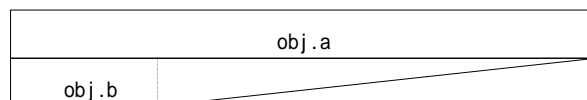
*1:構造体型の各メンバを割り付ける時、そのメンバのデータ型の境界調整数に合わせるために直前のメンバとの間に空き領域(境界調整領域)が生じる場合があります。

```
struct {
    char a;
    int b;}obj;
```



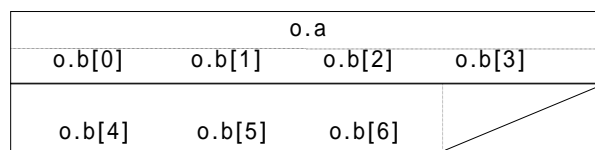
また、構造体/クラスが4バイトの境界調整数を持つ場合、最後のメンバが1,2,3 バイト目で終わっているとき、残りのバイトも含めて構造体型の領域として扱います。

```
struct {
    int a;
    char b;}obj;
```



*2:共用体が4バイトの境界調整数を持つ場合、メンバのサイズの最大値が4の倍数バイトでないとき、4の倍数になるまで残りのバイトも含めて共用体型の領域として扱います。

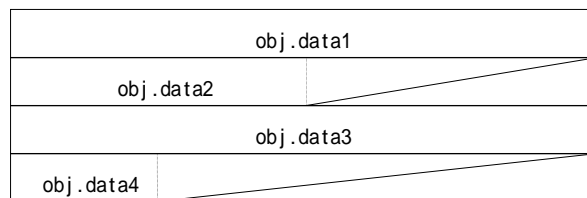
```
union {
    int a;
    char b[7];}o;
```



*3:クラス型割り付け例を示します。

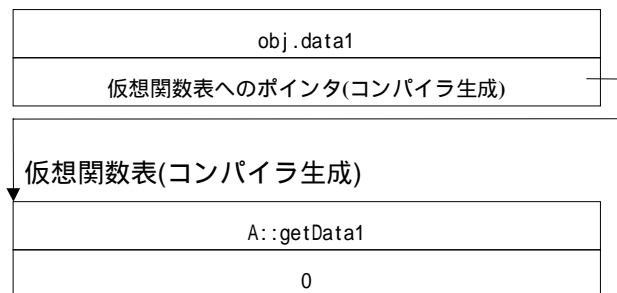
(a)仮想基底クラス、基底クラス、仮想関数がないクラスの場合

```
class A{
    int data1;
    short data2;
    int data3;
    char data4;
public:
    A();
    int getData1(){return data1;}
}obj;
```



(b)仮想関数があるクラスの場合

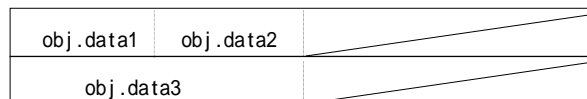
```
class A{
    int data1;
public:
    virtual int getData1();
}obj;
```



(c)基底クラスがあるクラスの場合

```
class A{
    char data1;
    char data2;
};

class B:public A{
    short data3;
}obj;
```

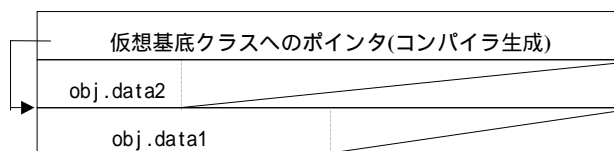


基底クラスのあるクラスの境界調整数は、4 バイトになります。

(d)仮想基底クラスがあるクラスの場合

```
class A{
    short data1;
};

class B: virtual protected A{
    char data2;
}obj;
```



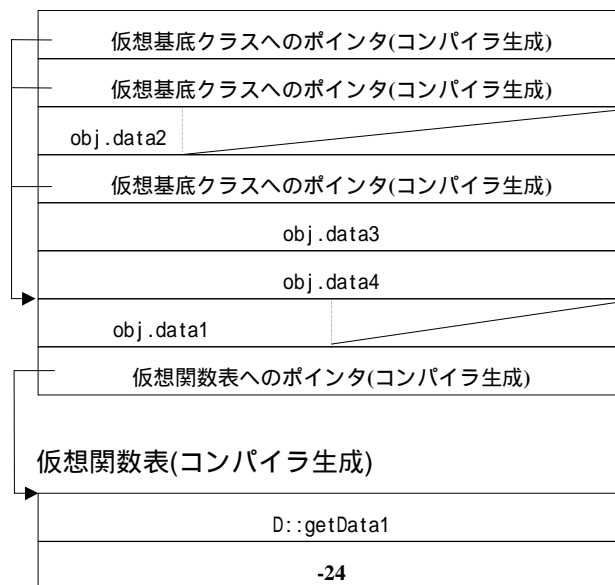
(e) 仮想基底クラス、基底クラス、仮想関数があるクラスの場合

```
class A{
    short data1 ;
    virtual short getData1();
};

class B:virtual public A{
    char data2;
    char getData2();
    short getData1();
};

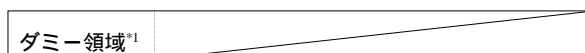
class C:virtual protected A{
    int data3;
};

class D:virtual public A,public B,public C{
public:
    int data4;
    short getData1();
}obj;
```



(f) 空クラス、単独クラスの場合

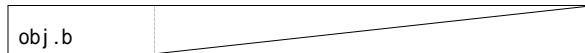
```
class A{
    void fun();
}obj;
```



*1: 「空クラスのサイズが0ではない(C++言語仕様で規定)」を実現するため、コンパイラがサイズ1バイトのデータを埋め込みます。

(g)空クラスを基底クラスに持つ場合

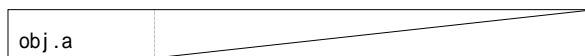
```
class A{
    void fun();
};
class B: A{
    char b;
}obj;
```



派生クラスで非静的データメンバが宣言されているときには、基底クラスのダミー領域はオブジェクト中に確保されません。

(h)空クラスを派生クラスに持つ場合

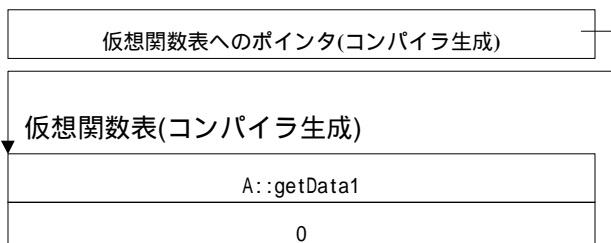
```
class A{
    char a;
};
class B:A{
}obj;
```



基底クラスで非静的データメンバが宣言されているときには、派生クラスのダミー領域はオブジェクト中に確保されません。

(i)仮想関数を持つ空クラスの場合

```
class A{
public:
    virtual int getData1();
}obj;
```



(3) ビットフィールド

ビットフィールドは、構造体、クラスの中にビット幅を指定して割り付けるメンバです。本項ではビットフィールド特有の割り付け規則について説明します。

(a) ビットフィールドのメンバ

表 2-5 にビットフィールドメンバの仕様を示します。

表 2-5 ビットフィールドメンバの仕様

No.	項目	仕様
1	ビットフィールドで許される型指定子	(signed)char, unsigned char, char ^{*1} , bool ^{*1} (signed)short, unsigned short, enum (signed)int, unsigned int (signed)long, unsigned long
2	宣言された型に拡張するときの符号の扱い ^{*2}	符号なし(unsigned を指定した型) ゼロ拡張 ^{*3} 符号あり(unsigned を指定しない型) 符号拡張 ^{*4}

*1: C++コンパイルのときのみ signed char, unsigned char, char, bool を指定できます。

*2: ビットフィールドのメンバを使用する場合、ビットフィールドに格納したデータを宣言した型に拡張して使用します。符号付き(signed)で宣言されたサイズが 1 ビットのビットフィールドのデータは、データそのものを符号として解釈します。したがって、表現できる値は 0 と -1 だけになります。0 と 1 を表現する場合には、必ず符号なし(unsigned)で宣言してください。

*3: ゼロ拡張: 拡張するときに上位のビットにゼロを補います。

*4: 符号拡張: 拡張するときにビットフィールドデータの最上位ビットを符号として解釈し、データより上位のビット全てに符号ビットを補います。

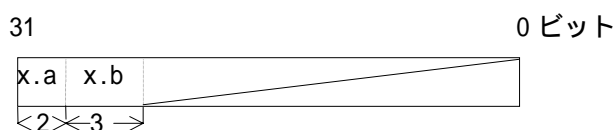
(b) ビットフィールドの割り付け方

ビットフィールドは、以下の 5 つの規則に従って割り付けます。

- (1) ビットフィールドのメンバは領域内で左(上位ビット側)から順に詰め込みます。

例

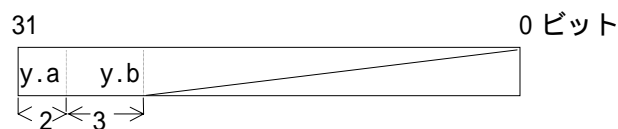
```
struct b1{
    int a:2;
    int b:3;
}x;
```



- (2) 同じサイズの型指定子が連続している場合、可能な限り同じ領域に詰め込みます。

例

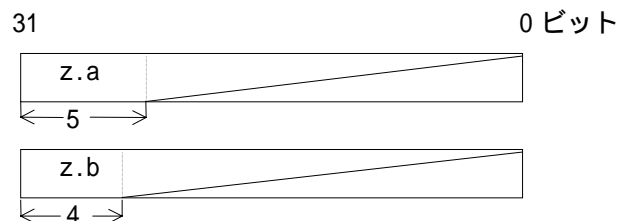
```
struct b1{
    long    a:2;
    unsigned int b:3;
}y;
```



- (3) 異なるサイズの型指定子で宣言されたメンバは、次の領域に割り付けます。

例

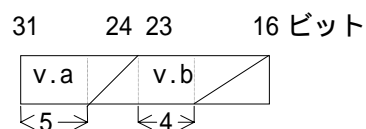
```
struct b1{
    int    a:5;
    char   b:4;
}z;
```



- (4) 同じサイズの型指定子が連続していても、詰め込み先の領域の残りビットが次のビットフィールドのサイズより小さい場合は、残りの領域は未使用領域となり、次の領域に割り付けます。

例

```
struct b2{
    char   a:5;
    char   b:4;
}v;
```

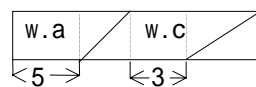


- (5) ビット幅 0 のビットフィールドのメンバを指定すると、次のメンバは強制的に次の領域に割り付けます。

例

```
struct b2{
    char a:5;
    char :0;
    char c:3;
}w;
```

31 24 23 16 ビット



(4) little endian のメモリ割り付け

little endian でのメモリ上のデータ配列は以下のとおりです。

- (a) 1 バイトデータ((signed) char, unsigned char, bool 型)

1 バイトデータの中のビット並び順は、**big endian** の場合も、**little endian** の場合も同じです。

- (b) 2 バイトデータ((signed) short, unsigned short 型)

2 バイトデータの中のバイト並び順は、上位、下位のバイトが逆になります。

例

0x100 番地に 2 バイトデータ 0x1234 がある場合：

big endian:	0x100 番地: 0x12	little endian:	0x100 番地: 0x34
	0x101 番地: 0x34		0x101 番地: 0x12

- (c) 4 バイトデータ((signed) int, unsigned int, (signed) long, unsigned long, float 型)

4 バイトデータの中のバイト並び順は、**big endian** と **little endian** で 4 バイトのデータの順序が逆になります。

例

0x100 番地に 4 バイトデータ 0x12345678 がある場合：

big endian:	0x100 番地: 0x12	little endian:	0x100 番地: 0x78
	0x101 番地: 0x34		0x101 番地: 0x56
	0x102 番地: 0x56		0x102 番地: 0x34
	0x103 番地: 0x78		0x103 番地: 0x12

(d) 8 バイトデータ(double 型)

8 バイトデータの中のバイト並び順は、**big endian** と **little endian** で 8 バイトのデータの順序が逆になります。

例

0x100 番地に 8 バイトデータ 0x0123456789abcdef がある場合：

big endian: 0x100 番地: 0x01	little endian: 0x100 番地: 0xef
0x101 番地: 0x23	0x101 番地: 0xcd
0x102 番地: 0x45	0x102 番地: 0xab
0x103 番地: 0x67	0x103 番地: 0x89
0x104 番地: 0x89	0x104 番地: 0x67
0x105 番地: 0xab	0x105 番地: 0x45
0x106 番地: 0xcd	0x106 番地: 0x23
0x107 番地: 0xef	0x107 番地: 0x01

(e) 複合型、クラス型データ

複合型、クラス型データの各メンバの割り付けは **big endian** のときと同様です。ただし、各メンバのバイト並び順はそのデータサイズの規則にしたがって反転します。

例

0x100 番地に、

```
struct {  
    short a;  
    int b;  
}z = {0x1234, 0x56789abc};
```

がある場合

big endian:	0x100 番地: 0x12	little endian:	0x100 番地: 0x34
	0x101 番地: 0x34		0x101 番地: 0x12
	0x102 番地: 空き領域		0x102 番地: 空き領域
	0x103 番地: 空き領域		0x103 番地: 空き領域
	0x104 番地: 0x56		0x104 番地: 0xbc
	0x105 番地: 0x78		0x105 番地: 0x9a
	0x106 番地: 0x9a		0x106 番地: 0x78
	0x107 番地: 0xbc		0x107 番地: 0x56

(f) ビットフィールド

ビットフィールドの各領域の割り付けも **big endian** のときと同様です。ただし、各領域のバイト並び順はそのデータサイズの規則に従って反転します。

例

0x100 番地に、

```
struct {  
    long a:16;  
    unsigned int b:15;  
    short c:5  
}y={1,1,1};
```

がある場合

big endian:	0x100 番地: 0x00	little endian:	0x100 番地: 0x02
	0x101 番地: 0x01		0x101 番地: 0x00
	0x102 番地: 0x00		0x102 番地: 0x01
	0x103 番地: 0x02		0x103 番地: 0x00
	0x104 番地: 0x08		0x104 番地: 0x00
	0x105 番地: 0x00		0x105 番地: 0x08
	0x106 番地: 空き領域		0x106 番地: 空き領域
	0x107 番地: 空き領域		0x107 番地: 空き領域

2.2.3 C プログラムとの結合

C++プログラムの中から C の関数を呼び出したり、C のデータを参照するための機能を用意しています。この機能を使うことにより、既存の C プログラム資産や C ライブラリを有効に活用することができます。C の関数参照する場合は、リンケージ指定子 `extern` “C”を使用します。グローバル変数の参照は、`extern` 宣言を行なってください C++プログラムから C の関数を呼び出す例を以下に示します。

例

```
extern "C" int f(int); //関数 f が C プログラムの関数であることを指定します。
extern int data; //C プログラムのグローバル変数
void g()
{
    ...
    a = f(1); //C の関数 f を呼び出します。
    data = 2; //C プログラムのグローバル変数を参照します。
    ...
}
```

2.2.4 アセンブリプログラムとの結合

コンパイラは、SuperH マイコン固有のレジスタへのアクセス等の機能を組み込み関数としてサポートしています(組み込み関数についての詳細は、「第 2 章 C/C++プログラミング 2.3.2 組み込み関数」を参照してください)。しかし、MAC 命令による積和演算など C/C++言語で記述できない処理はアセンブリ言語で記述し、C/C++言語と結合する必要があります。

本項では、C/C++プログラムとアセンブリプログラムの結合時に注意すべき以下の内容について述べます。

- ・ 外部名の相互参照方法
- ・ 関数呼び出しのインタフェース

2.2.4.1 外部名の相互参照方法

C/C++プログラムの中で外部名として宣言されたものは、アセンブリプログラムとの間で相互に参照あるいは更新することができます。コンパイラは、次のものを外部名として扱います。

- ・グローバル変数、かつ static 記憶クラスでないもの(C/C++プログラム)
- ・extern 記憶クラスで宣言されている変数名(C/C++プログラム)
- ・static 記憶クラスを指定されていない関数名(C プログラム)
- ・static 記憶クラスを指定されていない非メンバ非インライン関数名(C++プログラム)
- ・非インラインメンバ関数名(C++プログラム)
- ・静的データメンバ名(C++プログラム)

外部名となる変数名をアセンブリプログラムで指定する場合は、C/C++プログラム内の名前(最大 250 文字までが有効です)の先頭に下線(_)をつけたものになります。

例 1 アセンブリプログラムの外部名を C/C++プログラムで参照する方法

- ・アセンブリプログラムでは、「**.EXPORT**」制御命令を用いてシンボル名(先頭に下線(_)を付与)を外部定義宣言します。
- ・C/C++プログラムでは、シンボル名(先頭に下線(_)がない)を「**extern**」宣言します。

アセンブリプログラム(定義する側)

```
.EXPORT _a, _b
.SECTION D,DATA,ALIGN=4
_a: .DATA.L 1
_b: .DATA.L 1
.END
```

C/C++プログラム(参照する側)

```
extern int a,b;

void f()
{
    a+=b;
}
```

例2 C/C++プログラムの外部名をアセンブリプログラムから参照する方法

- ・ C/C++プログラムでは、シンボル名(先頭に下線(_)がない)を外部定義(グローバル変数)とします。

- ・ アセンブリプログラムでは、「**.IMPORT**」制御命令を用いてシンボル名(先頭に下線(_)を付与)を外部参照宣言します。

C/C++プログラム(定義する側)

```
int a;
```

アセンブリプログラム(参照する側)

```
.IMPORT    _a
.SECTION   P, CODE, ALIGN=2
MOV.L     A_a, R1
MOV.L     @R1, R0
ADD       #1, R0
RTS
MOV.L     R0, @R1
.ALIGN    4
A_a: .DATA.L    _a
.END
```

2.2.4.2 関数の呼び出し

C/C++プログラムとアセンブリプログラム間で相互に関数呼び出しを行うときに、アセンブリプログラム側で守るべき次の4つの規則について説明します。

- (1) スタックポインタに関する規則
- (2) スタックフレームの割り付け、解放に関する規則
- (3) レジスタに関する規則
- (4) 引数とリターン値の設定、参照に関する規則

(1) スタックポインタに関する規則

スタックポインタの指すアドレスよりも下位(0番地の方向)のスタック領域に有効なデータを格納してはいけません。スタックポインタより下位アドレスに格納されたデータは、割り込み処理で破壊される可能性があります。

(2) スタックフレームの割り付け、解放に関する規則

関数呼び出しが行われた時点(**JSR** または **BSR** 命令の実行直後)では、スタックポインタは呼び出した関数側で使用したスタックの最下位アドレスを指しています。このスタックポインタの指している領域より上位アドレスのデータの割り付け、設定は呼び出す側の関数の役目です。

関数のリターン時は、呼び出された関数で確保した領域を解放してから、通常 **RTS** 命令を用いて呼び出した関数へ返ります。これより上位アドレスの領域(リターン値アドレスおよび引数の領域)は、呼び出した側の関数で解放します。

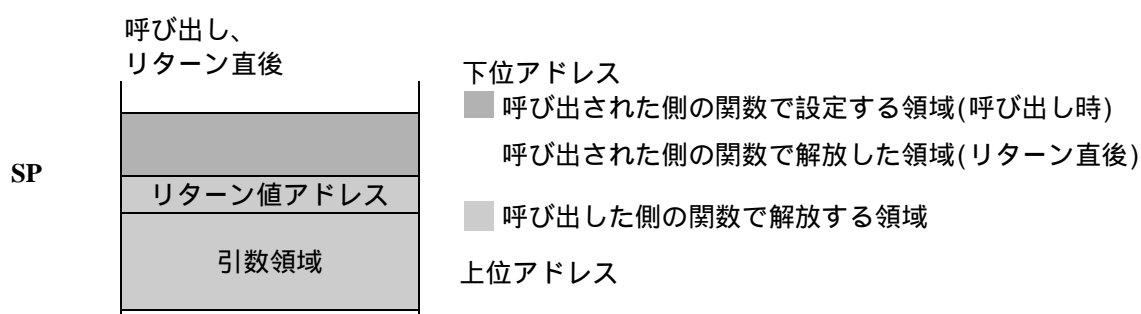


図 2-1 スタックフレームの割り付け、解放に関する規則

(3) レジスタに関する規則

関数呼び出し前後においてレジスタの値が同一であることを保証するレジスタと保証しないレジスタがあります。レジスタの保証規則を表 2-6 に示します。

表 2-6 関数呼び出し前後のレジスタ保証規則

No	項目	対象レジスタ	プログラミングにおける注意点
1	保証しないレジスタ	R0 ~ R7 FR0 ~ FR11 ^{*1,*2} DR0 ~ DR10 ^{*2} FPUL ^{*1,*2} , FPSCR ^{*1,*2}	関数呼び出し時に対象レジスタに有効な値があれば、呼び出し側で値を退避する。呼び出される側の関数では退避せずに使用できる。
2	保証するレジスタ	R8 ~ R15 MACH, MACL, PR FR12 ~ FR15 ^{*1,*2} DR12 ~ DR14 ^{*2}	対象レジスタのうち関数内で使用するレジスタの値を退避し、リターン時に回復する。ただし、macsave = 0 オプション指定時は MACH、MACL は保証しないレジスタ。

*1:SH2E,SH3E,SH4 の単精度浮動小数点用レジスタです。

*2:SH4 の倍精度浮動小数点用レジスタです。

以下、レジスタの保証規則の具体例を示します。

(a) アセンブリプログラムのサブルーチンを C/C++プログラムから呼び出す場合
アセンブリプログラム（呼び出される側）

```
.EXPORT _sub
.SECTION P, CODE, ALIGN=4
_sub: MOV.L    R14, @-R15
      MOV.L    R13, @-R15
      ADD     #-8, R15

      :

      ADD     #8, R15
      MOV.L    @R15+, R13
      RTS
      MOV.L    @R15+, R14
      .END
```

関数内で使用するレジスタの退避

関数本体の処理

(R0 ~ R7は関数呼び出し側で退避レジスタのため、関数内では退避せずに使用可能)

退避したレジスタの回復

C/C++プログラム（呼び出す側）

```
#ifdef __cplusplus
extern "C"
#endif
void sub();

void f()
{
    sub();
}
```

(b) C/C++プログラムの関数をアセンブリプログラムから呼び出す場合

C/C++プログラム（呼び出される側）

```
void sub ( )
{
    :
}
```

アセンブリプログラム（呼び出す側）

```
.IMPORT _sub
.SECTION P, CODE, ALIGN=4
:

STS.L    PR, @-R15
MOV.L    R1, @(1, R15)
MOV      R3, R12
MOV.L    A_sub, R0
JSR      @R0
NOP
LDS.L    @R15+, PR
:
A_sub: .DATA.L _sub
.END
```

呼び出す関数名の先頭に `_` を付けたものを「.IMPORT」制御命令で宣言(C)

コンパイラが関数宣言/定義から生成した外部名*を「.IMPORT」制御命令で宣言(C++)
外部名の生成規則は、「付録G エンコード規則」を参照してください。

関数呼び出しする場合は、PRレジスタ(リターンアドレス格納レジスタ)を退避
レジスタR0～R7に有効な値があれば空きレジスタまたはスタックに退避

関数「sub」の呼び出し

*:関数名、静的データメンバから生成する外部名は、C++コンパイルのとき一定の規則で変換を行っています。コンパイラが生成した外部名を知る必要があるときは、コンパイラオプション `code=asm` または、`listfile` にてコンパイラが生成する外部名を参照してください。「付録G エンコード規則」もあわせて参照してください。また、C++の関数を「extern "C"」を付与して関数定義を行えば、外部名はCの関数と同様の生成規則になります。ただし、その関数を多重定義できなくなります。

(4) 引数とリターン値の設定、参照に関する規則

以下、引数とリターン値の設定、参照方法について説明します。解説では、まず引数とリターン値に対する一般的な規則について述べたあと、引数の割り付け方とリターン値の設定場所について述べます。

(a) 引数とリターン値に対する一般的な規則

(i) 引数の渡し方

引数の値を、必ずレジスタまたはスタック上の引数の割り付け領域にコピーしたあとで関数を呼び出します。呼び出した側の関数では、リターン後に引数の割り付け領域を参照することはありませんので、呼び出された側の関数で引数の値を変更しても呼び出した側の処理は直接には影響を受けません。

(ii) 型変換の規則

引数を渡す場合、またはリターン値を返す場合、自動的に型変換を行う場合があります。以下、この型変換の規則について説明します。

(ア) 型の宣言された引数の型変換

プロトタイプ宣言によって型が宣言されている引数は、宣言された型に変換します。

(イ) 型の宣言されていない引数の型変換

プロトタイプ宣言によって型が宣言されていない引数の型変換は、以下の規則に従って変換します。

- ・(signed) char 型、unsigned char 型、(signed) short 型、unsigned short 型の引数は、(signed) int 型に変換します。
- ・float 型の引数は、double 型に変換します。
- ・上記以外の引数は、変換しません。

(ウ) リターン値の型変換

リターン値は、その関数の返す型に変換します。

例

(1)

```
long f( );  
long f( )  
{  
    float x;  
    return x; ← プロトタイプ宣言にしたがってリターン値はlong型に型変換  
}
```

されます。

(2)

```
void p( int, ... );  
void f( )  
{  
    char c;  
    p(1.0, c);  
}
```

→ cは、対応する引数の型宣言がないので、int型に変換されます。

→ 1.0は、対応する引数の型がint型なので、int型に変換されます。

(b) 引数の割り付け領域

引数は、レジスタに割り付ける場合とレジスタに割り付けられないときスタック上の引数領域に割り付ける場合があります。引数の割り付け領域を図 2-2 に、引数割り付け領域の一般規則を表 2-7 にそれぞれ示します。C++プログラムの非静的関数メンバの **this** ポインタは、**R4** に割り付けられます。

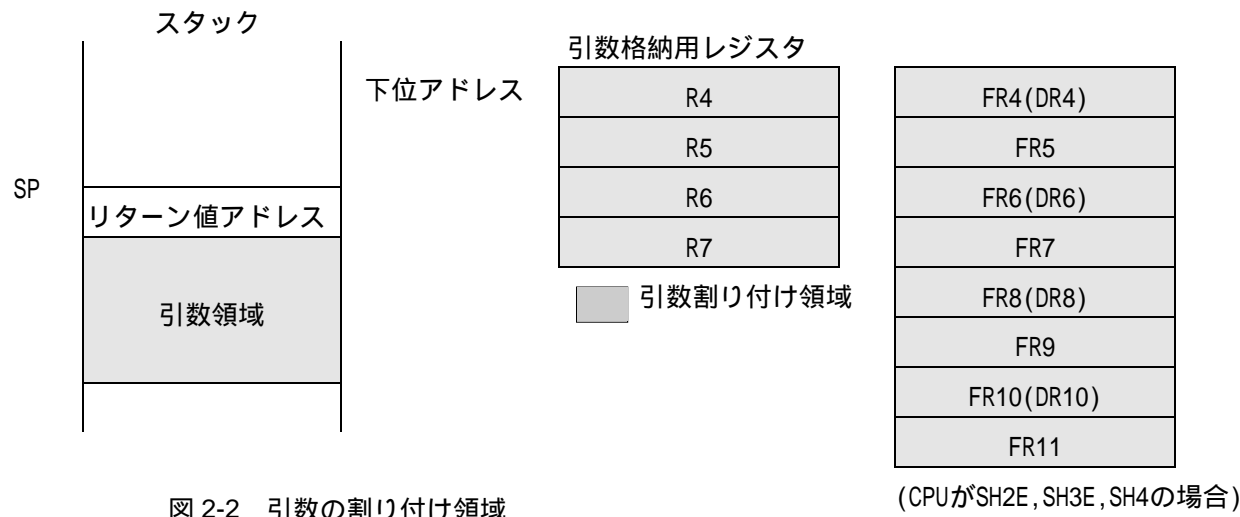


表 2-7 引数割り付け領域の一般規則

割り付け規則		
レジスタで渡される引数		スタックで渡される引数
引数格納用 レジスタ	対象の型	
R4 ~ R7	char, unsigned char, bool, short, unsigned short, int, unsigned int, long, unsigned long, float(CPU が SH1、SH2、SH3 の 場合),ポインタ,データメンバへの ポインタ,リファレンス	(1)引数の型がレジスタ渡しの対象 の型以外のもの (2)プロトタイプ宣言により可変個 の引数を持つ関数として宣言して いるもの* ³ (3)他の引数がすでに R4 ~ R7 に割 り付いている場合
FR4 ~ FR11 ^{*1}	SH2E、SH3E のとき ・引数が float 型 ・引数が double 型かつ double=float オプション指定 SH4 のとき ・引数型が float 型かつ fpu=double オプション指定なし ・引数型が double 型かつ fpu=single オプション指定	(4)他の引数がすでに FR4(DR4) ~ FR11(DR10)に割り付いている場 合
DR4 ~ DR10 ^{*2}	SH4 のとき ・引数型が double 型かつ fpu=single オプション指定なし ・引数型が float 型かつ fpu=double オプション指定	

*1:SH2E,SH3E,SH4 の単精度浮動小数点用のレジスタです。

*2:SH4 の倍精度浮動小数点用レジスタです。

*3:プロトタイプ宣言により可変個の引数をもつ関数として宣言している場合、宣言の中で対応する型のない引数およびその直前の引数はスタックに割り付けます。

例

```
int f2(int,int,int,int,...);
:
f2(a,b,c,x,y,z);    x、y、z はスタックに割り付けます。
```

(c) 引数の割り付け

(i) 引数格納用レジスタへの割り付け

引数格納用レジスタには、ソースプログラムの宣言順に番号の小さいレジスタから割り付けます。引数格納用レジスタの割り付け例を図 2-3 に示します。

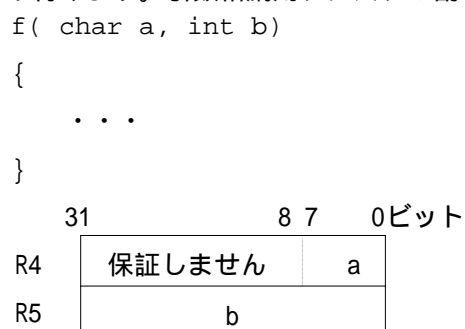


図 2-3 引数格納用レジスタの割り付け例

(ii) スタック上の引数領域への割り付け

スタック上の引数領域には、ソースプログラム上で宣言した順に下位アドレスから割り付けます。

【注】

構造体、共用体、クラス型の引数に関する注意

構造体、共用体、クラス型の引数を設定する場合は、その型の本来の境界調整にかかわらず 4 バイト境界に割り付けられ、しかもその領域として 4 の倍数バイトの領域が使用されます。これは、SuperH マイコンのスタックポインタが 4 バイト単位で変化するためです。

「付録 B. 引数割り付けの具体例」に、引数割り付けの具体例がありますので、あわせて参照してください。

(d) リターン値の設定場所

関数のリターン値の型によっては、リターン値をレジスタに設定する場合とメモリに設定場合があります。リターン値の型と設定場所の関係は表 2-8 を参照してください。

関数のリターン値をメモリに設定する場合、リターン値はリターン値アドレスの指す領域に設定します。呼び出す側では、引数領域のほかにリターン値設定領域を確保し、そのアドレスをリターン値アドレスに設定してから関数を呼び出します(図 2-4 参照)。関数のリターン値が void 型の場合、リターン値を設定しません。

表 2-8 リターン値の型と設定場所

No	リターン値の型	リターン値の設定場所
1	(signed) char, unsigned char, (signed) short, unsigned short, (signed) int, unsigned int, long, unsigned long float, ポインタ, bool リファレンス、データメンバへのポインタ	R0 : 32 ビット (signed) char,unsigned char の上位 3 バイト、(signed) short,unsigned short の上位 2 バイトの内容は保証しません。ただし、-rtnext オプション指定時は(signed) char,(signed) short 型は符号拡張、unsigned char,unsigned short 型はゼロ拡張を行います。) FR0 : 32 ビット (1)SH2E、SH3E のとき ・リターン値が float 型 ・リターン値が double 型かつ double=float オプション指定 (2)SH4 のとき ・リターン値が float 型かつ fpu=double オプション指定なし ・リターン値が浮動小数点型かつ fpu=single オプション指定
2	double, long double 構造体、共用体、クラス型、関数メンバへのポインタ	リターン値設定領域(メモリ) DR0:64 ビット SH4 のとき ・リターン値が double 型かつ fpu=single オプション指定なし ・リターン値が浮動小数点型かつ fpu=double オプション指定

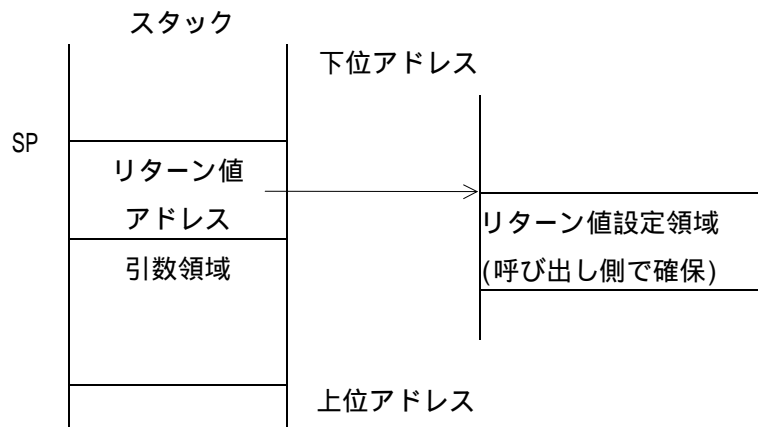


図 2-4 リターン値をメモリに設定する場合のリターン値の設定領域

2.3 拡張機能

本節では、本コンパイラの以下の拡張機能の使用方法について説明します。

- 割り込み関数
- 組み込み関数
- セクション切り替え機能
- 単精度浮動小数点ライブラリ
- 文字列内の日本語記述
- 関数のインライン展開
- アセンブラ埋め込みインライン展開
- 2 バイトアドレス変数の指定
- GBR ベース変数の指定
- レジスタ退避・回復の制御
- グローバル変数のレジスタ割り付け
- packed 構造体

上記拡張機能には、データメンバ、関数メンバが指定可能な機能があります。指定方法は(クラス名::メンバ名)です。指定可能なメンバの種類は、各機能の記述方法を参照してください。

2.3.1 割り込み関数

プリプロセッサ制御文(`#pragma`)を用いて、外部(ハードウェア)割り込み関数を C/C++ プログラムで記述することができます。以下、割り込み関数の作成方法を説明します。**SH3**、**SH3E**、**SH4** では割り込み時の動作が **SH1**、**SH2**、**SH2E** の場合と異なりますので、割り込みハンドラが必要になります。

(1) 記述方法

`#pragma interrupt` (関数名 [(割り込み仕様)] [, 関数名 [(割り込み仕様)]])

関数名には、グローバル関数および静的関数メンバを指定できます。

割り込み仕様の一覧を表 2-9 に示します。

表 2-9 割り込み仕様一覧

No.	項目	形式	オプション	指定内容
1	スタック 切り換え指定	<code>sp =</code>	<code>< 変数 > </code> <code>&< 変数 > </code> <code>< 定数 ></code>	新しいスタックのアドレスを変数または定数で指定 <code>< 変数 ></code> : 変数の値 <code>&< 変数 ></code> : 変数(ポインタ型)のアドレス <code>< 定数 ></code> : 定数値
2	トラップ命令 リターン指定	<code>tn =</code>	<code>< 定数 ></code>	終了を TRAPA 命令で指定 <code>< 定数 ></code> : 定数値 (トラップベクタ番号)

(2) 説明

`#pragma interrupt` を用いて割り込み関数となる関数を宣言します。

`#pragma interrupt` を用いて宣言した関数は、関数の処理の前後で全レジスタを保証(関数入口 / 出口において関数内で使用する全レジスタを退避・回復)し、通常 **RTE** 命令でリターンし、トラップ命令リターンを指定した場合は **TRAPA** 命令でリターンします。割り込み仕様を指定しない場合は単純な割り込み関数として処理します。また、スタック切り換え指定とトラップ命令リターン指定は重複して指定できます。

例

```
#pragma interrupt( f(sp = ptr, tn = 10),A::g)
extern int STK[100];
class A{
public:
    static void g();
};
int *ptr = STK + 100;
```

説明

(a) スタック切り替え指定

ptr を割り込み関数「f」で使用するスタックポインタとして設定します。

(b) トラップ命令リターン指定

割り込み関数終了時に TRAPA #10 でトラップ例外処理を開始します。トラップ例外処理開始時の SP は図 2-5 のようになっています。トラップルーチンの側で RTE 命令を使用して PC(プログラムカウンタ)、SR(ステータスレジスタ)を回復し、割り込み関数から復帰してください。

(c) C++プログラムで指定可能な関数メンバは、静的関数メンバです。例では、クラス A の静的関数メンバ g を割り込み関数として指定しています。非静的関数メンバは、指定できないので注意してください。

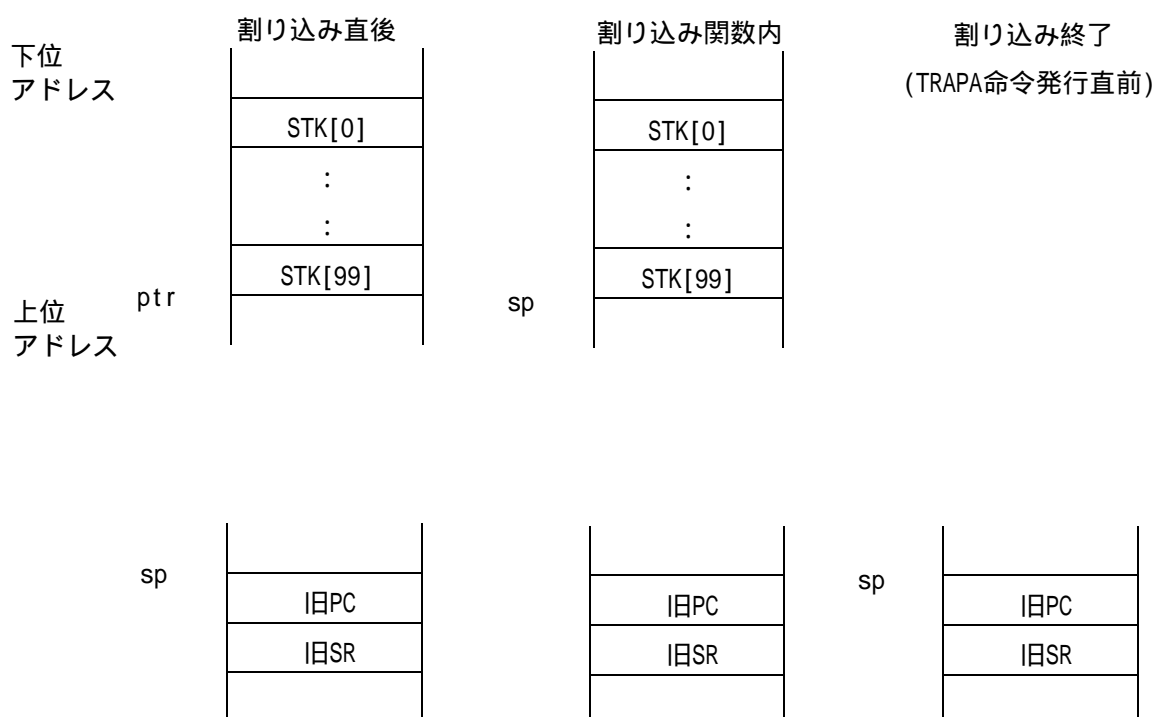


図 2-5 割り込み関数によるスタック使用例

(3) 使用上の注意事項

- (i) 割り込み関数の定義に対して指定できる関数は、グローバル関数(C/C++プログラム)と静的関数メンバ(C++プログラム)です。また、グローバル関数は、static と指定しても extern として処理します。

また、関数の返すデータ型は void のみです。return 文のリターン値を指定することはできません。指定があった場合はエラーを出力します。

例

```
#pragma interrupt (f1 (sp = 100), f2 )  
void f1(){...}.....(a)  
int f2(){...}.....(b)
```

説明

(a)は正しい宣言になります。

(b)は関数の返すデータ型が void ではないので誤った宣言です。エラーメッセージを出力します。

- (ii) 割り込み関数として宣言した関数をプログラムの中で呼び出すことはできません。呼び出しがあった場合はエラーメッセージを出力します。ただし、割り込み関数として定義した関数を割り込み関数の宣言のないプログラム内で呼び出した場合は、エラーメッセージは出力しません。この場合、実行時の動作は保証しません。

例

割り込み関数宣言のある場合

```
#pragma interrupt (f1)  
void f1(void){...}  
int f2(){ f1(); }.....(a)
```

説明

関数「f1」は割り込み関数として宣言しているのでプログラム中で呼び出すことはできません。(a)に対してエラーメッセージを出力します。

割り込み関数宣言がない場合

```
int f1();  
int f2(){ f1(); }.....(b)
```

説明

関数「f1」は割り込み関数としての宣言がないので非割り込み関数 `int f1()` ;としてオブジェクトを生成します。関数「f1」が別コンパイル単位で割り込み関数として宣言された場合、実行時の動作は保証しません。

2.3.2 組み込み関数

SuperH マイコン固有の命令に対応する組み込み関数を提供しています。

以下、組み込み関数の使用方法を説明します。

(1) 組み込み関数の機能

組み込み関数は以下の機能を記述することができます。

- (i) ステータスレジスタの設定、参照
- (ii) ベクタベースレジスタの設定、参照
- (iii) グローバルベースレジスタを利用した I/O 機能
- (iv) C/C++言語で使用するレジスタ資源と競合しないシステム命令
- (v) 浮動小数点ユニットを利用したマルチメディア命令、コントロールレジスタの設定、参照

(2) 説明

組み込み関数を使用する場合は、必ず<machine.h>、または<umachine.h>や<smachine.h>をインクルードしてください。

(3) 組み込み関数仕様

組み込み関数の一覧を表 2-10 に示します。

表 2-10 組み込み関数一覧

No.	項目	機能	仕様	説明
1	ステータスレジスタ(SR)	ステータスレジスタの設定	void set_cr(int cr)	ステータスレジスタに cr(32 ビット)を設定
2		ステータスレジスタの参照	int get_cr(void)	ステータスレジスタを参照
3		割り込みマスクの設定	void set_imask(int mask)	割り込みマスク(4 ビット)に mask (4 ビット)を設定
4		割り込みマスクの参照	int get_imask(void)	割り込みマスク(4 ビット)を参照
5	ベクタベースレジスタ(VBR)	ベクタベースレジスタの設定	void set_vbr(void **base)	VBR に**base(32 ビット)を設定
6		ベクタベースレジスタの参照	void **get_vbr(void)	VBR を参照

2. C/C++プログラミング

No.	項目	機能	仕様	説明
7	グローバル	GBR の設定	void set_gbr(void *base)	GBR に*base(32 ビット)を設定
8	ベース	GBR の参照	void *get_gbr(void)	GBR を参照
9	レジスタ(GBR)	GBR ベースのバイト参照	unsigned char gbr_read_byte(int offset)	GBR 相対 offset のバイトデータ(8 ビット)を参照
10		GBR ベースのワード参照	unsigned short gbr_read_word(int offset)	GBR 相対 offset のワードデータ(16 ビット)を参照
11		GBR ベースのロングワード参照	unsigned long gbr_read_long(int offset)	GBR 相対 offset のロングワードデータ(32 ビット)を参照
12		GBR ベースのバイト設定	void gbr_write_byte(int offset, unsigned char data)	GBR 相対 offset の data(8 ビット)を設定
13		GBR ベースのワード設定	void gbr_write_word(int offset, unsigned short data)	GBR 相対 offset の data(16 ビット)を設定
14		GBR ベースのロングワード設定	void gbr_write_long(int offset, unsigned long data)	GBR 相対 offset の data(32 ビット)を設定
15		GBR ベースのバイト AND	void gbr_and_byte(int offset, unsigned char mask)	GBR 相対 offset のバイトデータと mask の AND をとり、offset に設定
16		GBR ベースのバイト OR	void gbr_or_byte(int offset, unsigned char mask)	GBR 相対 offset のバイトデータと mask の OR をとり、offset に設定
17		GBR ベースのバイト XOR	void gbr_xor_byte(int offset, unsigned char mask)	GBR 相対 offset のバイトデータと mask の XOR をとり、offset に設定
18		GBR ベースのバイト TEST	int gbr_tst_byte(int offset, unsigned char mask)	GBR 相対 offset のバイトデータを mask と AND をとり、その値を 0 と判定し結果を T ビットにセット
19	その他の	SLEEP 命令	void sleep (void)	SLEEP 命令に展開
20	特殊命令	TAS 命令	int tas (char *addr)	TAS.B @addr に展開
21		TRAPA 命令	int trapa (int trap_no)	TRAPA #trap_no に展開

No.	項目	機能	仕様	説明
22	その他の 特殊命令	OS システムコールの 実現	int trapa_svc (int trap_no, int code, type1 para1,type2 para2, type3 para3, type4 para4) trap_no : トラップ番号 code : 機能コード para1 ~ 4 : パラメタ(0 ~ 4 個の可変) type1 ~ 4 : パラメタの型は、汎整数型 または ポインタ型	Hi7000 をはじめ、各種 OS のシステムコールを可能にする。trapa_svc を実行すると、R0 に code、R4 ~ R7 に para1 ~ para4 を設定し、TRAPA #trap_no 命令を実行する。
23		PREF 命令	void prefetch (void *p) 注意 : prefetch はコンパイラ オプション cpu=sh3,sh3e,sh4 指定時のみ使用可能	prefetch を実行すると、p の指す領域(16 バイト、ただし、領域は (int)p&0xfffff0)からの 16 バイトのデータをキャッシュに読み込む。プログラムの論理的な動作には全く影響を与えません。

No.	項目	機能	仕様	説明
24	積和演算	MAC.W 命令	<pre>int macw(short *ptr1, short *ptr2, unsigned int count) int macwl(short *ptr1, short *ptr2, unsigned int count, unsigned int mask) ptr1 : 積和演算するデータの先 頭アドレス ptr2 : 積和演算するデータの先 頭アドレス count : 積和演算を実行する回数 mask : リングバッファ対応のア ドレスマスク</pre>	<p>積和演算組み込み関数は、二つのデータテーブルの内容の積和を求める。</p> <p>例</p> <pre>short tbl1[] = {a1,a2,a3,a4}; short tbl2[] = {b1,b2,b3,b4};</pre> <p>この時</p> <pre>macw(tbl1, tbl2, 3)</pre> <p>は</p> $a1*b1 + a2*b2 + a3*b3$ <p>を求める。</p> <p>また、リングバッファ機能を用いて、tbl2 を繰り返して演算することができる。繰り返し回数は 2ⁿ 回。</p> <p>例</p> <p>データサイズが 2 バイトでリングバッファマスクを 4 バイト (0xffffffffb または ~0x4) とすると、</p>
25		MAC.L 命令	<pre>int macL(int *ptr1, int *ptr2, unsigned int count) int macLl (int *ptr1, int *ptr2, unsigned int count, unsigned int mask)</pre> <p>パラメタの仕様は No.24 と同様。</p> <p>注意:macL,macLl はコンパイラオプション cpu =sh2, sh2e, sh3, sh3e 指定時のみ使用可能。</p>	<p>macwl(tbl1,tbl2,4,0xffffffffb)は</p> $a1*b1 + a2*b2 + a3*b1 + a4*b2$ <p>を求める。</p>

No.	項目	機能	仕様	説明
26	浮動小数点ユニット	FPSCR の設定	void set_fpscr(int cr)	FPSCR に cr(32 ビット)を設定する。
27		FPSCR の参照	int get_fpscr()	FPSCR を参照する。
28	単精度浮動小数点ベクタ演算	FIPR 命令	float fipr(float vect1[4], float vect2[4])	2 つのベクタの内積を求める。 例 extern float data1[4],data2[4]; この時 fipr(data1,data2)は、ベクタ data1 と data2 の内積を求める。
29		FTRV 命令	float ftrv(float vec1[4], float vec2[4])	data1(ベクタ)を tbl(4 × 4 行列)で変換した結果を data2(ベクタ)に格納する。ただし、tbl は組み込み関数 ld_ext()でロードする必要があります。 data2 = data1 × tbl 例 extern float tbl[4][4]; extern float data1[4],data2[4]; ld_ext(tbl); この時 ftrv(data1,data2)は、data1 を 4 × 4 行列 tbl で変換した結果を data2 に格納する。 i = 0,1,2,3 として data2[i] = data1[0]*tbl[0][i] +data1[1]*tbl[1][i] +data1[2]*tbl[2][i] +data1[3]*tbl[3][i] が data2 の結果になります。

No.	項目	機能	仕様	説明
30	単精度浮動小数 点ベクタ演算	4次元ベクタの4×4 行列による変換と4次元 ベクタとの和	void ftrvadd(float vec1[4], float vec2[4], float vec3[4])	data1(ベクタ)をtbl(4×4行列)で変換した結果とdata2(ベクタ)の和をdata3(ベクタ)に格納する。ただし、tblは組み込み関数ld_ext()でロードする必要があります。 data3 = data1 × tbl + data2 例 extern float tbl[4][4]; extern float data1[4]; extern float data2[4]; extern float data3[4]; ld_ext(tbl); この時 ftrvadd(data1,data2,data3)は、data1を4×4行列tblで変換した結果とdata2の和をdata3に格納する。 i = 0,1,2,3として data3[i] = data1[0]*tbl[0][i] +data1[1]*tbl[1][i] +data1[2]*tbl[2][i] +data1[3]*tbl[3][i] +data2[i] がdata3の結果になります。

No.	項目	機能	仕様	説明
31	単精度浮動小数 点ベクタ演算	4次元ベクタの 4×4 行列による変換と4次元ベクタとの差	<pre>void ftrvsub(float vec1[4], float vec2[4], float vec3[4])</pre>	<p>data1(ベクタ)をtbl(4×4 行列)で変換した結果とdata2(ベクタ)の差をdata3(ベクタ)に格納する。ただし、tblは組み込み関数ld_ext()でロードする必要があります。</p> <p>data3=data1 × tbl-data2</p> <p>例</p> <pre>extern float tbl[4][4]; extern float data1[4]; extern float data2[4]; extern float data3[4]; ld_ext(tbl);</pre> <p>この時</p> <p>ftrvsub(data1,data2,data3)は、data1を4×4 行列tblで変換した結果とdata2の差をdata3に格納する。</p> <p>i = 0,1,2,3 として</p> <pre>data3[i] = data1[0]*tbl[0][i] +data1[1]*tbl[1][i] +data1[2]*tbl[2][i] +data1[3]*tbl[3][i] -data2[i]</pre> <p>がdata3の結果になります。</p>

2. C/C++プログラミング

No.	項目	機能	仕様	説明
32	単精度浮動小数 点ベクタ演算	4次元ベクタの和	<pre>void add4(float vec1[4], float vec2[4], float vec3[4])</pre>	<p>data1(ベクタ)と data2(ベクタ)の和を data3(ベクタ)に格納する。</p> <p>data3 = data1+data2</p> <p>例</p> <pre>extern float data1[4]; extern float data2[4]; extern float data3[4];</pre> <p>この時 add4(data1,data2,data3)は data1と data2 の和を data3 に格納する。</p>
33		4次元ベクタの差	<pre>void sub4(float vec1[4], float vec2[4], float vec3[4])</pre>	<p>data1(ベクタ)と data2(ベクタ)の差を data3(ベクタ)に格納する。</p> <p>data3=data1-data2</p> <p>例</p> <pre>extern float data1[4]; extern float data2[4]; extern float data3[4];</pre> <p>この時 sub4(data1,data2,data3)は、data1と data2 の差を data3 に格納する。</p>
34		4×4 行列の乗算	<pre>void mtrx4mul(float mat1[4][4], float mat2[4][4])</pre>	<p>tbl1(4×4 行列)を tbl(4×4 行列)で変換した結果 tbl2 に格納する。ただし、tbl は組み込み関数 ld_ext() でロードする必要があります。</p> <p>tbl2 = tbl1 × tbl</p> <p>例</p> <pre>extern float tbl[4][4]; extern float tbl1[4][4]; extern float tbl2[4][4]; ld_ext(tbl);</pre> <p>この時 mtrx4mul(tbl1,tbl2)は、tbl1 を 4×4 行列 tbl で乗算した結果を tbl2 に格納する。</p>

No.	項目	機能	仕様	説明
35	単精度浮動小数 点ベクタ演算	4 × 4 行列の乗算と和	<pre>void mtrx4muladd(float mat1[4][4], float mat2[4][4], float mat3[4][4])</pre>	<p>tbl1(4 × 4 行列)を tbl(4 × 4 行列)で変換した結果と tbl2(4 × 4 行列)の和を tbl3(4 × 4 行列)に格納する。ただし、tbl は組み込み関数 ld_ext() でロードする必要があります。</p> <p>tbl3=tbl1 × tbl+tbl2</p> <p>例</p> <pre>extern float tbl[4][4]; extern float tbl1[4][4]; extern float tbl2[4][4]; extern float tbl3[4][4]; ld_ext(tbl);</pre> <p>この時 mtrx4muladd(tbl1,tbl2,tbl3)は、tbl1 を 4 × 4 行列 tbl で乗算した結果と tbl2 との和を tbl3 に格納する。</p>
36		4 × 4 行列の乗算と差	<pre>void mtrx4mulsub(float mat1[4][4], float mat2[4][4], float mat3[4][4])</pre>	<p>tbl1(4 × 4 行列)を tbl(4 × 4 行列)で変換した結果と tbl2(4 × 4 行列)との差を tbl3 に格納する。ただし、tbl は組み込み関数 ld_ext() でロードする必要があります。</p> <p>tbl3=tbl1 × tbl - tbl2</p> <p>例</p> <pre>extern float tbl[4][4]; extern float tbl1[4][4]; extern float tbl2[4][4]; extern float tbl3[4][4]; ld_ext(tbl);</pre> <p>この時 mtrx4mulsub(tbl1,tbl2,tbl3)は、tbl1 を 4 × 4 行列 tbl で乗算した結果と tbl2 との差を tbl3 に格納する。</p>

No.	項目	機能	仕様	説明
37	拡張レジスタへのアクセス	拡張レジスタへのロード	void ld_ext(float mat[4][4])	tbl(4×4 行列)を拡張レジスタにロードする。 例 extern float tbl[4][4]; この時 ld_ext(tbl)は、tbl の内容を拡張レジスタにロードする。
38		拡張レジスタからのストア	void st_ext(float mat[4][4])	拡張レジスタの内容を tbl(4×4 行列)にストアする。 例 extern float tbl[4][4]; この時 st_ext(tbl)は、拡張レジスタの内容を tbl にストアする。

(4) 注意事項

(A)グローバルベースレジスタ(GBR)組み込み関数

表 2-10「組み込み関数一覧」で使用した **offset**(No.15 ~ 18 を除く)、**mask**(No.3 を除く)は定数でなければなりません。

また、**offset** 指定可能範囲は、アクセスサイズがバイトのとき+255 バイト、ワードのとき+510 バイト、ロングワードのとき+1020 バイトまでです。

GBR(グローバルベースレジスタ)相対のバイト論理演算(**AND**、**OR**、**XOR**、**TEST**)で指定できる **mask** は 0 ~ +255 です。

GBR はコントロールレジスタですので、本コンパイラでは関数ごとに内容を保証していません。

GBR の設定を変えるときには注意が必要です。

積和演算組み込み関数はパラメタのチェックを行いません。パラメタは次のことに注意してください。

- (i) ptr1、ptr2 の指すテーブルは、それぞれ 2 バイト、4 バイトで境界整合されていなければなりません。
- (ii) macwl、macll の ptr2 の指すテーブルはリングバッファマスク×2 のサイズで境界整合されていなければなりません。

(B)単精度浮動小数点ベクタ演算、拡張レジスタアクセス組み込み関数は、SH4 のみ有効です。

ベクタ演算組み込み関数は、割り込み関数で使用する時以下の点に注意してください。

(i)組み込み関数 `ld_ext(float[4][4])`と `st_ext(float[4][4])`は、浮動小数点ステータス制御レジスタ(FPSCR)の浮動小数点レジスタバンクビット(FR)を変更して拡張レジスタにアクセスするため、割り込み関数内で、組み込み関数 `ld_ext(float[4][4])`と `st_ext(float[4][4])`を使用しているときには、ベクタ演算組み込み関数の前後で割り込みマスクを変更してください。以下に例を示します。

例

```
#pragma interrupt (intfunc)
void intfunc(){
    ...
    ld_ext();
    ...
}
void normfunc(){
    ...
    int maskdata=get_imask();
    set_imask(15);
    ld_ext(mat1);
    ftrv(vec1,vec2);
    set_imask(maskdata);
    ...
}
```

(ii)組み込み関数 `mtrx4mul`、`mtrx4muladd`、`mtrx4mulsub` は 4×4 行列の演算のため非可換です。

例

```
extern float matA[4][4];
extern float matB[4][4];
int judge(){
    float data1[4][4], data2[4][4];
    set_imask(15);
    ld_ext(matA);
    mtrx4mul(matB,data1);/* data1 = matB × matA */
    ld_ext(matB);
    mtrx4mul(matA,data2);/* data2 = matA × matB */
    ..../*この時の data1[][]と data2[][]の各要素は必ずしも一致しません。*/
}
```

(5) 使用例

```

#include <machine.h>

#define CDATE1 0
#define CDATE2 1
#define CDATE3 2
#define SDATE1 4
#define IDATA1 8
#define IDATA2 12

struct {
    char  cdata1;           /*offset 0*/
    char  cdata2;           /*offset 1*/
    char  cdata3;           /*offset 2*/
    short sdata1;           /*offset 4*/
    int   idata1;           /*offset 8*/
    int   idata2;           /*offset 12*/
}table;

void f();

void f()
{
    set_gbr(&table);        /*GBR に table の先頭アドレス*/
    :                       /*を設定*/
    gbr_write_byte( CDATE2,10); /* table.cdata2 に 10 を設定*/
    gbr_write_long( IDATA2,100); /* table.idata2 に 100 を設定*/
    :
    if( gbr_read_byte( CDATE2) != 10) /* table.cdata2 の値を参照*/
        gbr_and_byte( CDATE2, 10); /* table.cdata2 の値と 10 の AND*/
        :                   /*をとって table.cdata2 に設定*/
    gbr_or_byte( CDATE2,0x0F); /* table.cdata2 の値と 0x0f の OR*/
    :                       /*をとって table.cdata2 に設定*/
    sleep();                /* sleep 命令に展開*/
}

```

組み込み関数の有効な使用法

- (i) 頻繁にアクセスするオブジェクトをメモリに割り付け、そのオブジェクトの先頭アドレスを GBR に設定する。
- (ii) 論理演算を多用するバイトデータをできるだけ構造体の先頭から 128 バイトまでに宣言する。これにより、構造体アクセスに必要な先頭アドレスロードと、論理演算に必要なメモリロード、ストアに対する命令が削減できます。

(6) <machine.h>の分割

SH3、SH3E、SH4 の実行モードに対応し<machine.h>の内容を以下のように分割しました。

- (a) <machine.h> 組み込み関数全体
- (b) <smachine.h>特権モードでのみ使用可能な組み込み関数
- (c) <umachine.h>(b)以外の組み込み関数

2.3.3 セクション切り替え機能

#pragma section を用いて、C/C++プログラムの中でコンパイラの出力するセクション名を切り替えることができます。

(1) 記述方法

```
#pragma section 名前 | 数値
    <ソースプログラム>
#pragma section
```

(2) 説明

“**#pragma section 名前**”または“**#pragma section 数値**”を用いて、セクション名を指定します。ソースプログラム中の宣言位置以降のセクションが、“**P セクション名+名前(数値)**”、“**D セクション名+名前(数値)**”、“**C セクション名+名前(数値)**”、“**B セクション名+名前(数値)**”になります。

“**#pragma section**”が宣言されると、以降はデフォルトのセクション名になります。

(3) 使用上の注意事項

- (i) **#pragma section** は関数定義の外に指定してください。
- (ii) 1 ファイルで宣言できるセクション名は最大 64 個です。

(4) セクション切り替え機能の使用例

例

```
#pragma section abc
int a; /* a は、セクション Babc に割り付きます。 */
extern const int c = 1; /* c は、セクション Cabc に割り付きます。 */
f(){ /* f は、セクション Pabc に割り付きます。 */
    a = c;
}

#pragma section
int b; /* b は、セクション B に割り付きます。 */
g(){ /* g は、セクション P に割り付きます。 */
    b = c;
}
```

上記例においてコンパイルオプション `section = P = PROG` が指定された場合、`f` はセクション `PROGabc`、`g` はセクション `PROG` にそれぞれ割り付きます。

2.3.4 単精度浮動小数点ライブラリ

ANSI 標準浮動小数点ライブラリ(`math.h`)のほかに、単精度の浮動小数点ライブラリ(`mathf.h`)を使用することができます。本ライブラリは表 2-11 に示す関数群から構成されています。

(1) 記述方法

各関数名は倍精度の ANSI 標準ライブラリの関数名の末尾に「f」を付加したものになっています。パラメタおよびリターンの型が、`double` 型または `double` 型へのポインタ型である場合、それぞれ、`float` 型、`float` 型へのポインタ型となります。それ以外の仕様は ANSI 標準 C ライブラリと同じです。

(2) 使用上の注意事項

本ライブラリを使用する場合は、必ず `#include <mathf.h>` と `#include <math.h>` を宣言してください。

表 2-11 単精度浮動小数点ライブラリ関数一覧

関数名	説明
float acosf (float x)	逆余弦 $\cos^{-1} x$
float asinf (float x)	逆正弦 $\sin^{-1} x$
float atanf (float x)	逆正接 $\tan^{-1} x$
float atan2f (float y, float x)	除算した結果の値の逆正接 $\tan^{-1}(y/x)$
float cosf (float x)	余弦 $\cos x$
float sinf (float x)	正弦 $\sin x$
float tanf (float x)	正接 $\tan x$
float coshf (float x)	双曲線余弦 $\cosh x$
float sinh (float x)	双曲線正弦 $\sinh x$
float tanhf (float x)	双曲線正接 $\tanh x$
float expf (float x)	指数関数 e^x
float frexpf (float x, int *p)	[0.5, 1.0)の値と2のべき乗の積に分解 result = frexp (x,p)とすると $x = 2^p \times \text{result}$ ($0.5 \leq \text{result} < 1.0$)
float ldexpf (float x, int i)	2のべき乗との乗算 $x \times 2^i$
float logf (float x)	自然対数 $\log x$
float log10f (float x)	常用対数 $\log_{10} x$ (底を10とする)
float modff (float x, float *p)	result = modff (x, y)とすると xを整数部分 *pと小数部分 resultに分解
float powf (float x, float y)	べき乗 x^y
float sqrtf (float x)	正の平方根 \sqrt{x}
float ceilf (float x)	xの小数点以下を切り上げて結果とします
float fabsf (float x)	絶対値 $ x $
float floorf (float x)	xの小数点以下を切り捨てて結果とします
float fmodf (float x, float y)	除算した余り result = fmodf (x,y)、qを商(整数)とすると $x = q \times y + \text{result}$

2.3.5 文字列内の日本語記述

文字列の中に日本語が記述できます。euc、または sjis オプションで文字コードを選択できます。本オプション省略時の設定は、ホストごとのデフォルトに従います(表 2-12 参照)。

表 2-12 日本語コードのデフォルト設定

ホスト	デフォルト
SPARC	EUC
HP9000/700	シフト JIS
PC	シフト JIS

オブジェクトプログラム内の文字コードは、ソースプログラムの文字コードと同一になります。

文字定数に日本語は指定できません。

2.3.6 関数のインライン展開

コンパイル時にインライン展開する関数名を指定します。

(1) 記述方法

```
#pragma inline (関数名,...)
```

関数名には、グローバル関数及び関数メンバを指定できます。

(2) 説明

#pragma inline で指定した関数名の関数と関数指定子 **inline**(C++言語)を指定した関数は、その関数を呼び出したところにインライン展開されます。

ただし、以下の場合はインライン展開しません。

- ・ #pragma inline 指定より前に関数の定義がある。
- ・ 可変パラメタを持つ関数である。
- ・ 関数内でパラメタのアドレスを参照している。
- ・ 展開対象関数のアドレスを介して呼び出しを行っている。

(3) 使用上の注意事項

- (a) `#pragma inline` は、関数本体の定義の前に指定してください。
- (b) `#pragma inline` で指定した関数に対しても外部定義を生成します。各プログラムファイル中にインライン関数の実体の記述がある場合は、必ず関数の宣言に `static` を指定してください。`static` を指定した場合は、外部定義を生成しません。`inline`(C++言語)指定された関数は、外部定義を生成しません。

(4) 使用例

<u>ソースプログラム</u>	<u>展開イメージ</u>
<pre>#pragma inline(func) static int func (int a, int b) { return (a+b)/2; } int x; main() { x = func(10,20); }</pre>	<pre>int x; main() { int func_result; { int a_1 = 10, b_1 = 20; func_result = (a_1+b_1)/2; } x = func_result; }</pre>

2.3.7 アセンブラ埋め込みインライン展開

C/C++プログラム内でアセンブリ言語で記述した関数をインライン展開します。

(1) 記述方法

`#pragma inline_asm (関数名[(size=数値)],...)`

関数名には、グローバル関数のみ指定できます。関数メンバは指定できません。

(2) 説明

アセンブラ埋め込みインライン関数のパラメタは、通常関数呼び出しと同様にレジスタ、あるいはスタックに設定されますので、`inline_asm` 関数から参照することができます。アセンブラ埋め込みインライン関数のリターン値は `R0`、`SH2E`、`SH3E`、`SH4` のとき単精度浮動小数点型のリターン値は `FR0`、`SH4` のとき倍精度浮動小数点型のリターン値は `DR0` に設定してください。オプションとの組み合わせによりリターン値を設定するレジスタは異なります。詳細は、「第2章 C/C++プログラミング 2.2.4.2 関数の呼び出し 表 2-8 リターン値の型と設定場所」を参照してください。

(size=数値)指定で、アセンブラ埋め込みインライン関数のサイズが指定できます。

(3) 使用上の注意事項

- (a) `#pragma inline_asm` は、関数本体の定義の前に指定して下さい。
- (b) `#pragma inline_asm` で指定した関数に対しても外部定義を生成します。各ソースプログラムファイル中にインライン関数の実体の記述がある場合は、必ず関数の宣言に `static` を指定して下さい。 `static` を指定した場合は、外部定義を生成しません。
- (c) アセンブラ埋め込みインライン関数中でラベルを使用する場合、必ずローカルラベルを使用して下さい。
- (d) アセンブラ埋め込みインライン関数中で R8 から R15 のレジスタを使用する場合は、アセンブラ埋め込みインライン関数の先頭と最後でこれらレジスタの退避・回復が必要です。また、FR12 から FR15 (CPU が SH2E、SH3E、SH4 の場合)、DR12 から DR14 (CPU が SH4 の場合) のレジスタを使用する場合もアセンブラ埋め込みインライン関数の先頭と最後でこれらのレジスタ退避・回復が必要です。
- (e) アセンブラ埋め込みインライン関数の最後に RTS を記述しないでください。
- (f) 本機能を使用する際は、オブジェクト形式指定オプション `code = asmcode` を用いてコンパイルしてください。
- (g) (`size = 数値`) で指定する数値は、実際のオブジェクトサイズ以上の値を指定してください。オブジェクトサイズより小さい値を指定した場合、動作は保証しません。また、数値が浮動小数点または 0 以下の数値の場合、エラーとなります。
- (h) `#pragma global_register` 機能で指定したレジスタを本関数内で使用する場合もアセンブラ埋め込みインライン関数の先頭と最後でこれらのレジスタ退避・回復が必要です。
- (i) 指定可能な関数は、グローバル関数のみです。関数メンバは指定できません。
- (j) リテラルを生成するような記述は使用しないでください。(MOV.L #100000,R0 等)

(4) 使用例

ソースプログラム	出力結果 (一部)
<pre>#pragma inline_asm(rotl) static int rotl (int a) { ROTL R4 MOV R4,R0 } int x; main() { x = 0x55555555; x = rotl(x); }</pre>	<pre>: ;function main ;frame size = 4 R14,@ - R15 MOV.L L220+2,R14 ;_x MOV.L L220+6,R3 ;H'55555555 MOV.L R3,@R14 MOV R3,R4 BRA L219 NOP L220: .RES.W 1 .DATA.L _x .DATA.L H'55555555 L219: ROTL R4 MOV R4,R0 .ALIGN 4 MOV.L R0,@R14 RTS MOV.L @R15+,R14 .SECTION B,DATA,ALIGN=4 ;static: x _x: .RES.L 1 .END</pre>

2.3.8 2バイトアドレス変数の指定

H'0000000 ~ H'0007FFF 番地および **H'FFF8000 ~ H'FFFFFFF** 番地に配置した変数に対して、アドレスの表現を2バイトで済ますことができます。

(1) 記述方法

```
#pragma abs16 (識別子,...)
```

識別子には、変数、グローバル関数、静的データメンバ及び関数メンバを指定できます。

(2) 説明

識別子で指定した変数、または関数のアドレスを2バイトの値として扱います。これによってプログラムサイズを削減することができます。

(3) 使用上の注意事項

- (a) #pragma abs16 は、自動オブジェクトや非静的データメンバを指定できません。
- (b) #pragma abs16 で宣言された変数は、必ず **H'0000000 ~ H'0007FFF** 番地または、**H'FFF8000 ~ H'FFFFFFF** 番地に配置してください。

2.3.9 GBR ベース変数の指定

変数を GBR レジスタからのオフセットでアクセスすることを指定します。

(1) 記述方法

```
#pragma gbr_base (変数名[=セクション名],...)
```

```
#pragma gbr_base1 (変数名[=セクション名],...)
```

変数名に、変数及び静的データメンバを指定できます。

セクション名に名前または数値を指定できます。

(2) 説明

セクション名の指定がない場合には、`#pragma gbr_base` で指定した変数は、セクション\$G0 に割り付けられます。`#pragma gbr_base1` で指定した変数は、セクション\$G1 に割り付けられます。セクション名の指定がある場合には、それぞれ”\$G0 セクション名+名前(数値)”、”\$G1 セクション名+名前(数値)”になります。

`#pragma gbr_base` は、変数が GBR レジスタの指すアドレスからオフセット 0 ~ 127 バイトにあることを指定します。

`#pragma gbr_base1` は、`#pragma gbr_base` でアクセス可能でない範囲(GBR レジスタの指すアドレスからオフセット 128 バイト以上)の変数に対して指定できます。GBR レジスタの指すアドレスからのオフセットが、`char` 型、`unsigned char` 型の場合は最大 255 バイト、`short` 型、`unsigned short` 型の場合は、最大 510 バイト、`int` 型、`unsigned` 型、`long` 型、`unsigned long` 型、`float` 型、`double` 型の場合は最大 1020 バイトであることを指定します。

コンパイラは、これらの指定に基づき、変数の参照、設定に対して、最適な GBR 相対アドレッシングでオブジェクトプログラムを生成します。また、セクション\$G0 内の `char` 型、`unsigned` 型のデータに対して、GBR 間接アドレッシングで最適なビット命令を生成します。

(3) 使用上の注意事項

- (a) セクション\$G0 のリンク後のサイズ合計が 128 バイトを超えた場合は動作を保証しません。また、セクション\$G1 内に、上記の `#pragma gbr_base1` の制約で各データ型に示した以上のオフセットを持つデータがある場合、動作を保証しません。
- (b) セクション\$G1 は、リンク時にセクション\$G0 の 128 バイト後に必ず配置してください。
- (c) 本機能を使用する場合は、プログラム実行開始時に、GBR レジスタにセクション\$G0 の先頭を設定してください。
- (d) 静的データメンバは指定可能ですが、非静的データメンバは指定できません。
- (e) Ver.5.0 では、セクション名の指定はできません。

2.3.10 レジスタ退避・回復の制御

関数のレジスタ退避・回復方法を変更します。

(1) 記述方法

```
#pragma noregsave (関数名,...)
```

```
#pragma noregalloc (関数名,...)
```

```
#pragma regsave (関数名,...)
```

関数名には、グローバル関数及び関数メンバを指定できます。

(2) 説明

- (a) #pragma noregsave で指定された関数は、関数の出入口で保証するレジスタ(表 2-6 参照)の退避・回復を行いません。
- (b) #pragma noregalloc で指定された関数は、関数の出入口で保証するレジスタの退避・回復を行いません。また、関数呼び出しを越えて R8～R14 を割り付けないオブジェクトを生成します。
- (c) #pragma regsave で指定された関数は、関数の出入口で保証するレジスタの退避・回復を行います。また関数呼び出しを越えて R8～R14 を割り付けないオブジェクトを生成します。
- (d) #pragma regsave と #pragma noregalloc は同一関数に対して重複指定できます。このとき、関数の出入口で保証するレジスタ R8～R14 を全て退避・回復し、関数呼び出しがあるとき、レジスタ(R8～R14)を割り付けないオブジェクトを生成します。

#pragma noregsave が指定された関数は、下記の条件で使うことができます。

- (i) 他の関数から呼び出されることなく、最初に起動する関数として使用する。
- (ii) #pragma regsave を指定した関数から呼び出す。
- (iii) #pragma regsave を指定した関数から、さらに #pragma noregalloc を介して呼び出す。

(3) 使用上の注意事項

上記以外の方法で**#pragma noregsave**を指定した関数を呼び出した場合の結果は保証されませんので注意が必要です。

(4) 使用例

```
#pragma noregsave(f,A::j)
#pragma noregalloc(g)
#pragma regsave(h)
class A{
public:
    void j();
};
void f();
void g();
void h();
void h()
{
    g();
    f(); /* #pragma regsave関数(h)から#pragma noregsave関数(f)の直後の呼び出し */
}

void g()
{
    f(); /* #pragma regsave関数(h)から#pragma noregsave関数(f,A::j)の */
        /* #pragma noregalloc関数(g)を介した呼び出し */
    A::j();
}

void f()
{
}
```

2.3.11 グローバル変数のレジスタ割り付け

レジスタにグローバル変数または、静的データメンバを割り付けます。

(1) 記述方法

```
#pragma global_register (<変数名>=<レジスタ名>,...)
```

変数名には、グローバル変数及び静的データメンバを指定できます。

(2) 説明

<変数名>で指定された変数を<レジスタ名>で指定したレジスタに割り付けます。

(3) 使用上の注意事項

- (i) グローバル変数で、単純型またはポインタ型の変数に使用できます。また、
double=float オプションを指定した場合を除き、double 型の変数は指定できません。(CPU が SH4 を除く)
- (ii) 指定可能なレジスタは、R8 ~ R14, FR12 ~ FR15 (CPU が SH2E、SH3E、SH4 の場合)、DR12 ~ DR14 (CPU が SH4 の場合) です。
- (iii) 初期値の設定はできません。また、アドレスの参照もできません。
- (iv) 指定された変数の、リンク先からの参照は保証されません。
- (v) 静的データメンバの指定は可能ですが、非静的データメンバの指定は不可能です。

FR12 ~ FR15 に設定可能な変数の型

SH2E、SH3E の場合

- ・float 型変数
- ・double 型変数(double=float オプション指定)

SH4 の場合

- ・float 型変数(fpu=double オプション指定なし)
- ・double 型変数(fpu=single オプション指定)

DR12 ~ DR15 に設定可能な変数の型

SH4 の場合

- ・float 型変数(fpu=double オプション指定)
- ・double 型変数(fpu=single オプション指定なし)

(4) 使用例

```
#pragma global_register(a = R8,A::b = R9)
class A{
public:
    static int b;
};
int a;
void g()
{
    a = A::b;
}
```

2.3.12 構造体/クラスメンバの境界調整

構造体/共用体/クラスの境界調整数を 1 バイト境界にします。

(1) 記述方法

```
#pragma pack1
#pragma unpack
```

(2) 説明

#pragma pack1 指定以降で宣言された構造体/共用体/クラスのメンバの境界調整数を 1 バイトとする。

#pragma unpack 指定以降で宣言された構造体/共用体/クラスの境界調整数はメンバの最大境界調整数とする。

(3) 使用例

```
#pragma pack1
struct A{ /* 1バイト境界になります */
    int a;
    char b;
};
#pragma unpack
struct B{ /* 4バイト境界になります */
    short a;
    int b;
    char c;
};
```

【注意】

本拡張機能は、Ver.5.0 ではサポートしていません。

2.4 プログラム作成上の注意事項

本節では、本コンパイラにおけるコーディング上の注意事項と、コンパイルからデバッグまでのプログラム開発上のトラブル対処方法を述べます。

2.4.1 コーディング上の注意事項

(1) float 型パラメタをもつ関数

float 型のパラメタを受け渡している関数は、必ず原型宣言を行うか、**float** 型を **double** 型に変更してください。原型宣言のない **float** 型パラメタの受け渡しによるデータの値は保証されません。

例

```
void f(float);  
void g()  
{  
    float a;  
    ...  
    f(a);  
}  
void f(float x)  
{...}
```

関数 **f** は、**float** 型のパラメタをもつ関数です。この場合、必ず原型宣言を行ってください。

(2) C/C++言語で評価順序を規定していない式

C/C++言語で評価順序を規定していない式で、評価順序で結果が変わるようなコーディングをした場合、その動作は保証しません。

例

```
a[i]=a[++i];
```

代入式の右辺を先に評価するか後に評価するかで、左辺の値が変わります。

```
sub(++i, i);
```

関数の第 1 引数を先に評価するか後に評価するかで第 2 引数の値が変わります。

(3) オーバフロー演算、ゼロ除算

オーバフロー演算や浮動小数点のゼロ除算があっても、エラーメッセージを出力しません。ただし、一つの定数または定数どうしの演算でのオーバフロー演算や、定数どうしの演算または整数型のゼロ除算があれば、コンパイル時にエラーメッセージを出力します。

例

```
void main()
{
    int ia;
    int ib;
    float fa;
    float fb;

    ib=32767;
    fb=3.4e+38f;

    /* 定数または定数どうしの演算時はオーバフロー、ゼロ除算に対する */
    /* コンパイルエラーメッセージを出力します */

    ia=999999999999; /* (W) 定数のオーバフローを検出します */
    fa=3.5e+40f;      /* (W) 浮動小数点演算のオーバフローを検出します */
    ia=1/0;           /* (E) ゼロ除算を検出します */
    fa=1.0/0.0;       /* (W) 浮動小数点のゼロ除算を検出します */

    /* 実行時のオーバフローに対するエラーメッセージは出力しません */

    ib=ib+32767;      /* 演算結果のオーバフローを無視します */
    fb=fb+3.4e+38f;   /* 浮動小数点演算結果のオーバフローを無視します */
}
```

(4) 最適化により削除される可能性のあるコーディング

連続した同一変数の参照や、結果を使用しない式を記述した場合、コンパイラの最適化により冗長コードとして削除される場合があります。常にアクセスを保証する場合は、宣言時に **volatile** を指定してください。

例 1

```
b = a; /* 1行目の式は冗長コードとして削除されることがあります。 */
b = a;
```

例 2

```
while(1) a; /* 変数 a の参照およびループ文は冗長コードとして削除 */
           /* されることがあります。 */
```

(5) const 型変数への書き込み

const 型の変数を宣言していても、型変換で const 型でない型に変換して代入した場合や、分割コンパイルしたプログラムの中で、型を統一して扱っていない場合は、const 型変数への書き込みをコンパイラでチェックできませんので、注意が必要です。

例

```
const char *p;      /* ライブラリ関数 strcat の第 1 引数は char 型 */
:                  /* へのポインタ型なので、引数の指す領域が書き換わ */
strcat(p, "abc");    /* ることがあります。 */
```

ファイル 1

```
const int i;
```

ファイル 2

```
extern int i;        /* 変数 i は、ファイル 2 では const 型で宣言してい */
:                  /* ませんのでファイル 2 の中で書き込んでもエラーに */
i=10;               /* なりません。 */
```

(6) 数学関数ライブラリの精度について

acos(x)、asin(x)関数では x = 1 で誤差が大きくなりますので注意が必要です。

誤差範囲は以下のとおりです。

acos(1.0 -)における絶対誤差	倍精度 2^{-39} ($= 2^{-33}$)
	単精度 2^{-21} ($= 2^{-19}$)
asin(1.0 -)における絶対誤差	倍精度 2^{-39} ($= 2^{-28}$)
	単精度 2^{-21} ($= 2^{-16}$)

2.4.2 プログラム開発上のトラブル対処方法

C/C++プログラムの作成からデバッグまでのプログラム開発上で、トラブルが発生したときの対処方法を表 2-13 に示します。

表 2-13 トラブル発生時の対処方法

No	現象	確認内容	対処方法	参照
1	リンク時にエラー No. 314 cannot found section が出力される。	リンケージエディタの start オプションにおいて、コンパイル出力のセクション名を大文字で指定しているか。	正しいセクション名を指定して下さい。	「第 2 章 C/C++プログラミング 2.2.1 オブジェクトプログラムの構造」
2	リンク時にエラー No. 105 undefined external symbol が出力される。	C/C++プログラムとアセンブリプログラム間で変数を相互参照している場合、アセンブリプログラム内で下線を付加しているか。	正しい変数名で参照して下さい。	「第 2 章 C/C++プログラミング 2.2.4.1 外部名の相互参照方法」
		C/C++プログラムで C ライブラリ関数を使用していないか。	リンク時に入力ライブラリとして標準ライブラリを指定して下さい。	標準ライブラリの指定： 「第 1 章 概要・操作 1.3.5 標準ライブラリとの対応」
		未定義参照シンボル名が__で始まっていないか。 (標準ライブラリ中の実行時ルーチンを使用しています)		ルーチン： 「第 3 章 システム組み込み 3.2.1(2)サイズの算出法」
		C ライブラリ関数の標準入力ライブラリを使用していないか。	低水準インタフェースルーチンを作成してリンクして下さい。	「第 3 章 システム組み込み 3.4(6)低水準インタフェースルーチン」
3	C/C++ソースレベルデバッグができない。	コンパイル時に debug オプション、リンク時に sdebug オプションを指定したか。	コンパイル時に debug オプション、リンク時に sdebug オプションを指定して下さい。	「第 1 章 概要・操作 1.3.3 コンパイラオプション」
		リンケージエディタの Ver.6.0 以上を使用しているか。	リンケージエディタの Ver.6.0 以上を使用してください。	
4	リンク時に、エラー No. 108 relocation	GBR ベース変数の指定で、指定した変数のオフセットは	制限を越えるデータに対し、#pragma gbr_base/	「第 2 章 C/C++プログラミング 2.3.9GBR ベー

2. C/C++プログラミング

	size overflow が出力される。	制限内におさまっているか。	gbr_base1 宣言を削除してください。	ス変数の指定」
No	現象	確認内容	対処方法	参照
5	リンク時に、エラー No. 104 duplicate symbol が出力される。	同じ名称の変数または関数を複数のファイル内で外部定義していないか。	名前を変更するかまたはstatic を指定してください。	
		複数のファイルでインクルードされるヘッダファイル内で変数または関数を外部定義していないか。 (#pragma inline/ inline_asm 指定した関数でも同様です)	static を指定してください。	「第2章 C/C++プログラミング 2.3.6(3)使用上の注意事項 2.3.7(3)使用上の注意事項」

3. システム組み込み

3.1 システム組み込みの概要

本節では、SuperH マイコンを応用したシステムに C/C++ プログラムを組み込む方法を説明します。

C/C++ プログラムをシステムに組み込むには、以下の準備が必要です。

(1) メモリの割り付け

C/C++ プログラムの各セクション、スタック領域、ヒープ領域をシステム上の ROM、RAM のメモリ領域に割り当てる必要があります。

(2) C/C++ プログラム実行環境の設定

C/C++ プログラムの実行環境を設定する処理には、レジスタの初期設定、メモリ領域の初期化、C/C++ プログラムの起動があります。これらの機能はアセンブリプログラムで実現する必要があります。

また、入出力等のライブラリ関数をご使用になる場合は、実行環境の設定時にライブラリの初期化をする必要があります。

2 節では C/C++ プログラムのメモリ領域のアドレスを決定する考え方を説明し、実際にアドレスを決定するためのリンケージエディタのコマンドの指定方法について実例を挙げて説明します。

3 節では実行環境設定の項目を説明し、設定プログラムの実例について説明します。

4 節ではライブラリ関数の初期設定処理と低水準ルーチンの作成方法を説明します。

【注】

入出力(stdio.h)とメモリ割り付け(stdlib.h)の機能をご使用になる場合は、システムのハードウェア構成にあわせて低水準の入出力ルーチンやメモリ割り付けルーチンを作成する必要があります。

3.2 メモリ領域の割り付け

本コンパイラの出力したオブジェクトプログラムをシステムに組み込むためには、プログラムの使用するメモリ領域のサイズを決定し、それぞれの領域を適切なメモリアドレスに割り付ける必要があります。

C/C++プログラムが使用するメモリ領域には、C/C++プログラム中の関数に対応する機械語や外部データ定義や静的データメンバで宣言したデータ領域のように静的に割り付ける領域とスタック領域のように動的に割り付ける領域があります。

以下、各領域の割り付け方を説明します。

3.2.1 静的領域の割り付け

(1) 静的領域の内容

オブジェクトプログラムの各セクション(プログラム領域、定数領域、初期化データ領域、未初期化データ領域、初期処理データ領域、後処理データ領域、仮想関数表領域)は静的領域に割り付けます。

(2) サイズの算出法

静的領域のサイズは、コンパイラが生成するオブジェクトプログラムサイズと C/C++プログラムが使用するライブラリ関数のサイズの合計になります。オブジェクトプログラムをリンクしたあと、リンケージマッピングリストにライブラリを含めた各セクションのサイズを出力しますので、静的領域のサイズを知ることができます。

リンク前に静的領域のサイズを概算する場合は、コンパイルリストの統計情報にセクションごとのサイズを出力しますので、これに基づいてサイズを算出することができます。

図 3-1 に統計情報の例を示します。各セクションごとの領域のトータルサイズは、Ver.5.0 では出力されません。

```
***** SECTION SIZE INFORMATION *****
PROGRAM SECTION(P) :0x00004A Byte(s)
CONSTANT SECTION(C) :0x000018 Byte(s)
DATA SECTION(D)      :0x000004 Byte(s)
BSS SECTION(B)       :0x000004 Byte(s)

TOTAL PROGRAM SECTION :0x00004A Byte(s)
TOTAL CONSTANT SECTION :0x000018 Byte(s)
TOTAL DATA SECTION   :0x000004 Byte(s)
TOTAL BSS SECTION     :0x000004 Byte(s)

TOTAL PROGRAM SIZE: 0x00006A Byte(s)
```

図 3-1 統計情報例

標準ライブラリを使用しない場合は、コンパイル単位ごとに出力する統計情報のセクションごとのサイズの合計が静的領域のサイズになります。

また、標準ライブラリを使用している場合、各セクションのメモリ領域サイズにはライブラリ関数の使用するメモリ領域サイズを加えなければなりません。標準ライブラリ関数には、C 言語仕様で規定したライブラリ関数の他に、C/C++プログラムを実行する上で必要な算術演算を行うルーチン(実行時ルーチン)も含まれています。そのため、C/C++ソースプログラム上でライブラリ関数の使用を指定しなくても、必ず標準ライブラリをリンクする必要があります。

本コンパイラが提供する標準ライブラリの中には、C 言語仕様で規定したライブラリ関数と、C/C++プログラムを実行する上で必要な算術演算を行うルーチン(実行時ルーチン)を含みます。実行時ルーチンのサイズもライブラリ関数と同じようにメモリ領域サイズに加える必要があります。

C/C++プログラムで使用する実行時ルーチンは、本コンパイラ出力のアセンブリプログラム(オプション `code = asmcode` 指定)に外部参照シンボルとして出力しますので、そのシンボル名を参照することによって C/C++プログラムで使用する実行時ルーチン名を知ることができます。また、`listfile` オプションでも知ることができます。

以下に具体例を示します。

例 C/C++プログラム例

```
f( int a, int b)
{
    a /= b;
    return a;
}
```

C コンパイル時に生成されるアセンブリプログラム例

```
.IMPORT    __divls      (実行時ルーチンの外部参照宣言)
.EXPORT    _f
.SECTION   P, CODE, ALIGN=4
_f:
                                ;function: f
                                ;frame size=4
                                ;used runtime library name:
                                ; __divls

    STS.L    PR, @-R15
    MOV      R5, R0
    MOV.L    L218, R3          ; __divls
    JSR      @R3
    MOV      R4, R1
    LDS.L    @R15+, PR
    RTS
    NOP
L218:
    .DATA.L   __divls
    .END
```

C++コンパイル時に生成されるアセンブリプログラム例

```

    .IMPORT    __divls      (実行時ルーチンの外部参照宣言)
    .EXPORT    __f__FSiT1
    .SECTION   P, CODE, ALIGN=4
__f__FSiT1:
                                ;function: f(signed int, signed int)
                                ;frame size=4
                                ;used runtime library name:
                                ;__divls

    STS.L      PR, @-R15
    MOV        R5, R0
    MOV.L      L218, R3        ;__divls
    JSR        @R3
    MOV        R4, R1
    LDS.L      @R15+, PR
    RTS
    NOP
L218:
    .DATA.L    __divls
    .END

```

上記例では `__divls` が C/C++ プログラムで使用する実行時ルーチンになります。

(3) ROM、RAMの割り付け

プログラムをメモリに割り付ける場合は、静的な領域を以下のように ROM と RAM に分けて割り付けます。

プログラム領域	(セクション P)	ROM
定数領域	(セクション C)	ROM
未初期化データ領域	(セクション B)	RAM
初期化データ領域	(セクション D)	ROM、RAM
(下記(4)参照)		
初期処理データ領域 ^{*1}	(セクション D_INIT_)	ROM
後処理データ領域 ^{*1}	(セクション D_END_)	ROM
仮想関数表領域 ^{*2}	(セクション C_\$VTBL)	ROM

^{*1}:C++コンパイル時にグローバルクラスオブジェクトがあるときにコンパイラが生成します。

^{*2}:C++コンパイル時に仮想関数宣言があるときにコンパイラが生成します。

(4) 初期化データ領域の割り付け

初期化データ領域は、初期値を持ったデータを集めた領域です。この領域にあるデータは値の変更が可能なので、リンク時には **ROM** 上に置き、プログラムの実行開始時に **RAM** 上にコピーする必要があります。したがって、初期化データの領域については、**ROM** 上と **RAM** 上に、二重に領域を確保しなければなりません。

ただし、初期値を指定した静的変数を変更しないようにプログラムを作成すれば、初期化データの領域は **ROM** 上に置くだけでよく、**RAM** 上に割り付ける必要はありません。

(5) メモリ割り付け例とリンク時のアドレス指定方法

アブソリュートロードモジュール作成時にリンカージェディタのオプションまたはサブコマンドで各セクションごとに割り付ける領域のアドレスを指定します。以下、静的領域のメモリ割り付け例とリンク時の指定方法について説明します。

図 3-2 に静的な領域の割り付け例を示します。

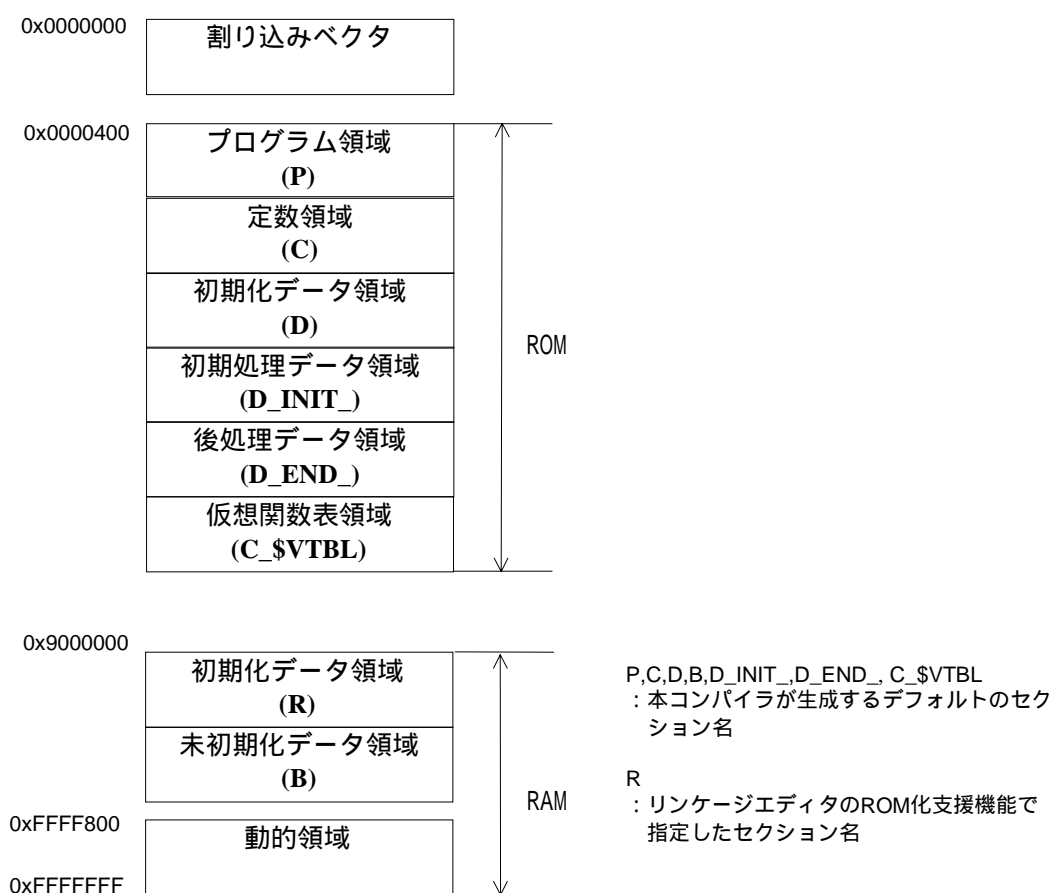


図 3-2 静的な領域の割り付け例

図 3-2 に示すメモリ割り付けを行う場合、リンク時に以下のサブコマンドを指定します。

```

:
ROM (D,R) (a)
START P,C,D,D_INIT_,D_END_,C_$VTBL(400),R,B(9000000) (b)
:

```

説明

- (a) セクション名 **D** と同じ大きさのセクション **R** を出力ロードモジュールに確保します。また、セクション **D** に割り付けられたシンボルを参照している場合、セクション **R** 上のアドレスとなるようリロケーションします。セクション **D** は **ROM** 上、セクション **R** は **RAM** 上の初期化データセクション名となります。
- (b) セクション **P**、**C**、**D**、**D_INIT_**、**D_END_**、**C_\$VTBL** を内蔵 **ROM** のアドレス **0x400** から連続した領域に割り付けます。また、セクション **R**、**B** を **RAM** のアドレス **0x9000000** から連続したアドレスに割り付けます。

3.2.2 動的領域の割り付け

(1) 動的領域の内容

C/C++ プログラムで使用する動的領域には以下の二つがあります。

- ・スタック領域
- ・ヒープ領域(メモリ割り付けライブラリ関数で使用)

(2) サイズの算出法

(a) スタック領域

C/C++ プログラムの使用するスタック領域は、関数の呼び出しのたびにスタック上に割り付け、関数のリターン時に解放します。スタック領域のサイズを算出するためには、まず各関数ごとのスタック使用量を算出し、関数の呼び出し関係から実際のスタック使用量を算出します。

(ア) 各関数の使用するスタック領域

各関数の使用するスタック領域は、コンパイラ出力のオブジェクトリスト中の **frame size** から分かります。

以下にオブジェクトリストとスタック上の割り付けの具体例を示し、そのスタック使用量の算出法について説明します。

例

次の C プログラムに対するオブジェクトリストとスタック使用量の算出法を示します。
C++ プログラムでも同様です。

```
extern int h(char, int *, double );
int h(char a, register int *b, double c)
{
    char    *d;

    d= &a;
    h(*d,b,c);
    {
        register int i;

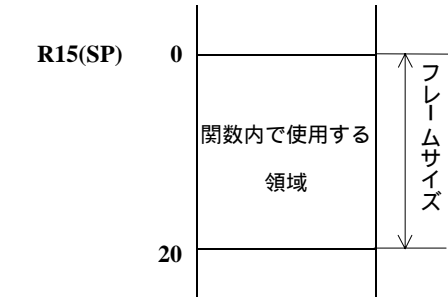
        i= *d;
        return i;
    }
}
```

***** OBJECT LISTING *****

FILE NAME: m0251.c

SCT	OFFSET	CODE	C LABEL	INSTRUCTION	OPERAND	COMMENT
P						
	00000000		_h:			; function: h
						; frame size=20
	00000000	2FE6		MOV.L	R14,@-R15	
	00000002	4F22		STS.L	PR,@-R15	
			:			

下位アドレス



上位アドレス スタック

関数の使用するスタック領域サイズは、フレームサイズの値と同じです。したがって、上記の例で関数 **h** の使用するスタック領域サイズは、オブジェクトリスト中の項目 **COMMENT** の **frame size** の値 20 バイトとなります。

スタック上の引数領域に割り付けられる引数については、「第 2 章 C/C++プログラミング 2.2.4.2 (4) 引数とリターン値の設定、参照に関する規則」を参照してください。

(b) スタック使用量の算出法

関数呼び出しの関係から使用するスタック領域のサイズを算出します。

例 関数呼び出しの関係と、各関数のスタック使用量の例を図 3-3 に示します。

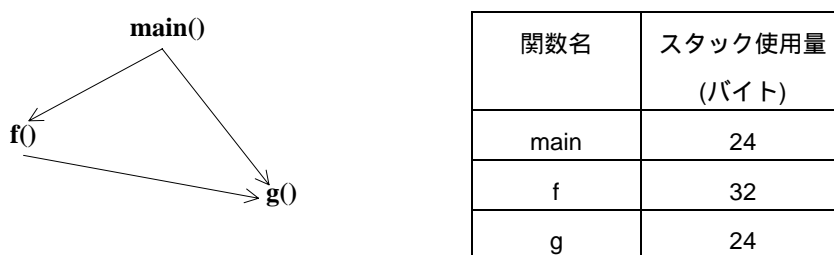


図 3-3 関数呼び出しの関係とスタック使用量の例

この場合、関数「f」を介して関数「g」が呼ばれた時のスタック領域のサイズは、表 3-1 によって計算します。

表 3-1 スタックサイズの計算例

呼び出し経路	スタックサイズ計(バイト)
main(24) f(32) g(24)	80
main(24) g(24)	48

このように、呼び出しレベルの一番深い関数についてスタック領域のサイズを計算し、その最大値(この場合 80 バイト)のスタック領域を最低限割り付ける必要があります。

標準ライブラリのライブラリ関数を使用する場合には、ライブラリ関数を含めたスタック領域のサイズを計算する必要があります。ライブラリ関数の使用するスタック領域のサイズについては、製品添付の「標準ライブラリのメモリ・スタック使用量一覧」を参照してください。

【注】

C/C++プログラムの中で再帰呼び出しを行っている場合は、再帰的に呼び出す回数の最大値を算出してから、その関数のスタック領域のサイズに再帰的に呼び出す回数をかけて計算してください。

(c) ヒープ領域

ヒープ領域で使用するメモリ領域のサイズは、C/C++プログラム内でメモリ管理ライブラリ関数(`calloc`,`malloc`,`realloc`,`new` 関数)によって割り付ける領域の合計です。ただし、メモリ管理ライブラリ関数は、1回の呼び出しのたびに管理用の領域として12バイト使用します。実際に確保する領域サイズにこの管理領域のサイズを加えて計算してください。

また、コンパイラは、ヒープ領域を1024バイト単位で管理しています。ヒープ領域として確保する領域サイズ(HEAPSIZE)は次のように計算してください。

$$\text{HEAPSIZE} = 1024 \times n (n - 1)$$

(メモリ管理ライブラリによって割り付ける領域サイズ)+管理領域サイズ **HEAPSIZE**

入出力ライブラリ関数は、内部処理の中でメモリ管理ライブラリ関数を使用しています。入出力の中で割り付ける領域のサイズは、

$$516 \text{ バイト} \times (\text{同時にオープンするファイルの数の最大値})$$

になります。

【注】

メモリ管理ライブラリ関数の `free`、または `delete` 関数で解放した領域は、再びメモリ管理ライブラリ関数で領域を確保するときに再利用しますが、割り付けを繰り返すことによって空き領域のサイズの合計は十分でも空き領域が小さな領域に分割しているために、新たに要求した大きなサイズの領域を確保できないという状況が生じることがあります。このような状況を避けるために、以下の注意に従ってヒープ領域を使用してください。

(ア) サイズの大きな領域は、なるべくプログラムの実行開始直後に確保してください。

(イ) 解放して再利用するデータ領域のサイズをなるべく一定にしてください。

(3) 動的領域の割り付け方

動的領域はRAM上に割り付けます。

スタック領域は、ベクタテーブルにスタック領域の最上位アドレスをSP(スタックポインタ)として設定することにより割り付ける場所が決まります。SH3、SH3E、SH4では割り込み時の動作がSH1、SH2、SH2Eの場合と異なるので、割り込みハンドラが必要になります。

ヒープ領域は、低水準インタフェースルーチン(`sbrk`)の初期設定で割り付ける場所が決まります。

それぞれ、「第3章 システム組み込み 3.3(1) ベクタテーブルの設定(VEC_TBL)」、「第3章 システム組み込み 3.4(6) 低水準インタフェースルーチン」を参照してください。

3.3 実行環境の設定

本節では、C/C++プログラムの実行に必要な環境を設定するための処理について説明します。ただし、C/C++プログラムを実行する環境はユーザシステムごとに異なりますので、使用するシステムの仕様に合わせて実行環境の設定プログラムを作成する必要があります。

ここでは、プログラムの実行環境の最も基本的な構成として、ライブラリ関数を使用しない場合について説明します。

ライブラリ関数、低水準の入出力ルーチン、メモリ割り付けルーチンを使用する場合は、「第3章 システム組み込み 3.4 ライブラリ関数の実行環境の設定」を参照してください。

図 3-4 にプログラムの構成例を示します。

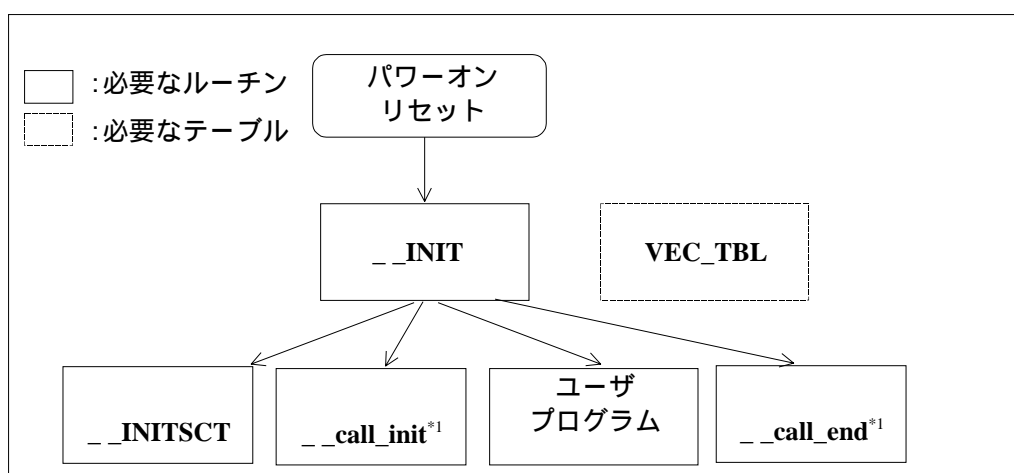


図 3-4 プログラムの構成例 (ライブラリ関数を使用しない場合)

*1:C++プログラム中にグローバルクラスオブジェクトの宣言があるとき必要になります。

各構成ルーチンの内容は以下のとおりです。

(1) ベクタテーブルの設定 (VEC_TBL)

パワーオンリセットでレジスタの初期設定プログラム(__INIT)が起動され、またスタックポインタ(SP)に値が設定されるように、ベクタテーブルを設定します。SH3、SH3E、SH4 では割り込み時の動作がSH1、SH2、SH2E の場合と異なるので割り込みハンドラが必要になります。

(2) 初期設定(__INIT)

レジスタの初期設定を行ったあと、初期設定ルーチンを順次呼び出します。

(3) セクションの初期化(__INITSCT)

初期値が設定されていない静的変数領域(未初期化データ領域)をゼロで初期化します。また、初期化データ領域の初期値を ROM 上から RAM 上にコピーします。

(4) グローバルクラスオブジェクト初期処理^{*1}(`_ _call_init`)

グローバルに宣言されたクラスオブジェクトに対してコンストラクタを呼び出します。

(5) グローバルクラスオブジェクト後処理^{*1}(`_ _call_end`)

main 関数の実行後、グローバルクラスオブジェクトに対してデストラクタを呼び出します。

^{*1}:C++ソースプログラム中にグローバルクラスオブジェクトの宣言があるときに必要な処理です。

以下、この構成に従って各処理の実現方法について説明します。

(1) ベクタテーブルの設定(VEC_TBL)

パワーオンリセットで、レジスタの初期設定を行う関数「`_ _INIT`」が呼び出されるようにするためには、ベクタテーブルの 0 番地に関数「`_ _INIT`」の先頭アドレスを設定します。また、スタックポインタ(SP)を設定するためには 4 番地にスタック領域の最上位アドレスを設定します。SH3、SH3E、SH4 では割り込み時の動作が SH1、SH2、SH2E の場合と異なるので割り込みハンドラが必要になります。

また、ユーザシステムで割り込み処理を使用する場合は、割り込みベクタの設定も本ルーチンで行います。以下にそのコーディング例を示します。

例(SH1、SH2、SH2E 用)

```
.SECTION VECT,DATA,LOCATE=H'0000
                                ; セクション制御命令で「VECTセクション」を0番地に配置

.IMPORT  _ _INIT
.IMPORT  _IRQ0
.DATA.L  _ _INIT      ; 「_ _INIT」の先頭アドレスを0x0 ~ 0x3番地に配置
.DATA.L  (a)          ; スタックポインタの値を0x4 ~ 0x7番地に配置
                                ; (a) : スタック領域の最上位アドレス

.ORG     H'00000100
.DATA.L  _IRQ0        ; 「IRQ0」の先頭アドレスを0x100 ~ 0x103番地に配置
.END
```

(2) 初期設定(_ _INIT)

ここでは、レジスタの初期設定を行い、初期設定ルーチンを順次呼び出したあと、**main**関数を呼び出します。

以下にコーディング例を示します。

例

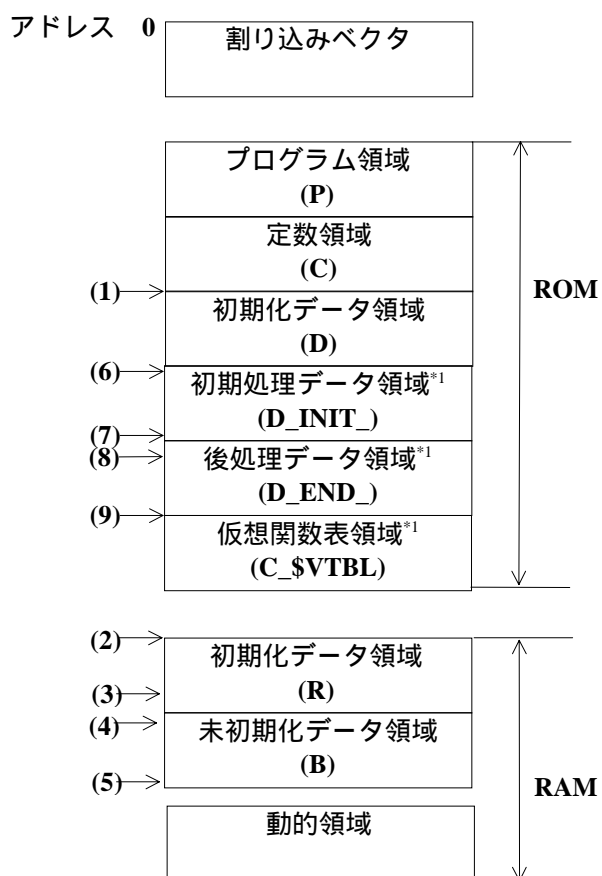
```
#ifdef __cplusplus
extern "C"{
#endif
void _INITSCT(void);
void main(void);
void _call_init(void);
void _call_end(void);
#ifdef __cplusplus
}
#endif
#ifdef __cplusplus
extern "C"
#endif
void _INIT()
{
    _INITSCT();
    /* セクションの初期化ルーチン「_ _INITSCT」の呼び出し*/
    _call_init();
    main();
    _call_end();
    /* メインルーチン「_main」の呼び出し*/
    for( ; ; )
        ;
    /* main 関数終了後、無限ループしてリセットを待つ*/
}
```

(3) セクションの初期化(_ _INITSCT)

C/C++プログラムの実行環境を設定するために、未初期化データ領域をゼロで初期化することとROM上にある初期化データをRAM上にコピーすることが必要です。

「_ _INITSCT」の処理を行うためには、次のアドレスを知る必要があります。

- ・初期化データ領域のROM上の先頭アドレス(1)
- ・初期化データ領域のRAM上の先頭アドレス(2)、最終アドレス(3)
- ・未初期化データ領域の先頭アドレス(4)、最終アドレス(5)
- ・初期処理データ領域^{*1}のROM上の先頭アドレス(6)、最終アドレス(7)
- ・後処理データ領域^{*1}のROM上の先頭アドレス(8)、最終アドレス(9)



これらのアドレスを知るためには、次のアセンブリプログラムを作成、リンクしてください。

```

        .SECTION D,DATA,ALIGN=4
        .SECTION R,DATA,ALIGN=4
        .SECTION B,DATA,ALIGN=4
        .SECTION D_INIT_,DATA,ALIGN=4*1
        .SECTION D_END_,DATA,ALIGN=4*1
        .SECTION C,DATA,ALIGN=4

__D_ROM    .DATA.L (STARTOF D)
;セクションDの先頭アドレス          (1)
__D_BGN    .DATA.L (STARTOF R)
;セクションRの先頭アドレス          (2)
__D_END    .DATA.L (STARTOF R) + (SIZEOF R)
;セクションRの最終アドレス          (3)
__B_BGN    .DATA.L (STARTOF B)
;セクションBの先頭アドレス          (4)
__B_END    .DATA.L (STARTOF B) + (SIZEOF B)
;セクションBの最終アドレス          (5)
__PRE_BGN  .DATA.L (STARTOF D_INIT_)*1
;セクションD_INIT_の先頭アドレス    (6)
__PRE_END  .DATA.L (STARTOF D_INIT_) + (SIZEOF D_INIT_)*1
;セクションD_INIT_の最終アドレス    (7)
__POST_BGN .DATA.L (STARTOF D_END_)*1
;セクションD_END_の先頭アドレス      (8)
__POST_END .DATA.L (STARTOF D_END_) + (SIZEOF D_END_)*1
;セクションD_END_の最終アドレス      (9)
        .EXPORT __D_ROM
        .EXPORT __D_BGN
        .EXPORT __D_END
        .EXPORT __B_BGN
        .EXPORT __B_END
        .EXPORT __PRE_BGN
        .EXPORT __PRE_END
        .EXPORT __POST_BGN
        .EXPORT __POST_END

```

【注】

(1)セクション名 B、D はコンパイラオプション section、拡張機能#pragma section で指定した未初期化データ領域、初期化データ領域のセクション名を指定してください。B、D はデフォルトのセクション名です。

(2)セクション名 R は、リンク時に ROM 化支援オプション ROM で指定した RAM 上のセクション名を指定してください。R はデフォルトのセクション名です。

*1:グローバルクラスオブジェクトのあるプログラムを C++コンパイルしたときに必要です。

上記の準備をすれば、セクションの初期化ルーチンは C/C++言語で記述することができます。以下にプログラム例を示します。

セクション初期化ルーチンの例

```

extern int *_D_ROM, *_B_BGN, *_B_END, *_D_BGN, *_D_END;
#ifdef __cplusplus
extern "C"
#endif
void _INITSCT( )
{
    int *p, *q;

    /*未初期化データ領域をゼロで初期化*/

    for( p = _B_BGN; p < _B_END; p++)
        *p = 0;

    /*初期化データをROM上からRAM上へコピー*/

    for(p = _D_BGN, q = _D_ROM; p < _D_END; p++, q++)
        *p = *q;
}

```

【注】セクションのサイズが4の倍数バイトでない場合は、p、qの宣言を char*にする必要があります。

グローバルオブジェクトの初期/後処理ルーチンの例

```

void (**_PRE_BGN)();
void (**_PRE_END)();
void (**_POST_BGN)();
void (**_POST_END)();
extern "C" {
    void _call_init();
    void _call_end();
}
extern "C" void _call_init()
{
    void (**ppf)();
    for( ppf = _PRE_BGN; ppf < _PRE_END; ppf++)
        (*ppf)();
    //グローバルクラスオブジェクトのコンストラクタ呼び出し
}

extern "C" void _call_end()
{
    void (**ppf)();
    for( ppf = _POST_BGN; ppf < _POST_END; ppf++)
        (*ppf)();
    //グローバルクラスオブジェクトのデストラクタ呼び出し
}

```

3.4 ライブラリ関数の実行環境の設定

ライブラリ関数を使用する場合は、C/C++プログラムの実行環境の設定としてライブラリ関数の初期化をする必要があります。特に入出力(`stdio.h`)とメモリ割り付け(`stdlib.h`)の機能を使用する場合や、プログラム終了処理を行うライブラリ関数を使用する場合は、システムごとに低水準の入出力ルーチンやメモリ割り付けルーチンを作成する必要があります。

本節では、ライブラリ関数使用時の C/C++プログラムの実行環境の設定方法について説明します。

図 3-5 にプログラムの構成例を示します。

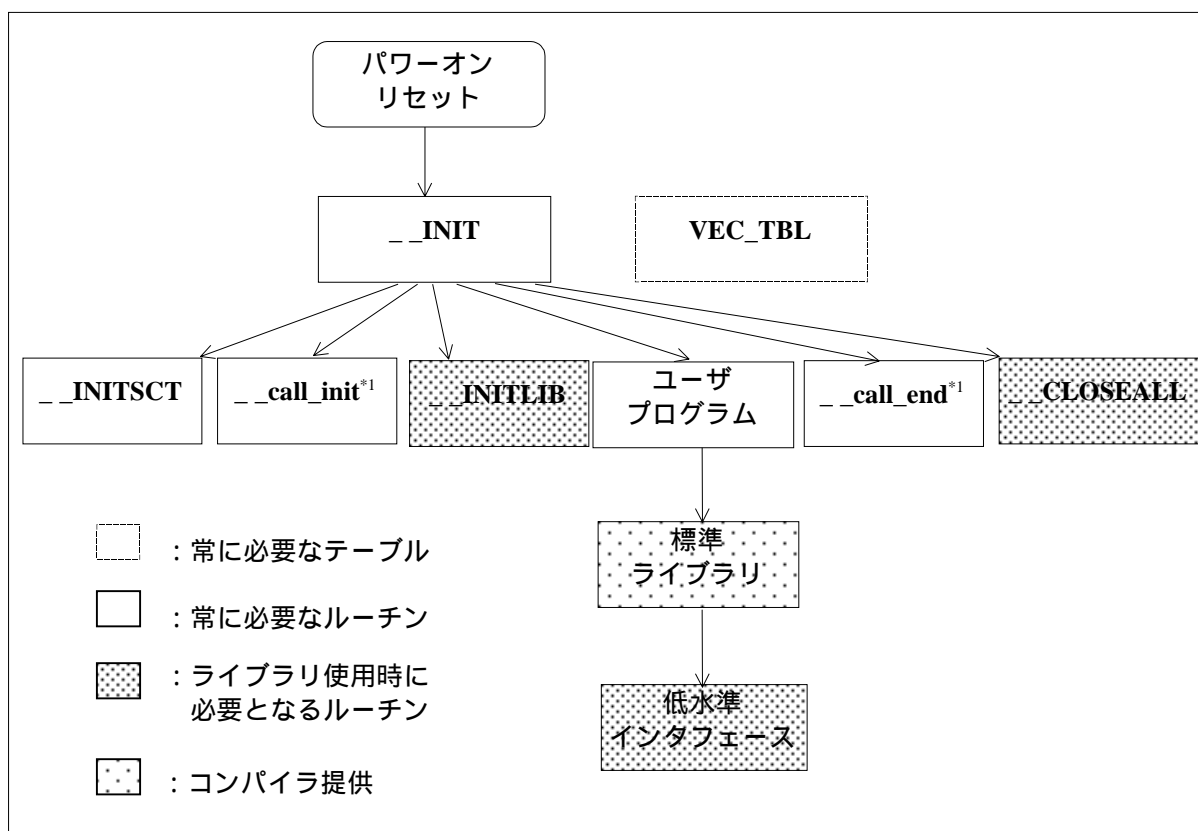


図 3-5 プログラムの構成例 (ライブラリ関数を使用する場合)

*1:グローバルクラスオブジェクトがあるときに必要になります。

プログラムの終了処理を行うライブラリ関数 `exit`、`onexit`、`abort` 関数を使用する場合は、ユーザシステムに合わせてこれらの関数を作成する必要があります。具体的なプログラム例は「付録 D. 終了処理関数の作成例」を参照してください。なお、ライブラリ関数 `assert` マクロを使用する場合、`abort` 関数は必ず作成する必要があります。

以下にライブラリ使用時に必要な各構成ルーチンの内容を示します。

(1) ベクタテーブルの設定(VEC_TBL)

パワーオンリセットでレジスタの初期設定プログラム(`_ _INIT`)が起動され、またスタックポインタ(SP)に値が設定されるように、ベクタテーブルを設定します。SH3、SH3E、SH4 では割り込み時の動作が SH1、SH2、SH2E の場合と異なるので割り込みハンドラが必要になります。

(2) 初期設定(`_ _INIT`)

レジスタの初期設定を行ったあと、初期設定ルーチンを順次呼び出します。

(3) セクションの初期化(`_ _INITSECT`)

初期値が設定されていない静的変数領域(未初期化データ領域)をゼロで初期化します。また、初期化データ領域の初期値を ROM 上から RAM 上にコピーします。

(4) ライブラリの初期設定(`_ _INITLIB`)

ライブラリ関数の中で、初期設定の必要なものについて、初期設定を行います。特に、標準入出力を行う準備をします。

(5) ファイルのクローズ(`_ _CLOSEALL`)

オープンしているファイルをすべてクローズします。

(6) 低水準インタフェースルーチン

標準入出力、メモリ管理ライブラリを使用する場合に必要なライブラリ関数とユーザシステムとの間のインタフェースルーチンです。

(7) グローバルクラスオブジェクト初期処理(`_ _call_init`)

グローバルに宣言されたクラスオブジェクトに対してコンストラクタを呼び出します。

(8) グローバルクラスオブジェクト後処理(`_ _call_end`)

`main` 関数の実行後、グローバルクラスオブジェクトに対してデストラクタを呼び出します。

以下、この構成に従って各処理の実現方法について解説します。

(1) ベクタテーブルの設定(VEC_TBL)

ベクタテーブルはライブラリ関数を使用しない場合と同じです。「第3章 システム組み込み 3.3 実行環境の設定」を参照してください。

(2) 初期設定(_ _INIT)

ライブラリ関数を使用する場合には、本関数でライブラリの初期設定を行う

「_ _INITLIB」とファイルのクローズ処理を行う「_ _CLOSEALL」を呼び出します。

以下に「_ _INIT」のコーディング例を示します。SH3、SH3E、SH4 では割り込み時の動作が SH1、SH2、SH2E の場合と異なるので割り込みハンドラが必要になります。

例

```
#ifdef __cplusplus
extern "C"{
#endif
void _INITSCT(void);
void _INITLIB(void);
void main(void);
void _CLOSEALL(void);
void _INIT(void)
void _call_init();
void _call_end();
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
extern "C"
#endif
void _INIT(void)
{
    _INITSCT();           /*セクションの初期化ルーチン「_ _INITSCT」の呼び出し*/
    _INITLIB();           /*ライブラリの初期化ルーチン「_ _INITLIB」の呼び出し*/
    _call_init();         /*「_ _call_init」の呼び出し*/
    main();               /*メインルーチン「_main」の呼び出し*/
    _call_end();          /*「_ _call_end」の呼び出し*/
    _CLOSEALL();          /*ファイルのクローズルーチン「_ _CLOSEALL」の呼び出し*/

    for( ; ; )            /*main 関数終了後、無限ループしてリセットを待つ*/
        ;
}
```

(3) セクションの初期化(_ _INITSCT)

セクションの初期化はライブラリ関数を使用しない場合と同じです。「第3章 システム組み込み 3.3 実行環境の設定」を参照してください。

(4) ライブラリ関数の初期設定(_ _INITLIB)

ライブラリ関数の中には、初期設定が必要な関数があります。それらの関数を使用する場合、使用する前に定められた初期設定を行わなければなりません。本項では、プログラム起動ルーチン中の「_ _INITLIB」の中で初期設定を行う場合を例に説明します。

実際に使用する機能に合わせた必要最低限の初期設定を行うために、以下の指針を参考にしてください。

- (a) ライブラリのエラー状態を示す「`errno`」の初期設定はすべてのライブラリ関数共通に必要です。
- (b) `<stdio.h>`の各関数と `assert` マクロを使用する場合、標準入出力の初期設定が必要です。作成した低水準インタフェースルーチンの中で初期設定が必要な場合、低水準インタフェースルーチンに合わせた初期設定が必要です。
- (c) `rand` 関数、`strtok` 関数を使用する場合、標準入出力以外の初期設定が必要です。

ライブラリの初期設定を行うプログラム例を以下に示します。

例

```
#include <errno.h>
#ifdef __cplusplus
extern "C"{
#endif
void _INIT_LOWLEVEL(void) ;
void _INIT_IOLIB(void) ;
void _INIT_OTHERLIB(void) ;
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
extern "C"
#endif
void _INITLIB(void)
    /* アセンブリルーチンのシンボル名から下線を一つ削除 */
{
    errno=0;                /* ライブラリ共通の初期設定 */

    _INIT_LOWLEVEL( ) ;
    /* 低水準インタフェースの初期設定ルーチンの呼び出し*/
    _INIT_IOLIB( ) ;
    /* 標準入出力の初期設定ルーチンの呼び出し */
    _INIT_OTHERLIB( ) ;
    /* 標準入出力以外の初期設定ルーチンの呼び出し */
}
```

以下、標準入出力の初期設定ルーチン(`_INIT_IOLIB`)、標準入出力以外の初期設定ルーチン(`_INIT_OTHERLIB`)の作成例を示します。低水準インタフェースルーチンの初期設定ルーチン(`_INIT_LOWLEVEL`)は、ユーザ作成の低水準インタフェースルーチンの仕様に合わせて作成してください。

(a) 標準入出力の初期設定ルーチン(_INIT_IOLIB)の作成例

標準入出力の初期設定では、ファイルを参照するために必要な FILE 型データ(図 3-6)の初期設定と標準入出力ファイルのオープンを行います。FILE 型データの初期設定は、必ず標準入出力ファイルのオープンの前に行ってください。

標準入出力の初期設定を行うプログラム例を以下に示します。

例

```
#include <stdio.h>
#ifdef __cplusplus
extern "C"
#endif
void _INIT_IOLIB(void)
{
    FILE *fp ;
/* FILE 型データの初期設定 */
for (fp=_iob; fp<_iob+_NFILE; fp++){
    fp -> _bufptr=NULL ; /* バッファポインタのクリア*/
    fp -> _bufcnt=0 ;    /* バッファカウンタのクリア*/
    fp -> _buflen=0 ;    /* バッファ長のクリア*/
    fp -> _bufbase=NULL ; /* ベースポインタのクリア*/
    fp -> _ioflag1=0 ;   /* I/O フラグのクリア*/
    fp -> _ioflag2=0 ;
    fp -> _iofd=0 ;
}

/* 標準入出力ファイルのオープン*/

if (freopen( "stdin"*1 , "r", stdin)==NULL) /* 標準入力ファイルのオープン*/
    stdin->_ioflag1=0xff ;                  /* ファイルアクセスの禁止*2 */
    stdin->_ioflag1 |= _IOUNBUF ;           /* データのバッファリング無*3*/

if (freopen( "stdout"*1 , "w", stdout)==NULL) /* 標準出力ファイルのオープン*/
    stdout->_ioflag1=0xff ;
    stdout->_ioflag1 |= _IOUNBUF ;

if (freopen( "stderr"*1 , "w", stderr)==NULL) /* 標準エラーファイルのオープン*/
    stderr->_ioflag1=0xff ;
    stderr->_ioflag1 |= _IOUNBUF ;
}
```

*1:標準入出力ファイルのファイル名を指定します。この名前は、低水準インタフェースルーチン「open」で使します。

*2:ファイルのオープンが失敗した場合、ファイルアクセス禁止のフラグを立てます。

*3:コンソール等の対話的な装置の場合、バッファリングを行わないためのフラグを立てます。

```

/* ファイル型データのC/C++言語での宣言*/
#define _NFILE 20
struct _iobuf{
    unsigned char *_bufptr;        /* バッファへのポインタ*/
    long _bufcnt;                 /* バッファカウンタ*/
    unsigned char *_bufbase;      /* バッファベースポインタ*/
    long _buflen;                 /* バッファ長*/
    char _ioflag1;                /* I/O フラグ*/
    char _ioflag2;                /* I/O フラグ*/
    char _iofd;                   /* I/O フラグ*/
}_iob[_NFILE];

```

図 3-6 FILE 型データ

(b) 標準入出力以外の初期設定ルーチン(_INIT_OTHERLIB)の作成例

標準入出力以外で初期設定が必要なライブラリ関数(rand 関数、strtok 関数)の初期設定プログラム例を以下に示します。

例

```

#include <stddef.h>

extern char *_slpstr ;
extern void srand(unsigned int) ;
#ifdef __cplusplus
extern "C"
#endif
void _INIT_OTHERLIB(void)
{
    srand(1) ;          /* rand 関数を使用する場合の初期値の設定*/
    _slpstr=NULL ;      /* strtok 関数で使用するポインタの初期設定*/
}

```

(5) ファイルのクローズ(_ _CLOSEALL)

通常ファイルへの出力は、メモリ領域上のバッファにためておき、バッファがいっぱいになったときに実際の外部記憶装置への書き出しを行います。したがってファイルのクローズを行わないと、ファイルへの出力内容が外部記憶装置へ書き出されないことがあります。

機器組み込み用のプログラムの場合、通常プログラムが終了することはありません。しかし、プログラムの誤りなどにより main 関数が終了する場合、オープンしているファイルはすべてクローズしなければなりません。

本処理は、main 関数終了時にオープンしているファイルのクローズを行います。

ファイルのクローズを行うプログラム例を以下に示します。

例

```
#include <stdio.h>
#ifdef __cplusplus
extern "C"
#endif
void _CLOSEALL(void)
    /* アセンブリルーチンのシンボル名から下線を一つ削除*/
{
    int i;

    for (i=0; i<_NFILE; i++)
        /* ファイルがオープンしているかどうかのチェック*/

        if(_iob[i]._ioflag1 & ( _IOREAD | _IOWRITE | _IORW))
            /* オープンしているファイルのクローズ*/

            fclose(&_iob[i]) ;
}
```

(6) 低水準インタフェースルーチン

標準入力、メモリ管理ライブラリを C/C++プログラムで使用する場合は、低水準インタフェースルーチンを作成しなければなりません。表 3-2 にライブラリ関数で使用している低水準インタフェースルーチンの一覧を示します。

表 3-2 低水準インタフェースルーチンの一覧

項番	名称	機能
1	open	ファイルのオープン
2	close	ファイルのクローズ
3	read	ファイルからの読み込み
4	write	ファイルへの書き出し
5	lseek	ファイルの読み込み / 書き出し位置の設定
6	sbrk	メモリ領域の確保

各ライブラリ関数に対して必要な低水準インタフェースルーチンについては、製品添付の「ソフトウェア添付資料」の中の「標準ライブラリのメモリスタック使用量一覧」を参照してください。

低水準インタフェースルーチンで必要な初期化は、プログラム起動時に行う必要があります。これは、「(4)ライブラリ関数の初期設定(_INITLIB)」の中の「_INIT_LOWLEVEL」という関数の中で行ってください。

以下、低水準入出力の基本的な考え方を説明したあと、各インタフェースルーチンの仕様を説明します。また、シミュレータ・デバッガ上で実行する低水準インタフェースルーチン例を「付録 E. 低水準インタフェースルーチンの作成例」に示しますので、あわせて参照してください。

(a) 入出力の考え方

標準入出力ライブラリでは、ファイルを **FILE** 型のデータによって管理しますが、低水準インタフェースルーチンでは、実際のファイルと 1 対 1 に対応する正の整数を与え、これによって管理します。この整数をファイル番号といいます。

open ルーチンでは、与えられたファイル名に対してファイル番号を与えます。**open** ルーチンでは、この番号によってファイルの入出力ができるように、以下の情報を設定する必要があります。

- (1) ファイルのデバイスの種類(コンソール、プリンタ、ディスクファイル等)。コンソールやプリンタ等の特殊なデバイスに対しては、特別なファイル名をシステムで決めておいて **open** ルーチンで判定する必要があります。
- (2) ファイルのバッファリングをする場合はバッファの先頭位置、サイズ等の情報。
- (3) ディスクファイルならば、ファイルの先頭から次に読み込み/書き出しを行う位置までのバイトオフセット。

open ルーチンで設定した情報に基づいて、以後、入出力(**read**,**write** ルーチン)、読み込み/書き出し位置の設定(**lseek** ルーチン)を行います。

close ルーチンでは、出力ファイルのバッファリングを行っている場合はバッファの内容を実際のファイルに書き出し、**open** ルーチンで設定したデータの領域が再使用できるようにしてください。

(b) 低水準インタフェースルーチンの仕様

本項では、低水準インタフェースルーチンを作成するための仕様を説明します。以下、各ルーチンごとに、ルーチンを呼び出す際のインタフェースとその動作および実現上の注意事項を示します。

各ルーチンのインタフェースは以下の形式で示します。なお、低水準インタフェースルーチンは、必ずプロトタイプ宣言してください。

【注】関数名 **open**,**close**,**read**,**write**,**lseek**,**sbrk** は、低水準インタフェースルーチン予約語です。ユーザプログラムでは使用しないでください。

凡例

(ルーチン名)				
機能	(ルーチンの機能概要を示します。)			
インタフェース	(ルーチンのCプログラムとしての宣言方法を示します。) (ルーチンのC++プログラムとして宣言する場合は、extern “C”を付与してください。)			
引数	No.	名前	型	意味
	1	引数の名前です。	引数の型を示します。	(引数として渡される値の意味を示します。)
	⋮	⋮	⋮	⋮
リターン値	型	(リターン値の型を示します。)		
	正常	(正常に終了した場合のリターン値の意味を示します。)		
	異常	(エラーが生じた場合のリターン値を示します。)		

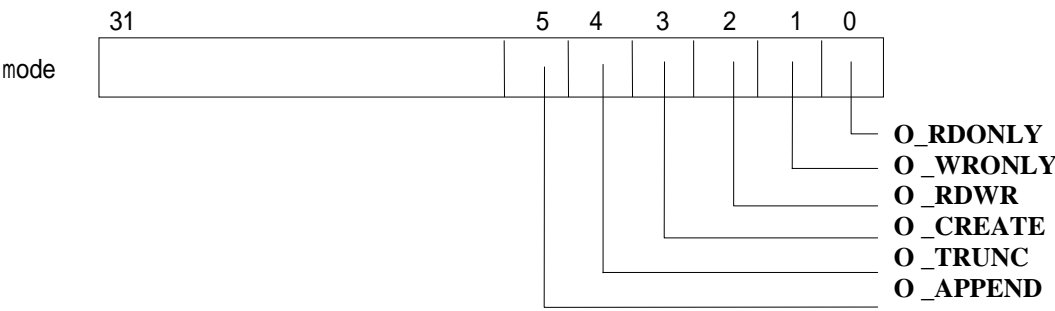
(a) open ルーチン				
機能	ファイルをオープンします。			
インタフェース	int open (char *name, int mode);			
引数	No.	名前	型	意味
	1	name	char型を指すポインタ	ファイルのファイル名を指す文字列
	2	mode	int	ファイルをオープンするときの処理の指定
リターン値	型	int		
	正常	オープンしたファイルのファイル番号		
	異常	-1		

説明

第1引数として渡されたファイル名に対応するファイル进行操作するための準備をします。

open ルーチンでは、後で読み込み/書き出しを行うために、ファイルの種類(コンソール、プリンタ、ディスクファイル等)を決定しなければなりません。ファイルの種類は、以後 **open** ルーチンで返したファイル番号を用いて読み込み/書き出しを行うたびに参照する必要があります。

第 2 引数の **mode** は、ファイルをオープンするときの処理の指定です。このデータの各ビットの意味について以下に示します。



O_RDONLY (0 ビット)	このビットが 1 のとき、ファイルを読み込み専用にオープン
O_WRONLY (1 ビット)	このビットが 1 のとき、ファイルを書き出し専用にオープン
O_RDWR (2 ビット)	このビットが 1 のとき、ファイルを読み込み、書き出し両用にオープン
O_CREATE (3 ビット)	このビットが 1 のとき、ファイル名で示すファイルが存在しない場合にファイルを新規に作成
O_TRUNC (4 ビット)	このビットが 1 のとき、ファイル名で示すファイルが存在する場合にファイルの内容を捨て、ファイルのサイズを 0 に更新
O_APPEND (5 ビット)	次の読み込み / 書き出しを行うファイル内の位置を設定 ビットが 0 のとき：ファイルの先頭に設定 ビットが 1 のとき：ファイルの最後に設定

mode で示したファイルの処理の指定と実際のファイルの性質が矛盾する場合はエラーにしてください。

正常にファイルがオープンできた場合は、以後の **read**、**write**、**lseek**、**close** ルーチンで使用するファイル番号(正の整数)を返してください。ファイル番号と実際のファイルの対応は、低水準インタフェースルーチンで管理する必要があります。オープンに失敗した場合は-1 を返してください。

(b) close ルーチン				
機能	ファイルをクローズします。			
インタフェース	int close (int fileno);			
引数	No.	名前	型	意味
	1	fileno	int	クローズするファイル番号
リターン値	型	int		
	正常	0		
	異常	-1		

説明

open ルーチンで得られたファイル番号が引数として渡されます。

open ルーチンで設定したファイル管理情報を再び使用できるように解放してください。

また、低水準インタフェースルーチン内で出力ファイルのバッファリングを行っている場合は、バッファの内容を実際のファイルに書き出してください。

ファイルを正常にクローズできた場合は 0、失敗した場合は -1 を返してください。

(c) read ルーチン				
機能	ファイルからデータの読み込みを行います。			
インタフェース	<pre>int read (int fileno, char *buf, unsigned int count);</pre>			
引数	No.	名前	型	意味
	1	fileno	int	読み込み対象となるファイル番号
	2	buf	char型を指すポインタ	読み込んだデータを設定する領域
	3	count	unsigned int	読み込むバイト数
リターン値	型	int		
	正常	実際に読み込まれたバイト数		
	異常	-1		

説明

第1引数(fileno)で示すファイルから、第2引数(buf)の指す領域へデータを読み込みます。読み込むデータのバイト数は第3引数(count)で示します。

ファイルが終了した場合、countで示されたバイト数以下のバイト数しか読み込むことができません。

ファイルの読み込み/書き出しの位置は、読み込んだバイト数だけ先に進みます。

正常に読み込みができた場合は、実際に読み込んだバイト数を返してください。読み込みに失敗した場合は-1を返してください。

(d) writeルーチン				
機能	ファイルへのデータの書き出しを行います。			
インタフェース	<pre>int write (int fileno, char *buf, unsigned int count);</pre>			
引数	No.	名前	型	意味
	1	fileno	int	書き出し対象となるファイル番号
	2	buf	char型を指すポインタ	書き出すデータの領域
	3	count	unsigned int	書き出すバイト数
リターン値	型	int		
	正常	実際に書き出されたバイト数		
	異常	-1		

説明

第2引数(buf)の指す領域から、第1引数(fileno)の示すファイルにデータを書き出します。書き込むデータのバイト数は第3引数(count)で示します。

ファイルを書き出そうとしているデバイス(ディスク等)が満杯の時は、countで示されたバイト数以下のバイト数しか書き出すことができません。実際に書き出すことのできたバイト数が何度か連続して0バイトの場合、ディスクが満杯であると判断してエラー(-1)を返すように実現することをお勧めします。

ファイルの読み込み/書き出しの位置は、書き出したバイト数だけ先に進みます。

正常に書き出しができた場合は、実際に書き出したバイト数を返してください。書き出しに失敗した場合は-1を返してください。

(e) lseekルーチン				
機能	ファイルの読み込み/書き出し位置を設定します。			
インタフェース	long lseek (int fileno, long offset, int base);			
引数	No.	名前	型	意味
	1	fileno	int	対象となるファイル番号
	2	offset	long	読み込み/書き出し位置を示すオフセット(バイト単位)
	3	base	int	オフセットの起点
リターン値	型	long		
	正常	新しいファイルの読み込み/書き出しの位置の先頭からのオフセット(バイト単位)		
	異常	-1		

説明

ファイルの読み込み/書き出しを行うファイル内の位置を、バイト単位で設定します。

新しいファイル内の位置は、第3引数(base)によって、以下の方法で計算し設定してください。

(1) base が 0 のとき

ファイルの先頭から offset バイトの位置に設定します。

(2) base が 1 のとき

現在の位置に offset バイトを加えた位置に設定します。

(3) base が 2 のとき

ファイルのサイズに offset バイトを加えた位置に設定します。

ファイルがコンソールやプリンタ等の対話的なデバイスの場合や、新しいオフセットの値が負になったり、(1)、(2)のときファイルのサイズを超える場合はエラーにします。

正しくファイル位置を設定できた場合は、新しい読み込み/書き出し位置のファイルの先頭からのオフセットを、そうでない場合は-1を返してください。

(f) sbrkルーチン				
機能	メモリ領域を割り付けます。			
インタフェース	char *sbrk (unsigned long size);			
引数	No.	名前	型	意味
	1	size	unsigned long	割り付けるデータのサイズ (バイト数)
リターン値	型	char型を指すポインタ		
	正常	割り付けた領域の先頭アドレス		
	異常	(char*) -1		

説明

メモリ領域を割り付けるサイズが引数として渡されます。

連続して sbrk ルーチンを呼び出す場合は、下位アドレスから順に連続した領域が割り付けられるようにしてください。

割り付けるメモリ領域が不足した場合はエラーにしてください。

正常に割り付けができた場合は、割り付けた領域の先頭アドレスを、失敗した場合は「(char *)-1」を返してください。

4. エラーメッセージ

4.1 コンパイラのエラーメッセージ

4.1.1 エラーメッセージ一覧

本章では、コンパイラの出力するエラーメッセージとエラー内容を説明します。

エラー番号 (エラーレベル) エラーメッセージ

エラーレベルは、エラーの重要度にしたがい5種に分類されます。

エラーレベル (I) インフォメーション
(W) ウォーニング
(E) エラー
(F) フェータル
(-) インターナル

0001 (I) "文字列" in comment

注釈の中に、"文字列"があります。

0002 (I) No declarator

宣言子のない宣言があります。

0003 (I) Unreachable statement

実行されることのない文があります。

0004 (I) Constant as condition

if 文または switch 文の条件を示す式として、定数式を指定しています。

0005 (I) Precision lost

代入において、右辺の式の値を左辺の型に変換するときに、精度が失われる可能性があります。

0006 (I) Conversion in argument

関数の引数の式が、原型宣言で指定した引数の型に変換されます。

0008 (I) Conversion in return

リターン文の式が、関数の返す値の型に変換されます。

0010 (I) Elimination of needless expression

不要な式があります。

0011 (I) Used before set symbol : "変数名"

値を設定せずに使用している変数 " 変数名 " があります。

0012 (I) Unused variable "変数名"

使用していない変数 " 変数名 " があります。

0015 (I) No return value

void 型以外の型を返す関数の中で、リターン文が値を返していないか、またはリターン文がありません。

0100 (I) Function "関数名" not optimized

関数 " 関数名 " のサイズが大きすぎるため、最適化できません。

0200 (I) No prototype function

関数のプロトタイプ宣言がされていません。

1000 (W) Illegal pointer assignment

ポインタ型どうしの代入で、それぞれのポインタ型の指す型が異なります。

1001 (W) Illegal comparison in "演算子"

二項演算子 `==` または `!=` の被演算子が、一方がポインタ型で他方が値 0 以外の汎整数型を指しています。

1002 (W) Illegal pointer for "演算子"

二項演算子 `=`、`!=`、`>`、`<`、`>=` または `<=` の被演算子が、同じ型へのポインタ型を指していません。

1005 (W) Undefined escape sequence

文字定数または文字列の中で、文法上定義していない拡張表記（逆スラッシュとそれに続く文字）を用いています。

1007 (W) Long character constant

文字定数の長さが 2 文字以上になっています。

1008 (W) Identifier too long

識別子の長さが 250 文字を超えています。

1010 (W) Character constant too long

文字定数の長さが 4 文字を超えています。

1012 (W) Floating point constant overflow

浮動小数点定数の値が値の範囲を超えています。符号にしたがって + または - に対応する内部表現の値を仮定します。

1013 (W) Integer constant overflow

整数定数の値が **unsigned long** 型のとり得る値の範囲を超えています。オーバフローした上位ビットを無視した値を仮定します。

1014 (W) Escape sequence overflow

文字定数あるいは文字列の中でのビットパターンを示す拡張表記の値が 255 を超えています。下位 1 バイトの値を有効とします。

1015 (W) Floating point constant underflow

浮動小数点定数の値の絶対値が表現できる最小値よりも小さな値となっています。定数の値を 0.0 と仮定します。

1016 (W) Argument mismatch

原型宣言の中の引数と関数呼び出しの対応する引数の型がポインタ型で、それぞれの指す型が異なります。関数呼び出しの引数のポインタの内部表現をそのまま設定します。

1017 (W) Return type mismatch

関数の返す型とリターン文の式の型がポインタ型で、それぞれの指す型が異なります。リターン文の式のポインタの内部表現をそのまま設定します。

1019 (W) Illegal constant expression

定数式において関係演算子 <、>、<= または >= の被演算子が、同じ型へのポインタ型を指していません。結果の値を 0 と仮定します。

1020 (W) Illegal constant expression of "-"

定数式において二項演算子 - の被演算子が、同じ型へのポインタ型を指していません。結果の値を 0 と仮定します。

1021 (W) Register saving pragma conflicts in interrupt function "関数名"

"関数名"で示す割り込み関数に対するレジスタ退避・回復を制御する**#pragma** が不適切です。**#pragma** 指定を無視します。

1022 (W) First operand of "演算子" is not lvalue

第 1 オペランドの"演算子"は、左辺値になりません。

1023 (W) Can not convert Japanese code "コード" to output type

日本語コード"コード"を指定の出力コードに変換できません。

1200 (W) Division by floating point zero

定数式の中で浮動小数点数 **0.0** を除数とする割り算を行っています。符号にしたがって、
+ または - に対応する内部表現の値を仮定します。

1201 (W) Ineffective floating point operation

定数式の中で - 、**0.0/0.0** 等の無効演算を行っています。無効演算の結果を表わす非数に対応する内部表現の値を仮定します。

1300 (W) Command parameter specified twice

同じコンパイラオプションを 2 度以上指定しています。同じコンパイラオプションの中で最後に指定したものを有効とします。

1301 (W) "browser" option ignored

C コンパイル時に、**browser** オプションを指定しています。**browser** オプションを無視します。

1302 (W) "double=float" option ignored

double=float、**cpu=sh4** オプションを同時に指定しています。**double=float** オプションを無視し、**fpu=single** オプションが指定されていると解釈してコンパイルをします。

1400 (W) Function "関数名" in **#pragma inline** is not expanded

#pragma inline で指定した関数がインライン展開されませんでした。コンパイル処理を継続します。

1500 (W) EC++ does not support "クラス名"

多重継承、仮想基底クラスで指定された"クラス名"は、EC++でサポートされていません。

2000 (E) Illegal preprocessor keyword

プリプロセッサ文で、誤ったキーワードを使用しています。

2001 (E) Illegal preprocessor syntax

プリプロセッサ文またはマクロ呼び出しの指定方法に誤りがあります。

2002 (E) Missing ","

引数のある**#define** 文で引数の並びを区切るコンマ(,)が抜けています。

2003 (E) Missing ")"

名前が**#define** 文で定義されているかどうかを判定する **defined** 式で名前の次の右括弧「)」が抜けています。

2004 (E) Missing ">"

#include 文のファイル名の指定でファイル名の次の > がありません。

2005 (E) Cannot open include file "ファイル名"

#include 文で指定したファイル名のファイルがオープンできません。

2006 (E) Multiple **#define**'s

#define 文で同じマクロ名を再定義しています。

2008 (E) Processor directive **#elif** mismatches

#elif 文に対応する**#if** 文、**#ifdef** 文、**#ifndef** 文、**#elif** 文がありません。

2009 (E) Processor directive **#else** mismatches

#else 文に対応する**#if** 文、**#ifdef** 文、**#ifndef** 文がありません。

2010 (E) Macro parameters mismatch

マクロ呼び出しの引数の数がマクロ定義の引数の数と異なっています。

2011 (E) Line too long

マクロ展開後のソースプログラムの行が限界値を超えています。

2012 (E) Keyword as a macro name

プリプロセッサで規定しているキーワードを**#define** 文または、**#undef** 文のマクロ名として定義しています。

2013 (E) Processor directive #endif mismatches

#endif 文に対応する#if、#ifdef、#ifndef 文がありません。

2014 (E) Missing #endif

#if 文、#ifdef 文、#ifndef 文に対応する#endif 文がないままファイルが終了しました。

2016 (E) Preprocessor constant expression too complex

#if、#elif 文で指定した定数式の演算子と被演算子の合計が限界値を超えています。

2017 (E) Missing "

#include 文のファイル名の指定で、ファイル名の次に " がありません。

2018 (E) Illegal #line

#line 文で指定した行数が限界値を超えています。

2019 (E) File name too long

ファイル名の長さが 128 文字を超えています。

2020 (E) System identifier "名前" redefined

実行時ルーチンと同名のシンボルを定義しています。

2100 (E) Multiple storage classes

宣言の中で二つ以上の記憶クラス指定子を指定しています。

2101 (E) Address of register

レジスタ記憶クラスを持つ変数に対して、単項演算子&を適用しています。

2102 (E) Illegal type combination

型指定子の組み合わせが誤っています。

2103 (E) Bad self reference structure

構造体、共用体のメンバの型を、親の構造体または共用体と同じ型で宣言しています。

2104 (E) Illegal bit field width

ビットフィールド幅を示す定数式が整数型ではありません。あるいはビットフィールド幅として負の整数を指定しています。

2105 (E) Incomplete tag used in declaration

構造体または共用体で仮宣言されたタグ名または、未宣言のタグ名を `typedef` 宣言、ポインタを指す型あるいは関数の返す型以外の宣言で使用しています。

2106 (E) Extern variable initialized

複文内で `extern` 記憶クラスを指定した宣言に対して初期値を指定しています。

2107 (E) Array of function

要素の型が関数型となる配列型を指定しています。

2108 (E) Function returning array

リターン値の型が配列型となる関数型を指定しています。

2109 (E) Illegal function declaration

複文内の関数型の変数の宣言において、`extern` 以外の記憶クラスを指定しています。

2110 (E) Illegal storage class

外部定義の中で記憶クラスとして `auto` または `register` を指定しています。

2111 (E) Function as a member

構造体または共用体のメンバの型に関数型を指定しています。

2112 (E) Illegal bit field

ビットフィールドに整数型以外の型を指定しています。

2113 (E) Bit field too wide

ビットフィールド幅が型指定子で指定したサイズ(8 ビット、16 ビット、32 ビット)を超えています。

2114 (E) Multiple variable declarations

変数名を同じ有効範囲の中で重複して宣言しています。

2115 (E) Multiple tag declarations

構造体、共用体、列挙型のタグ名を同じ有効範囲の中で重複して宣言しています。

2117 (E) Empty source program

ソースプログラム内に外部定義が含まれていません。

2118 (E) Prototype mismatch "関数名"

関数の型が以前になされている宣言で指定した型と一致しません。

2119 (E) Not a parameter name "引数名"

関数の引数宣言列にない識別子に対して引数宣言を行っています。

2120 (E) Illegal parameter storage class

関数の引数宣言で **register** 以外の記憶クラスを指定しています。

2121 (E) Illegal tag name

構造体、共用体または列挙型とタグ名の組み合わせが、以前に宣言した型とタグ名の組み合わせと異なっています。

2122 (E) Bit field width 0

メンバ名を指定しているビットフィールドの幅が 0 になっています。

2123 (E) Undefined tag name

列挙型の宣言で未定義のタグ名を使用しています。

2124 (E) Illegal enum value

列挙型のメンバに整数でない定数式を指定しています。

2125 (E) Function returning function

リターン値の型が関数型となる関数型を指定しています。

2126 (E) Illegal array size

配列の要素数の値が 1 以上 2147483647 以下の整数値以外の値を指定しています。

2127 (E) Missing array size

配列の要素数の指定がありません。

2128 (E) Illegal pointer declaration for ""

ポインタ型の宣言を示す * の直後に **const**、**volatile** 以外の型指定子を指定しています。

2129 (E) Illegal initializer type

変数の初期値指定において初期値の型が変数に代入可能な型ではありません。

2130 (E) Initializer should be constant

構造体型、共用体型、配列型の変数の初期値、または静的に割り付けられる変数の初期値に定数式でないものを指定しています。

2131 (E) No type nor storage class

外部データ定義において記憶クラスまたは型の指定がありません。

2132 (E) No parameter name

関数の引数宣言列が空であるにもかかわらず引数の宣言を行っています。

2133 (E) Multiple parameter declarations

(マクロ)関数定義の引数宣言列の中で同一名の引数を重複して宣言しているか、または引数宣言が関数宣言子の中と外の 2 箇所で行われています。

2134 (E) Initializer for parameter

引数の宣言において初期値を指定しています。

2135 (E) Multiple initialization

同一の変数に対して、初期化を重複して行っています。

2136 (E) Type mismatch

extern あるいは **static** 記憶クラスを持つ変数あるいは関数を 2 度以上宣言しており、その型が一致していません。

2137 (E) Null declaration for parameter

関数の引数宣言で識別子を指定していません。

2138 (E) Too many initializers

構造体、共用体または配列の初期値指定において、構造体のメンバ数または配列の要素数より多く初期値の数を指定しています。あるいは、共用体の最初のメンバがスカラ型のときに 2 個以上の初期値を指定しています。

2139 (E) No parameter type

関数宣言の引数宣言に型指定がありません。

2140 (E) Illegal bit field

共用体にビットフィールドを指定しています。

2141 (E) Struct has no member name

構造体のメンバ名が指定されていません。

2142 (E) Illegal void type

void 型の指定方法に誤りがあります。**void** 型を指定できるのは以下の三つの場合です。

(1)ポインタの指す先の型として指定する場合。

(2)関数の返す型として指定する場合。

(3)原型宣言の関数が引数を持たないことを明示的に指定する場合。

2143 (E) Illegal static function

ソースファイル内に定義のない **static** 記憶クラスを持つ関数宣言があります。

2144 (E) Type mismatch

extern 記憶クラスを持つ同じ名前の変数あるいは関数の型が一致していません。

2145 (E) Const/volatile specified for incomplete type

不完全型に対して **const** または **volatile** が指定されています。

2200 (E) Index not integer

配列の添字の式が整数型ではありません。

2201 (E) Cannot convert parameter "n"

関数呼び出しにおける **n** 番目の引数に対応する原型宣言の引数の型に変換できません。

2202 (E) Number of parameters mismatch

関数呼び出しにおける引数の数が原型宣言の引数の数と一致しません。

2203 (E) Illegal member reference for "."

演算子 **.** の左側の式の型が構造体型、共用体型ではありません。

2204 (E) Illegal member reference for "->"

演算子 **->** の左側の式の型が構造体型または共用体型へのポインタではありません。

2205 (E) Undefined member name

構造体、共用体への参照で宣言していないメンバ名を使用しています。

2206 (E) Modifiable lvalue required for "演算子"

前置または後置演算子 `++`、`--` を代入可能な左辺値(配列型、`const` 型を除く左辺値)でない式に使用しています。

2207 (E) Scalar required for "!"

単項演算子 `!` をスカラ型でない式に使用しています。

2208 (E) Pointer required for "**"

単項演算子 `*` をポインタ型でない式か、または `void` 型へのポインタ型の式に使用しています。

2209 (E) Arithmetic type required for "演算子"

単項演算子 `+` または `-` を算術型でない式に使用しています。

2210 (E) Integer required for "~"

単項演算子 `~` を汎整数型でない式に使用しています。

2211 (E) Illegal sizeof

`sizeof` 演算子をビットフィールドの指定のあるメンバ、関数型、`void` 型またはサイズの指定していない配列に使用しています。

2212 (E) Illegal cast

キャスト演算子で指定している型が配列型、構造体型または共用体型です。あるいはキャスト演算子の被演算子が `void` 型、構造体型か共用体型で型変換できません。

2213 (E) Arithmetic type required for "演算子"

二項演算子 `*`、`/`、`*=` または `/=` を算術型でない式に適用しています。

2214 (E) Integer required for "演算子"

二項演算子 `<<`、`>>`、`&`、`|`、`^`、`%`、`<<=`、`>>=`、`&=`、`|=`、`^=` または `%=` を汎整数型でない式に適用しています。

2215 (E) Illegal type for "+"

二項演算子 `+` の被演算子の型の組み合わせが許されていません。

2216 (E) Illegal type for parameter

関数呼び出しの引数の型に `void` 型を指定しています。

2217 (E) Illegal type for "-"

二項演算子 - の被演算子の型の組み合わせが許されていません。

2218 (E) Scalar required

条件演算子 ?: の第 1 被演算子の型がスカラー型ではありません。

2219 (E) Type not compatible in "? :"

条件演算子 ?: の第 2 被演算子と第 3 被演算子の型が合っていないです。

2220 (E) Modifiable lvalue required for "演算子"

代入演算子 =、*=、/=、%=、+=、-=、<<=、>>=、&=、^= または |= の左辺の式に代入可能な左辺値(配列型、const 型を除く左辺値)以外の式を指定しています。

2221 (E) Illegal type for "演算子"

後置演算子 ++ または -- の被演算子にスカラー型以外の型、関数型または void 型へのポインタ型を指定しています。

2222 (E) Type not compatible for "="

代入演算子 = の両辺の式の型が合っていないです。

2223 (E) Incomplete tag used in expression

構造体または共用体で仮宣言されたタグ名を式中使用しています。

2224 (E) Illegal type for assign

代入演算子 += または -= の両辺の型が正しくありません。

2225 (E) Undeclared name "名前"

宣言していない名前を式の中で用いています。

2226 (E) Scalar required for "演算子"

二項演算子 && または || をスカラー型でない式に適用しています。

2227 (E) Illegal type for equality

等値演算子 == または != の被演算子の型の組み合わせが許されていません。

2228 (E) Illegal type for comparison

関係演算子 >、<、>= または <= の被演算子の型の組み合わせが許されていません。

2230 (E) Illegal function call

関数呼び出しにおいて、関数型あるいは関数型へのポインタ型でない式を用いています。

2231 (E) Address of bit field

単項演算子 `&` をビットフィールドに適用しています。

2232 (E) Illegal type for "演算子"

前置演算子 `++`, または `--` の被演算子にスカラ型以外の型、関数型または `void` 型へのポインタ型を指定しています。

2233 (E) Illegal array reference

配列型、関数型または `void` 型を除くポインタ型以外の式を配列として使用しています。

2234 (E) Illegal typedef name reference

`typedef` 宣言された名前を式の中で変数として使用しています。

2235 (E) Illegal cast

ポインタを浮動小数点型にキャストしています。

2236 (E) Illegal cast in constant

定数式でポインタ型を `char` 型または `short` 型にキャストしています。

2237 (E) Illegal constant expression

定数式の中でポインタ型の定数を整数型へキャストした結果に対して演算を行っています。

2238 (E) Lvalue or function type required for "&"

単項演算子 `&` を左辺値あるいは関数型以外の式に適用しています。

2300 (E) Case not in switch

`case` ラベルを `switch` 文以外に指定しています。

2301 (E) Default not in switch

`default` ラベルを `switch` 文以外に指定しています。

2302 (E) Multiple labels

一つの関数内にラベル名を重複して定義しています。

2303 (E) Illegal continue

continue 文を **while** 文、**for** 文または **do** 文以外に指定しています。

2304 (E) Illegal break

break 文を **while** 文、**for** 文、**do** 文または **switch** 文以外に指定しています。

2305 (E) Void function returns value

void 型を返す関数の中の **return** 文でリターン値を指定しています。

2306 (E) Case label not constant

case ラベルの式が汎整数型の定数式ではありません。

2307 (E) Multiple case labels

同一の値を持つ **case** ラベルを一つの **switch** 文の中に重複して指定しています。

2308 (E) Multiple default labels

default ラベルを一つの **switch** 文の中に重複して指定しています。

2309 (E) No label for goto

goto 文で指定した行き先のラベルがありません。

2310 (E) Scalar required

while 文、**for** 文または **do** 文の制御式(文の実行を判定する式)がスカラ型ではありません。

2311 (E) Integer required

switch 文の制御式(文の実行を判定する式)が汎整数型ではありません。

2312 (E) Missing (

if 文、**while** 文、**for** 文、**do** 文または **switch** 文の制御式(文の実行を判定する式)の左括弧「 (」がありません。

2313 (E) Missing ;

do 文の最後のセミicolon(;)がありません。

2314 (E) Scalar required

if 文の制御式(文の実行を判定する式)がスカラ型ではありません。

2316 (E) Illegal type for return value

return 文の式の型を関数の返す型に変換することができません。

2400 (E) Illegal character "文字"

不正な印字文字があります。

2401 (E) Incomplete character constant

文字定数の途中で改行があります。

2402 (E) Incomplete string

文字列の途中で改行があります。

2403 (E) EOF in comment

コメントの途中でファイルが終了しました。

2404 (E) Illegal character code "文字コード"

不正な文字コードがあります。

2405 (E) Null character constant

文字定数の中に文字を指定していません。すなわち `' '` という形式の文字定数を指定しています。

2406 (E) Out of float

浮動小数点定数の有効桁数が 17 桁を超えています。

2407 (E) Incomplete logical line

空でないソースファイルの最後の文字に、バックスラッシュ(`\`)またはバックスラッシュのあとに改行文字(`\(RET)`)を指定しています。

2408 (E) Comment nest too deep

コメントのネストが限界値を超えています。限界値は 255 レベルです。

2500 (E) Illegal token "語句"

語句の並びが文法に合っていないです。

2501 (E) Division by zero

定数式中で整数型データのゼロ除算が行われました。

2600 (E) 文字列

nolist オプションが指定されていないければ、**#error** の文字列で指定されたエラーメッセージをリストファイルに表示します。

2650 (E) Invalid pointer reference

指定されたアドレス値が境界調整数と一致しません。

2700 (E) Function "関数名" in **#pragma interrupt** already declared

割り込み関数宣言**#pragma interrupt** で指定した関数が、すでに通常の関数として宣言されています。

2701 (E) Multiple interrupt for one function

一つの関数に対して割り込み関数宣言**#pragma interrupt** を重複して宣言しています。

2702 (E) Multiple **#pragma interrupt** options

同種の割り込み仕様が重複して指定されています。

2703 (E) Illegal **#pragma interrupt** declaration

割り込み関数宣言**#pragma interrupt** の仕様の指定が異なります。

2704 (E) Illegal reference to interrupt function

割り込み関数を不正に参照しています。

2705 (E) Illegal parameter in interrupt function

割り込み関数で使用する引数の型が一致していません。

2706 (E) Missing parameter declaration in interrupt function

割り込み関数のオプション指定で使用する変数の宣言がありません。

2707 (E) Parameter out of range in interrupt function

割り込み関数のパラメタ **tn** の値が 256 を超えています。

2709 (E) Illegal section name declaration

#pragma section 指定に誤りがあります。

2710 (E) Section name too long

指定したセクション名の長さが 31 文字を超えています。

2711 (E) Section name table overflow

指定したセクションの数が 1 ファイルで 64 個を超えています。

2712 (E) GBR based displacement overflow

#pragma gbr_base で宣言した変数の領域がオーバーフローしました。

2713 (E) Illegal #pragma interrupt function type

#pragma interrupt 指定した関数の型が不正です。

2800 (E) Illegal parameter number in in-line function

組み込み関数で使用する引数の数が一致しません。

2801 (E) Illegal parameter type in in-line function

組み込み関数で引数の型が一致しません。

2802 (E) Parameter out of range in in-line function

組み込み関数で引数の大きさが指定可能範囲を超えています。

2803 (E) Invalid offset value in in-line function

組み込み関数で引数の指定が不適当です。

2804 (E) Illegal in-line function

指定された CPU オプションでは使用できない組み込み関数があります。

2805 (E) Function "関数名" in #pragma inline/inline_asm already declared

"関数名"で示す関数の本体が、**#pragma** 指定よりも前にあります。

2806 (E) Multiple #pragma for one function

一つの関数に対して複数の矛盾した**#pragma** 指定をしています。

2807 (E) Illegal #pragma inline/inline_asm declaration

#pragma inline または**#pragma inline_asm** の指定方法に誤りがあります。

2808 (E) Illegal option for #pragma inline_asm

#pragma inline_asm の指定があるにもかかわらず、**-code=machinecode** オプションを指定しています。

2809 (E) Illegal #pragma inline/inline_asm function type

#pragma inline または **#pragma inline_asm** を指定した識別子の種類が誤っています。

2810 (E) Global variable "変数名" in #pragma gbr_base/gbr_base1 already declared

"変数名"で示す変数の定義が**#pragma** 指定よりも前にあります。

2811 (E) Multiple #pragma for one global variable

変数に対して複数の矛盾する**#pragma** が指定されています。

2812 (E) Illegal #pragma gbr_base/gbr_base1 declaration

#pragma gbr_base、**#pragma gbr_base1** の指定方法が誤っています。

2813 (E) Illegal #pragma gbr_base/gbr_base1 global variable type

#pragma gbr_base、**#pragma gbr_base1** を指定した識別子の種類が誤っています。

2814 (E) Function "関数名" in #pragma noregsave/noregalloc/regsave already declared

"関数名"で示す関数の本体が、**#pragma** 指定よりも前にあります。

2815 (E) Illegal #pragma noregsave/noregalloc/regsave declaration

#pragma noregsave、**#pragma noregalloc**、**#pragma regsave** の指定方法が誤っています。

2816 (E) Illegal #pragma noregsave/noregalloc/regsave function type

#pragma noregsave、**#pragma noregalloc**、**#pragma regsave** を指定した識別子の種類が誤っています。

2817 (E) Symbol "識別子" in #pragma abs16 already declared

"識別子"で示す名前の宣言が、**#pragma** 指定よりも前にあります。

2818 (E) Multiple #pragma for one symbol

同一の識別子に対して、複数の矛盾した**#pragma** が指定されています。

2819 (E) Illegal #pragma abs16 declaration

#pragma abs16 の指定方法が誤っています。

2820 (E) Illegal #pragma abs16 symbol type

#pragma abs16 を指定した識別子の種類が誤っています。

2821 (E) Global variable "変数名" in #pragma global_register already declared
#pragma global_register を指定した変数名はすでに定義されています。

2822 (E) Illegal register "レジスタ" in #pragma global_register
#pragma global_register を指定したレジスタが不正です。

2823 (E) Illegal #pragma global_register declaration
#pragma global_register の指定方法が誤っています。

2824 (E) Illegal #pragma global_register type
#pragma global_register を指定できない変数が指定されています。

3000 (F) Statement nest too deep
if 文、while 文、for 文、do 文および switch 文のネストが限界値を超えています。

3001 (F) Block nest too deep
複文のネストが限界値を超えています。

3002 (F) #if nest too deep
条件コンパイル(#if、#ifdef、#ifndef、#elif、#else)のネストが限界値を超えています。

3006 (F) Too many parameters
関数の宣言または呼び出しにおいて引数の数が限界値を超えています。

3007 (F) Too many macro parameters
マクロの定義または呼び出しにおいて、引数の数が限界値を超えています。

3008 (F) Line too long
マクロ展開後の 1 行の長さが限界値を超えています。

3009 (F) String literal too long
文字列の長さが 512 文字を超えています。文字列の長さは、連続して指定した文字列を連結した後のバイト数です。ここでいう文字列の長さとは、ソースプログラム上の長さではなく文字列のデータに含まれるバイト数で、拡張表記も 1 文字に数えます。

3010 (F) Processor directive #include nest too deep
#include 文によるファイルの取り込みのネストが限界値を超えています。

3011 (F) Macro expansion nest too deep

`#define` 文によるマクロ名の再置換が限界値を超えています。

3012 (F) Too many function definitions

関数定義の数が限界値を超えています。

3013 (F) Too many switches

`switch` 文の数が限界値を超えています。

3014 (F) For nest too deep

`for` 文のネストが限界値を超えています。

3015 (F) Symbol table overflow

コンパイラが生成するシンボルの数が限界値を超えています。

3016 (F) Internal label overflow

コンパイラが生成する内部ラベルの数が限界値を超えています。

3017 (F) Too many case labels

一つの `switch` 文の中の `case` ラベルの数が限界値を超えています。

3018 (F) Too many goto labels

一つの関数の中で定義している `goto` ラベルの数が限界値を超えています。

3019 (F) Cannot open source file "ファイル名"

一つの関数の中で定義している `goto` ラベルの数が限界値を超えています。
ソースファイルをオープンすることができません。

3020 (F) Source file input error "ファイル名"

ソースファイルまたはインクルードファイルを読み込むことができません。

3021 (F) Memory overflow

コンパイラが内部で使用するメモリ領域を割り当てることができません。

3022 (F) Switch nest too deep

`switch` 文のネストが限界値を超えています。

3023 (F) Type nest too deep

基本型を修飾する型(ポインタ型、配列型、関数型)の数が 16 個を超えています。

3024 (F) Array dimension too deep

配列の次元数が 6 次元を超えています。

3025 (F) Source file not found

コマンドラインの中にソースファイル名の指定がありません。

3026 (F) Expression too complex

式が複雑すぎます。

3027 (F) Source file too complex

プログラムの文のネストが深いあるいは、式が複雑すぎます。

3028 (F) Source line number overflow

ソース行番号が限界値を超えています。

3030 (F) Too many compound statements

複文の数が限界値を超えました。

3031 (F) Data size overflow

配列または構造体の大きさが、2147483647 バイトを超えています。

3100 (F) Misaligned pointer access

境界整合が正しくないポインタを用いて参照または設定をしようとしています。

3201 (F) Object size overflow

オブジェクトサイズが 4G バイトを超えています。

3202 (F) Too many source lines for debug

ソース行数が多すぎて、デバッグのための情報が出力できません。

3203 (F) Assembly source line too long

出力するアセンブリソースの 1 行が長すぎます。

3204 (F) Illegal stack access

関数内で使用するスタックのサイズ（局所変数領域、レジスタ退避領域その他関数呼び出しのためのパラメタプッシュ領域等含む）または、その関数呼び出しのためのパラメタ領域が 2G バイトを超えています。

3300 (F) Cannot open internal file

以下、四つの場合のいずれかでエラーが起こっている可能性があります。

- (1) コンパイラが内部で生成する中間ファイルをオープンすることができません。
- (2) 中間ファイルと同じ名前のファイルが既に存在しています。
- (3) リストファイル仕様のパス名が 128 文字を超えています。
- (4) コンパイラが内部で使用するファイルをオープンすることができません。

3301 (F) Cannot close internal file

コンパイラが内部で生成する中間ファイルをクローズすることができません。コンパイラのインストール手順に誤りがないことを確認してください。

3302 (F) Cannot input internal file

コンパイラが内部で生成する中間ファイルを読み込むことができません。コンパイラのインストール手順に誤りがないことを確認してください。

3303 (F) Cannot output internal file

コンパイラが内部で生成する中間ファイルに書き込むことができません。

3304 (F) Cannot delete internal file

コンパイラが内部で生成する中間ファイルを削除することができません。

3305 (F) Invalid command parameter "オプション名"

コンパイラオプションの指定方法が誤っています。

3306 (F) Interrupt in compilation

コンパイル処理中に標準入力端末から(CNTL) C コマンドによる割り込みを検出しました。

3307 (F) Compiler version mismatch

コンパイラを構成するファイル間のバージョンが一致していません。

3308 (F) Cannot create file "ファイル名"

コンパイラが生成するファイルを作成できません。

3320 (F) Command parameter buffer overflow

コマンドラインの指定が 256 文字を超えています。

3321 (F) Illegal environment variable

以下のどれかの場合でエラーが起っています。

(1) 環境変数 **SHC_LIB** が設定されていません。

(2) 環境変数 **SHC_LIB** の設定でファイル名の規約に反した指定をしているか、パス名の長さが 118 文字を超えています。

(3) 環境変数 **SHCPU** に、"**SH1**", "**SH2**", "**SH2E**", "**SHDSP**", "**SH3**", "**SH3E**", "**SH4**" 以外の設定がされています。

4000 - 4999 (-) Internal error

コンパイラの内部処理で何らかの障害が生じました。本コンパイラをお求めになった営業所あるいは代理店にエラーの発生状況をご連絡ください。

5001 (W) Linkage specification ignored for 'static' functions and objects

リンケージ指定の { } の中で、**static** と宣言された関数やオブジェクトの **extern** リンケージ指令は無視されます。

6001 (E) Overloaded function "関数名" cannot specify #pragma function

多重定義関数を **#pragama** 指定できません。

6002 (E) Preprocessing numerical token is not floating constant value or integer constant value

前処理数字字句が浮動小数点定数、整数定数ではありません。

6101 (E) Illegal storage class 'auto'

auto 記憶クラスは、ブロック内か、仮引数内のオブジェクト宣言に指定しなければなりません。

6102 (E) Illegal storage class 'register'

register 記憶クラスは、ブロック内か、仮引数内のオブジェクト宣言に指定しなければなりません。

6103 (E) 'static' keyword must be applied to objects, functions and anonymous unions

static 記憶クラスは、オブジェクト名、関数名または名前なし共用体以外に指定できません。

6104 (E) Function declaration "関数名" cannot have 'static' storage in a block
ブロック内で **static** 関数を宣言することはできません。

6105 (E) Static linkage specifications for "名前" must agree
名前に対する全てのリンケージ指定が一致していません。

6106 (E) Illegal storage class 'extern'
extern 記憶クラスは、オブジェクト名、関数名または名前なし共用体以外に指定できません。

6108 (E) Inline member function "関数名" must be declared before it is called
メンバ関数を呼んだ後でその関数を **inline** と宣言することはできません。

6109 (E) Illegal function specifier 'virtual'
virtual 指定子は、クラス定義の非静的クラス・メンバ関数の宣言以外に指定できません。

6110 (E) Illegal 'typedef' declaration in function definition "名前"
typedef 指定子は、関数定義に指定できません。

6111 (E) 'typedef' "名前" cannot re-define other types in the same scope
同一スコープで宣言された型の名前を、別の型を参照するように再定義することはできません。

6112 (E) Cannot specify 'typedef' "名前" after 'enum'
typedef で定義された **enum** 型名を **enum** 接頭辞のあとで使用することはできません。

6114 (E) Integral type data is not assigned to an enumeration
整数型あるいは整数型への昇格の結果を列挙型オブジェクトに代入することはできません。

6116 (E) Undefined linkage-specification "文字列"
リンケージ指定に未定義の文字列が指定されました。

6117 (E) Multiple linkage-specification "名前"
関数やオブジェクトが複数のリンケージ指定を持つ時のリンケージ指定文字列が一致していません。

- 6118 (E) 'static' function "関数名" with linkage-specification
リンケージ指定された関数に `static` 指定の関数宣言があります。
- 6119 (E) Multiple 'C' linkage overloaded function "関数名"
多重定義関数の集合の中で C リンケージを持つものが複数あります。
- 6120 (E) Illegal 'void &' type
`void` 型へのリファレンスは指定できません。
- 6123 (E) Cannot specify a reference to `&`, to bit-fields, to array or to pointer type
リファレンス型、ビットフィールド、配列またはポインタ型に対してのリファレンス型は指定できません。
- 6124 (E) Initial value required for declaration & "名前"
リファレンス型宣言に初期値の指定がありません。
- 6128 (E) New type defined in a return type of a function or in a parameter
戻り値や仮引数型の中で、型を定義しています。
- 6129 (E) Non-static members or 'auto' variables cannot be used as default parameter
デフォルト引数に非静的メンバが使われています。
- 6130 (E) Default parameter re-defined in function "名前"
デフォルト引数は、後の宣言で再定義することができません。
- 6131 (E) Non-default parameters found following default parameters
デフォルト引数の後にデフォルト引数以外の引数が存在します。
- 6133 (E) Overloaded operators cannot have default parameters
多重定義演算子はデフォルト引数を持ってません。
- 6139 (E) Overloaded functions "関数名" have the same type of parameters except & type
関数の引数の型がリファレンス型の違いだけなので多重定義できません。
- 6140 (E) Functions "関数名" have the same type of parameters except `const/volatile` type
関数の引数の型が `const/volatile` 型の違いだけなので多重定義できません。

6141 (E) Functions "関数名" have the same type of parameters except return type
関数の戻り値が型の違いだけなので多重定義できません。

6142 (E) Cannot overload function "関数名"
メンバ関数が **static** であるかどうかの違いだけなので多重定義できません。

6143 (E) Operator overloaded function "関数名" does not have correct parameters
演算子関数は、以下のいずれかでなければなりません。

- (1) メンバ関数である。
- (2) クラスまたは列挙の引数を 1 つ以上持つ。
- (3) クラスまたは列挙へのリファレンスの引数を 1 つ以上持つ。

6144 (E) Operator overloaded function "=" is not a nonstatic member function
operator=() が非静的メンバ関数ではありません。

6145 (E) Operator overloaded function "(" is not a nonstatic member function
operator()() が非静的メンバ関数ではありません。

6146 (E) Operator overloaded function "[" is not a nonstatic member function
operator[]() が非静的メンバ関数ではありません。

6147 (E) Operator overloaded function "->" returns illegal type value
operator->() がクラスへのポインタ、または、**operator->()** を定義しているクラスのオブジェクト、あるいはリファレンスを返していません。

6148 (E) Operator overloaded function "->" is not a nonstatic member function
operator->() が非静的メンバ関数ではありません。

6149 (E) The second parameter of the postfix operator function should have type int
後置の **operator++()** または **operator--()** の第 2 引数が **int** 型ではありません。

6150 (E) 'const' object should have an initial value
const 型のオブジェクトは外部リンケージを持たないので初期値が必要です。

6151 (E) 'friend' should be specified in a class declaration
friend 指定子がクラス宣言外で使われています。

6152 (E) 'friend' keyword specified twice

friend 指定子が 2 度指定されています。

6153 (E) 'inline' keyword specified twice

inline 指定子が 2 度指定されています。

6154 (E) 'virtual' keyword specified twice

virtual 指定子が 2 度指定されています。

6155 (E) 'inline' must be specified for functions

inline 指定子が関数以外に指定されています。

6156 (E) Parameters cannot have 'inline' specifier

引数に **inline** 指定子が指定されています。

6157 (E) Cannot specify 'inline' and 'extern' together

inline と **extern** 指定子が同時に指定されています。

6158 (E) Object "名前" initialized with { } format

{ } 形式で初期化できるオブジェクトは、次のいずれかです。

(1) 配列

(2) 非公開/限定公開のメンバ、基底クラス、コンストラクタ、仮想関数のいずれも持たないクラス

6159 (E) 'typedef' cannot specify 'friend'

typedef と **friend** が合わせて指定されています。

6162 (E) Cannot declare 'typedef' name qualified by a class

typedef 宣言の名前をクラスで限定することはできません。

6163 (E) Missing the number of operator function parameters

多重定義演算子の引数の数が間違っています。

6165 (E) Operator member function cannot specify 'static'

多重定義演算子は静的メンバ関数になれません。

6166 (E) Operator function or conversion function has function type

多重定義演算子、変換関数が関数型ではありません。

6167 (E) Conversion function should be a nonstatic member function

変換関数が非静的メンバ関数ではありません。

6169 (E) A destructor should be a function type

デストラクタが関数型ではありません。

6170 (E) Non-member functions cannot specify 'const' or 'volatile'

メンバ関数以外では、関数自身に **const** または **volatile** 型修飾できません。

6171 (E) Declarations qualified by a class cannot specify storage class specifier

クラスで限定された宣言に記憶クラスは指定できません。

6172 (E) Cannot declare parameter declarations qualified by a class

引数宣言にクラスで限定された名前を指定しています。

6203 (E) Ambiguous name "名前"

関数名、オブジェクト名、または、型名が曖昧です。

6209 (E) Array "名前" does not have dimension size

配列が非静的メンバの型として使われていますが、全ての次元数が指定されていません。

6210 (E) Member declarator be omitted

クラスのメンバ宣言子を省略することはできません。

6211 (E) Non-virtual function "関数名" cannot specify "=0"

純粋指定子 **=0** 指定できません。純粋指定子は仮想関数の宣言でのみ使用できます。

6212 (E) Member "名前" cannot have the same name of that class

静的データメンバ、列挙子、名前なしの共用体メンバあるいは入れ子型がクラスと同じ名前を持っています。

6213 (E) Static member function cannot access "名前"

静的メンバ関数ができるのは、静的メンバ、列挙子、入れ子型だけです。

6219 (E) Static data member "名前" cannot exist in a local class

局所クラスでは、静的データメンバを宣言できません。

6221 (E) Union "名前" cannot have virtual functions

共用体は、仮想関数を持つことができません。

6222 (E) Union "名前" cannot have base classes

共用体は、基底クラスを持つことができません。

6223 (E) Union "名前" cannot be a base class

共用体を基底クラスとして使用することはできません。

6224 (E) Union member cannot have a constructor, destructor or class with a user defined operator =()

コンストラクタ、デストラクタまたはユーザ定義の代入演算子を持つクラスオブジェクトが共用体のメンバに存在します。

6225 (E) Union "名前" cannot have static data members

static データメンバが共用体のメンバに存在します。

6227 (E) Anonymous union must specify 'static'

static 指定のない大域的な名前無し共用体宣言が存在します。

6228 (E) Anonymous union cannot have private and protected members

名前無し共用体に **private** や **protected** メンバが存在します。

6229 (E) Anonymous union cannot have member functions

名前無し共用体に、関数メンバが存在します。

6233 (E) Nested class declaration cannot access "名前"

入れ子クラスの宣言では、それを囲むクラスからのオブジェクト、静的メンバ、列挙子以外を使用できません。

6234 (E) "名前" cannot be specified in a local class declaration

局所クラス宣言内では、型名、静的変数、**extern** 変数、**extern** 関数、列挙子以外使用できません。

6236 (E) Member function "名前" should be defined in the local class declaration

局所クラス宣言内のメンバ関数をクラス内で定義していません。

- 6240 (E) Member "名前" defined more than once
クラスメンバが2度宣言されています。
- 6241 (E) Undeclared member "名前"
クラス宣言中に未定義のメンバ関数をクラス外でメンバ関数として定義しています。
- 6242 (E) "名前" declared multiple
クラス再評価時に名前が確定しません。
- 6243 (E) Undeclared base class "名前"
基底クラスに指定したクラス名が定義されていません。
- 6244 (E) Base class "名前" defined more than once in the derived class declaration
派生クラスの直接基底クラスが2度以上指定されています。
- 6246 (E) Virtual base class "名前" casted to the derived class
仮想基底クラスを派生クラスへキャストすることはできません。
- 6248 (E) Returning type mismatch in virtual function "関数名"
基底クラスの仮想関数と派生クラスの仮想関数の戻り型が異なります。
- 6250 (E) Illegal 'static' virtual function "関数名"
仮想関数を **static** 指定できません。
- 6251 (E) Cannot create abstract class object "名前"
抽象クラスのオブジェクトを作成することはできません。
- 6254 (E) Implicit pure virtual function "関数名" call
純粋仮想関数は明示的に限定して呼び出さなければなりません。
- 6256 (E) "名前" is not a member of the class
指定されたクラスのメンバが定義されていません。
- 6263 (E) "名前" should be defined as a class or a struct
class または、**struct** 指定子で宣言した名前が、異なる指定子で定義されています。
- 6267 (E) Illegal declaration "名前" in type-specifier
クラス、列挙、**typedef** 名は、型指定並びの中で宣言できません。

- 6268 (E) "関数名" cannot have parameters and return value
変換関数に引数型や戻り型を指定できません。
- 6269 (E) Ambiguous user defined conversion
利用者定義変換が曖昧です。
- 6270 (E) Destructor "関数名" cannot have parameters and return value
デストラクタに引数や戻り型を指定できません。
- 6271 (E) Constructor and destructor cannot have 'const', 'volatile' or 'static' keywords
コンストラクタ、デストラクタに **const**、**volatile**、**static** を指定できません。
- 6272 (E) Constructor and destructor "関数名" do not have addresses
コンストラクタ、デストラクタのアドレスを求めることはできません。
- 6274 (E) The first parameter type of operator new() should be 'size_t'
クラスの **operator new()**関数の第 1 引数は、整数型の **size_t** でなければなりません。
- 6275 (E) The return type of operator new() should be 'void *'
クラスの **operator new()**関数の戻り型は、**void***型でなければなりません。
- 6276 (E) The first parameter type of operator delete() should be 'void *'
クラスまたは大域の **operator delete()**関数の第 1 引数は、**void***でなければなりません。
- 6277 (E) The return type of operator delete() should be 'void'
クラスの **::operator delete()**の戻り型は **void** 型でなければなりません。
- 6278 (E) Operator delete() function cannot be overloaded
operator delete()は多重定義できません。
- 6280 (E) The second parameter type of operator delete() should be 'size_t'
クラスの **operator delete()**関数の第 2 引数は、整数型の **size_t** でなければなりません。
- 6281 (E) Illegal virtual operator new() or delete() function
operator new()、**operator delete()**は、仮想関数指定できません。

6282 (E) Default constructor required

初期設定子ならびに対応する初期値がそろっていないか、初期値がない場合のデフォルトコンストラクタが定義されていません。

6283 (E) Class "名前" requires a constructor

仮想基底クラスの初期設定に誤りがあります。最派生クラスのコンストラクタが仮想基底クラスのメンバ初期設定子を指定していなければ、その仮想基底クラスはデフォルトコンストラクタを持つか、コンストラクタを持たないかのどちらかでなければなりません。

6284 (E) Undefined class member "名前"

クラスメンバの初期設定に誤りがあります。クラスメンバのオブジェクトは、コンストラクタを持たないか、デフォルトコンストラクタを持つか、あるいは、クラスメンバを宣言しているクラスのコンストラクタの中でクラスメンバを初期化しなければなりません。

6285 (E) Multiple implicit conversion

式の値に暗黙のうちに適用される利用者定義変換が複数存在します。

6286 (E) 'public' copy constructor "関数名" required

コピーコンストラクタを **private** メンバとして宣言しているのでアクセスできません。

6287 (E) Illegal nonstatic 'const' array initialization

非静的 **const** 配列のメンバをコンストラクタで初期設定することはできません。

6288 (E) Constructors cannot initialize indirect base classes or derived members

間接基底クラスや派生メンバをコンストラクタで初期設定することはできません。

6289 (E) Cannot generate default assignment operator

デフォルトの代入演算子は生成されません。クラスが **const** メンバ、リファレンスメンバ、非公開の **operator=()** を持つクラスのメンバ、あるいは、非公開の **operator=()** を持つ基底クラスのメンバを持っています。

6290 (E) Cannot generate default copy constructor "名前"

デフォルトのコピーコンストラクタは生成されません。クラスが名前なしクラスか、すでにクラスのメンバまたは基底クラスのメンバが非公開コピーコンストラクタを持っています。

6291 (E) Cannot assign "名前"

基底クラスのオブジェクトを派生クラスのオブジェクトに代入することはできません。

6292 (E) Cannot access private member "名前"

非公開メンバを参照することはできません。

6293 (E) Cannot access protected member "名前"

限定公開メンバは参照することはできません。

6294 (E) Illegal access declaration "名前"

アクセス宣言において、基底クラスのメンバで宣言したアクセス指定を変更することはできません。

6296 (E) Cannot change the access control of overloaded function "関数名"

アクセス指定が同じでない多重定義関数をアクセス宣言で調整することはできません。

6297 (E) Cannot change the access control of redefined member "名前"

派生クラスにおいてメンバが再定義された場合、基底クラスのメンバアクセスを派生クラスで調整することはできません。

6298 (E) 'friend' declaration syntax error

フレンド宣言の中で、クラスを定義しようとしています。

6303 (E) Cannot access base class "クラス名"

その基底クラスにアクセスできません。

6304 (E) Cannot define 'friend' data members

データメンバを **friend** 宣言しています。

6305 (E) Illegal pure virtual function specifier

純粋仮想関数指定に=0 以外を指定しています。

6306 (E) Cannot access class "名前" member

アクセスできないクラスメンバを指定しています。

6307 (E) Cannot access base class member "名前"

基底クラスメンバをアクセスできません。

- 6308 (E) Constructors cannot have return type
コンストラクタにリターン型を指定しています。
- 6310 (E) Cannot define class member "名前"
別クラスの中で別クラスのメンバを宣言しています。
- 6311 (E) Illegal constructor/destructor declaration
コンストラクタ / デストラクタの宣言に誤りがあります。
- 6313 (E) Cannot declare nonstatic data member "名前"
非静的データメンバをクラス外部で再宣言することはできません。
- 6314 (E) Illegal data member initializer
データメンバの初期値指定の方法が正しくありません。
- 6401 (E) 'this' should be referred to in a nonstatic member function
キーワード **this** は、非静的メンバ関数以外で参照することはできません。
- 6402 (E) "名前" does not exist in this file scope
:: 演算子に続く識別子がファイルスコープに存在しません。
- 6404 (E) "名前" is not a member
:: 演算子に続く名前がそのクラスのメンバではありません。
- 6407 (E) The type of the second operand of '? :' is different from the third operand
?: 演算子の第2オペランドと第3オペランドの型が異なるクラスの場合は、共通の基底クラスを持たなければなりません。
- 6408 (E) Base class objects cannot be assigned to derived class objects
基底クラスのオブジェクトは、派生クラスのオブジェクトに代入できません。
- 6409 (E) '? :' operands should have integral types
クラス・オブジェクト、リファレンスは使用できません。
- 6417 (E) Illegal type conversion
関数呼び出し形式の明示的型変換において、対応するコンストラクが宣言されていません。

6421 (E) Invalid 'new' operand

new 演算子のオペランドが正しくありません。

6425 (E) An array cannot be initialized in a 'new' operator

new 演算子のオペランドで指定された配列を、初期設定子を使って初期化することはできません。

6428 (E) Invalid 'delete' operand

delete 演算子のオペランドが正しくありません。

6430 (E) Cannot convert an object or data to a class object

コンストラクタや変換演算子が正しく宣言されていないため、オブジェクトや値をクラスオブジェクトに変換できません。

6431 (E) Cannot convert a virtual base class to a derived class

仮想基底クラスを派生クラスにキャストすることはできません。

6432 (E) Illegal explicit type conversion

メンバへのポインタを別のメンバへのポインタ型へ変換することはできません。

6433 (E) '->*' operands type mismatch

->* 演算子のオペランドの型が合致しません。

6434 (E) The second operand of '->*' and '.*' operator should be a pointer to member of a class

->* 演算子、**.*** 演算子の第 2 オペランドはクラスのメンバへのポインタ型でなければなりません。

6435 (E) '.*' operands type mismatch

.* 演算子のオペランドの型が合致しません。

6437 (E) Unary '&' operand require a qualified name

単項演算子 **&** のオペランドで限定した名前がクラスメンバ名ではありません。

6441 (E) 'typedef' "名前" used as a constructor or a destructor

typedef 宣言されたクラスの **typedef** 名をコンストラクタやデストラクタの名前として使用できません。

6442 (E) Cannot refer to undefined class

未定義のクラスは式中などで使用できません。

6446 (E) Ambiguous overloaded function call

多重定義関数呼び出しの関数が曖昧です。

6449 (E) Ambiguous function name referred to

多重定義関数のアドレスを示す関数名の式が曖昧です。

6450 (E) Illegal function pointer conversion

関数へのポインタ型を他の型に標準変換できません。

6451 (E) Undeclared function call

該当する関数が宣言されていません。

6452 (E) Pointer-to-Member cannot be used as an operand of a function call

メンバ関数へのポインタ型が関数呼び出し演算子以外のオペランドに使用できません。

6503 (E) Statements is required in selection statements or iteration statements

'if'、'switch'文の中には 1 つ以上の文を含まなくてはなりません。または、反復文の文は宣言であってなりません。

6504 (E) Declaration with initialization is not executed

明示的あるいは暗黙の初期設定子を持った宣言を飛び越えている。

6508 (E) Illegal conditional expression

'while'、'do'、'for' 文の条件式は、算術型、ポインタ型または、算術型かポインタ型への型変換が存在するクラス型でなければなりません。

6513 (E) Illegal jump

不当なジャンプです。

6514 (E) "xxxx" is not supported

本バージョンでサポートしていない機能を使用しています。

7001 (E) Too many identifiers

識別子の数が制限値を超えています。

4.2 標準ライブラリのエラーメッセージ

ライブラリ関数の中には、ライブラリ関数を実行中にエラーが発生した場合、標準ライブラリのヘッダファイル<errno.h>で定義しているマクロ `errno` にエラー番号を設定するものがあります。エラー番号には、対応するエラーメッセージが定義されており、エラーメッセージを出力することができます。エラーメッセージを出力するプログラム例を以下に示します。

例

```
#include      <stdio.h>
#include      <string.h>
#include      <stdlib.h>
#include      <errno.h>

main()
{
    FILE *fp;

    fp=fopen("file", "w");
    fp=NULL;

    fclose(fp);                                /* error occurred */

    printf("%s\n", strerror(errno)); /* print error message */
}
```

説明

- (1) `fclose` 関数に値 `NULL` のファイルポインタを実引数として渡しているため、エラーとなります。このとき `errno` に対応するエラー番号が設定されます。
- (2) `strerror` 関数は、エラー番号を実引数として渡すと、対応するエラーメッセージの文字列のポインタを返します。`printf` 関数の文字列出力指定によりエラーメッセージを出力します。

標準ライブラリエラーメッセージ一覧

エラー番号	エラーメッセージ / 説明	エラー番号を設定する関数
1 1 0 0 (ERANGE)	Data out of range オーバーフローが発生しました。	atan, cos, sin, tan, cosh, sinh, tanh, exp, fabs, frexp, ldexp, modf, ceil, floor, strtol, atoi, fscanf, scanf, sscanf, atol
1 1 0 1 (EDOM)	Data out of domain 数学関数の引数に対する結果の値が定義 されません。	acos, asin, atan2, log, log10, sqrt, fmod, pow
1 1 0 2 (EDIV)	Division by zero ゼロによる除算を行っています。	divbs, divws, divls, divbu, divwu, divlu
1 1 0 4 (ESTRN)	Too long string 文字列の長さが 512 文字を超えています。	strtol, strtod, atof, atoi, atol
1 1 0 6 (PTRERR)	Invalid file pointer ファイルポインタの値に NULL ポインタ 定数を指定しています。	fclose, fflush, freopen, setbuf, setvbuf, fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, vsprintf, fgetc, fgets, fputc, fputs, ungetc, fread, fwrite, fseek, ftell, rewind, perror
1 2 0 0 (ECBASE)	Invalid radix 基数の指定が誤っています。	strtol, atol, atoi
1 2 0 2 (ETLN)	Number too long 数値を表現する文字列の長さが 17 桁を 超えています。	strtod, fscanf, scanf, sscanf, atof
1 2 0 4 (EEXP)	Exponent too large 指数部の桁数が 3 桁を超えています。	strtod, fscanf, scanf, sscanf, atof

1 2 0 6 (EEXPN)	Normalized exponent too large 文字列を一度 IEEE 規格の 10 進形式に 正規化したとき指数部の桁数が 3 桁を超 えています。	strtod, fscanf, sscanf, atof
1 2 1 0 (EFLOAT0)	Overflow out of float float 型の 10 進数値が, float 型の範 囲を超えています(オーバフロー)。	strtod, fscanf, scanf, sscanf, atof
1 2 2 0 (EFLOATU)	Underflow out of float float 型の 10 進数値が, float 型の範 囲を超えています(アンダフロー)。	strtod, fscanf, scanf, sscanf, atof
1 2 5 0 (EDBL0)	Overflow out of double double 型の 10 進数値が, double 型の 範囲を超えています(オーバフロー)。	strtod, fscanf, scanf, sscanf, atof
1 2 6 0 (EDBLU)	Underflow out of double double 型の 10 進数値が, double 型の 範囲を超えています(アンダフロー)。	strtod, fscanf, scanf, sscanf, atof
1 2 7 0 (ELDBL0)	Overflow out of long double long double 型の 10 進数値が, long double 型の範囲を超えています (オーバフロー)。	fscanf, scanf, sscanf
1 2 8 0 (ELDBLU)	Underflow out of long double long double 型の 10 進数値が, long double 型の範囲を超えています (アンダフロー)。	fscanf, scanf, sscanf
1 3 0 0 (NOTOPN)	File not open ファイルがオープンされていません。	fclose, fflush, setbuf, setvbuf, fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, vsprintf, fgetc, fgets, fputc, fputs, gets, puts, ungetc, fread, fwrite, fseek, ftell, rewind, perror, freopen

4. エラーメッセージ

1 3 0 2 (EBADF)	Bad file number 入力専用ファイルに対して出力関数,あるいは出力専用ファイルに対して入力関数を発行しています。	fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, vsprintf, fgetc, fgets, fputc, fputs, gets, puts, ungetc, perror, fread, fwrite
1 3 0 4 (ECSPEC)	Error in format 書式付き入出力関数で指定している書式が誤っています。	fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, vsprintf, perror

5. モジュール間最適化 ツール

5.1 モジュール間最適化ツール概要

モジュール間最適化ツールは、コンパイラが出力した複数のオブジェクトプログラムを入力としオブジェクトプログラムをまたがって最適化を実行した上で、リンケージエディタを起動し結合および編集するソフトウェアシステムです。従来コンパイラでは、最適化できなかった、複数オブジェクトの最適化を行います。

また、本最適化ツールを使用することによりロードモジュールのフォーマットとして

- ・ELF/DWARF フォーマット (オブジェクト部:ELF、デバッグ情報部:DWARF)
- ・SYSROF フォーマット (オブジェクト部:SYSROF、デバッグ情報部:SYSROF)
- ・SYSROF PLUS フォーマット(オブジェクト部:SYSROF、デバッグ情報部:DWARF)

の3通りの出力が可能です。

【注】ロードモジュールをSYSROFフォーマット以外のフォーマットにするには、本ツールを起動させなければなりません。

本最適化ツールをご使用になる際には、以下のソフトウェアが必要です。

SuperH RISC engine C/C++コンパイラ (Ver.5.0)

H シリーズ リンケージエディタ (Ver.6.0)

【注】

- (1)本最適化ツールは、コンパイル時に `goptimize` オプションを指定したファイル、最適化付加情報ファイルを生成したオブジェクトプログラムが最適化の対象になります。
`goptimize` オプション指定なしで生成したオブジェクトプログラム、SH シリーズ C コンパイラ Ver.4.1 以前、SH シリーズ C++コンパイラ Ver.1.1 以前のオブジェクトプログラムは、本最適化ツールの入力ファイルとして混在することが可能ですが、最適化の対象となるのは、`goptimize` オプションを指定して生成されたオブジェクトプログラムのみです。
- (2)コンパイラは、オブジェクトプログラム出力ディレクトリに「`shiop`」という名前のディレクトリを自動生成し、そのディレクトリの下に最適化付加情報ファイル(拡張子 `iop`)を格納します。
- (3)本最適化ツールからリンケージエディタを起動するので、H シリーズリンケージエディタが必要になります。

5.2 最適化ツールの起動方法

最適化ツールを起動するコマンドラインの形式は次のとおりです。

```
optlnksh[ <オプション>...]
```

最適化ツールを実行するためには、関連ソフトウェアを使用して、次のファイルを作成する必要があります。（括弧内は関連ソフトウェア名称）

オブジェクトプログラム（SuperH RISC engine C/C++コンパイラ）

リンケージエディタ用サブコマンドファイル

以下、最適化ツールの基本的な操作方法をサンプルプログラムを用いて説明します。

test1.c C プログラム

test1.sub 最適化ツール用サブコマンドファイル

(1) プログラムのコンパイル

test1.c をコンパイルします。このとき必ず `goptimize` オプションを指定します。また、ここで `debug` オプションを指定することにより、ソースレベルデバッグを行うためのデバッグ情報を出力することができます。

```
shc -goptimize -debug test1.c (RET)
```

(2) デフォルトライブラリの設定

リンク時に使用する標準ライブラリをデフォルトライブラリとして設定します。デフォルトライブラリについての詳細は「Hシリーズ リンケージエディタ、ライブラリアン、オブジェクトコンバータユーザズマニュアル」を参照してください。

PC 版(DOS プロンプト使用時) : `set HLNK_LIBRARY1= <ライブラリパス>%shc.lib (RET)`

UNIX 版 : `setenv HLNK_LIBRARY1 <ライブラリディレクトリ>/shc.lib (RET)`

(3) 最適化ツールの実行

test1.obj の最適化を実施後、ロードモジュールを作成します。

ここでは、リンカージェディタ用サブコマンドファイルの指定は省略できません。必ず指定してください。

例 1 オブジェクトプログラムの最適化、結合

```
optlnksh -optimize -subcommand=test1.sub (RET)
```

<test1.sub>

input	test1	; 入力ファイル名を指定します
entry	_main	; 実行開始の関数名を指定します
debug		; デバッグ情報出力を指定します
start	P,C(200),D,B(08000)	; 各セクションの開始アドレスを指定します
exit		; 処理を終了します

説明：

リンカージェディタ用サブコマンドファイルの作成方法は、「Hシリーズ リンカージェディタ、ライブラリアン、オブジェクトコンバータユーザズマニュアル」を参照してください。

例 2 最適化オプションの指定

optimize オプションのサブオプションによって、最適化の内容を指定することができます。

```
optlnksh -optimize=speed -subcommand=test1.sub (RET)
```

例 3 サブコマンドによる最適化オプションの指定

最適化ツールのオプションはサブコマンドとして、リンカージェディタ用サブコマンドファイル内で指定することもできます。

```
optlnksh -subcommand=test2.sub (RET)
```

<test2.sub>

optimize	speed	; 最適化ツール用オプションを指定します
input	test1	; 入力ファイル名を指定します
entry	_main	; 実行開始の関数名を指定します
debug		; デバッグ情報出力を指定します
start	P,C(200),D,B(08000)	; 各セクションの開始アドレスを指定します
exit		; 処理を終了します

(4) コマンド入力形式、オプションの表示

標準出力画面上にコマンドの入力形式、オプションの一覧を表示します。

```
optlnksh (RET)
```

5.3. オプション / サブコマンド

オプション / サブコマンドの形式は次のとおりです。

オプション : - <オプション>[=<パラメタ>[,<パラメタ>...]]

サブコマンド^{*1} : <サブコマンド> [<パラメタ>[,<パラメタ>...]]

オプション / サブコマンドと短縮形および省略時解釈の一覧を表 5-1 に示します。

下線部 () は短縮形指定時の文字を示します。

【注】 " * " のないものは、Ver.5.0 で有効なオプションです。 " * " のあるものは、Ver.5.1 以降でサポートいたします。

表 5-1 オプション一覧

No.	項目	オプション/サブコマンド名	パラメタ	省略時解釈	指定内容
1	最適化内容	<u>optimize</u> *	<u>string_unify</u> <u>symbol_delete</u> <u>register</u> <u>same_code</u> <u>branch</u> <u>speed</u> <u>safe</u>	optimize= string_unify, symbol_delete, register, same_code, branch (または optimize)	最適化内容の指定 定数 / 文字列の統合 未参照シンボルの削除 レジスタの再割付 共通コードの統合 分岐命令の最適化 スピード重視の最適化(op=st,sy, r,b) 安全な最適化(op=st,r,b)
		<u>nooptimize</u> *	-	なし	最適化の抑止指定
		<u>same</u> size*	<size> size : 16 進数	samesize=1E	共通コード統合(optimize=same_code)の対象サイズ指定
2	最適化抑止	<u>symbol_forbid</u> *	<シンボル名> [,<シンボル名>...] シンボル名 : <変数名> <関数名>	なし	未参照シンボル削除(optimize=symbol_delete)の最適化を抑止する変数 / 関数名を指定
		<u>same</u> code_forbid*	<関数名> [,<関数名>...]	なし	共通コード統合(optimize=same_code)の最適化を抑止する関数名を指定
3	オブジェクトフォーマット指定	<u>elf</u> <u>sysrof</u> <u>sysrof</u> plus	- - -		ELF/DWARF フォーマット SYSROF フォーマット SYSROF PLUS フォーマット
4	最適化情報	<u>in</u> formation*	-	なし	最適化された関数名の表示指定

No.	項目	オプション/サブコマンド名	パラメタ	省略時解釈	指定内容
5	サブコマンドファイル	<code>subcommand</code>	<ファイル名>	なし	リンケージエディタ用サブコマンドファイルの指定

*1: サブコマンドファイルの記述規則および制限事項は、「Hシリーズ リンケージエディタ、ライブラリアン、オブジェクトコンバータユーザズマニュアル」を参照してください。

*2: オブジェクトフォーマットオプションを指定しないときの省略時解釈は、`sysrof` です。

5.3.1 最適化内容指定

(1) optimize オプション / サブコマンド

形式: オプション : `-optimize[=<パラメタ>[,<パラメタ>...]]`

サブコマンド : `optimize[<パラメタ>[,<パラメタ>...]]`

パラメタ : `string_unify | symbol_delete |`

`register | same_code | branch | speed | safe`

説明

モジュール間最適化を実行します。また、パラメタを指定することにより、最適化内容を指定することができます。パラメタは複数指定することができます。

パラメタなし

全ての最適化を実行します。`optimize=string_unify,symbol_delete,register,branch`を指定した時と同じです。

`string_unify`

`const` 属性を持つ定数 / 文字列に対し、同一値定数および同一文字列の統合を、モジュール間に亘って実施します。この最適化は、コンパイラ出力のオブジェクトプログラムに対してのみ実施します。`const` 属性を持つ定数/文字列には、次のものが含まれます。

- ・C プログラム中で `const` 宣言した変数
- ・文字列データの初期値
- ・C++ プログラム中で `extern const` 宣言した変数(D セクションに割り付いたクラスオブジェクト除く)

symbol_delete

一度も参照のない変数 / 関数を削除します。この最適化は、コンパイラ出力のオブジェクトプログラムに対してのみ実施します。この最適化を指定する場合は、必ずリンケージエディタ用サブコマンドファイル内で `entry` サブコマンドを指定してください。

register

関数の呼出関係を解析し、冗長なレジスタ退避・回復コードを削除します。また、呼出前後のレジスタ使用状況により、使用レジスタ番号を変更することもあります。この最適化は、コンパイラ出力のオブジェクトプログラムに対してのみ実施します。

same_code

複数の同一命令列をサブルーチン化して、コードサイズを削減します。この最適化は、コンパイラ出力のオブジェクトプログラムに対してのみ実施します。

branch

プログラムの配置情報に基づいて、分岐命令サイズを最適化します。この最適化は、コンパイラ出力のオブジェクトプログラムに対してのみ実施します。また、他の最適化項目をひとつでも実行すると、本最適化は指定の有無に関わらず、必ず実行します。

speed

最適化項目のうち、同一命令列のサブルーチン化のようなオブジェクトスピード低下を招く可能性のある最適化以外を実施します。`optimize=speed` は、`optimize=string_unify,symbol_delete,register,branch` を指定したときと同じ効果になります。

safe

メモリ割り付け位置が固定でなければならない変数や、スピードを優先したい関数など、部分的に最適化を抑止したい場合があります。

`optimize=safe` は、変数や関数の属性によって制限される可能性のある最適化以外を実施します。`optimize=safe` は、`optimize=string_unify,register,branch` を指定したときと同じ効果になります。

【注意】

`optimize` オプション / サブコマンドのパラメタは、指定されたパラメタの論理和が有効となります。例えば、`optimize=speed,same_code` が指定された場合、`optimize=string_unify,symbol_delete,register,branch,same_code` が有効になります。

(2) nooptimize オプション / サブコマンド

形式: オプション : -nooptimize

 サブコマンド : nooptimize

 パラメタ : なし

説明

モジュール間最適化を実行せずに、**subcommand** オプションで指定したリンケージエディタコマンドに従って、結合・編集作業のみ行い、リンケージエディタを直接起動した時と同じロードモジュールを生成します。

(3) samesize オプション / サブコマンド

形式: オプション : -samesize=<パラメタ>

 サブコマンド : samesize <パラメタ>

 パラメタ : <数値>

説明

optimize=same_code で、最適化の対象となる共通コードのサイズを指定します。このオプションで指定するサイズは、オブジェクトプログラムの実際のバイト数を指します。

optimize=same_code オプションが有効でない場合には、本オプションは無視されます。

数値

16 進数で指定します。指定したサイズ以上の命令列について、共通コードのサブルーチン化を実施します。本オプション省略時は、**samesize=1E** を仮定します。指定できる範囲は 8 数値 7FFF です。

5.3.2 最適化抑止指定

(1) symbol_forbid オプション / サブコマンド

形式： オプション : -symbols_forbid=<パラメタ>[,<パラメタ>...]

サブコマンド : symbol_forbid <パラメタ>[,<パラメタ>...]

パラメタ : <シンボル名>

説明

未参照シンボル削除(`optimize=symbol_delete`)の最適化を抑止する変数名 / 関数名を指定します。`optimize=symbol_delete` オプションが有効でない場合には、本オプション / サブコマンドは無視されます。

シンボル名

変数名、関数名はCプログラム中での定義名の先頭に_を付加します。最適化によって削除してはならない変数名、関数名を指定してください。C++プログラム中の関数名は、付録 H に示すエンコード規則によって定義名を変更していません。

(2) samecode_forbid オプション / サブコマンド

形式： オプション : -samecode_forbid=<パラメタ>[,<パラメタ>...]

サブコマンド : samecode_forbid <パラメタ>[,<パラメタ>...]

パラメタ : <関数名>

説明

共通コード統合(`optimize=same_code`)の最適化を抑止する関数名を指定します。`optimize=same_code` オプションが有効でない場合には、本オプション / サブコマンドは無視されます。

関数名

関数名は C プログラム中での定義名の先頭に_を付加します。C++プログラムのとき一定の規則で変換を行っています。コンパイラが生成した関数名を知る必要があるときは、コンパイラオプション`-code=asm` または、`-lis` にてコンパイラが生成する関数名を参照してください。「付録 H エンコード規則」もあわせて参照してください。共通コード統合の最適化は、コードサイズ削減には効果がありますが、実行速度が低下する可能性があります。スピードを重視する関数等、共通コード統合化を抑止したい関数名を指定してください。

5.3.3 オブジェクトフォーマット指定

(1) オブジェクトフォーマット指定オプション/サブコマンド

形式：	サブコマンド	:	elf
			sysrof
			sysrofplus
パラメタ	:	なし	

説明

指定されたオブジェクトフォーマットに変換します。

5.3.4 最適化情報

(1) information オプション / サブコマンド

形式： オプション : -information

サブコマンド : information

パラメタ : なし

説明

最適化された関数名の表示を指定します。

5.3.5 サブコマンドファイル

(1) subcommand オプション

形式： オプション : -subcommand=<パラメタ>

パラメタ : <ファイル名>

説明

リンケージエディタ用サブコマンドファイルを指定します。本最適化ツールは、最適化処理後、リンケージエディタを自動的に起動し、結合・編集処理を行います。そのため、本オプションを省略することはできません。

ファイル名

リンケージエディタ用サブコマンドファイル名称を指定します。サブコマンドファイル名には、-を含めることはできません。

【注意】

リンケージエディタ用サブコマンドファイルおよび、サブコマンドに関する説明は、「H シリーズリンケージエディタ、ライブラリアン、オブジェクトコンバータ ユーザーズマニュアル」を参照してください。また、以下の注意事項に注意して、サブコマンドファイルを作成してください。

- (a) subcommand オプションで指定するリンケージエディタ用サブコマンドファイルには、最適化ツール用サブコマンドを指定することができます。ただし、最適化ツール用サブコマンドを指定したサブコマンドファイルを、Hシリーズリンケージエディタの subcommand オプションで指定した場合、エラーになります。
- (b) print サブコマンドを指定した場合、マップリストには最適化ツール出力のテンポラリサブコマンドファイル名が出力されます。このテンポラリサブコマンドファイルは、最適化ツール終了後削除されます。
- (c) リンケージエディタサブコマンドの delete, rename, exchange は使用できません。

6. モジュール間最適化時の エラーメッセージ

6.1 モジュール間最適化ツールのエラーメッセージ

6.1.1 エラーメッセージ一覧

本章では、コンパイラの出力するエラーメッセージとエラー内容を説明します。

エラー番号 (エラーレベル) エラーメッセージ

エラーレベルは、エラーの重要度にしたい5種に分類されます。

エラーレベル (I) インフォメーション
(W) ウォーニング
(E) エラー
(F) フェータル

0010 (I) <ユニット名 1> IS REPLACED WITH <ユニット名 2> (<ファイル名>)
<ユニット名 1>を(<ファイル名>)中の<ユニット名 2>に置き換えました。

0020 (I) <外部名 1> IS RENAMED TO <外部名 2>
<外部名 1>を<外部名 2>に変更しました。

0030 (I) <外部名> IS DELETED
<外部名>を削除しました。

0040 (I) DUPLICATE UNIT - (<ユニット名>) in (<ファイル名>) IS DELETED
<ユニット名>のユニットを複数見つけたため、<ファイル名>中のユニット名を削除しました。

0050 (I) <外部参照シンボル名> CANNOT BE DEFINED
<外部参照シンボル名>が見つからないため、強制定義できません。

0060 (I) <外部名> CANNOT BE RENAMED
<外部名>が見つからないため、変更できません。

0070 (I) <外部名> CANNOT BE DELETED
<外部名>が見つからないため、削除できません。

0080 (I) <ユニット名> CANNOT BE REPLACED
<ユニット名>が見つからないため、置き換えができません。

0200 (I) <最適化種別> OPTIMIZE :<セクション名> SECTION IS CREATED

<最適化種別>の最適化によって<セクション名>を作成しました。

0210 (I) <最適化種別> OPTIMIZE:<ユニット名>.<シンボル名> MOVED <セクション名>
> SECTION

<最適化種別>の最適化によって<ユニット名>.<シンボル名>を<セクション名>に移動しました。

0220 (I) <最適化種別> OPTIMIZE:<ユニット名>.<シンボル名> IS CREATED

<最適化種別>の最適化によって<ユニット名>.<シンボル名>を作成しました。

0230 (I) <最適化種別> OPTIMIZE:<ユニット名>.<シンボル名> IS DELETED

<最適化種別>の最適化によって<ユニット名>.<シンボル名>を削除しました。

0240 (I) <ユニット名>.<シンボル名> IS OPTIMIZED

<ユニット名>.<シンボル名>を最適化しました。

1010 (W) DUPLICATE OPTION/SUBCOMMAND (<オプション / サブコマンド名>)

同じオプションまたはサブコマンドを重複して指定しています。後に指定したオプションまたはサブコマンドが有効になります。

1020 (W) IDENTIFIER CHARACTER EXCEEDS 251 (<名前>)

251 文字を超える名前（ユニット名、セクション名、シンボル名）を指定しています。
251 文字までが有効になります。

1040 (W) DUPLICATE SYMBOL (<シンボル名>)

外部定義シンボルが重複しています。先に現われた外部定義シンボルが有効になります。

1050 (W) UNDEFINED EXTERNAL SYMBOL (<ユニット名>.<シンボル名>)

未定義の外部シンボルを参照しています。外部参照は無効になり、0 を仮定します。

1060 (W) REDEFINED SYMBOL (<シンボル名>)

定義済みのシンボルを **DEFINE** オプション / サブコマンドで定義しています。
DEFINE オプション / サブコマンドの指定を無視します。

1070 (W) SECTION ATTRIBUTE MISMATCH (<セクション名>)

属性または境界調整数の異なる同名セクションを入力しました。別セクションとして扱います。

1080 (W) RELOCATION SIZE OVERFLOW (<ユニット名>.<セクション名> - <オフセット値>)

リロケーションの結果がリロケーションサイズを超えました。

1090 (W) ENTRY POINT MULTIPLY DEFINED

実行開始アドレスの指定があるオブジェクトモジュールを複数指定しています。先に現われた実効開始アドレスの指定が有効になります。

1110 (W) DUPLICATE SECTION NAME (<セクション名>)

オプション / サブコマンドで同一セクション名を指定しています。最初に指定したセクション名を有効にします。

1130 (W) CONFLICTING DEVICE TYPE

入力オブジェクトモジュールの対象 CPU と異なる CPU 情報ファイルを指定しています。

1140 (W) SECTION IS NOT IN SAME MEMORY AREA (<セクション名>:xxxx-yyyy)

セクションが一つのメモリ領域に入りきらず、xxxx 番地から yyyy 番地が異なるメモリ領域に割り付けられています。

1150 (W) INACCESSIBLE ADDRESS RANGE (<セクション名>)

セクションが使用できない領域に割り付けられています。

1160 (W) INVALID CPU OPTION/SUBCOMMAND

ロードモジュールファイルをリロケータブル形式に指定して、CPU オプション / サブコマンドを指定しています。

1170 (W) ADDRESS SPACE DUPLICATE

セクションが重複しています。

1180 (W) INVALID UDF OPTION/SUBCOMMAND

出力ロードモジュール形式がアブソリュート指定に対し、**NOUDF** オプション / サブコマンドを指定しています。**NOUDF** オプション / サブコマンドを無視します。

1190 (W) RELOCATION VALUE IS ODD (<ユニット名>.<セクション名> - <オフセット値>)

ディスプレイメントに対するリロケーション結果が奇数になりました。最下位ビットを切り捨てます。

1200 (W) START ADDRESS NOT SPECIFIED FOR SECTION (<セクション名>)

START オプション / サブコマンドで指定していないセクションが存在します。

1210 (W) CANNOT FIND SECTION (<セクション名>)

指定したセクションが見つかりません。

1220 (W) TOO LONG SUBCOMMAND LINE

ディレクトリ名の置き換えで文字数が 511 文字を超えました。511 文字までを有効とします。

1230 (W) TOO MANY DIRECTORY COMMANDS

DIRECTORY サブコマンドで 16 個を超えたディレクトリを指定しています。16 個までを有効とします。

1240 (W) NO DEBUG INFORMATION

デバッグ情報の全くないファイルに対して **DEBUG**、**SDEBUG** オプション / サブコマンドを指定しています。コンパイル、アセンブル時にデバッグオプションを指定してください。

1250 (W) CANNOT SET ENTRY POINT

出力ロードモジュールがリロケータブル形式のとき、実行開始アドレスに定数の外部参照シンボルを指定しています。出力ロードモジュールをアブソリュート形式にするか、実行開始アドレスの指定を削除してください。

1260 (W) TOO LONG CHARACTER

FSYMBOL サブコマンドで指定したセクション内のシンボルの文字数が 238 文字を超えています。

1270 (W) EXTERNAL SYMBOL 0 (<セクション名>)

FSYMBOL サブコマンドで指定したセクション内に外部定義シンボルが存在しません。

1280 (W) ILLEGAL SYMBOL REFERENCE

start サブコマンドで同一アドレスに割り付けたセクション間でシンボルを参照しています。

1600 (W) INVALID SYMBOL_FORBID OPTION

SYMBOL_FORBID の指定が無効です。

1610 (W) INVALID SAMECODE_FORBID OPTION

SAMECODE_FORBID の指定が無効です。

1700 (W) CANNOT FIND SYMBOL SPECIFIED SYMBOL_FORBID (<シンボル名>)

SYMBOL_FORBID で指定したシンボル名が見つかりません。

1710 (W) CANNOT FIND SYMBOL SPECIFIED SAMECODE_FORBID (<シンボル名>)

SAMECODE_FORBID で指定したシンボル名が見つかりません。

1800 (W) <最適化種別> OPTIMIZE:SECTION OVERLAP

<最適化種別>の最適化でサイズ増加により隣接するセクションと重複しました。<最適化種別>の最適化指定を無効にします。

1810 (W) DIFFERENT SYMBOL ASSIGNED TO A GLOBAL REGISTER AMONG FILES (<シンボル名>:<レジスタ番号>)

グローバルレジスタに割り付けるシンボル名、レジスタ番号がファイル間で異なります。

1820 (W) STACK ACCESS SIZE OVERFLOW

レジスタ最適化でスタックアクセスコードがコンパイラのスタック量制限値を超えました。レジスタ最適化指定を無視します。

1830 (W) RELOCATION VALUE EXISTS IN BSR (<ユニット名>)

<ユニット名>のアセンブリプログラムで **BSR** に未解決のシンボルがあります。<ユニット名>を最適化対象外にします。

2010 (E) ILLEGAL SUBCOMMAND/OPTION

不正なサブコマンド名（またはオプション名）を指定しています。

2020 (E) SYNTAX ERROR

指定されたサブコマンド（またはオプション）に構文上の不正があります。

- 2030 (E) TOO LONG SUBCOMMAND LINE
サブコマンドの長さが 511 文字を超えています。
- 2040 (E) ILLEGAL SUBCOMMAND SEQUENCE
サブコマンドの指定順序が不正です。
- 2070 (E) ILLEGAL SECTION NAME (<セクション名>)
不正なセクション名を指定しています。
- 2080 (E) ILLEGAL SYMBOL NAME (<シンボル名>)
不正なシンボル名を指定しています。
- 2100 (E) TOO MANY INPUT FILES
入力ファイル数が 256 個を超えています。
- 2110 (E) CANNOT FIND FILE (<ファイル名>)
指定したファイルが見つかりません。
- 2120 (E) CANNOT FIND UNIT (<ユニット名>)
指定したユニットが見つかりません。
- 2130 (E) CANNOT FIND MODULE (<モジュール名>)
指定したモジュールが見つかりません。
- 2140 (E) DUPLICATE START ADDRESS SPECIFIED
同じ先頭アドレスを重複して指定しています。
- 2170 (E) SUBCOMMAND COMMAND IN SUBCOMMAND FILE
サブコマンドファイル中に **SUBCOMMAND** サブコマンドを指定しています。
- 2190 (E) INVALID ADDRESS (<アドレス>)
指定したアドレスが CPU のアドレス範囲を超えています。
- 2200 (E) TOO MANY ROM COMMANDS
ROM サブコマンドで 64 組を超えたセクションを指定しています。
- 2210 (E) CANNOT CREATE ABSOLUTE MODULE (<モジュール名>)
未定義の外部参照シンボルが存在しています。

2220 (E) DIVISION BY ZERO IN RELOCATION VALUE (<ユニット名>.<セクション名>.<オフセット値>)

0 除算を含むオブジェクトファイルを入力しました。

2600 (E) COMPILER SUPPLEMENTARY INFORMATION FILE MISMATCH (<ファイル名>)

コンパイラ付加情報ファイルの作成日付がオブジェクトと一致しません。

2610 (E) ILLEGAL DUPLICATE SYMBOL (<シンボル名>)

外部定義シンボルが重複しています。

2730 (E) ILLEGAL SAMESIZE SPECIFIED

共通コードサイズ指定が正しくありません。

2740 (E) CANNOT OPTIMIZE RELOCATABLE FILE

出力ロードモジュールファイル形式にリロケータブルを指定しています。

2750 (E) NOT SPECIFIED ENTRY SUBCOMMAND

`optimize=symbol_delete` を指定していますが、`entry` サブコマンド指定がありません。

2760 (E) <サブコマンド名> NOT SUPPORT

該当サブコマンドはモジュール間最適化ツールではサポートしていません。一旦リンケージエディタでリロケータブルファイル出力後、再度 `optlnksh` を実行してください

3010 (F) ILLEGAL COMMAND PARAMETER

不正なコマンドパラメータを指定しています。

3020 (F) CANNOT OPEN FILE (<ファイル名>)

ファイルをオープンできません。

3030 (F) CANNOT READ INPUT FILE (<ファイル名>)

ファイルを読み込むことができません。

3040 (F) CANNOT WRITE OUTPUT FILE (<ファイル名>)

ファイルに書き込むことができません。

3050 (F) CANNOT CLOSE FILE (<ファイル名>)

ファイルをクローズできません。

3060 (F) ILLEGAL FILE FORMAT (<ファイル名>)

指定したファイルのフォーマットが不正です。または、**RENAME** サブコマンドで指定した外部シンボル名が既に存在します。

3070 (F) ILLEGAL RECORD FORMAT (<ファイル名>)

指定したファイル中に不正なレコードがあります。または、除数が 0 の除算があります。

3080 (F) SECTION ADDRESS OVERFLOW (<セクション名>)

セクションの割り付けアドレスが CPU で許されるアドレス範囲を超えています。

3090 (F) ADDRESS OVERFLOW

指定したアドレスが CPU で許されるアドレス範囲を超えています。

3100 (F) MEMORY OVERFLOW

最適化ツールが内部で使用するメモリ領域を割り当てることができません。

3110 (F) PROGRAM ERROR (<nnnn>)

最適化ツールの内部処理で何らかの障害が発生しました。プログラムエラー番号 (nnnn)を確認の上、当社営業担当までご連絡ください。

3120 (F) ILLEGAL START ADDRESS ALIGNMENT (<アドレス>)

オブジェクトモジュールの境界調整数と矛盾するアドレスを指定しています。

3140 (F) CANNOT FIND SECTION (<セクション名>)

指定したセクションが見つかりません。

3230 (F) SECTION SPECIFIED AT ROM OPTION/SUBCOMMAND DOES NOT EXIST (<セクション名>)

ROM コマンドで指定したセクションが存在しません。

3250 (F) ILLEGAL START SECTION (<セクション名>)

START コマンドで指定したセクションの属性が不正です。

3260 (F) CANNOT READ

指定したファイル (標準入力を含む) から入力できません。

3270 (F) SYMBOL ADDRESS OVERFLOW (<シンボル名>)

シンボルの割り付けアドレスが CPU のアドレス範囲を超えています。

3280 (F) ILLEGAL ROM SECTION (<セクション名>)

ROM オプション / サブコマンドの指定で、転送先セクションにサイズ 0 以外のセクションあるいは絶対番地セクションを指定しています。または、転送元と転送先のセクションの属性が異なっています。

3300 (F) ILLEGAL FILE FORMAT (INPUT ABSOLUTE FILE)

アブソリュートロードモジュールを入力ファイルに指定しています。

3310 (F) ILLEGAL FILE FORMAT (MISMATCH OBJECT FORMAT VERSION)

オブジェクト形式の異なるファイルを入力しました。

3320 (F) ILLEGAL FILE FORMAT (INPUT MISMATCH CPU TYPE)

H シリーズ、**SH** シリーズ以外のファイルを入力しました。

3330 (F) CANNOT OPEN INTERNAL FILE

中間ファイルをオープンできません。ディスク容量に空きがないか、またはディスクにハード的なエラーがある場合があります。確認の上、再実行してください。

3340 (F) CANNOT CLOSE INTERNAL FILE

中間ファイルをクローズできません。ディスク容量に空きがないか、またはディスクにハード的なエラーがある場合があります。確認の上、再実行してください。

3350 (F) CANNOT DELETE INTERNAL FILE

中間ファイルを削除できません。ディスク容量に空きがないか、またはディスクにハード的なエラーがある場合があります。確認の上、再実行してください。

3360 (F) CANNOT OUTPUT INTERNAL FILE

中間ファイルに書き込みできません。ディスク容量に空きがないか、またはディスクにハード的なエラーがある場合があります。確認の上、再実行してください。

3370 (F) CANNOT READ INTERNAL FILE

中間ファイルを読み込みできません。ディスク容量に空きがないか、またはディスクにハード的なエラーがある場合があります。確認の上、再実行してください。

3390 (F) TOO MANY UNITS

指定したユニット数が 65,535 を超えました。

3400 (F) TOO MANY SECTIONS

指定したセクション数が 65,535 を超えました。

3700 (F) CANNOT OPEN CPU INFORMATION FILE

CPU 情報ファイルがオープンできません。

3710 (F) CANNOT OPEN INTERNAL FILE

中間ファイルがオープンできません。

3720 (F) CANNOT WRITE INTERNAL FILE

中間ファイルに書き込むことができません。

3730 (F) CANNOT CLOSE INTERNAL FILE

中間ファイルをクローズできません。

3740 (F) CANNOT EXECUTE (<ロードモジュール名>)

optlnksh または **lnk** を起動できません。正しくインストールされているか確認してください。

3750 (F) CANNOT CREATE INTERNAL FILE

中間ファイルを作成することができません。

3760 (F) INTERRUPT BY USER

処理中に標準入力端末から「(CNTL) + C」コマンドによる割り込みを検出しました。

3770 (F) CANNOT ANALYZE OBJECT (<ユニット名>)

オブジェクトコードを解析することができません。プログラムセクション内の **.DATA** 制御命令を削除してください。

3800 (F) TOO MANY EXTERNAL DEFINE SYMBOLS (<ユニット名>)

ユニット内の定義シンボル数が 65535 を超えました。ファイルを分割するか、または当該ユニットのコンパイル時の **GOPTIMIZE** オプション指定を外してください。

3810 (F) TOO MANY EXTERNAL REFERENCE SYMBOLS (<ユニット名>)

ユニット内の参照シンボル数が 65535 を超えました。ファイルを分割するか、または当該ユニットのコンパイル時の **GOPTIMIZE** オプション指定を外してください。

3820 (F) TOO MANY SECTIONS (<ユニット名>)

ユニット内のセクション数が 65535 を超えました。ファイルを分割するか、または当該ユニットのコンパイル時の **GOPTIMIZE** オプション指定を外してください。

3830 (F) <最適化種別> OPTIMIZE:SECTION OVERLAP

<最適化種別>の最適化でサイズ増加により隣接するセクションと重複しました。

7. 標準ライブラリ

7.1 ライブラリの概要

本章では、C/C++言語の中で標準的に利用できる関数であるライブラリ関数の仕様について説明します。本節では、ライブラリの構成を概説し、本章の読み方および用語について説明します。以下の節ではライブラリの構成に従って各ライブラリ関数の仕様を説明します。「付録 A.2 ライブラリ関数仕様」もあわせて参照してください。

(1) ライブラリの種類

ライブラリとは、入出力、文字列操作等の標準的な処理を C/C++言語の関数の形式で実現したものです。また、これらのライブラリは、各処理単位ごとに対応した標準インクルードファイルを取り込むことによって使用可能となります。

標準インクルードファイルには、対応するライブラリの宣言とそれらを使用するために必要なマクロ名が定義されています。

表 7-1 にライブラリの種類と対応する標準インクルードファイルを示します。

表 7-1 ライブラリの種類と対応する標準インクルードファイル

項番	ライブラリの種類	内 容	標準インクルードファイル
1	プログラム診断用ライブラリ	プログラムの診断情報の出力を行うライブラリです。	< assert.h >
2	文字操作用ライブラリ	文字の操作およびチェックを行うライブラリです。	< ctype.h >
3	数値計算用ライブラリ	三角関数等の数値計算を行うライブラリです。	< math.h >
4	プログラムの制御移動用ライブラリ	関数間の制御の移動をサポートするライブラリです。	< setjmp.h >
5	割り込み操作用ライブラリ	プログラムの実行時に生じる割り込み等の条件を発生させたり、その条件が生じた時の処理の登録を行うライブラリです。	< signal.h >
6	可変個の実引数アクセス用ライブラリ	可変個の実引数を持つ関数に対し、その実引数へのアクセスをサポートするライブラリです。	< stdarg.h >
7	入出力用ライブラリ	入出力操作を行うライブラリです。	< stdio.h >
8	標準処理用ライブラリ	記憶域管理等の C プログラムでの標準的処理を行うライブラリです。	< stdlib.h >
9	文字列操作用ライブラリ	文字列の比較、複写等を行うライブラリです。	< string.h >
10	時間操作用ライブラリ	時間に関する操作を行うライブラリです。	< time.h >

また、以上の標準インクルードファイルの他にプログラムの作成作業の効率向上を図るため表 7-2 に示すマクロ名の定義だけからなる標準インクルードファイルがあります。

表 7-2 マクロ名定義からなる標準インクルードファイル

項番	標準インクルードファイル	内容
1	<stddef.h>	各標準インクルードファイルで共通に使用するマクロ名を定義します。
2	<float.h>	浮動小数点数の内部表現に関する各種制限値を定義します。
3	<limits.h>	コンパイラの内部処理に関する各種制限値を定義します。

(2) ライブラリ編の説明形式

次にライブラリ編の説明形式について説明します。

ライブラリの各関数を標準インクルードファイルごとに分類し、その標準インクルードファイルごとに説明してゆきます。その各分類は、まず、標準インクルードファイルの中で定義されているマクロ名や関数宣言に対する説明を行い(図 7-1 参照)、その後、各関数ごとの説明を行う(図 7-2 参照)という形式で構成されています。

図 7-1 に標準インクルードファイル説明の凡例、図 7-2 に関数説明の凡例を示します。

項番<標準インクルードファイル名>		
機能概要 : 本標準インクルードファイルがもつ全体的な機能の概要を説明します。		
定義名一覧 : 本標準インクルードファイル内で定義されるマクロ名、あるいは関数宣言のひとつひとつに対する説明を行います。		
定 義 名	種 類	説 明
本標準インクルードファイル内で定義されるマクロ名、または宣言される関数名を示します。	定義名の分類を示します。定義名の分類は以下のとおりです。 定義名の分類 ・マクロ名 : 引数を持たない形式のマクロ名称であることを意味します。 ・マクロ : 引数を持つ形式のマクロ名称であることを意味します。 ・関数 : 関数により実現されているライブラリの関数宣言であることを意味します。 ・タグ名 : 構造体宣言における構造体宣言名であることを示しています。	定義名に対する説明を行います。
その他、本標準インクルードファイル内で宣言されている関数に共通する注意事項等を説明します。		

図 7-1 標準インクルードファイル説明の凡例

項番 関数名

関数名を示します。ライブラリが関数として実現されているかマクロとして実現されているかを示します。*

機能

ライブラリ関数の機能概要を説明します。

呼び出し手順

ライブラリ関数の書式、およびライブラリ関数使用時に必要となるデータの宣言を示します。

パラメタ

No.	名前	型	意味
項番で す。	パラメタの名称を 示します。	パラメタの型を示 します。	パラメタの意味を説明します。

リターン値

型 : リターン値の型を示します。

正常 : ライブラリ関数が正常終了した時のリターン値を示します。

異常 : ライブラリ関数が異常終了した時のリターン値を示します。

ライブラリ関数の詳細な仕様を説明します。

【注意】

ライブラリ関数の使用上の注意事項を説明します。

【エラー条件】

ライブラリ関数の処理でリターン値からでは、判断できないエラーが発生する条件を示します。

このようなエラーが発生した時、エラーの種類に対応する、コンパイラごとに定義された値が `errno`** に設定されます。（処理系定義の値はユーザーズマニュアルを参照してください）

図 7-2 関数説明の凡例

【注】 * 関数とマクロとの相違は、「(3)(c) 関数とマクロ」を参照してください。

** `errno` は、ライブラリ関数実行中に生じたエラーの種類を格納する変数です。
詳細については「7.4.2 <stddef.h>」を参照してください。

(3) ライブラリ関数の説明で使用する用語

(a) ストリーム入出力

データの入出力において、1文字ごとの入出力関数の呼び出しの度に入出力装置を駆動したり、OSの機能呼び出しでは、効率が悪くなります。そこで、通常はバッファと呼ばれる記憶域を用意しておき、バッファ内のデータを一括して入出力を行います。

一方、プログラムの側から見ると、1文字ごとに入出力関数を呼び出せた方が便利です。

ライブラリ関数では、バッファの管理を自動的に行うことにより、プログラム内でバッファの状態を意識することなしに、1文字単位ごとの入出力を効率よく行うことができます。

このように、データの入出力を効率よく実現するために詳細な手段を意識せず、入出力をひとつのデータの流れ（ストリーム）と考えてプログラムを作ることのできる機能をストリーム入出力といいます。

(b) FILE 構造体およびファイルポインタ

(a)で述べたストリーム入出力に必要なバッファや、その他の情報は、ひとつの構造体の中に記憶されており、標準インクルードファイル<stdio.h>の中でFILEという名前で定義されています。

ストリーム入出力においては、ファイルはすべてFILE構造体のデータ構造を持つものとして扱います。このようなファイルをストリームファイルと呼びます。このファイル構造体へのポインタをファイルポインタと呼び、入出力ファイルを指定するために用います。

ファイルポインタは、

```
FILE *fp ;
```

と定義します。

fopen関数等でファイルをオープンすると、ファイルポインタが得られますが、オープン処理が失敗するとNULLが返ってきます。NULLポインタを、他のストリーム入出力関数に指定すると、その関数は異常終了しますので、注意が必要です。ファイルをオープンした時は、成功したか、失敗したか必ずファイルポインタの値をチェックするようにしてください。

(c) 関数とマクロ

ライブラリ関数の実現方法としては、関数とマクロの二通りがあります。

関数は、通常のユーザ作成の関数と同じインタフェースを持ち、リンク時に取り込みます。

マクロは、その関数に関連した標準インクルードファイルの中で `#define` 文を用いて定義されています。

マクロについては、以下の点に注意する必要があります。

- (1) マクロは、プリプロセッサによって自動的に展開されてしまうので、ユーザが同じ名前の関数を宣言してもマクロを無効にすることはできません。
- (2) マクロのパラメタとして副作用のある式（代入式、インクリメント、デクリメント）を指定した時、その効果は保証されません。

例

パラメタの絶対値を求める **MACRO** を以下のようにマクロ定義します。

```
#define MACRO(a) (a) > = 0 ? (a) : -(a)
```

と定義されている時、

```
X=MACRO(a++)
```

がプログラム内にあると、

```
X = (a++) > = 0 ? (a++) : -(a++)
```

と展開され、`a` は 2 回インクリメントされることになり、また結果の値も最初の `a` の値の絶対値とは異なります。

(d) EOF

`getc` 関数、`getchar` 関数、`fgetc` 関数等のファイルからデータを入力する関数において、ファイル終了 (End Of File) 時に返される値です。EOF という名前は、標準インクルードファイル `<stdio.h>` の中で定義されています。

(e) NULL

ポインタが何も指していない時の値です。NULL という名前は、標準インクルードファイル `<stddef.h>` の中で定義されています。

(f) ヌル文字

C 言語における文字列の終わりは、文字 `'\0'` によって示されることになっています。ライブラリ関数における文字列のパラメタも、すべてこの約束に従っていなければなりません。この文字列の終わりを示す文字 `'\0'` を、以下ヌル文字と呼びます。

(g) リターンコード

ライブラリ関数の中には、リターン値によって、指定された処理が成功したか、失敗したか等の結果を判断するものがあります。このような場合のリターン値を特にリターンコードと呼びます。

(h) テキストファイルとバイナリファイル

多くのシステムでは、データを格納するために、特殊なファイル形式を持っています。これをサポートするために、ライブラリ関数には、テキストファイルとバイナリファイルの2種類のファイル形式があります。

(1) テキストファイル

テキストファイルは、通常のテキストを格納するためのファイルで、行の集まりとして構成されています。テキストファイルの入力の時、行の区切りとして改行文字（`'\n'`）が入力されます。また、出力の時、改行文字（`'\n'`）を出力することにより、現在の行の出力を終了します。テキストファイルは、処理系ごとの標準的なテキストを格納するファイルの入出力を行うためのファイルです。テキストファイルでは、ライブラリ関数で入出力する文字は必ずしもファイル内の物理的なデータの並びと対応していません。テキストファイルの実現法についてはユーザズマニュアルを参照してください。

(2) バイナリファイル

バイナリファイルは、バイトデータの列として構成されているファイルです。ライブラリ関数で入出力するデータは、ファイル内の物理的なデータの並びと対応しています。

(i) 標準入出力ファイル

入出力のライブラリ関数で、ファイルのオープン等の準備を行わずに標準的に使用できるファイルを標準入出力ファイルといいます。標準入出力ファイルには、標準入力ファイル（`stdin`）、標準出力ファイル（`stdout`）、標準エラー出力ファイル（`stderr`）があります。

(1) 標準入力ファイル（`stdin`）

プログラムへの入力となる標準的なファイルです。

(2) 標準出力ファイル（`stdout`）

プログラムからの出力となる標準的なファイルです。

(3) 標準エラー出力ファイル（`stderr`）

プログラムからのエラーメッセージ等の出力を行うための標準的なファイルです。

(j) 浮動小数点数

浮動小数点数は、実数を近似して表現したものです。C 言語のソースプログラム上では浮動小数点数を 10 進数で表現していますが、計算機の内部では通常 2 進数で表現されます。

2 進数の場合の浮動小数点数の表現は次のようになります。

$$2^n \times m \text{ (} n: \text{整数、} m: \text{2 進小数) }$$

ここで n を浮動小数点数の指数部、 m を仮数部といいます。浮動小数点数を一定のデータサイズで表現するために、 n と m のビット数は通常固定されています。

以下、浮動小数点数に関する用語を説明します。

(1) 基数

浮動小数点数が何進法で表現されているかを示す整数値です。通常、基数は 2 です。

(2) 丸め

浮動小数点数よりも精度の高い演算の途中結果を浮動小数点数に格納する場合に丸めが行われます。丸めには、切り上げ、切り捨て、四捨五入 (2 進小数の場合は、0 捨 1 入となります。) があります。

(3) 正規化

浮動小数点数を、 $2^n \times m$ の形式で表現する場合、同一の数値を表わす異なる表現が可能です。

例

$$2^5 \times 1.0_{(2)} \quad ((2) \text{ は 2 進数を示します。})$$

$$2^6 \times 0.1_{(2)}$$

どちらも同じ値です。

通常は、有効桁数を確保するために、先頭の桁が 0 でないような表現を用います。これを正規化された浮動小数点数といい、浮動小数点数をこのような表現に変換する操作を正規化といいます。

(4) ガードビット

浮動小数点数の演算の途中結果を保持する場合、通常は、丸めを行うために実際の浮動小数点数よりも 1 ビット多いデータを用意します。しかし、これだけでは桁落ち等が生じた時に正確な結果を求めることができません。このために、もう 1 ビット設けて演算の途中結果を保持する手法があります。このビットをガードビットといいます。

(k) ファイルアクセスモード

ファイルをオープンする時にどのような処理をファイルに行うかを示す文字列です。文字列の種類には表 7-3 に示す 12 種類があります。

表 7-3 ファイルアクセスモードの種類

項番	アクセスモード	意 味
1	" r "	テキストファイルを読み込み用にオープンします
2	" w "	テキストファイルを書き出し用にオープンします。
3	" a "	テキストファイルを追加用にオープンします。
4	" rb "	バイナリファイルを読み込み用にオープンします。
5	" wb "	バイナリファイルを書き出し用にオープンします。
6	" ab "	バイナリファイルを追加用にオープンします。
7	" r+ "	テキストファイルを読み込み用でかつ更新用にオープンします。
8	" w+ "	テキストファイルを書き出し用でかつ更新用にオープンします。
9	" a+ "	テキストファイルを追加用でかつ更新用にオープンします。
10	" r+b "	バイナリファイルを読み込み用でかつ更新用にオープンします。
11	" w+b "	バイナリファイルを書き出し用でかつ更新用にオープンします。
12	" a+b "	バイナリファイルを追加用でかつ更新用にオープンします。

(l) 処理系定義

コンパイラが異なることによって定義が異なるという意味です。各コンパイラの定義は、「付録 A.2 ライブラリ関数仕様」を参照してください。

(m) エラー指示子、ファイル終了指示子

ストリームファイルごとに、ファイルの入出力の際にエラーが生じたかどうかを示すエラー指示子、入力ファイルが終了したかどうかを示すファイル終了指示子というデータを保持しています。

これらのデータは、それぞれ `ferror` 関数、`feof` 関数によって参照することができます。

ストリームファイルを扱う関数のうち、そのリターン値だけでは、エラーの発生やファイルの終了の情報が得られないものがあります。エラー指示子とファイル終了指示子は、このような関数の実行後にファイルの状態を調べるために有効です。

(n) 位置指示子

ディスク上のファイル等、ファイル内の任意の位置からの読み書きができるストリームファイルは、現在読み書きしているファイル内の位置を示すデータを保持しています。これを位置指示子といいます。

端末装置等、ファイル内の読み書きの位置を変更できないストリームファイルでは、位置指示子は使用しません。

(4) ライブラリ使用時の注意事項

- (a) ライブラリの中で定義されているマクロの内容は、コンパイラごとに異なります。
ライブラリを使用する場合、これらのマクロの内容を再定義した場合、動作は保証されません。
- (b) ライブラリは、すべての場合についてエラーを検出しているわけではありません。
2 節以降の説明に示す以外の形式でライブラリ関数を呼び出した場合、動作は保証されません。

7.2 < stddef.h >

機能概要

標準インクルードファイルの中で共通に使用されるマクロ名を定義します。

定義名一覧

定義名	種類	説 明
ptrdiff_t	マクロ名	二つのポインタを減算した結果の型を示します。
size_t	マクロ名	sizeof 演算子による演算結果の型を示します。
NULL	マクロ名	ポインタが何も指していない時の値を示します。 この値は、0 と等値演算子 (==) による比較結果が真になるような値です。
errno	マクロ名	ライブラリ関数の処理中にエラーが発生した場合、そのライブラリごとに定義されたエラーコードがこの errno に設定されます。ライブラリ関数を呼び出す前に errno に 0 を設定しておき、ライブラリ関数の処理終了後に errno に設定されているコードを調べることによってライブラリ関数の処理中に発生したエラーをチェックすることができます。

上記のマクロ名は、すべて処理系定義です。

7.3 < assert.h >

機能概要

プログラム中に診断機能を付け加えます。

定義名一覧

定義名	種類	説 明
assert	マクロ	プログラム中に診断機能を付け加えます。

< assert.h > で定義される診断機能を無効にするためには、< assert.h > を取り込む前に **NDEBUG** というマクロ名を **#define** 文で定義してください (**#define NDEBUG**)。

assert 関数はマクロとして実現されています。

assert というマクロ名に対して **#undef** 文を使用すると、それ以降の **assert** の呼び出しの効果は保証されません。

7.3.1 assert マクロ

機能

プログラム中に診断機能を付け加えます。

呼び出し手順

```
#include <assert.h>

int expression;

assert (expression);
```

パラメタ

No.	名前	型	意味
1	expression	int	評価する式

リターン値

型 : void

正常 : -

異常 : -

`assert` マクロは `expression` が真の時は値を返さずに処理を終了します。`expression` が偽の時は、診断情報をコンパイラによって定義された書式で標準エラーファイルに出力し、その後 `abort` 関数を呼び出します。

診断情報の中には、パラメタのプログラムテキスト、ソースファイル名、ソース行番号が含まれています。

7.4 < ctype.h >

機能概要

文字に対して、その種類の判定や変換を行います。

定義名一覧

定義名	種類	説 明
isalnum	関数	英字または 10 進数字かどうかを判定します。
isalpha	関数	英字かどうかを判定します。
isctrl	関数	制御文字かどうかを判定します。
isdigit	関数	10 進数字かどうかを判定します。
isgraph	関数	空白を除く印字文字かどうかを判定します。
islower	関数	英小文字かどうかを判定します。
isprint	関数	空白を含む印字文字かどうかを判定します。
ispunct	関数	特殊文字かどうかを判定します。
isspace	関数	空白類文字かどうかを判定します。
isupper	関数	英大文字かどうかを判定します。
isxdigit	関数	16 進数字かどうかを判定します。
tolower	関数	英大文字を英小文字に変換します。
toupper	関数	英小文字を英大文字に変換します。

上記の関数において、入力パラメタの値が **unsigned char** 型で表現できる範囲に含まれず、なおかつ EOF でない場合、その関数の動作は保証されません。また、文字の種類の一覧を表 7-4 に示します。

表 7-4 文字の種類

項番	文字の種類	内 容
1	英大文字	以下の 26 文字のいずれかの文字です。 'A'、'B'、'C'、'D'、'E'、'F'、'G'、'H'、'I'、'J'、'K'、'L'、'M'、'N'、'O'、'P'、'Q'、'R'、'S'、'T'、'U'、'V'、'W'、'X'、'Y'、'Z'
2	英小文字	以下の 26 文字のいずれかの文字です。 'a'、'b'、'c'、'd'、'e'、'f'、'g'、'h'、'i'、'j'、'k'、'l'、'm'、'n'、'o'、'p'、'q'、'r'、's'、't'、'u'、'v'、'w'、'x'、'y'、'z'
3	英字	英大文字と英小文字のいずれかの文字です。
4	10 進数字	以下の 10 文字のいずれかの文字です。 '0'、'1'、'2'、'3'、'4'、'5'、'6'、'7'、'8'、'9'
5	印字文字	空白 (' ') を含む、ディスプレイ上に表示される文字のことです。 ASCII コードの 0x20 ~ 0x7E に対応します。
6	制御文字	印字文字以外の文字のことです。
7	空白類文字	以下の 6 文字のいずれかの文字です。 空白 (' ')、書式送り (' \f')、改行 (' \n')、復帰 (' \r')、 水平タブ (' \t')、垂直タブ (' \v')、
8	16 進数字	以下の 22 文字のいずれかの文字です。 '0'、'1'、'2'、'3'、'4'、'5'、'6'、'7'、'8'、'9'、 'A'、'B'、'C'、'D'、'E'、'F'、'a'、'b'、'c'、'd'、'e'、'f'
9	特殊文字	空白 (' ')、英字、及び 10 進数字を除く任意の印字文字のことです。

7.4.1 isalnum 関数

機能

文字が英字または 10 進数字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>

int c, ret;

ret=isalnum(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型 : int

正常: 文字 c が英字または 10 進数字の時 : 0 以外
文字 c が英字または 10 進数字以外の時 : 0

異常: -

7.4.2 isalpha 関数

機能

文字が英字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>

int c , ret;

ret=isalpha(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型 : int

正常: 文字 c が英字の時 : 0 以外
文字 c が英字以外の時 : 0

異常: -

7.4.3 iscntrl 関数

機能

文字が制御文字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>

int c, ret;

ret=iscntrl(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型 : int

正常: 文字 c が制御文字の時 : 0 以外
文字 c が制御文字以外の時 : 0

異常: -

7.4.4 isdigit 関数

機能

文字が 10 進数字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>

int c, ret;

ret=isdigit(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型 : int

正常: 文字 c が 10 進数字の時 : 0 以外

文字 c が 10 進数字以外の時 : 0

異常: -

7.4.5 isgraph 関数

機能

文字が空白（ ' ' ）を除く任意の印字文字かどうかを判定します。

呼び出し手順

```
#include <ctype.h>

int c, ret;

ret=isgraph(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型 : int

正常： 文字 c が空白を除く印字文字の時 : 0 以外
文字 c が空白を除く印字文字以外の時 : 0

異常： -

7.4.6 islower 関数

機能

文字が英小文字であるかどうか判定します

呼び出し手順

```
#include <ctype.h>

int c, ret;

ret=islower(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型 : int

正常: 文字 c が英小文字の時 : 0 以外
文字 c が英小文字以外の時 : 0

異常: -

7.4.7 isprint 関数

機能

文字が空白文字（' '）を含む印字文字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>

int c, ret;

ret=isprint(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型 : int

正常: 文字 c が空白文字を含む印字文字の時 : 0 以外

文字 c が空白文字を含む印字文字以外の時 : 0

異常: -

7.4.8 ispunct 関数

機能

文字が特殊文字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>

int c, ret;

ret=ispunct(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型 : int

正常: 文字 c が特殊文字の時 : 0 以外
文字 c が特殊文字以外の時 : 0

異常: -

7.4.9 isspace 関数

機能

文字が空白類文字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>
```

```
int c, ret;
```

```
ret=isspace(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型 : int

正常: 文字 c が空白類文字の時 : 0 以外

文字 c が空白類文字以外の時 : 0

異常: -

7.4.10 isupper 関数

機能

文字が英大文字であるかどうか判定します。

呼び出し手順

```
#include <ctype.h>

int c, ret;

ret=isupper(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型 : int

正常: 文字 c が英大文字の時 : 0 以外
文字 c が英大文字以外の時 : 0

異常: -

7.4.11 isxdigit 関数

機能

文字が 16 進数字かどうか判定します。

呼び出し手順

```
#include <ctype.h>

int c, ret;

ret=isxdigit(c);
```

パラメタ

No.	名前	型	意味
1	c	int	判定する文字

リターン値

型 : int

正常: 文字 c が 16 進数字の時 : 0 以外
文字 c が 16 進数字以外の時 : 0

異常: -

7.4.12 tolower 関数

機能

英大文字を対応する英小文字に変換します。

呼び出し手順

```
#include <ctype.h>

int c, ret;

ret=tolower(c);
```

パラメタ

No.	名前	型	意味
1	c	int	変換する文字

リターン値

型 : int

正常: 文字 c が英大文字の時: 文字 c に対応する英小文字
文字 c が英大文字以外の時: 文字 c

異常: -

7.4.13 toupper 関数

機能

英小文字を対応する英大文字に変換します。

呼び出し手順

```
#include <ctype.h>
```

```
int c, ret;
```

```
ret=toupper(c);
```

パラメタ

No.	名前	型	意味
1	c	int	変換する文字

リターン値

型 : int

正常: 文字 c が英小文字の時: 文字 c に対応する英大文字

文字 c が英小文字以外の時: 文字 c

異常: -

7.5 < float.h >

機能概要

浮動小数点数の内部表現に関する各種制限値を定義します。

定義名一覧

定義名	種類	説 明
FLT_RADIX	マクロ名	指数部表現における基数を示します。
FLT_ROUNDS	マクロ名	<p>加算演算結果が丸められるかどうかを示します。</p> <p>本マクロの定義の意味は以下のとおりです。</p> <ul style="list-style-type: none"> ・加算演算結果を丸める場合：正の値 ・加算演算結果を切り捨てる場合：0 ・特に規定しない場合：-1 <p>丸め、切り捨ての方法は、処理系定義です。</p>
FLT_GUARD	マクロ名	<p>乗算演算結果においてガードビットが用いられるかどうかを示します。</p> <p>本マクロの定義の意味は以下のとおりです。</p> <ul style="list-style-type: none"> ・ガードビットが用いられる場合：1 ・ガードビットが用いられない場合：0
FLT_NORMALIZE	マクロ名	<p>浮動小数点数の値が正規化されているかどうかを示します。</p> <p>本マクロの定義の意味は以下のとおりです。</p> <ul style="list-style-type: none"> ・正規化されている場合：1 ・正規化されていない場合：0
FLT_MAX	マクロ名	float 型の浮動小数点数値として表現できる最大値を示します。
DBL_MAX	マクロ名	double 型の浮動小数点数値として表現できる最大値を示します。
LDBL_MAX	マクロ名	long double 型の浮動小数点数値として表現できる最大値を示します。
FLT_MIN	マクロ名	float 型の浮動小数点数値として表現できる正の値での最小値を示します。
DBL_MIN	マクロ名	double 型の浮動小数点数値として表現できる正の値での最小値を示します。
LDBL_MIN	マクロ名	long double 型の浮動小数点数値として表現できる正の値での最小値を示します。

定義名	種類	説明
FLT_MAX_EXP	マクロ名	float 型の浮動小数点数値として表現できる基数のべき乗の最大値を示します。
DBL_MAX_EXP	マクロ名	double 型の浮動小数点数値として表現できる基数のべき乗の最大値を示します。
LDBL_MAX_EXP	マクロ名	long double 型の浮動小数点数値として表現できる基数のべき乗の最大値を示します。
FLT_MIN_EXP	マクロ名	float 型の正の値として表現できる浮動小数点数値の基数のべき乗の最小値を示します。
DBL_MIN_EXP	マクロ名	double 型の正の値として表現できる浮動小数点数値の基数のべき乗の最小値を示します。
LDBL_MIN_EXP	マクロ名	long double 型の正の値として表現できる浮動小数点数値の基数のべき乗の最小値を示します。
FLT_MAX_10_EXP	マクロ名	float 型の浮動小数点数値として表現できる 10 のべき乗の最大値を示します。
DBL_MAX_10_EXP	マクロ名	double 型の浮動小数点数値として表現できる 10 のべき乗の最大値を示します。
LDBL_MAX_10_EXP	マクロ名	long double 型の浮動小数点数値として表現できる 10 のべき乗の最大値を示します。
FLT_MIN_10_EXP	マクロ名	float 型の正の値として表現できる浮動小数点数値の 10 のべき乗の最小値を示します。
DBL_MIN_10_EXP	マクロ名	double 型の正の値として表現できる浮動小数点数値の 10 のべき乗の最小値を示します。
LDBL_MIN_10_EXP	マクロ名	long double 型の正の値として表現できる浮動小数点数値の 10 のべき乗の最小値を示します。
FLT_DIG	マクロ名	float 型の浮動小数点数値の 10 進精度の最大桁数を示します。
DBL_DIG	マクロ名	double 型の浮動小数点数値の 10 進精度の最大桁数を示します。
LDBL_DIG	マクロ名	long double 型の浮動小数点数値の 10 進精度の最大桁数を示します。
FLT_MANT_DIG	マクロ名	float 型の浮動小数点数値を基数に合わせて表現した時の仮数部の最大桁数を示します。
DBL_MANT_DIG	マクロ名	double 型の浮動小数点数値を基数に合わせて表現した時の仮数部の最大桁数を示します。
LDBL_MANT_DIG	マクロ名	long double 型の浮動小数点数値を基数に合わせて表現した時の仮数部の最大桁数を示します。

定義名	種類	説明
FLT_EXP_DIG	マクロ名	float 型の浮動小数点数値を基数に合わせて表現した時の指数部の最大桁数を示します。
DBL_EXP_DIG	マクロ名	double 型の浮動小数点数値を基数に合わせて表現した時の指数部の最大桁数を示します。
LDBL_EXP_DIG	マクロ名	long double 型の浮動小数点数値を基数に合わせて表現した時の指数部の最大桁数を示します。
FLT_POS_EPS	マクロ名	float 型において、 $1.0 + x$ 1.0 である最小の浮動小数点数値 x を示します。
DBL_POS_EPS	マクロ名	double 型において、 $1.0 + x$ 1.0 である最小の浮動小数点数値 x を示します。
LDBL_POS_EPS	マクロ名	long double 型において、 $1.0 + x$ 1.0 である最小の浮動小数点数値 x を示します。
FLT_NEG_EPS	マクロ名	float 型において、 $1.0 - x$ 1.0 である最小の浮動小数点数値 x を示します。
DBL_NEG_EPS	マクロ名	double 型において、 $1.0 - x$ 1.0 である最小の浮動小数点数値 x を示します。
LDBL_NEG_EPS	マクロ名	long double 型において、 $1.0 - x$ 1.0 である最小の浮動小数点数値 x を示します。
FLT_POS_EPS_EXP	マクロ名	float 型において、 $1.0 + (\text{基数})^n$ 1.0 となる最小の整数 n を示します。
DBL_POS_EPS_EXP	マクロ名	double 型において、 $1.0 + (\text{基数})^n$ 1.0 となる最小の整数 n を示します。
LDBL_POS_EPS_EXP	マクロ名	long double 型において、 $1.0 + (\text{基数})^n$ 1.0 となる最小の整数 n を示します。
FLT_NEG_EPS_EXP	マクロ名	float 型において、 $1.0 - (\text{基数})^n$ 1.0 となる最小の整数 n を示します。
DBL_NEG_EPS_EXP	マクロ名	double 型において、 $1.0 - (\text{基数})^n$ 1.0 となる最小の整数 n を示します。
LDBL_NEG_EPS_EXP	マクロ名	long double 型において、 $1.0 - (\text{基数})^n$ 1.0 となる最小の整数 n を示します。

上記のマクロ名はすべて処理系定義です。

7.6 < limits.h >

機能概要

整数型データの内部表現に関する各種制限値を定義します。

定義名一覧

定義名	種類	説 明
CHAR_BIT	マクロ名	char 型が何ビットから構成されるかを示します。
CHAR_MAX	マクロ名	char 型の変数が値として持つことのできる最大値を示します。
CHAR_MIN	マクロ名	char 型の変数が値として持つことのできる最小値を示します。
SCHAR_MAX	マクロ名	signed char 型の変数が値として持つことのできる最大値を示します。
SCHAR_MIN	マクロ名	signed char 型の変数が値として持つことのできる最小値を示します。
UCHAR_MAX	マクロ名	unsigned char 型の変数が値として持つことのできる最大値を示します。
SHRT_MAX	マクロ名	short int 型の変数が値として持つことのできる最大値を示します。
SHRT_MIN	マクロ名	short int 型の変数が値として持つことのできる最小値を示します。
USHRT_MAX	マクロ名	unsigned short int 型の変数が値として持つことのできる最大値を示します。
INT_MAX	マクロ名	int 型の変数が値として持つことのできる最大値を示します。
INT_MIN	マクロ名	int 型の変数が値として持つことのできる最小値を示します。
UINT_MAX	マクロ名	unsigned int 型の変数が値として持つことのできる最大値を示します。
LONG_MAX	マクロ名	long 型の変数が値として持つことのできる最大値を示します。
LONG_MIN	マクロ名	long 型の変数が値として持つことのできる最小値を示します。
ULONG_MAX	マクロ名	unsigned long 型の変数が値として持つことのできる最大値を示します。

上記のマクロ名はすべて処理系定義です。

7.7 <math.h>

機能概要

各種の数値計算を行います。

定義名一覧

定義名	種類	説明
EDOM	マクロ名	関数に入力するパラメタの値が関数内で定義している値の範囲を超える時、errno に設定する値を示しています。
ERANGE	マクロ名	関数の計算結果が double 型の値として表わせない時、あるいはオーバーフロー / アンダフローとなった時、errno に設定する値を示しています。
HUGE_VAL	マクロ名	関数の計算結果がオーバーフローした時に、関数のリターン値として返す値を示しています。
acos	関数	浮動小数点数の逆余弦を計算します。
asin	関数	浮動小数点数の逆正弦を計算します。
atan	関数	浮動小数点数の逆正接を計算します。
atan2	関数	浮動小数点数どうしを除算した結果の値の逆正接を計算します。
cos	関数	浮動小数点数のラディアン値の余弦を計算します。
sin	関数	浮動小数点数のラディアン値の正弦を計算します。
tan	関数	浮動小数点数のラディアン値の正接を計算します。
cosh	関数	浮動小数点数の双曲線余弦を計算します。
sinh	関数	浮動小数点数の双曲線正弦を計算します。
tanh	関数	浮動小数点数の双曲線正接を計算します。
exp	関数	浮動小数点数の指数関数を計算します。
frexp	関数	浮動小数点数を [0.5, 1.0) の値として 2 のべき乗の積に分解します。
ldexp	関数	浮動小数点数と 2 のべき乗の乗算を計算します。
log	関数	浮動小数点数の自然対数を計算します。
log10	関数	浮動小数点数の 10 を底とする対数を計算します。
modf	関数	浮動小数点数を整数部分と小数部分に分解します。
pow	関数	浮動小数点数のべき乗を計算します。
sqrt	関数	浮動小数点数の正の平方根を計算します。
ceil	関数	浮動小数点数の小数点以下を切り上げた整数値を求めます。
fabs	関数	浮動小数点数の絶対値を計算します。
floor	関数	浮動小数点数の小数点以下を切り捨てた整数値を求めます。

定義名	種類	説 明
fmod	関数	浮動小数点数どうしを除算した結果の余りを計算します。

上記のマクロ名はすべて処理系定義です。

エラーが発生した時の動作を以下に説明します。

(1) 定義域エラー

関数に入力するパラメタの値が関数内で定義している値の範囲を超えている時、定義域エラーというエラーが発生します。この時 `errno` には `EDOM` の値が設定されます。また、関数のリターン値は、コンパイラによって異なりますので、ユーザーズマニュアルを参照してください。

(2) 範囲エラー

関数における計算結果が `double` 型の値として表わせない時には範囲エラーというエラーが発生します。この時、`errno` には `ERANGE` の値が設定されます。また、計算結果がオーバーフローの時は、正しく計算が行われた時と同様の符号の `HUGE_VAL` の値をリターン値として返します。逆に計算結果がアンダフローの時は、`0` をリターン値として返します。

【注】 1: `<math.h>` の関数の呼び出しによって定義域エラーが発生する可能性がある場合は、結果の値をそのまま用いるのは危険です。必ず `errno` をチェックしてから用いてください。

例

```

        :
1   x=asin(a);
2   if (errno==EDOM)
3       printf ("error¥n");
4   else
5       printf ("result is : %d¥n", x);
        :

```

1 行目で、asin 関数を使って逆正弦値を求めます。このとき、引数 a の値が、asin 関数の定義域 $[-1.0, 1.0]$ の範囲を超えていると、errno に値 EDOM が設定されます。2 行目で定義域エラーが生じたかどうかの判定をします。定義域エラーが生じれば、3 行目で、error を出力します。定義域エラーが生じなければ 4 行目で、逆正弦値を出力します。

- 2: 範囲エラーが発生するかどうかは、コンパイラによって定まる、浮動小数点数の内部表現形式によって異なります。たとえば無限大を値として表現できる内部表現形式を採用している場合、範囲エラーの生じないように < math.h > のライブラリ関数を実現することができます。

詳細については「付録 A.2 ライブラリ関数仕様」を参照してください。

7.7.1 acos 関数

機能

浮動小数点数の逆余弦を計算します。

呼び出し手順

```
#include <math.h>
```

```
double d, ret;
```

```
ret=acos(d);
```

パラメタ

No.	名前	型	意味
1	d	double	逆余弦を求める浮動小数点数

リターン値

型 : **double**

正常: **d** の逆余弦値

異常: 定義域エラーの時: 結果はコンパイラによって異なります。

acos 関数のリターン値の範囲は (0,) です。

【エラー条件】

d の値が (-1.0, 1.0) の範囲を超えている時、定義域エラーになります。

7.7.2 asin 関数

機能

浮動小数点数の逆正弦を計算します。

呼び出し手順

```
#include <math.h>

double d, ret;

ret=asin(d);
```

パラメタ

No.	名前	型	意味
1	d	double	逆正弦を求める浮動小数点数

リターン値

型 : **double**

正常 : **d** の逆正弦値

異常 : 定義域エラーの時 : 結果はコンパイラによって異なります。

asin 関数のリターン値の範囲は (- $\pi/2$, $\pi/2$) です。

【エラー条件】

d の値が (-1.0, 1.0) の範囲を超えている時、定義域エラーになります。

7.7.3 atan 関数

機能

浮動小数点数の逆正接を計算します。

呼び出し手順

```
#include <math.h>
```

```
double d, ret;
```

```
ret=atan(d);
```

パラメタ

No.	名前	型	意味
1	d	double	逆正接を求める浮動小数点数

リターン値

型 : double

正常 : d の逆正接値

異常 : -

atan 関数のリターン値の範囲は $(-\pi/2, \pi/2)$ です。

7.7.4 atan2 関数

機能

浮動小数点数どうしを除算した結果の値の逆正接を計算します。

呼び出し手順

```
#include <math.h>

double x, y, ret;

ret=atan2(y, x);
```

パラメタ

No.	名前	型	意味
1	x	double	除数
2	y	double	被除数

リターン値

型 : double

正常 : y を x で除算したときの逆正接値

異常 : 定義域エラーの時 : 結果はコンパイラによって異なります。

atan2 関数のリターン値の範囲は (- ,) です。atan2 関数の示す意味を図 7-3 に示します。図に示すように、atan2 関数の結果は、点 (x , y) と原点を通る直線と x 軸をなす角を求めます。

y = 0.0 で x が負の時、結果は π 、x = 0.0 の時、y の値の正負に従って結果は $\pm \pi / 2$ となります。

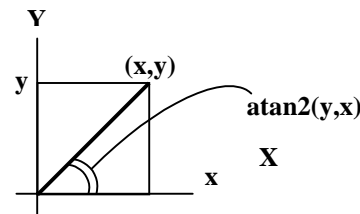


図 7-3 atan2 関数の意味

【エラー条件】

x, y の値がともに 0.0 の時、定義域エラーになります。

7.7.5 cos 関数

機能

浮動小数点数のラディアン値の余弦を計算します。

呼び出し手順

```
#include <math.h>
```

```
double d, ret;
```

```
ret=cos(d);
```

パラメタ

No.	名前	型	意味
1	d	double	余弦を求めるラディアン値

リターン値

型 : **double**

正常 : **d** の余弦値

異常 : -

7.7.6 sin 関数

機能

浮動小数点数のラディアン値の正弦を計算します。

呼び出し手順

```
#include <math.h>

double d, ret;

ret=sin(d);
```

パラメタ

No.	名前	型	意味
1	d	double	正弦を求めるラディアン値

リターン値

型 : **double**

正常 : **d** の正弦値

異常 : -

7.7.7 tan 関数

機能

浮動小数点数のラディアン値の正接を計算します。

呼び出し手順

```
#include <math.h>
```

```
double d, ret;
```

```
ret=tan(d);
```

パラメタ

No.	名前	型	意味
1	d	double	正接を求めるラディアン値

リターン値

型 : **double**

正常 : **d** の正接値

異常 : -

7.7.8 cosh 関数

機能

浮動小数点数の双曲線余弦を計算します。

呼び出し手順

```
#include <math.h>

double d, ret;

ret=cosh(d);
```

パラメタ

No.	名前	型	意味
1	d	double	双曲線余弦を求める浮動小数点数

リターン値

型 : **double**

正常 : **d** の双曲線余弦値

異常 : -

7.7.9 sinh 関数

機能

浮動小数点数の双曲線正弦を計算します。

呼び出し手順

```
#include <math.h>
```

```
double d, ret;
```

```
ret=sinh(d);
```

パラメタ

No.	名前	型	意味
1	d	double	双曲線正弦を求める浮動小数点数

リターン値

型 : **double**

正常 : **d** の双曲線正弦値

異常 : -

7.7.10 tanh 関数

機能

浮動小数点数の双曲線正接を計算します。

呼び出し手順

```
#include <math.h>

double d, ret;

ret=tanh(d);
```

パラメタ

No.	名前	型	意味
1	d	double	双曲線正接を求める浮動小数点数

リターン値

型 : **double**

正常 : **d** の双曲線正接値

異常 : -

7.7.11 exp 関数

機能

浮動小数点数の指数関数を計算します。

呼び出し手順

```
#include <math.h>
```

```
double d, ret;
```

```
ret=exp(d);
```

パラメタ

No.	名前	型	意味
1	d	double	指数関数を求める浮動小数点数

リターン値

型 : **double**

正常 : **d** の指数関数値

異常 : -

7.7.12 frexp 関数

機能

浮動小数点数を [0.5, 1.0) の値と 2 のべき乗の積に分解します。

呼び出し手順

```
#include <math.h>

double ret, value;
int *e;

ret=frexp(value, e);
```

パラメタ

No.	名前	型	意味
1	value	double	[0.5, 1.0) の値と 2 のべき乗の積に分解する浮動小数点数
2	e	int 型を指すポインタ	2 のべき乗値を格納する記憶域へのポインタ

リターン値

型 : double

正常 : value が 0.0 の時 : 0.0

value が 0.0 でない時 : $ret \cdot 2^e$ の指している領域の値 = value で定義される ret の値

異常 : -

frexp 関数は value を [0.5, 1.0) の値と 2 のべき乗の積に分解します。e の指す領域には、分解した結果の 2 のべき乗値を設定します。

リターン値 ret の値の範囲は [0.5, 1.0) または 0.0 になります。

value が 0.0 ならば、e の指す int 型の記憶域の内容と ret の値は 0.0 になります。

7.7.13 ldexp 関数

機能

浮動小数点数と 2 のべき乗の積を計算します。

呼び出し手順

```
#include <math.h>

double ret, e;
int f;

ret=ldexp(e, f);
```

パラメタ

No.	名前	型	意味
1	e	double	2 のべき乗値を求める浮動小数点数
2	f	int	2 のべき乗値

リターン値

型 : **double**

正常 : $e \cdot 2^f$ の演算結果の値

異常 : -

7.7.14 log 関数

機能

浮動小数点数の自然対数を計算します。

呼び出し手順

```
#include <math.h>

double d, ret;

ret=log(d);
```

パラメタ

No.	名前	型	意味
1	d	double	自然対数を求める浮動小数点数

リターン値

型 : **double**

正常 : **d** の自然対数の値

異常 : 定義域エラーの時 : 結果はコンパイラによって異なります。

【エラー条件】

d の値が負の時、定義域エラーになります。

d の値が **0.0** の時、範囲エラーになります。

7.7.15 log10 関数

機能

浮動小数点数の **10** を底とする対数を計算します。

呼び出し手順

```
#include <math.h>

double d, ret;

ret=log10(d);
```

パラメタ

No.	名前	型	意味
1	d	double	10 を底とする対数を求める浮動小数点

リターン値

型 : **double**

正常: **d** は **10** を底とする対数値

異常: 定義域エラーの時: 結果はコンパイラによって異なります。

【エラー条件】

d の値が負の値の時、定義域エラーになります。

d の値が **0.0** の時、範囲エラーになります。

7.7.16 modf 関数

機能

浮動小数点数を整数部分と小数部分に分解します。

呼び出し手順

```
#include <math.h>

double a, *b, ret;

ret=modf(a, b);
```

パラメタ

No.	名前	型	意味
1	a	double	整数部分と小数部分に分解する浮動小数点数
2	b	double 型を指すポインタ	整数部分を格納する記憶域を指すポインタ

リターン値

型 : **double**

正常 : **a** の小数部分

異常 : -

7.7.17 pow 関数

機能

浮動小数点数のべき乗を計算します。

呼び出し手順

```
#include <math.h>
```

```
double x, y, ret;
```

```
ret=pow(x, y);
```

パラメタ

No.	名前	型	意味
1	x	double	べき乗される値
2	y	double	べき乗する値

リターン値

型 : **double**

正常 : **x** の **y** 乗の値

異常 : 定義域エラーの時 : 結果はコンパイラによって異なります。

【エラー条件】

x の値が **0.0** で、かつ **y** の値が **0.0** 以下の時、あるいは **x** の値が負で **y** の値が整数値でない時、定義域エラーになります。

7.7.18 sqrt 関数

機能

浮動小数点数の正の平方根を計算します。

呼び出し手順

```
#include <math.h>

double d, ret;

ret=sqrt(d);
```

パラメタ

No.	名前	型	意味
1	d	double	正の平方根を求める浮動小数点数

リターン値

型 : **double**

正常: **d** の正の平方根の値

異常: 定義域エラーの時: 結果はコンパイラによって異なります。

【エラー条件】

d の値が負の値の時、定義域エラーになります。

7.7.19 ceil 関数

機能

浮動小数点数の小数点以下を切り上げた整数値を求めます。

呼び出し手順

```
#include <math.h>
```

```
double d, ret;
```

```
ret=ceil(d);
```

パラメタ

No.	名前	型	意味
1	d	double	小数点以下を切り上げる浮動小数点数

リターン値

型 : **double**

正常 : **d** の小数点以下を切り上げた整数値

異常 : -

ceil 関数は **d** の値より大きいまたは等しい最小の整数値を **double** 型の値として返す関数です。したがって **d** の値が負の値の時は小数点以下を切り捨てた時の値を返します。

7.7.20 fabs 関数

機能

浮動小数点数の絶対値を計算します。

呼び出し手順

```
#include <math.h>
```

```
double d, ret;
```

```
ret=fabs(d);
```

パラメタ

No.	名前	型	意味
1	d	double	絶対値を求める浮動小数点数

リターン値

型 : **double**

正常 : **d** の絶対値

異常 : -

7.7.21 floor 関数

機能

浮動小数点数の小数点以下を切り捨てた整数値を求めます。

呼び出し手順

```
#include <math.h>
```

```
double d, ret;
```

```
ret=floor(d);
```

パラメタ

No.	名前	型	意味
1	d	double	小数点以下を切り捨てる浮動小数点数

リターン値

型 : **double**

正常 : **d** の小数点以下を切り捨てた整数値

異常 : -

floor 関数は **d** の値を超えない範囲の整数の最大値を、**double** 型の値として返す関数です。したがって **d** の値が負の値の時は小数点以下を切り上げた時の値を返します。

7.7.22 fmod 関数

機能

浮動小数点数どうしを除算した結果の余りを計算します。

呼び出し手順

```
#include <math.h>

double x, y, ret;

ret=fmod(x, y);
```

パラメタ

No.	名前	型	意味
1	x	double	被除数
2	y	double	除数

リターン値

型 : **double**

正常 : y の値が **0.0** の時 : x

y の値が **0.0** でない時 : x を y で除算した結果の余り

異常 : -

fmod 関数では、パラメタ x, y、リターン値 **ret** の間には、次に示す関係が成立します。

$x=y*i+ret$ (ただし i は整数値)

また、リターン値 **ret** の符号は x の符号と同じ符号になります。

x/y の商を表現できない場合、結果の値は、保証されません。

7.8 < setjmp.h >

機能概要

関数間の制御の移動をサポートします。

定義名一覧

定義名	種類	説 明
jmp_buf	マクロ名	関数間の制御の移動を可能とする情報を保存しておくための記憶域に対応する型名を示しています。
setjmp	関数	現在実行中の関数の jmp_buf で定義した実行環境を指定した記憶域に退避します。
longjmp	関数	setjmp 関数で退避していた関数の実行環境を回復し、setjmp 関数を呼び出したプログラムの位置に制御を移動します。

上記のマクロ名は、処理系定義です。

setjmp 関数は現在の関数の実行環境を退避します。その後 **longjmp** 関数を呼び出すことにより、**setjmp** 関数を呼び出したプログラム上の位置にもどることができます。以下に **setjmp**、**longjmp** 関数を使用して関数間の制御の移動をサポートした例を示します。

例

```

1  #include <stdio.h>
2  #include <setjmp.h>
3  jmp_buf env;
4  main( )
5  {
6
7
8      if (setjmp(env)!=0){
9          printf("return from longjmp¥n");
10         exit(0);
11     }
12     sub( );
13 }
14
15 sub( )
16 {
17     printf("subroutine is running¥n");
18     longjmp(env, 1);
19 }
```

【説明】

8 行目で `setjmp` 関数を呼んでいます。この時、`setjmp` 関数の呼び出された環境を、`jmp_buf` 型の変数 `env` に退避します。この時のリターン値は `0` なので、次に関数 `sub` が呼び出されます。

関数 `sub` の中で呼び出される `longjmp` 関数により、変数 `env` に退避した環境を回復します。その結果、プログラムは、あたかも 8 行目の `setjmp` 関数からリターンしたかのようふるまいます。ただし、この時のリターン値は `longjmp` 関数の第 2 パラメタで指定した値 (`1`) になります。その結果、9 行目以降が実行されます。

7.8.1 setjmp 関数

機能

現在実行中の関数の実行環境を、指定した記憶域に退避します。

呼び出し手順

```
#include <setjmp.h>

int ret;
jmp_buf env;

ret=setjmp(env);
```

パラメタ

No.	名前	型	意味
1	env	jmp_buf	実行環境を退避する記憶域へのポインタ

リターン値

型 : int

正常 : setjmp 関数を呼び出した時 : 0

longjmp 関数からのリターン時 : 0 以外

異常 : -

setjmp 関数により退避された実行環境は、longjmp 関数において使用されます。setjmp 関数として呼び出された時のリターン値は 0 ですが、longjmp 関数からリターンしてきた時のリターン値は、longjmp 関数で指定した第 2 パラメタの値となります。

【注意】

setjmp 関数を複雑な式から呼び出す場合、式の評価の途中結果等の現在の実行環境の一部が失われる可能性があります。setjmp 関数は setjmp 関数の結果と定数式の比較という形態だけで使用し、複雑な式の中では呼び出さないようにしてください。

7.8.2 longjmp 関数

機能

setjmp 関数で退避していた関数の実行環境を回復し、**setjmp** 関数を呼び出したプログラムの位置に制御を移動します。

呼び出し手順

```
#include <setjmp.h>

int ret;
jmp_buf env;

longjmp(env, ret);
```

パラメタ

No.	名前	型	意味
1	env	jmp_buf	実行環境を退避した記憶域へのポインタ
2	ret	int	setjmp 関数へのリターンコード

リターン値

型 : void

正常 : -

異常 : -

longjmp 関数は同じプログラム中で最後に呼び出された **setjmp** 関数によって退避された関数の実行環境を **env** で指定された記憶域から回復し、その **setjmp** 関数を呼び出したプログラムの位置に制御を移します。この時 **longjmp** 関数の **ret** が **setjmp** 関数のリターン値として返ります。ただし、**ret** が 0 の時は **setjmp** 関数へのリターン値としては 1 が返ります。

【注意】

setjmp 関数が呼び出されていない時、あるいは **setjmp** 関数を呼び出した関数がすでに **return** 文を実行している時は、**longjmp** 関数の動作は保証されません。

7.9 < signal.h >

機能概要

プログラム実行時に生じる割り込み等の条件を発生させたり、その条件が生じた時の処理の登録を行います。

定義名一覧

定義名	種類	説 明
SIGABRT	マクロ名	プログラムの続行が不可能なエラー条件に対応するシグナル番号を示します。
SIGFPE	マクロ名	ゼロによる除算あるいは結果としてオーバフローを導く操作のようなエラーのある算術操作に対応するシグナル番号を示します。
SIGILL	マクロ名	不正な関数のコードの検出に対応するシグナル番号を示します。
SIGINT	マクロ名	ユーザがプログラムに対して意識的に端末等から、発生させた割り込みのシグナルを受け取ったことを示すシグナル番号を示します。
SIGSEGV	マクロ名	不正なデータオブジェクトへのアクセスであることを示すシグナル番号を示します。
SIGTERM	マクロ名	プログラムに送られる終了要求を示すシグナル番号を示します。
SIG_IGN	マクロ名	シグナル発生時には、そのシグナルを無視するという処理に対応するマクロ名を示しています。
SIG_DFL	マクロ名	シグナル発生時には、処理系定義の既定の処理を行うという処理に対応するマクロ名を示しています。
SIG_ERR	マクロ名	signal 関数においてエラーが発生したことを示すためのマクロ名を示しています。
signal	関数	シグナルが発生した時、どのような処理を行うかを登録します。
raise	関数	実行中のプログラムにおいてシグナルを発生させます。

プログラム実行時に発生する割り込み等の条件をシグナルといいます。各シグナルの種類に対応し、以下のマクロ名が定義されています。

SIGABRT, SIGFPE, SIGILL, SIGINT, SIGSEGV, SIGTERM

これらに対しては、それぞれシグナル番号という整数が対応しています。上記のマクロ名はすべて処理系定義です。

割り込み発生時に起動される処理は、**signal** 関数で登録することができます。登録時に、**SIG_IGN**,**SIG_DFL** で定義されている値を設定することにより、標準的な処理を行うことを指定できます。

以下に **signal** 関数を使用したプログラムの例を示します。

例

```
1  #include <stdio.h>
2  #include <signal.h>
3
4  void interrupt (int sig)
5  {
6      printf("interrupt¥n");
7  }
8
9  main( )
10 {
11     sub1( );
12     signal(SIGINT, SIG_IGN);
13     printf("interrupt disabled¥n");
14     sub2( );
15     signal(SIGINT, interrupt);
16     sub3( );
17     raise(SIGINT);
18 }
```

【説明】

11 行目で呼び出す関数 **sub1** の中では、割り込み等の条件の発生に対して、システムで標準の処理が行われます。

12 行目で端末等からの割り込み (**SIGINT**) に対し、それを無視する処理 (**SIG_IGN**) を **signal** 関数で登録します。これにより、14 行目で呼び出す関数「**sub2**」の中では端末等からの割り込みは無視されます。

15 行目で、**SIGINT** に対して 4 行目で定義した関数 **interrupt** を登録します。これにより、16 行目で呼び出す関数 **sub3** の中では、端末等からの割り込みに対して関数 **interrupt** の処理が行われるようになります。

17 行目の **raise** 関数は、端末等からの割り込み条件を、プログラム内で発生させます。この結果、やはり関数 **interrupt** の処理が行われます。

7.9.1 signal 関数

機能

各種シグナル発生時の処理を登録します。

呼び出し手順

```
include <signal.h>

int sig;

void (*func)(int);

void (*ret)(int);

ret=signal(sig, func);
```

パラメタ

No.	名前	型	意味
1	sig	int	シグナル番号
2	func	void 型のリターン値を持つ関数へのポインタ	シグナル発生時に、実行する関数へのポインタ

リターン値

型 : void 型のリターン値を持つ関数へのポインタ

正常 : 以前に同じ sig に対して登録されていた関数へのポインタ

異常 : SIG_ERR の値

sig で指定されたシグナル番号に対して、func で指定した処理を登録します。func には、通常の間数を指定する以外に、SIG_DFL、SIG_IGN のマクロ名を指定することができます。

(1) func が SIG_DFL の時

指定されたシグナルが発生した時、処理系定義の既定の処理を実行します。

(2) func が SIG_IGN の時

指定されたシグナルの発生を無視します。

(3) func が通常の間数の時

指定されたシグナルが発生した時、まずそのシグナルに対する既定の処理を実行し、その後ここで登録した関数を実行します。

【注意】

`func` で指定した関数が `return` 文を実行した時、プログラムは、シグナルが発生した位置から実行を再開しますが、もし、シグナルが `SIGFPE` または処理系で定義している例外処理に対応する値の時は、その動作は保証されません。

【エラー条件】

シグナル発生時の処理を登録することができない時には、`errno` に `SIG_ERR` の値が設定されます。

7.9.2 raise 関数

機能

実行中のプログラムにおいてシグナルを発生させます。

呼び出し手順

```
#include <signal.h>

int sig;
int ret;

ret=signal(sig);
```

パラメタ

No.	名前	型	意味
1	sig	int	シグナル番号

リターン値

型 : int

正常 : 0

異常 : 0 以外の値

sig で指定されたシグナル番号に対するシグナルを実行中のプログラムにおいて発生させます。

7.10 <stdarg.h>

機能概要

可変個の引数を持つ関数に対し、その引数の参照を可能にします。

定義名一覧

定義名	種類	説 明
va_list	マクロ名	可変個の引数を参照するために、va_start, va_arg, va_end マクロで共通に使用される変数の型を示しています。
va_start	マクロ	可変個の引数の参照を行うため、初期設定処理を行います。
va_arg	マクロ	可変個の引数を持つ関数に対して、現在参照中引数の次の引数への参照を可能とします。
va_end	マクロ	可変個の引数を持つ関数の引数への参照を終了させます。

上記のマクロ名はすべて処理系定義です。

本標準インクルードファイルで定義しているマクロを使用したプログラムの例を以下に示します。

例

```
1  #include <stdio.h>
2  #include <stdarg.h>
3
4  extern void prlist(int count, ...);
5
6  main( )
7  {
8      prlist(1, 1);
9      prlist(3, 4, 5, 6);
10     prlist(5, 1, 2, 3, 4, 5);
11 }
12
13 void prlist(int count, ...)
14 {
15     va_list ap;
16     int i;
17
18     va_start(ap, count);
19     for(i=0; i<count; i++)
20         printf("%d", va_arg(ap, int));
21     putchar('\n');
22     va_end(ap);
23 }
```

【説明】

この例では、第 1 引数に出力するデータの数を指定し、以下の引数をその数だけ出力する関数 `prlist` を実現しています。

18 行目で、可変個の引数への参照を `va_start` で初期化します。その後引数を一つ出力するたびに、`va_arg` マクロによって次の引数を参照します（20 行目）。`va_arg` マクロでは、引数の型名（この場合は `int` 型）を第 2 引数に指定します。

引数の参照が終了したら、`va_end` マクロを呼び出します（22 行目）。

7.10.1 va_start マクロ

機能

可変個のパラメタへの参照を行うため、初期設定処理を行います。

呼び出し手順

```
#include <stdarg.h>

va_list ap;

        va_start(ap, parmN);
```

パラメタ

No.	名前	型	意味
1	ap	va_list	可変個のパラメタにアクセスするための変数
2	parmN	parmN の型	最右端の引数の識別子

リターン値

型 : **void**

正常: -

異常: -

va_start マクロは、**va_arg**, **va_end** マクロによって使用される **ap** の初期化を行います。

また、**parmN** には、外部関数定義におけるパラメタの並びの最右端のパラメタの識別子、すなわち「,...」の直前の識別子を指定します。

【注意】

可変個の名前のない引数を参照するためには、**va_start** マクロ呼び出しを一番初めに実行する必要があります。

7.10.2 va_arg マクロ

機能

可変個のパラメタを持つ関数に対して、現在参照中のパラメタの次のパラメタへの参照を可能とします。

呼び出し手順

```
#include <stdarg.h>

va_list ap;

type      ret;

      ret=va_arg(ap, type);
```

パラメタ

No.	名前	型	意味
1	ap	va_list	可変個のパラメタにアクセスするための変数
2	type	型名	アクセスするパラメタの型

リターン値

型 : パラメタの第 2 引数で指定した型 **type**

正常 : パラメタの値

異常 : -

va_start マクロで初期化した **va_list** 型の変数を第 1 パラメタに指定します。**ap** の値は **va_arg** を使用する度に更新され、結果として可変個のパラメタが順次本マクロのリターン値として返されます。

呼び出し手順の **type** のところには、参照する引数の型を指定してください。

【注意】

ap は **va_start** によって初期化された **ap** と同じでなければなりません。

type の型が **char** 型、**unsigned char** 型、**short** 型、**unsigned short** 型、**float** 型を関数の引数に指定した時に型変換によってサイズが変わる型を指定した場合、正しくパラメタを参照することができなくなります。このような型を指定すると動作は保証されません。

7.10.3 va_end マクロ

機能

可変個の引数を持つ関数の引数への参照を終了させます。

呼び出し手順

```
#include <stdarg.h>

va_list ap;

...

va_end(ap);
```

パラメタ

No.	名前	型	意味
1	ap	va_list	可変個の引数を参照するための変数

リターン値

型 : -
正常 : -
異常 : -

【注意】

ap は va_start によって初期化された ap と同じでなければなりません。また、関数からの return 前に va_end マクロが呼び出されない時は、その関数の動作は保証されません。

7.11 <stdio.h>

機能概要

ストリーム入出力用ファイルの入出力に関する処理を行います。

定義名一覧

定義名	種類	説 明
FILE	マクロ名	ストリーム入出力処理で必要とするバッファへのポインタやエラー指示子、終了指示子などの各種制御情報を保存しておく構造体の型を示しています。
_IOBF	マクロ名	バッファ領域の使用方法として、入出力処理はすべてバッファ領域を使用することを示しています。
_IOLBF	マクロ名	バッファ領域の使用方法として、入出力処理は行単位でバッファ領域を使用することを示しています。
_IONBF	マクロ名	バッファ領域の使用方法として、入出力処理はバッファ領域を使用しないことを示しています。
BUFSIZ	マクロ名	入出力処理において必要とするバッファの大きさを示しています。
EOF	マクロ名	ファイルの終わり (end of file) すなわちファイルからそれ以上の入力が無いことを示しています。
L_tmpnam	マクロ名	tmpnam 関数によって生成される一時ファイル名の文字列を格納するのに十分な大きさの配列のサイズを示しています。
SEEK_CUR	マクロ名	ファイルの現在の読み書き位置を現在の位置からのオフセットに移すことを示しています。
SEEK_END	マクロ名	ファイルの現在の読み書き位置をファイルの終了位置からのオフセットに移すことを示しています。
SEEK_SET	マクロ名	ファイルの現在の読み書き位置をファイルの先頭位置からのオフセットに移すことを示しています。
SYS_OPEN	マクロ名	処理系が同時にオープンすることができることを保証するファイルの数を示しています。
TMP_MAX	マクロ名	tmpnam 関数によって生成される一意なファイル名の個数の最小値を示します。
stderr	マクロ名	標準エラーファイルに対するファイルポインタを示します。
stdin	マクロ名	標準入力ファイルに対するファイルポインタを示します。
stdout	マクロ名	標準出力ファイルに対するファイルポインタを示します。

定義名	種類	説 明
remove	関数	指定されたファイルを削除します。
rename	関数	指定されたファイルの名前を新たに指定したファイル名に変更します。
tmpfile	関数	バイナリー時ファイルを生成します。
tmpnam	関数	呼ばれる度に異なったファイル名を生成します。
fclose	関数	ストリーム入出力用ファイルをクローズします。
fflush	関数	ストリーム入出力用ファイルのバッファの内容をファイルへ出力します。
fopen	関数	ストリーム入出力用ファイルを指定したファイル名によってオープンします。
freopen	関数	現在オープンされているストリーム入力出力用ファイルをクローズし、新しいファイルを指定したファイル名で再オープンします。
setbuf	関数	ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。
setvbuf	関数	ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。
fprintf	関数	書式に従ってストリーム入出力用ファイルへデータを出力します。
fscanf	関数	ストリーム入出力用ファイルからデータを入力し、書式に従って変換します。
printf	関数	データを書式に従って変換し、標準出力ファイル (stdout) へ出力します。
scanf	関数	標準入力ファイル (stdin) からデータを入力し、書式に従って変換します。
sprintf	関数	データを書式に従って変換し、指定した領域へ出力します。
sscanf	関数	指定した記憶域からデータを入力し、書式に従って変換します。
vfprintf	関数	可変個のパラメタリストを書式に従って指定したストリーム入出力用ファイルに出力します。
vprintf	関数	可変個のパラメタリストを書式に従って標準出力ファイルに出力します。
vsprintf	関数	可変個のパラメタリストを書式に従って指定した記憶域に出力します。
fgetc	関数	ストリーム入出力用ファイルから 1 文字入力します。
fgets	関数	ストリーム入出力用ファイルから文字列を入力します。
fputc	関数	ストリーム入出力用ファイルへ 1 文字出力します。

定義名	種類	説明
fputs	関数	ストリーム入出力用ファイルへ文字列を出力します。
getc	関数	ストリーム入出力用ファイルから 1 文字入力します。
getchar	関数	標準入力ファイルから 1 文字入力します。
gets	関数	標準入力ファイルから文字列を入力します。
putc	関数	ストリーム入出力用ファイルへ 1 文字出力します。
putchar	関数	標準出力ファイルへ 1 文字出力します。
puts	関数	標準出力ファイルへ文字列を出力します。
ungetc	関数	ストリーム入出力用ファイルへ 1 文字をもどします。
fread	関数	ストリーム入出力用ファイルから指定した記憶域にデータを入力します。
fwrite	関数	記憶域からストリーム入出力用ファイルにデータを出力します。
fseek	関数	ストリーム入出力用ファイルの現在の読み書き位置を移動させます。
ftell	関数	ストリーム入出力用ファイルの現在の読み書き位置を求めます。
rewind	関数	ストリーム入出力用ファイルの現在の読み書き位置をファイルの先頭に移動します。
clearerr	関数	ストリーム入出力用ファイルのエラー状態をクリアします。
feof	関数	ストリーム入出力用ファイルが終わりであるかどうかを判定します。
ferror	関数	ストリーム入出力用ファイルがエラー状態であるかどうかを判定します。
perror	関数	標準エラーファイル (stderr) に、エラー番号に対応したエラーメッセージを出力します。

上記のマクロ名は全て処理系定義です。

ストリーム入出力用ファイルに対する一連の入出力処理を行ったプログラムの例を以下に示します。

例

```
1  #include <stdio.h>
2
3  main( )
4  {
5      int c;
6      FILE *ifp, *ofp;
7
8      if ((ifp=fopen("INPUT.DAT ", "r"))==NULL){
9          fprintf(stderr, "cannot open input file¥n");
10         exit(1);
11     }
12     if ((ofp=fopen("OUTPUT.DAT ", "w"))==NULL){
13         fprintf(stderr, "cannot open output file¥n");
14         exit(1);
15     }
16     while ((c=getc(ifp))!=EOF)
17         putc(c, ofp);
18     fclose(ifp);
19     fclose(ofp);
20 }
```

【説明】

ファイル **INPUT.DAT** の内容をファイル **OUTPUT.DAT** へコピーするプログラムです。

8 行目の **fopen** 関数で入力ファイル **INPUT.DAT** を、12 行目の **fopen** 関数で出力ファイル **OUTPUT.DAT** をオープンします。オープンに失敗した場合、**fopen** 関数のリターン値として **NULL** が返されますので、エラーメッセージを出力してプログラムを終了させます。

fopen 関数が正常に終了した時、オープンしたファイルの情報を格納するデータ (**FILE** 型) へのポインタが返されますので、これらを変数 **ifp**、**ofp** に設定します。

オープンが成功した後は、これらの **FILE** 型のデータを用いて入出力を行います。

ファイルの処理が終了したら、**fclose** 関数でファイルをクローズします。

7.11.1 remove 関数

機能

指定されたファイルを削除します。

呼び出し手順

```
#include <stdio.h>

const char *pathname;

int ret;

ret=remove(pathname);
```

パラメタ

No.	名前	型	意味
1	pathname	const char 型を指すポインタ	削除するファイル名を指すポインタ

リターン値

型 : int

正常 : 0

異常 : 0 以外

pathname によって指されるファイル名のファイルを削除します。

【注意】

オープンされているファイルに対して、**remove** 関数を実行した時、その動作はコンパイラによって異なります。

7.11.2 rename 関数

機能

指定されたファイルの名前を新たに指定したファイル名に変更します。

呼び出し手順

```
#include <stdio.h>

const char *old, *new;

int ret;


ret=rename(old, new);
```

パラメタ

No.	名前	型	意味
1	old	const char 型を指すポインタ	旧ファイル名を指すポインタ
2	new	const char 型を指すポインタ	新ファイル名を指すポインタ

リターン値

型 : int

正常 : 0

異常 : 0 以外

old で指されるファイル名を持つファイルをこの後 new によって指されるファイル名によって識別できるようにします。旧ファイル名は削除されます。

もし、この操作が失敗した時は、ファイルのファイル名はもとのままです。

7.11.3 tmpfile 関数

機能

バイナリー時ファイルを生成します。

呼び出し手順

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

```
FILE *ret;
```

```
ret=tmpfile( );
```

パラメタ

No.	名前	型	意味
-	-	-	-

リターン値

型 : **FILE** 型へのポインタ

正常 : 生成されたファイルに対するファイルポインタ

異常 : **NULL**

tmpfile 関数で生成したファイルは、クローズされる時、あるいはプログラムが終了する時に自動的に削除されます。

また、このファイルは更新用として使用することができます。

7.11.4 tmpnam 関数

機能

呼ばれるたびに異なったファイル名を生成します。

呼び出し手順

```
#include <stdio.h>

char *s, *ret;

ret=tmpnam(s)
```

パラメタ

No.	名前	型	意味
1	s	char 型を指すポインタ	生成したファイル名を格納するための記憶域へのポインタ

リターン値

型 : char 型へのポインタ

正常 : ファイル名へのポインタ

異常 : -

tmpnam 関数は、呼ばれるたびに、最低 **TMP_MAX** 個までの異なったファイル名を生成します。

s が **NULL** の時は、**tmpnam** 関数はファイル名を格納する記憶域を割り付けてそこにファイル名を生成し、その先頭アドレスを返します。

s が **NULL** でない時は、s は **L_tmpnam** バイトのサイズを持つ記憶域を指していると仮定され、その記憶域にファイル名を書き込み、リターン値として s の値をそのまま返します。

【注意】

ファイル名の領域として、最低 **L_tmpnam** バイトの記憶域を用意してください。また、**tmpnam** 関数が **TMP_MAX** 回をこえて呼び出された時の動作は処理系定義です。

7.11.5 fclose 関数

機能

ストリーム入出力用ファイルをクローズします。

呼び出し手順

```
#include <stdio.h>

FILE *fp;

int ret;

ret=fclose(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : int

正常 : 0

異常 : 0 以外

fclose 関数はファイルポインタ **fp** の示すストリーム入出力用ファイルをクローズします。

fclose 関数は、ストリーム入出力用ファイルの出力ファイルがオープンされており、まだ出力されていないデータがバッファに残っている時は、それをファイルに出力してからクローズします。

また、入出力用のバッファがシステムによって自動的に割り付けられていた場合は、それを解放します。

7.11.6 fflush 関数

機能

ストリーム入出力用ファイルのバッファの内容をファイルへ出力します。

呼び出し手順

```
#include <stdio.h>

FILE *fp;
int ret;

ret=fflush(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : int

正常 : 0

異常 : 0 以外

fflush 関数はストリーム入出力用ファイルの出力ファイルがオープンされている時、ファイルポインタ **fp** で指定されたストリーム入出力用ファイルのバッファの未出力内容をファイルに出力します。また、入力ファイルがオープンされている時、**ungetc** 関数の指定を無効にします。

7.11.7 fopen 関数

機能

ストリーム入出力用ファイルを、指定したファイル名によってオープンします。

呼び出し手順

```
#include <stdio.h>

FILE *ret;

const char *fname, *mode;

ret=fopen(fname, mode);
```

パラメタ

No.	名前	型	意味
1	fname	const char 型を指すポインタ	ファイル名を示す文字列へのポインタ
2	mode	const char 型を指すポインタ	ファイルアクセスモードを示す文字列へのポインタ

リターン値

型 : **FILE** 型へのポインタ

正常: オープンしたファイルのファイル情報を指すファイルポインタ

異常: **NULL**

fopen 関数は **fname** が指す文字列をファイル名とするストリーム入出力用ファイルをオープンします。書き出しモードあるいは追加モードで存在しないファイルをオープンしようとした時は、可能な限り新しいファイルを作成します。また既存のファイルに対して書き出しモードでオープンした時は、ファイルの先頭から書き込みが行われ、以前に書き込まれていたファイルの内容は消去されます。

追加モードでオープンしたファイルは、そのファイルの終わりの位置から書き出しの処理が行われます。更新モードでオープンしたファイルは、このファイルに対して入力と出力の両方の処理を行うことができます。

ただし、出力処理は後に **fflush** , **fseek** , **rewind** 関数が実行されることなしに入力処理を続けることはできません。

また同様に入力処理の後に **fflush** , **fseek** , **rewind** 関数が実行されることなしに出力処理を続けることはできません。

また、ファイルアクセスモードを示す文字列の後にオープンの方法を指示する文字が付くこともあります。

7.11.8 freopen 関数

機能

現在オープンされているストリーム入出力用ファイルをクローズし、新しいファイルを指定したファイル名で再オープンします。

呼び出し手順

```
#include <stdio.h>

const char *fname, *mode;
FILE *ret, *fp;

ret=freopen(fname, mode, fp);
```

パラメタ

No.	名前	型	意味
1	fname	const char 型を指すポインタ	新しいファイル名を示す文字列へのポインタ
2	mode	const char 型を指すポインタ	ファイルアクセスモードを示す文字列へのポインタ
3	fp	FILE 型を指すポインタ	現在オープンされているストリーム入出力用ファイルのファイルポインタ

リターン値

型 : FILE 型へのポインタ

正常 : fp

異常 : NULL

freopen 関数は、まず、ファイルポインタ **fp** の示すストリーム入出力用ファイルをクローズします。（このクローズ処理が正しく行われない時でも以下の処理は続けます。）次に、その **fp** の指す **FILE** 構造体を再使用して、ファイル名 **fname** で示すファイルを、ストリーム入出力用にオープンします。

freopen 関数は一時にオープンするファイル数の限られているときなどに有効です。

freopen 関数は通常、**fp** と同じ値を返しますが、エラーが発生した時は、**NULL** を返します。

7.11.9 setbuf 関数

機能

ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。

呼び出し手順

```
#include <stdio.h>

FILE *fp;

char buf [BUFSIZ];

setbuf(fp, buf);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ
2	buf	char 型の配列を指すポインタ	バッファ領域へのポインタ

リターン値

型 : **void**

正常: -

異常: -

setbuf 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルに対して **buf** の指す記憶域を入出力用のバッファ領域として使用するよう定義します。この結果、大きさが **BUFSIZ** のバッファ領域を使用した入出力処理が行われます。

7.11.10 setvbuf 関数

機能

ストリーム入出力用のバッファ領域をユーザプログラムの側で定義して設定します。

呼び出し手順

```
#include <stdio.h>

FILE *fp;
char *buf;
int type, ret;
size_t size;

ret=setvbuf(fp, buf, type, size);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ
2	buf	char 型を指すポインタ	バッファ領域へのポインタ
3	type	int	バッファの管理方式
4	size	size_t	バッファ領域の大きさ

リターン値

型 : int

正常 : 0

異常 : 0 以外

setvbuf 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルに対して **buf** の指す記憶域を入出力用のバッファ領域として使用するよう定義します。

このバッファ領域の使用方法としては、以下の三通りの方法があります。

(1) type に **_IOFBF** を指定した時

入出力処理はすべてバッファ領域を使用して行います。

(2) type に **_IOLBF** を指定した時

入出力処理は行単位でバッファ領域を使用して行います。すなわち、入出力データは、改行文字が書かれた時、バッファ領域が一杯になった時、入力が必要された時にバッファ領域から取り出されることになります。

(3) `type` に `_IONBF` を指定した時

入出力処理はバッファ領域を使用せず行います。

`setvbuf` 関数は通常 0 を返しますが、`type` あるいは `size` に不正な値が与えられた時、あるいはバッファ領域の使用方法等の要求が受け入れられなかった時には 0 以外の値を返します。

【注意】

バッファ領域は、オープンされているストリーム入出力用ファイルがクローズされる前に解放してはいけません。また、`setvbuf` 関数は、ストリーム入出力用ファイルがオープンされてから入出力用処理が行われるまでの間で使用してください。

7.11.11 fprintf 関数

機能

書式に従って、ストリーム入出力用ファイルヘデータを出力します。

呼び出し手順

```
#include <stdio.h>

FILE *fp;

const char *control;

int ret;

ret=fprintf(fp, control [ ,arg] ...);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ
2	control	const char 型を指すポインタ	書式を示す文字列へのポインタ
3	arg, ...	特に規定せず	書式に従って出力されるデータの並び

リターン値

型 : int

正常 : 変換し出力した文字数

異常 : 負の値

fprintf 関数は、**control** が指す書式を示す文字列に従って、引数 **arg** を変換、編集し、ファイルポインタ **fp** の示すストリーム入出力用ファイルへ出力します。

fprintf 関数は、通常は変換し出力したデータの個数を返しますが、エラー発生時には負の値を返します。

書式の仕様は以下のとおりです。

(1) 書式の概要

書式を表わす文字列は、2 種類の文字列から構成されます。

(a) 通常の文字

(b) に示す % で始まる指定以外の文字はそのまま出力されます。

(b) 変換仕様

変換仕様は、%で始まる文字列で、後に続く引数の変換方法を指定します。変換仕様の形式は次の規則に従います。

$$\% [\text{フラグ}\dots] \left\{ \begin{array}{l} [\text{※}] \\ [\text{フィールド幅}] \end{array} \right\} \left[\begin{array}{l} \text{.} \\ \left\{ \begin{array}{l} [\text{※}] \\ [\text{精度}] \end{array} \right\} \end{array} \right] [\text{パラメタのサイズ指定}] \text{変換文字}$$

この変換仕様に対して、実際に出力するパラメタが無い時は、その動作は保証されません。また、変換仕様よりも実際に出力するパラメタの個数が多い時は、余分なパラメタはすべて無視されます。

(2) 変換仕様の説明

(a) フラグ

符号を付けるなどの出力するデータに対する修飾を指定します。指定できるフラグの種類と意味を表 7-5 に示します。

表 7-5 フラグの種類と意味

項番	種類	意 味
1	-	変換したデータの文字数が指定したフィールド幅より少ない時、そのデータをフィールド内で左詰めにして出力します。
2	+	符号付きのデータに変換する時、そのデータの符号に従って、変換したデータの先頭にプラスあるいはマイナス符号を付けます。
3	空白	符号付きのデータの変換において、変換したデータの先頭に符号が付かない時、そのデータの先頭に空白を付けます。 「+」と共に使用した時、本フラグは無視されます。
4	#	表 7-7 で説明する変換の種類に従って、変換後のデータに修飾を行います。 1. c, d, i, s, u 変換の時 本フラグは無視されます。 2. o 変換の時 変換したデータの先頭に 0 を付けます。 3. x (あるいは X) 変換の時 変換したデータの先頭に 0x (あるいは 0X) を付けます。 4. e, E, f, g, G 変換の時 変換したデータに小数点以下がない時でも、小数点を出力します。 また、g, G 変換の時は、変換後のデータの後に付く 0 は取り除きません。

(b) フィールド幅

変換したデータを出力する文字数を任意の 10 進数で指定します。

変換したデータの文字数がフィールド幅より少ない時、フィールド幅までそのデータの先頭に空白が付けられます。(ただし、フラグとして ' - ' を指定した時は、データの後に空白が付けられます。)

もし、変換したデータの文字数がフィールド幅より大きい時は、フィールド幅は、変換結果を出力できる幅に拡張されます。

また、フィールド幅指定の先頭が 0 で始まっている時は、出力するデータの先頭には空白ではなく文字「0」が付けられます。

(c) 精度

表 7-7 で説明する変換の種類に従って変換したデータの精度を指定します。

精度は、ピリオド (.) の後に 10 進整数を続ける形式で指定します。10 進整数を省略した時は、0 を指定したものと仮定します。

精度を指定した結果、フィールド幅の指定との間に矛盾が生じれば、フィールド幅の指定を無効とします。

各変換の種類と精度指定の意味を以下に示します。

- **d , i , o , u , x , X** 変換の時

変換したデータの最小の桁数を示します。

- **e , E , f** 変換の時

変換したデータの小数点以下の桁数を示します。

- **g , G** 変換の時

変換したデータの最大有効桁数を示します。

- **s** 変換の時

印字される最大文字数を示します。

(d) パラメタのサイズ指定

d , i , o , u , x , X , e , E , f , g , G 変換の時 (表 7-7 参照)

変換するデータのサイズ (short 型、long 型、long double 型) を指定します。これ以外の変換の時は、本指定を無視します。表 7-6 にサイズ指定の種類とその意味を示します。

表 7-6 パラメタのサイズ指定の種類とその意味

項番	種類	意 味
1	h	d , i , o , u , x , X 変換において、変換するデータが short 型あるいは unsigned short 型であることを指定します。
2	l	d , i , o , u , x , X 変換において、変換するデータが long 型、unsigned long 型あるいは、double 型であることを指定します。
3	L	e , E , f , g , G 変換において、変換するデータが long double 型であることを指定します。

(e) 変換文字

変換するデータをどのような形式に変換するかを指定します。

もし、変換するデータが構造体や配列型の時や、それらの型を指すポインタの時は、s 変換で文字の配列を変換する時、p 変換でポインタを変換する時を除いてその動作は保証されません。

表 7-7 に変換文字と変換方式を示します。この表に述べられていない英小文字を変換文字として指定した時は、その動作は保証されません。また、それ以外の文字を指定した時の動作はコンパイラによって異なります。

表 7-7 変換文字と変換の方式

項 番	変換 文字	変換の 種 類	変換の方式	変換の対象とす るデータの型	精度に対する注意事項
1	d	d 変換	int 型データを符号付き 10 進数の文字列に変換しま	int 型	精度指定は、最低で何文字出力されるかを示しています。もし、変換後の文字数が精度の値より少ない時は、文字列の先頭に 0 が付きます。また、精度を省略した時は、1 が仮定されます。さらに、0 の値を持つデータを精度に 0 を指定して変換し出力しようとした時は、何も出力されません。
2	i	i 変換	す。d 変換と i 変換は同じ仕様です。	int 型	
3	o	o 変換	int 型データを符号無しの 8 進数の文字列に変換します。	int 型	
4	u	u 変換	int 型データを符号無しの 10 進数の文字列に変換します。	int 型	
5	x	x 変換	int 型データを符号無しの 16 進数に変換します。16 進文字には a , b , c , d , e , f を用います。	int 型	
6	X	X 変換	int 型データを符号無しの 16 進数に変換します。16 進文字には A , B , C , D , E , F を用います。	int 型	精度の指定は、小数点以降の桁数を表わします。小数点以降の文字が存在する時には、必ず小数点の前に 1 桁の数字が出力されます。精度を省略した時は、6 が仮定されます。また、精度に 0 を指定した時は、小数点と小数点以降の文字は出力しません。出力するデータは丸められます。
7	f	f 変換	double 型データを [-] ddd.ddd の形式の 10 進数の文字列に変換します。	double 型	

項 番	変換 文字	変換の 種 類	変換の方式	変換の対象とす るデータの型	精度に対する注意事項
8	e	e 変換	double 型データを「[-]d.ddde ± dd」の形式の 10 進数の文字列に変換します。指数は、少なくとも 2 桁出力されます。	double 型	精度の指定は、小数点以降の桁数を表わします。変換した文字は小数点の前に 1 桁の数字が出力され、小数点以降に精度に等しい桁数の数字が出力される形式となります。精度を省略した時は 6 が
	E	E 変換	double 型データを「[-]d.dddE ± dd」の形式の 10 進数の文字列に変換します。指数は、少なくとも 2 桁出力されます。	double 型	仮定されます。また、精度に 0 を指定した時は、小数点以降の文字は出力しません。出力するデータは丸められます。
9	g	g 変換 (ある いは G 変換)	変換する値と有効桁数を指定する精度の値から f 変換の形式で出力するか e 変換 (あるいは E 変換) の形式で出力するかを決め double 型データを	double 型	精度の指定は、変換されたデータの最大有効桁数を示します。
10	G		出力します。もし、変換されたデータの指数が - 4 より小さいか、有効桁数を指定する精度より大きい時には e 変換 (あるいは E 変換) の形式に変換します。	double 型	
11	c	c 変換	int 型のデータを unsigned char 型データとし、そのデータに対応する文字に変換します。	int 型	精度の指定は無効です。

項 番	変換 文字	変換の 種 類	変換の方式	変換の対象とす るデータの型	精度に対する注意事項
12	s	s 変換	char 型へのポインタ型 データが指す文字列を文 字列の終了を示すヌル文 字まで、あるいは、精度 で指定された文字数分 出力します。（ただしヌル 文字は出力されません。 また、空白、水平タブ、 改行文字は変換文字列に ふくまれません。）	char 型へのポイ ンタ型	精度の指定は出力する文字数を示 します。もし、精度が省略された 時は、データが指す文字列のヌル 文字までの文字が出力されます。 （ただし、ヌル文字は出力されま せん。また、空白、水平タブ、改 行文字は変換文字列にふくまれま せん。）
13	p	p 変換	データをポインタとし て、コンパイラごとに定 義された印字可能な文字 列に変換します。	void 型へのポイ ンタ	精度の指定は無効です。
14	n	データ の変換 は生じ ませ ん。	データは int 型へのポイン タ型とみなされ、このデー タが指す記憶域にいまま で、出力したデータの文 字数を設定します。	int 型へのポイン タ型	
15	%	この変 換では データ の変換 は生じ ませ ん。	%を出力します。	無し	

(f) フィールド幅あるいは精度に対する*指定

フィールド幅あるいは精度指定の値として*を指定することができます。この時は、この変換仕様に対応するパラメタの値がフィールド幅あるいは精度指定の値として使用されます。このパラメタが負のフィールド幅を持つ時は、正のフィールド幅にフラグの - が指定されたと解釈します。また、負の精度を持つ時は、精度が省略されたものと解釈します。

7.11.12 fscanf 関数

機能

ストリーム入出力用ファイルからデータを入力し、書式に従って変換します。

呼び出し手順

```
#include <stdio.h>

FILE *fp;

const char *control;

int ret;

ret=fscanf(fp, control [,ptr] ...);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ
2	control	const char 型を指すポインタ	書式を示す文字列へのポインタ
3	ptr	データ型を指すポインタ	入力したデータを格納する記憶域へのポインタ

リターン値

型 : int

正常 : 入力変換に成功したデータの個数

異常 : 入力データの変換を行う前に入力データが終了した時 : EOF

fscanf 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルからデータを入力し、**control** が指す書式を文字列に従って変換、編集して、その結果を **ptr** の指す記憶域へ格納します。

データを入力するための書式の仕様を以下に示します。

(1) 書式の概要

書式を表わす文字列は、以下の 3 種類の文字列から構成されます。

(a) 空白文字

空白 (' ') 水平タブ (' \t ') あるいは改行文字 (' \n ') を指定すると、入力データを次の空白類文字でない文字まで読み飛ばす処理を行います。

(b) 通常の文字

(a) の空白文字でも % でもない文字を指定すると、入力データを 1 文字入力します。ここで入力した文字は書式を表わす文字列の中に指定した文字と一致していなければなりません。

(c) 変換仕様

変換仕様は、% で始まる文字列で、書式を表わす文字列の後に続く引数の指す領域に入力データを変換して格納する方法を指定します。変換仕様の形式は次の規則に従います。

% [*] [フィールド幅] [変換後のデータのサイズ] 変換文字

書式中の変換仕様に対して入力したデータを格納する記憶域へのポインタがない時は、その動作は保証されません。また、書式が終了したにもかかわらず、入力データを格納する記憶域へのポインタが残っている時は、そのポインタは無視されます。

(2) 変換仕様の説明

(a) *指定

入力したデータをパラメタが指す記憶域に格納することを抑止します。

(b) フィールド幅

入力するデータの最大文字数を 10 進数字で指定します。

(c) 変換後のデータのサイズ

d, i, o, u, x, X, e, E, f 変換の時、(表 7-9 参照) 変換後のデータのサイズ (short 型、long 型、long double 型) を指定します。これ以外の変換の時は、本指定を無視します。表 7-8 にサイズ指定の種類とその意味を示します。

表 7-8 変換後のデータのサイズ指定の種類とその意味

項番	種類	意 味
1	h	d, i, o, u, x, X 変換において、変換後のデータは short 型であることを指定します。
2	l	d, i, o, u, x, X 変換において、変換後のデータは long 型であることを指定します。 また、e, E, f 変換において、変換後のデータは double 型であることを指定します。
3	L	e, E, f 変換において、変換後のデータは、long double 型であることを指定します。

(d) 変換文字

入力するデータは、各変換文字が指定する変換の種類に従って変換します。ただし、空白類文字を読み込んだ場合、変換の対象として許されていない文字を読み込んだ場合、あるいは指定されたフィールド幅を超えた場合は処理を終了します。

表 7-9 変換文字と変換の内容

項番	変換文字	変換の種類	変換の方式	対応するパラメタのデータ型
1	d	d 変換	10 進数字の文字列を整数型データに変換します。	整数型
2	i	i 変換	先頭に符号が付いている 10 進数字の文字列、あるいは最後に u (U) または l (L) が付いている 10 進数字の文字列を整数型データに変換します。また、先頭が 0x (あるいは 0X) で始まっている文字列は、16 進数字として解釈し、文字列を int 型データに変換します。さらに、先頭が 0 で始まっている文字列は、8 進数字として解釈し文字列を int 型データに変換します。	整数型
3	o	o 変換	8 進数字の文字列を整数型データに変換します。	整数型
4	u	u 変換	符号無しの 10 進数字の文字列を整数型データに変換します。	整数型
5	x	x 変換	16 進数字の文字列を整数型データに変換します。	整数型
6	X	X 変換	x 変換と X 変換に意味の違いはありません。	
7	s	s 変換	空白、水平タブ、改行文字を読み込むまでをひとつの文字列として変換します。文字列の最後にはヌル文字を付加します。(変換したデータを設定する文字列は、ヌル文字を含めて格納できるサイズが必要です。)	文字型
8	c	c 変換	1 文字を入力します。この時、入力する文字が空白類文字であっても読み飛ばすことはしません。もし、空白類文字以外の文字だけを読み込む時は、%1S と指定してください。また、フィールド幅が指定されている時は、その指定分の文字が読み込まれます。したがって、この時、変換したデータを格納する記憶域は、指定分の大きさが必要です。	char 型
9	e	e 変換	浮動小数点数を示す文字列を浮動小数点型データに変換します。e 変換と E 変換、g 変換と G 変換にそれぞれ意味の違いはありません。入力形式は strtod 関数で表現できる浮動小数点数です。	浮動小数点型
10	E	E 変換		
11	f	f 変換		
12	g	g 変換		
13	G	G 変換		

項 番	変換 文字	変換の種類	変換の方式	対応するパラメ タのデータ型
14	p	p 変換	fprintf 関数において、p 変換で変換される形式の文字列をポインタ型データに変換します。	void 型へのポ インタ型
15	n	データの変換は生じません。	データの入力を行わず、いままでに入力したデータの文字数が設定されます。	整数型
16	[[変換	[の後に文字の集合、その後に] を指定します。この文字集合は、文字列を構成する文字の集合を定義しています。もし、文字集合の最初の文字が ^ でない時は、入力データはこの文字集合にない文字が最初に読み込まれるまでをひとつの文字列として入力します。もし、最初の文字が ^ の時は、^ を除いた文字集合の文字が最初に読み込まれるまでをひとつの文字列として入力します。入力した文字列の最後には自動的にヌル文字を付加します（変換したデータを設定する文字列は、ヌル文字を含めて格納できるサイズが必要です）。	文字型
17	%	データの変換は生じません。	%を読み込みます。	無し

変換文字が表 7-9 に示す文字以外の英小文字の時は、その動作は保証されません。また、その他の文字の時は、その動作は処理系定義です。

7.11.13 printf 関数

機能

データを書式に従って変換し、標準出力ファイル（`stdout`）へ出力します。

呼び出し手順

```
#include <stdio.h>

const char *control;

int ret;

ret=printf(control [ ,arg] ...);
```

パラメタ

No.	名前	型	意味
1	control	const char 型を指すポインタ	書式を示す文字列へのポインタ
2	arg	書式に従った型	書式に従って出力されるデータ

リターン値

型 : `int`

正常 : 変換し出力した文字数

異常 : 負の値

`printf` 関数は `control` が指す書式を示す文字列に従って、パラメタ `arg` を変換、編集し、標準出力ファイル（`stdout`）へ出力します。

書式の仕様の詳細は「7.11.11 `fprintf` 関数」を参照してください。

7.11.14 scanf 関数

機能

標準入力ファイル (`stdin`) からデータを入力し、書式に従って変換します。

呼び出し手順

```
#include <stdio.h>

const char *control;

int ret;

ret=scanf(control [ ,ptr ] ...);
```

パラメタ

No.	名前	型	意味
1	control	const char 型を指すポインタ	書式を示す文字列へのポインタ
2	ptr	任意のデータを指すポインタ	入力変換したデータを格納する記憶域へのポインタ

リターン値

型 : `int`

正常 : 入力変換に成功したデータの個数

異常 : `EOF`

`scanf` 関数は標準入力ファイル (`stdin`) からデータを入力し、`control` が指す書式を示す文字列に従って、そのデータを変換、編集して、その結果を `ptr` の指す記憶域へ格納します。

`scanf` 関数は、入力変換に成功したデータの個数をリターン値として返します。最初の変換の前に標準入力ファイルが終了した時には `EOF` を返します。

書式の仕様の詳細は「7.11.12 fscanf 関数」を参照してください。

[注意] `%e` 変換について `double` 型の場合は `l`、`long double` 型の場合は `L` で指定することになっています。デフォルトの型は `float` 型です。

7.11.15 sprintf 関数

機能

データを書式に従って変換し、指定した領域へ出力します。

呼び出し手順

```
#include <stdio.h>

char *s;
const char *control;
int ret;

ret=sprintf(s, control [, arg] ...);
```

パラメタ

No.	名前	型	意味
1	s	char 型を指すポインタ	データを出力する記憶域へのポインタ
2	control	const char 型を指すポインタ	書式を示す文字列へのポインタ
3	arg	書式に従った型	書式に従って出力されるデータ

リターン値

型 : int

正常 : 変換した文字数

異常 : -

sprintf 関数は、**control** が指す書式を示す文字列に従って、パラメタ **arg** を変換、編集し、**s** の指す記憶域へ出力します。

変換して出力した文字の列の最後には、ヌル文字が付加されます。このヌル文字はリターン値である出力した文字数の中には含まれません。

書式の仕様の詳細は「7.11.11 fprintf 関数」を参照してください。

7.11.16 sscanf 関数

機能

指定した記憶域からデータを入力し、書式に従って変換します。

呼び出し手順

```
#include <stdio.h>

const char *s, *control;

int ret;

ret=sscanf(s, control [, ptr] ...);
```

パラメタ

No.	名前	型	意味
1	s	const char 型を指すポインタ	入力するデータがある記憶域
2	control	const char 型を指すポインタ	書式を示す文字列へのポインタ
3	ptr	データ型を指すポインタ	入力変換したデータを格納する記憶域へのポインタ

リターン値

型 : int

正常 : 入力変換に成功したデータの個数

異常 : EOF

sscanf 関数は、s の指す記憶域からデータを入力し、control が指す書式を示す文字列に従って、そのデータを変換、編集して、その結果を ptr の指す記憶域へ格納します。

sscanf 関数は、入力変換に成功したデータの個数を返します。また、最初の変換の前に入力するデータが終了した時には EOF を返します。

書式の仕様の詳細は「7.11.12 fscanf 関数」を参照してください。

7.11.17 vfprintf 関数

機能

可変個のパラメタリストを書式に従って、指定したストリーム入出力用ファイルに出力します。

呼び出し手順

```
#include <stdarg.h>
#include <stdio.h>
FILE *fp;
const char *control;
va_list arg;
int      ret;

ret=vfprintf(fp, control, arg);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ
2	control	const char 型を指すポインタ	書式を示す文字列へのポインタ
3	arg	va_list	引数リスト

リターン値

型 : int

正常 : 変換し出力した文字数

異常 : 負の値

vfprintf 関数は、**control** が指す書式を示す文字列に従って、可変個の引数リストを順に変換、編集し、**fp** の示すストリーム入出力用ファイルへ出力します。

vfprintf 関数は、変換し出力したデータの個数を返しますが出力エラーが発生した時は負の値を返します。

また、**vfprintf** 関数では **va_end** マクロは呼び出しません。

書式の仕様の詳細は「7.11.11 fprintf 関数」を参照してください。

【注意】

引数リストを示す `arg` は、`va_start` および `va_arg` マクロによって初期化されていなければなりません。

7.11.18 vprintf 関数

機能

可変個のパラメタリストを書式に従って標準出力ファイル（`stdout`）に出力します。

呼び出し手順

```
#include <stdarg.h>
#include <stdio.h>
const char *control;
va_list     arg;
int         ret;

ret=vprintf(control, arg);
```

パラメタ

No.	名前	型	意味
1	control	const char 型を指すポインタ	書式を示す文字列へのポインタ
2	arg	va_list	引数リスト

リターン値

型 : `int`

正常 : 変換し出力した文字数

異常 : 負の値

`vprintf` 関数は、`control` が指す書式を示す文字列に従って、可変個のパラメタリストを順に変換、編集し、標準出力ファイルへ出力します。

`vprintf` 関数は、変換し出力したデータの個数を返しますが出力エラーが発生した時は負の値を返します。

また、`vprintf` 関数では `va_end` マクロは呼び出しません。

書式の仕様の詳細は「7.11.11 printf 関数」を参照してください。

【注意】

引数リストを示す `arg` は、`va_start` および `va_arg` マクロによって初期化されていなければなりません。

7.11.19 vsprintf 関数

機能

可変個のパラメタリストを書式に従って、指定した記憶域に出力します。

呼び出し手順

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

char *s;
const char *control;
va_list arg;
int ret;

ret=vsprintf(s, control, arg);
```

パラメタ

No.	名前	型	意味
1	s	char 型を指すポインタ	データを出力する記憶域へのポインタ
2	control	const char 型を指すポインタ	書式を示す文字列へのポインタ
3	arg	va_list	引数リスト

リターン値

型 : int

正常 : 変換した文字数

異常 : 負の数

vsprintf 関数は、**control** が指す書式を示す文字列に従って、可変個の引数リストを順に変換、編集し、s により指される記憶域へ出力します。

変換して出力した文字列の最後にはヌル文字が付加されます。このヌル文字はリターン値である出力した文字数の中には含まれません。

書式の仕様の詳細は「7.11.11 fprintf 関数」を参照してください。

【注意】

引数リストを示す **arg** は、**va_start** および **va_arg** マクロによって初期化されていなければなりません。

7.11.20 fgetc 関数

機能

ストリーム入出力用ファイルから 1 文字入力します。

呼び出し手順

```
#include <stdio.h>

FILE *fp;

int ret;

ret=fgetc(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : int

正常: ファイルの終了の時: EOF

ファイルの終了でない時: 入力した文字

異常: EOF

fgetc 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルから 1 文字入力します。

fgetc 関数は、通常入力した 1 文字を返しますが、ファイルの終了やエラー発生の際は、EOF を返します。また、ファイルの終了の時には、そのファイルに対するファイル終了指示子が設定されます。

【エラー条件】

読み込みエラーが発生した時、そのファイルに対してのエラー指示子が設定されます。

7.11.21 fgets 関数

機能

ストリーム入出力用ファイルから文字列を入力します。

呼び出し手順

```
#include <stdio.h>

char *s, *ret;

int n;

FILE *fp;

ret=fgets(s, n, fp);
```

パラメタ

No.	名前	型	意味
1	s	char 型を指すポインタ	文字列を入力する記憶域へのポインタ
2	n	int	文字列を入力する記憶域のバイト数
3	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : char 型へのポインタ

正常 : ファイルの終了の時 ; NULL
ファイルの終了でない時 ; s

異常 : NULL

fgets 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルから、ポインタ **s** の指す記憶域に文字列を入力します。

fgets 関数は、**n-1** 文字まであるいは改行文字を入力するまで、またはファイルの終わりになるまで文字を入力し、入力文字列の最後にヌル文字を付け加えます。

fgets 関数は通常、文字列を入力する記憶域へのポインタ **s** を返しますが、ファイルが終了した時やエラー発生の際は **NULL** を返します。

【注意】

ファイルが終了した時は、**s** が指す記憶域の内容は変化しませんが、エラー発生の際は、**s** が指す記憶域の内容は保証されません。

7.11.22 fputc 関数

機能

ストリーム入出力用ファイルへ 1 文字出力します。

呼び出し手順

```
#include <stdio.h>

FILE *fp;

int c, ret;

ret=fputc(c, fp)
```

パラメタ

No.	名前	型	意味
1	c	int	出力する文字
2	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : int

正常 : 出力した文字

異常 : EOF

fputc 関数は、文字 **c** をファイルポインタ **fp** の示すストリーム入出力ファイルへ出力します。

fputc 関数は、通常出力した文字 **c** を返しますが、エラー発生の際は、**EOF** を返します。

【エラー条件】

書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。

7.11.23 fputs 関数

機能

ストリーム入出力用ファイルへ文字列を出力します。

呼び出し手順

```
#include <stdio.h>

const char *s;
int ret;
FILE *fp;

ret=fputs(s, fp);
```

パラメタ

No.	名前	型	意味
1	s	const char 型を指すポインタ	出力する文字列へのポインタ
2	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : int

正常 : 0

異常 : 0 以外

fputs 関数は、s の指すヌル文字の直前までの文字列をファイルポインタ fp の示すストリーム入出力用ファイルへ出力します。この時、文字列の終了を示すヌル文字は出力されません。

fputs 関数は、通常 0 を返しますが、エラー発生の際は、0 以外の値を返します。

7.11.24 getc 関数

機能

ストリーム入出力用ファイルから 1 文字入力します。

呼び出し手順

```
#include <stdio.h>

FILE *fp;

int ret;

ret=getc(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : int

正常 : ファイルの終了の時 : EOF

ファイルの終了でない時 : 入力した文字

異常 : EOF

getc 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルから 1 文字入力します。getc 関数は、通常入力した 1 文字を返しますがファイルの終了やエラー発生の際は、EOF を返します。またファイルの終了の時には、そのファイルに対するファイル終了指示子が設定されます。

【注意】

コンパイラによっては getc 関数はマクロで実現しているためアドレスをとることができません。関数として（アドレスをとるなどの用途で）使用する時は、fgetc 関数を用いてください。

【エラー条件】

読み込みエラーが発生した時、そのファイルに対してエラー指示子が設定されます。

7.11.25 getchar 関数

機能

標準入力ファイル (`stdin`) から、1 文字入力します。

呼び出し手順

```
#include <stdio.h>

int ret;

ret=getchar( );
```

パラメタ

No.	名前	型	意味
-	-	-	-

リターン値

型 : `int`

正常: ファイルの終了の時: `EOF`
ファイルの終了でない時: 入力した文字

異常: `EOF`

`getchar` 関数は標準入力ファイル (`stdin`) から 1 文字入力します。

`getchar` 関数は、通常入力した 1 文字を返しますが、ファイルの終了やエラー発生の際は `EOF` を返します。また、ファイルの終了の時には、そのファイルに対するファイル終了指示子が設定されます。

【注意】

コンパイラによっては `getchar` 関数はマクロで実現しているためアドレスをとることができません。関数として (アドレスをとるなどの用途で) 使用する時は、`fgetc` 関数を用いてください。

【エラー条件】

読み込みエラーが発生した時、そのファイルに対してエラー指示子が設定されます。

7.11.26 gets 関数

機能

標準入力ファイル (`stdin`) から文字列を入力します。

呼び出し手順

```
#include <stdio.h>

char *ret, *s;

ret=gets(s);
```

パラメタ

No.	名前	型	意味
1	s	char 型を指すポインタ	文字列を入力する記憶域へのポインタ

リターン値

型 : `char` 型へのポインタ

正常 : ファイルの終了の時 : `NULL`

ファイルの終了でない時 : s

異常 : `NULL`

`gets` 関数は、標準入力ファイル (`stdin`) から、s で始まる記憶域へ文字列を入力します。

`gets` 関数は、ファイルの終了か、改行文字を入力するまで文字を入力し、改行文字の代わりにヌル文字を付け加えます。

`gets` 関数は、通常文字列を入力する記憶域へのポインタ s を返しますが、標準入力ファイルの終了やエラー発生の際は、`NULL` を返します。

【注意】

標準入力ファイルが終了した時は、s が指す記憶域の内容は変化しませんが、エラー発生の際は s が指す記憶域の内容は保証されません。

7.11.27 putc 関数

機能

ストリーム入出力用ファイルへ 1 文字出力します。

呼び出し手順

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

```
FILE *fp;
```

```
int c, ret;
```

```
ret=putc(c, fp);
```

パラメタ

No.	名前	型	意味
1	c	int	出力する文字
2	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : int

正常 : 出力した文字

異常 : EOF

putc 関数は、文字 c をファイルポインタ fp の示すストリーム入出力ファイルへ出力します。

putc 関数は、通常出力した文字 c を返しますがエラー発生の際は、EOF を返します。

【注意】

コンパイラによっては putc 関数はマクロで実現しているため、アドレスをとることができません。関数として（アドレスをとるなどの用途で）使用する時は fputc 関数を用いてください。

【エラー条件】

書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。

7.11.28 putchar 関数

機能

標準出力ファイル (`stdout`) へ 1 文字出力します。

呼び出し手順

```
#include <stdio.h>

int c, ret;

ret=putchar(c);
```

パラメタ

No.	名前	型	意味
1	c	int	出力する文字

リターン値

型 : `int`

正常 : 出力した文字

異常 : `EOF`

`putchar` 関数は、文字 `c` を標準出力ファイル (`stdout`) へ出力します。`putchar` マクロは、通常出力した文字 `c` を返しますが、エラー発生の際は `EOF` を返します。

【注意】

コンパイラによっては `putchar` 関数はマクロで実現しているため、アドレスをとることができません。関数として (アドレスをとるなどの用途で) 使用する時は、`fputc` 関数を用いてください。

【エラー条件】

書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。

7.11.29 puts 関数

機能

標準出力ファイル (`stdout`) へ文字列を出力します。

呼び出し手順

```
#include <stdio.h>

const char *s;

int ret;

ret=puts(s);
```

パラメタ

No.	名前	型	意味
1	s	const char 型を指すポインタ	出力する文字列へのポインタ

リターン値

型 : `int`

正常 : `0`

異常 : `0` 以外

`puts` 関数は、s の指す文字列を標準出力ファイル (`stdout`) へ出力します。この時、文字列の終了を示す文字は出力されず、代わりに改行文字を出力します。

`puts` 関数は、通常 `0` を返しますが、エラー発生の時、`0` 以外の値を返します。

7.11.30 ungetc 関数

機能

ストリーム入出力用ファイルへ 1 文字をもどします。

呼び出し手順

```
#include <stdio.h>

int c, ret;

FILE *fp;

ret=ungetc(c, fp);
```

パラメタ

No.	名前	型	意味
1	c	int	もどす文字
2	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : int

正常 : もどした文字

異常 : EOF

ungetc 関数は、文字 **c** を、ファイルポインタ **fp** に示すストリーム入出力用ファイルへもどします。また、ここでもどされた文字は、**fflush** , **fseek** , **rewind** 関数を呼び出さなければ次の入力データとなります。

ungetc 関数は、通常もどした文字 **c** を返しますが、エラー発生の際は、**EOF** を返します。

【注意】

ungetc 関数が **fflush** , **fseek** , **rewind** 関数を実行することなく 2 回以上呼び出された時の動作は保証されません。また、**ungetc** 関数が実行されるとファイルに対する現在の位置指示子がひとつもどされますが、この位置指示子がすでにファイルの先頭に位置している時は、位置指示子は保証されなくなります。

7.11.31 fread 関数

機能

ストリーム入出力用ファイルから、指定した記憶域にデータを入力します。

呼び出し手順

```
#include <stdio.h>

void *ptr;
size_t size;
size_t n, ret;
FILE *fp;

ret=fread(ptr, size, n, fp);
```

パラメタ

No.	名前	型	意味
1	ptr	void 型を指すポインタ	データを入力する記憶域へのポインタ
2	size	size_t	1 メンバのバイト数
3	n	size_t	入力するメンバの数
4	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : size_t

正常 : size もしくは n が 0 の時 : 0

size, n がともに 0 でない時 : 入力に成功したメンバ数

異常 : -

fread 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルから **ptr** が指す記憶域に **size** で指定したバイト数を 1 メンバとしたデータを **n** メンバ入力します。この時ファイルに対する位置指示子は入力したバイト数分進められます。

fread 関数は、実際に入力に成功したメンバ数を返しますので、通常 **n** と同じ値になります。しかし、ファイルが終了した時やエラー発生の際は、それまで入力に成功したメンバ数を返しますので、**n** より小さな値となります。ファイルの終了かエラー発生かの区別は、**ferror** , **feof** 関数を用いて行ってください。

【注意】

`size` もしくは `n` が `0` の時、リターン値として `0` を返し `ptr` の指す記憶域の内容は変化しません。また、エラーが発生した時、または、メンバの途中までしか入力できなかった時は、そのファイルの位置指示子は保証されません。

7.11.32 fwrite 関数

機能

メモリ領域からストリーム入出力用ファイルにデータを出力します。

呼び出し手順

```
#include <stdio.h>

const void *ptr;
size_t size;
size_t n, ret;
FILE *fp;

ret=fwrite(ptr, size, n, fp);
```

パラメタ

No.	名前	型	意味
1	ptr	const void 型を指すポインタ	出力するデータを格納している記憶域へのポインタ
2	size	size_t	1 メンバのバイト数
3	n	size_t	出力するメンバの数
4	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : size_t

正常 : 出力に成功したメンバ数

異常 : -

fwrite 関数は、**ptr** の指す記憶域から、ファイルポインタ **fp** の示すストリーム入出力用ファイルに、**size** で指定したバイト数を 1 メンバとしたデータを **n** メンバ出力します。この時、ファイルに対する位置指示子は出力したバイト数進められます。

fwrite 関数は、実際に出力に成功したメンバ数を返しますので、通常 **n** と同じ値になります。しかし、エラー発生の際はそれまで出力に成功したメンバ数を返しますので、それより小さな値となります。

【注意】

エラー発生の時、そのファイルの位置指示子は保証されません。

7.11.33 fseek 関数

機能

ストリーム入出力用ファイルの現在の読み書き位置を移動させます。

呼び出し手順

```
#include <stdio.h>

FILE *fp;
long offset;
int type, ret;

ret=fseek(fp, offset, type);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ
2	offset	long	オフセットの種類で指定された位置からのオフセット
3	type	int	オフセットの種類

リターン値

型 : int

正常 : 0

異常 : 0 以外

fseek 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルの現在の読み書き位置をオフセットの種類 **type** で指定した場所から **offset** バイト先の位置に移動します。

オフセットの種類を表 7-10 に示します。

fseek 関数は、通常は 0 を返しますが、不適当な要求に対しては 0 以外の値を返します。

表 7-10 オフセットの種類

項番	オフセットの種類	意 味
1	SEEK_SET	ファイルの先頭から offset バイト先の位置に移動します。この時、offset で指定する値は 0 か正でなければなりません。
2	SEEK_CUR	ファイルの現在位置から offset バイト先の位置に移動します。この時、offset で指定する値が正ならばファイルの後方に、負ならばファイルの先頭に向かって移動します。
3	SEEK_END	ファイルの終わりから offset バイト先の位置に移動します。この時 offset で指定する値は 0 か負でなければなりません。

【注意】

テキストファイルの時は、オフセットの種類は `SEEK_SET` でかつ、offset は 0 かそのファイルに対する `ftell` 関数によって返された値でなければなりません。また、`fseek` 関数を呼び出すことによって `ungetc` 関数の効果はなくなりますので注意が必要です。

7.11.34 ftell 関数

機能

ストリーム入出力用ファイルの現在の読み書き位置を求めます。

呼び出し手順

```
#include <stdio.h>

FILE *fp;

long ret;

ret=ftell(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : long

正常: 現在の位置指示子の位置 (テキストファイル)

ファイルの先頭から現在位置までのバイト数 (バイナリファイル)

異常: -

ftell 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルの現在の読み書き位置を求めます。

ftell 関数は、バイナリファイルの時、ファイルの先頭から現在位置までのバイト数を返しますが、テキストファイルの時は、ここで返した値が **fseek** 関数でできるように処理系定義の値を位置指示子の位置として返します。

【注意】

ftell 関数を 2 回テキストファイルに適用した時、そのリターン値の差が実際のファイル上の隔たりを表わすことにはなりません。

7.11.35 rewind 関数

機能

ストリーム入出力用ファイルの現在の読み書き位置を、ファイルの先頭に移動します。

呼び出し手順

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

```
FILE *fp;
```

```
rewind(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : void

正常 : -

異常 : -

rewind 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルの現在の読み書き位置をファイルの先頭に移動します。

また、**rewind** 関数は、そのファイルに対する終了指示子とエラー指示子をクリアします。

【注意】

rewind 関数を呼び出すことによって、**ungetc** 関数の効果はなくなりますので、注意が必要です。

7.11.36 clearerr 関数

機能

ストリーム入出力用ファイルのエラー状態をクリアします。

呼び出し手順

```
#include <stdio.h>

FILE *fp;

clearerr(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : void

正常 : -

異常 : -

clearerr 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルに対するエラー指示子と終了指示子をクリアします。

7.11.37 feof 関数

機能

ストリーム入出力用ファイルが終わりであるかどうかを判定します。

呼び出し手順

```
#include <stdio.h>

FILE *fp;

int ret;

ret=feof(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : **int**

正常 : ファイルが終わりの時 : **0** 以外

ファイルが終わりでない時 : **0**

異常 : -

feof 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルが終了したかどうかを判定します。

feof 関数は、指定したストリーム入出力用ファイルに対するファイル終了指示子を調べ、設定されていればファイルが終わりであるとして、**0** 以外の値を返します。設定されていなければ、ファイルはまだ終わりではないとして **0** を返します。

7.11.38 ferror 関数

機能

ストリーム入出力用ファイルがエラー状態であるかどうかを判定します。

呼び出し手順

```
#include <stdio.h>

FILE *fp;

int ret;

ret=ferror(fp);
```

パラメタ

No.	名前	型	意味
1	fp	FILE 型を指すポインタ	ファイルポインタ

リターン値

型 : int

正常: ファイルがエラー状態の時: 0 以外

ファイルがエラー状態でない時: 0

異常: -

ferror 関数は、ファイルポインタ **fp** の示すストリーム入出力用ファイルがエラー状態であるかどうかを判定します。

ferror 関数は、指定したストリーム入出力用ファイルに対するエラー指示子を調べ、設定されていれば、エラー状態にあるとして 0 以外の値を返します。設定されていなければ、エラー状態ではないとして 0 を返します。

7.11.39 perror 関数

機能

標準エラーファイル（`stderr`）に、エラー番号に対応したエラーメッセージを出力します。

呼び出し手順

```
#include <stdio.h>

const char *s;

perror(s);
```

パラメタ

No.	名前	型	意味
1	s	const char 型を指すポインタ	エラーメッセージへのポインタ

リターン値

型 : `void`

正常 : -

異常 : -

`perror` 関数は標準エラーファイル（`stderr`）へ `s` で示されるエラーメッセージと `errno` とを対応させ出力します。

出力するメッセージは、もし、`s` が `NULL` でなく、`s` の指す文字列がヌル文字でないならば、`s` の指す文字列にコロンと空白とその後に処理系定義のエラーメッセージを続け最後に改行文字を付けた形式で出力されます。

7.12 <stdlib.h>

機能概要

C プログラムでの標準的処理を行う関数を定義しています。

定義名一覧

定義名	種類	説明
onexit_t	マクロ名	onexit 関数で登録する関数の返す型および onexit 関数のリターン値の型を示しています。
div_t	マクロ名	div 関数のリターン値の構造体の型を示しています。
ldiv_t	マクロ名	ldiv 関数のリターン値の構造体の型を示しています。
RAND_MAX	マクロ名	rand 関数において生成する擬似乱数整数の最大値を示しています。
atof	関数	数を表現する文字列を double 型の浮動小数点数値に変換します。
atoi	関数	10 進数を表現する文字列を int 型の整数値に変換します。
atol	関数	10 進数を表現する文字列を long 型の整数値に変換します。
strtod	関数	数を表現する文字列を double 型の浮動小数点数値に変換します。
strtol	関数	数を表現する文字列を long 型の整数値に変換します。
rand	関数	0 から RAND_MAX の間の擬似乱数整数を生成します。
srand	関数	rand 関数で生成する擬似乱数列の初期値を設定します。
calloc	関数	記憶域を割り当てて、すべての割当てられた記憶域を 0 によって初期化します。
free	関数	指定された記憶域を解放します。
malloc	関数	記憶域を割り当てます。
realloc	関数	記憶域の大きさを指定された大きさに変更します。
abort	関数	プログラムを異常終了させます。
exit	関数	プログラムを正常終了させます。
getenv	関数	プログラムを起動する側の環境における名前に対する定義を参照します。
onexit	関数	プログラム終了時に呼び出すべき関数を登録します。
system	関数	指定された文字列を OS のコマンドとして実行します。
bsearch	関数	2 分割検索を行います。
qsort	関数	ソートを行います。
abs	関数	int 型整数の絶対値を計算します。
div	関数	int 型整数の除算の商と余りを計算します。
labs	関数	long 型整数の絶対値を計算します。

定義名	種類	説 明
ldiv	関数	long 型整数の除算の商と余りを計算します。

上記のマクロ名は、処理系定義です。

7.12.1 atof 関数

機能

数を表現する文字列を、**double** 型の浮動小数点数値に変換します。

呼び出し手順

```
#include <stdlib.h>

const char *nptr;

double ret;

ret=atof(nptr);
```

パラメタ

No.	名前	型	意味
1	nptr	const char 型を指すポインタ	変換する数を表現する文字列のポインタ

リターン値

型 : **double**

正常 : 変換された **double** 型の浮動小数点数値

異常 : -

変換は、浮動小数点数の形式に合わない最初の文字までに対して行います。

【注意】

atof 関数は、オーバーフロー等のエラーが生じてでも **errno** を設定しません。また、エラーが生じた場合、結果の値は保証されません。変換のエラーが生じる可能性がある場合は、**strtod** 関数を使用してください。

7.12.2 atoi 関数

機能

10 進数を表現する文字列を、`int` 型の整数値に変換します。

呼び出し手順

```
#include <stdlib.h>

const char *nptr;

int ret;

ret=atoi(nptr);
```

パラメタ

No.	名前	型	意味
1	nptr	const char 型を指すポインタ	変換する数を表現する文字列のポインタ

リターン値

型 : `int`

正常 : 変換された `int` 型の整数値

異常 : -

変換は、10 進数の形式に合わない最初の文字までに対して行います。

【注意】

`atoi` 関数は、オーバーフロー等のエラーが生じても `errno` を設定しません。また、エラーが生じた場合、結果の値を保証しません。変換のエラーが生じる可能性がある場合は、`strtol` 関数を使用してください。

7.12.3 atol 関数

機能

10 進数を表現する文字列を、**long** 型の整数値に変換します。

呼び出し手順

```
#include <stdlib.h>

const char *nptr ;
long ret;

ret=atol(nptr);
```

パラメタ

No.	名前	型	意味
1	nptr	const char 型を指すポインタ	変換する数を表現する文字列のポインタ

リターン値

型 : **long**

正常 : 変換された **long** 型の整数値

異常 : -

変換は、10 進数の形式に合わない最初の文字までに対して行います。

【注意】

atol 関数は、オーバーフロー等のエラーが生じても **errno** を設定しません。また、エラーが生じた場合、結果の値を保証しません。変換のエラーが生じる可能性がある場合は、**strtol** 関数を使用してください。

7.12.4 strtod 関数

機能

数を表現する文字列を **double** 型の浮動小数点数値に変換します。

呼び出し手順

```
#include <stdlib.h>

const char *nptr;
char **endptr;
double ret;

ret=strtod(nptr, endptr);
```

パラメタ

No.	名前	型	意味
1	nptr	const char 型を指すポインタ	変換する数を表現する文字列へのポインタ
2	endptr	char 型を指すポインタへのポインタ	浮動小数点数値を構成していない最初の文字へのポインタを格納する記憶域へのポインタ

リターン値

型 : **double**

正常 : **nptr** が指している文字列が浮動小数点数を構成しない文字で始まっている時
: **0**

nptr が指している文字列が浮動小数点数を構成する文字で始まっている時
: 変換された **double** 型の浮動小数点数値

異常 : 変換後の値がオーバーフローの時 : 変換する文字列の符号と同符号をもつ
HUGE_VAL

変換後の値がアンダフローの時 : **0**

strtod 関数は、最初の数字もしくは小数点から浮動小数点数値を構成しない文字の直前までを C 言語仕様の規則に従って **double** 型の浮動小数点数値に変換します。ただし、指数部も小数点も現われなかった時は、小数点は文字列の最後の数字の後に続くと仮定されます。**endptr** の指す領域には、浮動小数点数を構成しない最初の文字へのポインタを設定します。数字を読み込む前に浮動小数点数を構成しない文字がある場合は **nptr** の値を設定します。**endptr** が **NULL** の場合、この設定は行われません。

【エラー条件】

変換後の値がオーバーフロー / アンダフローをおこす時は、**errno** に **ERANGE** が設定されます。

7.12.5 strtol 関数

機能

数を表現する文字列を `long` 型の整数値に変換します。

呼び出し手順

```
#include <stdlib.h>

long ret;

const char *nptr;
char **endptr;
int base;

ret=strtol(nptr, endptr, base);
```

パラメタ

No.	名前	型	意味
1	<code>nptr</code>	<code>const char</code> 型を指すポインタ	変換する数を表現する文字列へのポインタ
2	<code>endptr</code>	<code>char</code> 型を指すポインタへのポインタ	整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
3	<code>base</code>	<code>int</code>	変換の基数 (0 又は 2 ~ 36)

リターン値

型 : `long`

正常 : `nptr` が指している文字列が整数を構成しない文字で始まっている時 : 0

`nptr` が指している文字列が整数を構成する文字で始まっている時

: 変換された `long` 型の整数値

異常 : 変換後の値がオーバーフローの時

: 変換する文字列の符号に従って `LONG_MAX` あるいは `LONG_MIN`

`strtol` 関数は、最初の数字から整数を構成しない最初の文字の前までを `long` 型の整数値に変換します。

`endptr` の指す記憶域に、整数を構成しない最初の文字へのポインタを設定します。最初の数字を読み込む前に整数を構成しない文字がある場合は `nptr` の値を設定します。

`endptr` が `NULL` 場合、この設定は行われません。

`base` の値が 0 の時は、C 言語仕様の規則に従って変換されます。`base` の値が 2 から 36

の間の時は、変換する時の基数を示しています。ここで変換する文字列中の `a` (もしくは `A`) から `z` (もしくは `Z`) までの文字は、10 から 35 の値に対応付けられます。`base` の値より大きいか等しい文字が、変換する文字列の中にある時は、そこで変換処理を終了します。また、符号の後にある `0` は、変換の時は無視され、また、`base` が 16 の時の `0x` (もしくは `0X`) も無視されます。

【エラー条件】

変換後の値がオーバーフローをおこす時は、`errno` に `ERANGE` が設定されます。

7.12.6 rand 関数

機能

0 から **RAND_MAX** の間の擬似乱数整数を生成します。

呼び出し手順

```
#include <stdlib.h>

int ret;

ret=rand( );
```

パラメタ

No.	名前	型	意味
-	-	-	-

リターン値

型 : **int**

正常 : 擬似乱数整数値

異常 : -

7.12.7 srand 関数

機能

rand 関数で生成する擬似乱数列の初期値を設定します。

呼び出し手順

```
#include <stdlib.h>

unsigned int seed;

srand(seed);
```

パラメタ

No.	名前	型	意味
1	seed	unsigned int	擬似乱数列生成の初期値

リターン値

型 : void

正常 : -

異常 : -

srand 関数は、**rand** 関数が擬似乱数列を生成するための初期値を設定します。したがって、**rand** 関数で擬似乱数値を生成している時に、再度 **srand** 関数で、同じ値の初期値を設定すると、擬似乱数列はくり返し生成されることになります。

【注意】

rand 関数が **srand** 関数より先に呼ばれた時は、擬似乱数列の生成の初期値として 1 が設定されます。

7.12.8 calloc 関数

機能

記憶域を割り当てて、すべての割り当てられた記憶域を 0 によって初期化します。

呼び出し手順

```
#include <stdlib.h>

size_t nelem, elsize;

void *ret;

ret=calloc(nelem, elsize);
```

パラメタ

No.	名前	型	意味
1	nelem	size_t	要素の数
2	elsize	size_t	ひとつの要素の占めるバイト数

リターン値

型 : void 型へのポインタ

正常 : 割り当てられた記憶域の先頭のアドレス

異常 : 記憶域の割り当てができなかった時、またはパラメタのいずれかが 0 の時 :
NULL

elsize バイト単位の記憶域を **nelem** 個記憶域に割り当てます。また、その割り当てられた記憶域のすべてのビットは 0 で初期化されます。

7.12.9 free 関数

機能

指定された記憶域を解放します。

呼び出し手順

```
#include <stdlib.h>

void *ptr;

free(ptr);
```

パラメタ

No.	名前	型	意味
1	ptr	void 型を指すポインタ	解放する記憶域のアドレス

リターン値

型 : void

正常 : -

異常 : -

ptr が指す記憶域を解放し、再度割り当てて使用することを可能とします。ptr が NULL であれば何もしません。

【注意】

解放しようとした記憶域が、calloc、malloc、realloc 関数で割り当てられた記憶域でない時、または、すでに free、realloc 関数によって解放されていた時の動作は保証されません。また、解放された後の記憶域を参照した時の動作も保証されません。

7.12.10 malloc 関数

機能

記憶域を割り当てます。

呼び出し手順

```
#include <stdlib.h>

size_t size;

void *ret;

ret=malloc(size);
```

パラメタ

No.	名前	型	意味
1	size	size_t	割り当てる記憶域のバイト数

リターン値

型 : void 型へのポインタ

正常 : 割り当てられた記憶域の先頭アドレス

異常 : 記憶域の割り当てができなかった時、または size が 0 の時 : NULL

size で示されるバイトの分だけ記憶域を割り当てます。

7.12.11 realloc 関数

機能

記憶域の大きさを指定された大きさに変更します。

呼び出し手順

```
#include <stdlib.h>

size_t size;

void *ptr, *ret;

ret=realloc(ptr, size);
```

パラメタ

No.	名前	型	意味
1	ptr	void 型を指すポインタ	変更する記憶域の先頭アドレス
2	size	size_t	変更後の記憶域のバイト数

リターン値

型 : void 型へのポインタ

正常 : 変更した記憶域の先頭アドレス

異常 : 記憶域の割り当てができなかった時、または size が 0 の時 : NULL

ptr の指す記憶域の大きさを size で示されるバイト分の大きさの記憶域に変更します。もし、新しく割り当てられた記憶域の大きさが、変更前の記憶域の大きさより小さい時は、新しく割り当てられた記憶域の大きさまでの内容は変化しません。

【注意】

ptr が calloc、malloc、realloc 関数で割り当てられた記憶域へのポインタでない時、またはすでに free、realloc 関数によって解放されている記憶域へのポインタの時、動作は保証されません。

7.12.12 abort 関数

機能

プログラムを異常終了させます。

呼び出し手順

```
#include <stdlib.h>
```

```
abort( );
```

パラメタ

No.	名前	型	意味
-	-	-	-

リターン値

型 : void

正常 : -

異常 : -

abort 関数を実行したプログラムを異常終了させます。

もし、**signal** 関数で **SIGABRT** が登録されている時は、**abort** 関数の実行が無視されます。

【注意】

abort 関数の実行後は、この関数を呼び出した元のプログラムへはもどりません。また、プログラムを異常終了する時に、オープンされているファイルが出力ファイルのとき、ストリームバッファに残ったデータを出力ファイルに掃き出すか、オープンされたファイルが、クローズされるか、一時ファイルが削除されるかは、処理系定義です。

7.12.13 exit 関数

機能

プログラムを正常終了させます。

呼び出し手順

```
#include <stdlib.h>

int status;

exit(status);
```

パラメタ

No.	名前	型	意味
1	status	int	終了コード

リターン値

型 : void

正常 : -

異常 : -

`exit` 関数は、次の三つの処理を順次行います。

- (1) `onexit` 関数によって登録されたすべての関数を登録した時とは逆の順番で呼び出します。
- (2) すべての出力用にオープンされたファイルを、フラッシュ（出力バッファの内容をファイルに出力）します。また、すべてのオープンされたファイルをクローズし、一時ファイルを削除します。
- (3) 制御をプログラム起動時の環境にもどします。

【注意】

`status` の値が 0 の時は、正常終了の状態として制御をプログラム起動時の環境にもどしますが、0 以外の時は、処理系定義の状態として制御をプログラム起動時の環境にもどします。

7.12.14 getenv 関数

機能

プログラムを起動する側の環境における名前に対する定義を参照します。

呼び出し手順

```
#include <stdlib.h>

const char *name;

char          *ret;

          ret=getenv(name);
```

パラメタ

No.	名前	型	意味
1	name	const char 型を指すポインタ	参照したい名前と一致する文字列へのポインタ

リターン値

型 : char 型へのポインタ

正常 : name に対応する名前が見つかった時
 : その名前に対する定義の先頭へのポインタ
 name に対応する名前が見つからなかった時 : NULL

異常 : -

プログラムを起動する側の環境における名前に対する定義は環境リストと呼ばれ、次のような形をしています。

```
名前1 = 定義1
名前2 = 定義2
      ⋮
名前n = 定義n
```

getenv 関数はこれらの環境リストの中から name と一致する名前を見つけ出し、その名前に対する定義の先頭位置（アドレス）を返します。

7.12.15 onexit 関数

機能

プログラム終了時に呼び出すべき関数を登録します。

呼び出し手順

```
#include <stdlib.h>
onexit_t (*func)();
onexit_t ret;

ret=onexit(func);
```

パラメタ

No.	名前	型	意味
1	func	関数へのポインタ	プログラム終了時に呼び出す関数名

リターン値

型 : onexit_t

正常 : NULL と等しくない値

異常 : NULL と等しい値

【注意】

同じ関数が 2 度以上登録された時その動作は保証されません。

7.12.16 system 関数

機能

指定された文字列をオペレーティングシステムのコマンドとして実行します。

呼び出し手順

```
#include <stdlib.h>

const char *string;

int ret;

ret=system(string);
```

パラメタ

No.	名前	型	意味
1	string	const char 型を指すポインタ	実行するコマンド名へのポインタ

リターン値

型 : int

正常 : 処理系定義の値

異常 : string が NULL の時 : 0

string で指定された文字列をオペレーティングシステムのコマンド名とし、コンパイラによって定められた方法で、コマンドを実行します。

7.12.17 bsearch 関数

機能

二分探索を行います。

呼び出し手順

```
#include <stdlib.h>

const void *key, *base;

size_t nmemb, size;

int (*compar) (const void *, const void *);

void *ret;

ret=bsearch(key, base, nmemb, size, compar);
```

パラメタ

No.	名前	型	意味
1	key	const void 型を指すポインタ	検索するデータへのポインタ
2	base	const void を指すポインタ	検索対象となるテーブルへのポインタ
3	nmemb	size_t	検索対象のメンバの数
4	size	size_t	検索対象のメンバのバイト数
5	compar	int 型を返す関数へのポインタ	比較を行う関数へのポインタ

リターン値

型 : void 型へのポインタ

正常 : 一致するメンバが検索できた時 : 一致したメンバへのポインタ

一致するメンバが検索できなかった時 : NULL

異常 : -

key の指すデータと一致するメンバを、base の指すテーブルの中で二分探索法によって検索します。比較を行う関数は、比較する二つのデータへのポインタ p1 (第 1 引数)、p2 (第 2 引数) を受け取り、以下の仕様に従って結果を返してください。

*p1 < *p2 の時 負の値を返します。

*p1 == *p2 の時 0 を返します。

*p1 > *p2 の時 正の値を返します。

【注意】

検索対象となる各メンバは、昇順に並んでいる必要があります。

7.12.18 qsort 関数

機能

ソートを行います。

呼び出し手順

```
#include <stdlib.h>

const void *base;
size_t nmemb, size;
int (*compar) (const void *, const void *);

qsort(base, nmemb, size, compar);
```

パラメタ

No.	名前	型	意味
1	base	const void を指すポインタ	ソート対象となるテーブルへのポインタ
2	nmemb	size_t	ソート対象のメンバの数
3	size	size_t	ソート対象のメンバのバイト数
4	compar	int 型を返す関数へのポインタ	比較を行う関数へのポインタ

リターン値

型 : void

正常 : -

異常 : -

base の指すテーブルのデータをソートします。データの並べる順序は、比較を行う関数へのポインタによって指定します。この関数は、比較する二つのデータへのポインタ **p1** (第 1 引数)、**p2** (第 2 引数) を受け取り、以下の仕様に従って結果を返してください。

***p1** < ***p2** の時 負の値を返します。

***p1** = ***p2** の時 0 を返します。

***p1** > ***p2** の時 正の値を返します。

7.12.19 abs 関数

機能

int 型整数の絶対値を計算します。

呼び出し手順

```
#include <stdlib.h>

int i, ret ;

ret=abs(i);
```

パラメタ

No.	名前	型	意味
1	i	int	絶対値を求める整数

リターン値

型 : **int**

正常 : **i** の絶対値

異常 : -

【注意】

i の絶対値を求めた結果、**int** 型整数値として表現できない時の動作は保証されません。

7.12.20 div 関数

機能

int 型整数の除算の商と余りを計算します。

呼び出し手順

```
#include <stdlib.h>

int numer, denom;
div_t ret;

ret=div(numer, denom);
```

パラメタ

No.	名前	型	意味
1	numer	int	被除数
2	denom	int	除数

リターン値

型 : **div_t**

正常: **numer** を **denom** で除算した結果の商と余り。

異常: -

7.12.21 labs 関数

機能

long 型整数の絶対値を計算します。

呼び出し手順

```
#include <stdlib.h>

long j;
long ret;

ret=labs(j);
```

パラメタ

No.	名前	型	意味
1	j	long	絶対値を求める整数

リターン値

型 : **long**

正常 : j の絶対値

異常 : -

【注意】

j の絶対値を求めた結果、**long** 型の整数値として表現できない時の動作は保証されません。

7.12.22 ldiv 関数

機能

long 型整数の除算の商と余りを計算します。

呼び出し手順

```
#include <stdlib.h>

long numer, denom;
ldiv_t ret;

ret=ldiv(numer, denom);
```

パラメタ

No.	名前	型	意味
1	numer	long	被除数
2	denom	long	除数

リターン値

型 : **ldiv_t**

正常 : **numer** を **denom** で除算した結果の商と余り。

異常 : -

7.13 < string.h >

機能概要

文字配列の操作に必要な種々の関数を定義します。

定義名一覧

定義名	種類	説 明
memcpy	関数	複写元の記憶域の内容を指定した大きさ分、複写先の記憶域に複写します。
strcpy	関数	複写元の文字列の内容を、複写先の記憶域にヌル文字も含めて複写します。
strncpy	関数	複写元の文字列を指定された文字数分、複写先の記憶域に複写します。
strcat	関数	文字列の後に、文字列を連結します。
strncat	関数	文字列に文字列を指定した文字数分、連結します。
memcmp	関数	指定された二つの記憶域の比較を行います。
strcmp	関数	指定された二つの文字列を比較します。
strncmp	関数	指定された二つの文字列を指定された文字数分まで比較します。
memchr	関数	指定された記憶域において、指定された文字が最初に現われる位置を検索します。
strchr	関数	指定された文字列において、指定された文字が最初に現われる位置を検索します。
strcspn	関数	指定された文字列を先頭から調べ、別に指定した文字列中の文字以外の文字が先頭から何文字続くか求めます。
strpbrk	関数	指定された文字列において、別に指定された文字列中の文字が最初に現われる位置を検索します。
strrchr	関数	指定された文字列において指定された文字が最後に現われる位置を検索します。
strspn	関数	指定された文字列を先頭から調べ別に指定した文字列中の文字が先頭から何文字続くかを求めます。
strstr	関数	指定された文字列において、別に指定した文字列が最初に現われる位置を検索します。
strtok	関数	指定した文字列をいくつかの字句に切り分けます。
memset	関数	指定された記憶域の先頭から指定された文字を指定された文字数分設定します。

定義名	種類	説 明
strerror	関数	エラーメッセージを設定します。
strlen	関数	文字列の長さを計算します。

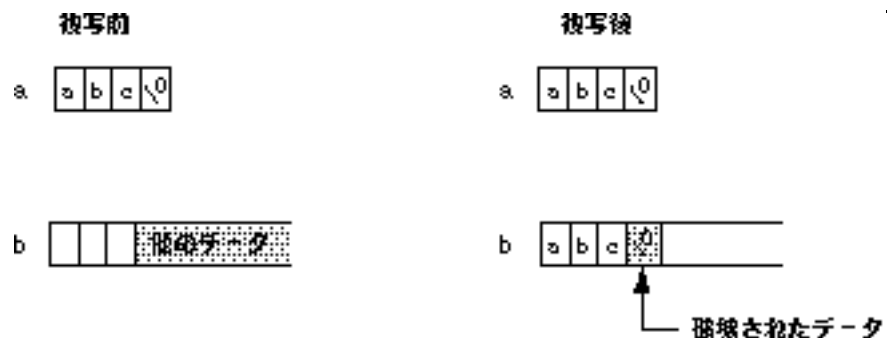
本標準インクルードファイル内で定義されている関数を使用する時は、以下の二つの事項に注意する必要があります。

- (1) 文字列の複写を行う時、複写先の領域が複写元の領域よりも、小さい場合、動作は保証されませんので注意が必要です。

例

```
char a[ ]="abc";
char b[3];
:
strcpy(b, a);
```

この場合、配列 **a** のサイズは (ヌル文字を含めて) 4 バイトです。したがって、**strcpy** 関数によって複写を行うと、配列 **b** の領域以外のデータを書き換えることになります。



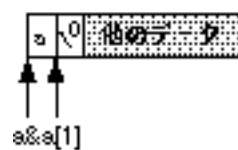
- (2) 文字列の複写を行う時、複写元の領域と複写先の領域が重なっていると正しい動作が保証されませんので注意が必要です。

例

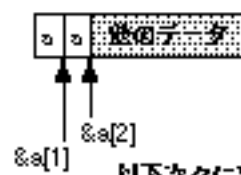
```
int a[ ]="a";
:
strcpy(&a[1], a);
:
:
```

この場合、複写元の文字列がヌル文字に達する以前に、ヌル文字の上に文字 '**a**' を書き込むことになります。したがって、複写元の文字列のデータに続くデータを書き換えることになります。

複写前



複写後



以下次々に文字を複写し続けます。

7.13.1 memcpy 関数

機能

複写元の記憶域の内容を、指定した大きさ分、複写先の記憶域に複写します。

呼び出し手順

```
#include <string.h>

void *ret, *s1;
const void *s2;
size_t n;

ret=memcpy(s1, s2, n);
```

パラメタ

No.	名前	型	意味
1	s1	void 型を指すポインタ	複写先の記憶域へのポインタ
2	s2	const void 型を指すポインタ	複写元の記憶域へのポインタ
3	n	size_t	複写する文字数

リターン値

型 : void 型へのポインタ

正常 : s1 の値

異常 : -

7.13.2 strcpy 関数

機能

複写元の文字列の内容を、複写先の記憶域にヌル文字も含めて複写します。

呼び出し手順

```
#include <string.h>

char *s1, *ret;
const char *s2;

ret=strcpy(s1, s2);
```

パラメタ

No.	名前	型	意味
1	s1	char 型を指すポインタ	複写先の記憶域へのポインタ
2	s2	const char 型を指すポインタ	複写元の文字列へのポインタ

リターン値

型 : char 型へのポインタ

正常 : s1 の値

異常 : -

7.13.3 strncpy 関数

機能

複写元の文字列を指定された文字数分、複写先の記憶域に複写します。

呼び出し手順

```
#include <string.h>

char *s1, *ret;
const char *s2;
size_t n;

ret=strncpy(s1, s2, n);
```

パラメタ

No.	名前	型	意味
1	s1	char 型を指すポインタ	複写先の記憶域へのポインタ
2	s2	const char 型を指すポインタ	複写元の文字列へのポインタ
3	n	size_t	複写する文字数

リターン値

型 : char 型へのポインタ

正常: s1 の値

異常: -

s2 で指された文字列から最高 n 文字を s1 で指される記憶域に複写します。s2 で指定された文字列の長さが n 文字より短い時は、n 文字になるまでヌル文字が付加されます。

【注意】

s2 で指された文字列の長さが n 文字より長い時は、s1 に複写された文字列はヌル文字で終了しないことになります。

7.13.4 strcat 関数

機能

文字列の後に、文字列を連結します。

呼び出し手順

```
#include <string.h>

char *s1, *ret;
const char *s2;

ret=strcat(s1, s2);
```

パラメタ

No.	名前	型	意味
1	s1	char 型を指すポインタ	連結される文字列へのポインタ
2	s2	const char 型を指すポインタ	連結する文字列へのポインタ

リターン値

型 : char 型へのポインタ

正常 : s1 の値

異常 : -

s1 で指された文字列の最後に、s2 で指された文字列を連結します。この時、s2 の指す文字列の最後を示すヌル文字も複写します。また、s1 で指された文字列の最後のヌル文字は削除されます。

7.13.5 strncat 関数

機能

文字列に文字列を指定した文字数分連結します。

呼び出し手順

```
#include <string.h>

char *s1, *ret;
const char *s2;
size_t n;

ret=strncat(s1, s2, n);
```

パラメタ

No.	名前	型	意味
1	s1	char 型を指すポインタ	連結される文字列へのポインタ
2	s2	const char 型を指すポインタ	連結する文字列へのポインタ
3	n	size_t	連結する文字数

リターン値

型 : char 型へのポインタ

正常 : s1 の値

異常 : -

s2 で指された文字列の先頭から n 文字を s1 で指された文字列の最後に付加します。s1 で指された文字列の最高のヌル文字は s2 の先頭文字で置き換えられます。

また、連結された後の文字列の最後には、必ずヌル文字が付加されます。

7.13.6 memcmp 関数

機能

指定された二つの記憶域の内容を比較します。

呼び出し手順

```
#include <string.h>

const void *s1, *s2;
size_t n;
int ret;

ret=memcmp(s1, s2, n);
```

パラメタ

No.	名前	型	意味
1	s1	const void 型を指すポインタ	比較される記憶域へのポインタ
2	s2	const void 型を指すポインタ	比較する記憶域へのポインタ
3	n	size_t	比較する記憶域の文字数

リターン値

型 : int

正常: s1 で指された記憶域 > s2 で指された記憶域の時: 正の値

s1 で指された記憶域 = s2 で指された記憶域の時: 0

s1 で指された記憶域 < s2 で指された記憶域の時: 負の値

異常: -

s1 で指された記憶域と s2 で指された記憶域の最初の n 文字分の内容を比較します。比較するための基準は処理系定義です。

7.13.7 strcmp 関数

機能

指定された二つの文字列の内容を比較します。

呼び出し手順

```
#include <string.h>

const char *s1, *s2;

int ret;

ret=strcmp(s1, s2);
```

パラメタ

No.	名前	型	意味
1	s1	const char 型を指すポインタ	比較される文字列へのポインタ
2	s2	const char 型を指すポインタ	比較する文字列へのポインタ

リターン値

型 : int

正常: s1 で指された文字列 > s2 で指された文字列の時: 正の値

s1 で指された文字列 = s2 で指された文字列の時: 0

s1 で指された文字列 < s2 で指された文字列の時: 負の値

異常: -

s1 で指された文字列と、s2 で指された文字列の内容を比較し、その結果をリターン値として設定します。比較するための基準は処理系定義です。

7.13.8 strncmp 関数

機能

指定された二つの文字列を指定された文字分まで比較します。

呼び出し手順

```
#include <string.h>

const char *s1, *s2;
size_t n;
int ret;

ret=strncmp(s1, s2, n);
```

パラメタ

No.	名前	型	意味
1	s1	const char 型を指すポインタ	比較される文字列へのポインタ
2	s2	const char 型を指すポインタ	比較する文字列へのポインタ
3	n	size_t	比較する文字数の最大値

リターン値

型 : int

正常: s1 で指された文字列 > s2 で指された文字列の時: 正の値

s1 で指された文字列 = s2 で指された文字列の時: 0

s1 で指された文字列 < s2 で指された文字列の時: 負の値

異常: -

s1 で指された文字列と、s2 で指された文字列を最初の n 文字以内の範囲で、その内容を比較します。比較するための基準は処理系定義です。

7.13.9 memchr 関数

機能

指定された記憶域において、指定された文字が最初に現われる位置を検索します。

呼び出し手順

```
#include <string.h>

const void *s;
int c;
size_t n ;
void *ret;

ret=memchr(s, c, n);
```

パラメタ

No.	名前	型	意味
1	s	const void 型を指すポインタ	検索を行う記憶域へのポインタ
2	c	int	検索する文字
3	n	size_t	検索を行う文字数

リターン値

型 : void 型へのポインタ

正常 : 検索の結果見つかった時 : 見つけられた文字へのポインタ

検索の結果見つからなかった時 : NULL

異常 : -

s で指定された記憶域の先頭から n 文字の中で最初に現われた c の文字と同一文字の位置へのポインタをリターン値として返します。

7.13.10 strchr 関数

機能

指定された文字列において、指定された文字が最初に現われる位置を検索します。

呼び出し手順

```
#include <string.h>

const char *s;
int c;
char *ret;

ret=strchr(s, c);
```

パラメタ

No.	名前	型	意味
1	s	const char 型を指すポインタ	検索を行う文字列へのポインタ
2	c	int	検索する文字

リターン値

型 : char 型へのポインタ

正常 : 検索の結果見つかった時 : 見つけられた文字へのポインタ

検索の結果見つからなかった時 : NULL

異常 : -

s で指定された文字列中で最初に現われた c の文字と同一文字へのポインタをリターン値として返します。

【注意】

s によって指される文字列の終了を現わすヌル文字も検索の対象として含まれます。

7.13.11 strcspn 関数

機能

指定された文字列を先頭から調べ、別に指定した文字列中の文字以外の文字が先頭から何文字続くか求めます。

呼び出し手順

```
#include <string.h>

const char *s1, *s2;

size_t ret;

ret=strcspn(s1, s2);
```

パラメタ

No.	名前	型	意味
1	s1	const char 型を指すポインタ	調べられる文字列へのポインタ
2	s2	const char 型を指すポインタ	s1 を調べるための文字列へのポインタ

リターン値

型 : size_t

正常 : s2 が指す文字列を構成する文字以外の文字が構成される文字列 s1 の先頭からの長さ

異常 : -

s2 が指す文字列を構成する文字以外の文字が、文字列として何文字続くかを s1 で指された文字列の先頭から調べ、その文字列の長さをリターン値として返します。

【注意】

s2 によって指される文字列の終了を表わすヌル文字は、s2 で指された文字列の一部とはみなされません。

7.13.12 strpbrk 関数

機能

指定された文字列内において、別に指定された文字列中の文字が最初に現われる位置を検索します。

呼び出し手順

```
#include <string.h>

const char *s1, *s2;

char *ret;

ret=strpbrk(s1, s2);
```

パラメタ

No.	名前	型	意味
1	s1	const char 型を指すポインタ	検索を行う文字列へのポインタ
2	s2	const char 型を指すポインタ	s1 内で検索する文字を示す文字列へのポインタ

リターン値

型 : char 型へのポインタ

正常: 検索の結果見つかった時: 見つかった文字へのポインタ
検索の結果見つからなかった時: NULL

異常: -

s1 で指された文字列において、s2 で指された文字列中の文字の一つが最初に現われる所を検索し、そのポインタをリターン値として返します。

7.13.13 strrchr 関数

機能

指定された文字列において、指定された文字が最後に現われる位置を検索します。

呼び出し手順

```
#include <string.h>

const char *s;
int c;
char *ret;

ret=strrchr(s, c) ;
```

パラメタ

No.	名前	型	意味
1	s	const char 型を指すポインタ	検索を行う文字列へのポインタ
2	c	int	検索する文字

リターン値

型 : **char** 型へのポインタ

正常 : 検索の結果見つかった時 : 見つかった文字へのポインタ

検索の結果見つからなかった時 : **NULL**

異常 : -

s で指された文字列の中で c で指定する文字と同一の文字が最後に現われた位置へのポインタをリターン値として返します。

【注意】

s によって指される文字列の終了を表わすヌル文字も検索の対象として含まれます。

7.13.14 strspn 関数

機能

指定された文字列を先頭から調べ、別に指定した文字列中の文字が先頭から何文字続くかを求めます。

呼び出し手順

```
#include <string.h>

const char *s1, *s2;
size_t      ret;

      ret=strspn(s1, s2);
```

パラメタ

No.	名前	型	意味
1	s1	const char 型を指すポインタ	調べられる文字列へのポインタ
2	s2	const char 型を指すポインタ	s1 を調べるための文字列へのポインタ

リターン値

型 : size_t

正常: s1 の先頭から、s2 で指定した文字が続いている文字数

異常: -

s2 が指す文字列を構成する文字が文字列として何文字続くかを s1 で指された文字列の先頭から調べ、その文字列の長さをリターン値として返します。

7.13.15 strstr 関数

機能

指定された文字列において、別に指定した文字列が最初に現われる位置を検索します。

呼び出し手順

```
#include <string.h>

const char *s1, *s2;
char *ret;

ret=strstr(s1, s2);
```

パラメタ

No.	名前	型	意味
1	s1	const char 型を指すポインタ	検索を行う文字列へのポインタ
2	s2	const char 型を指すポインタ	検索する文字列へのポインタ

リターン値

型 : char 型へのポインタ

正常 : 検索の結果見つかったとき : 見つけられた文字へのポインタ

検索の結果見つからなかったとき : NULL

異常 : -

s1 で指された文字列において、s2 で指された文字列が最初に現われる所を検索し、そのポインタをリターン値として返します。

7.13.16 strtok 関数

機能

指定した文字列をいくつかの字句に切り分けます。

呼び出し手順

```
#include <string.h>

char *s1, *ret;
const char *s2;

ret=strtok(s1, s2);
```

パラメタ

No.	名前	型	意味
1	s1	char 型を指すポインタ	いくつかの字句に切り分ける文字列へのポインタ
2	s2	const char 型を指すポインタ	文字列を切り分けるための文字からなる文字列へのポインタ

リターン値

型 : char 型へのポインタ

正常 : 字句に切り分けられた時 : 切り分けた字句の先頭へのポインタ
字句に切り分けられなかった時 : NULL

異常 : -

strtok 関数は文字列を切り分けるために連続的に呼び出されます。

(1) 最初の呼び出し時

s1 で指された文字列を先頭から s2 で指された文字列中の文字によって字句に切り分けます。その結果字句に切り分けられれば、その字句の先頭へのポインタを、分けられなければ NULL をリターン値として返します。

(2) 2 回目以降の呼び出し時

以前に切り分けられた字句の次の文字から、s2 で指された文字列中の文字によって字句に切り分けます。その結果字句に切り分けられれば、その字句の先頭へのポインタを、分けられなければ NULL をリターン値として返します。

2 回目以降の呼び出しの時は、第 1 パラメタには `NULL` を指定します。また、`s2` で指された文字列は呼び出しのたびに異なってもかまいません。切り出された字句の最後にはヌル文字が付きます。

`strtok` 関数の使用例を以下に示します。

例

```
1  #include <string.h>
2  static char s1[ ]="a@b, @c/@d";
3  char *ret;
4
5  ret = strtok(s1, "@");
6  ret = strtok(NULL, ",@");
7  ret = strtok(NULL, "/@");
8  ret = strtok(NULL, "@");
```

【説明】

この例は、文字列「`" a@b, @c / @d "`」を `strtok` 関数を用いて `a`, `b`, `c`, `d` という字句に切り分けるプログラムを示しています。

2 行目で文字列 `s1` に初期値として、文字列 `" a@b, @c / @d "` を設定しています。

5 行目では、「`@`」を区切り文字として字句を切り分けるため、`strtok` 関数を呼び出します。この結果、文字 '`a`' へのポインタがリターン値として得られ、文字 '`a`' の次の最初の区切り文字である「`@`」にヌル文字を埋め込みます。この結果、文字列 `" a "` が切り出されます。

以下、同一の文字列から次々に字句を切り出すために第 1 引数に `NULL` を指定して `strtok` 関数を呼び出します。

この結果、文字列 `" b "`、`" c "`、`" d "` が次々に切り出されます。

7.13.17 memset 関数

機能

指定された記憶域の先頭から、指定された文字を指定された文字数分設定します。

呼び出し手順

```
#include <string.h>

void *s, *ret;
int c;
size_t n;

ret=memset(s, c, n);
```

パラメタ

No.	名前	型	意味
1	s	void 型を指すポインタ	文字が設定される記憶域へのポインタ
2	c	int	設定する文字
3	n	size_t	設定する文字数

リターン値

型 : void 型へのポインタ

正常 : s の値

異常 : -

s で指された記憶域に n 文字分、文字 c を設定します。

7.13.18 strerror 関数

機能

エラー番号を指定して、それに対するエラーメッセージを返します。

呼び出し手順

```
#include <string.h>

char *ret;

int s;

ret=strerror(s);
```

パラメタ

No.	名前	型	意味
1	s	int	エラー番号

リターン値

型 : char 型へのポインタ

正常 : エラー番号に対応するエラーメッセージ (文字列) へのポインタ

異常 : -

エラー番号 s に対応するエラーメッセージへのポインタをリターン値として返します。
エラーメッセージの内容に関しては処理系定義です。

【注意】

リターン値として返されたエラーメッセージを修正した時、動作は保証されません。

7.13.19 strlen 関数

機能

文字列の長さを計算します。

呼び出し手順

```
#include <string.h>

const char *s;
size_t ret;

ret=strlen(s);
```

パラメタ

No.	名前	型	意味
1	s	const char 型を返すポインタ	長さを求める文字列へのポインタ

リターン値

型 : size_t

正常 : 文字列の文字数

異常 : -

【注意】

s が指す文字列の終了を表わすヌル文字は、文字列の長さとしては計算に入れません。

7.14 < time.h >

機能概要

時間に関する関数類を定義しています。

定義名一覧

定義名	種類	説明
CLK_TCK	マクロ名	1秒間に使用されるプロセッサの時間を示します。本マクロは、clock関数のリターン値を秒に変換する時に使用します。
clock_t	マクロ名	プロセッサの使用時間を表わす型を示します。
time_t	マクロ名	暦時間を表わす型を示します。
struct tm	マクロ名	詳細時間を保存しておくための構造体のタグ名を示します。
clock	関数	現在までの、プロセッサ使用時間を決定します。
difftime	関数	2つの暦時間の差を計算します。
time	関数	現在の暦時間を決定します。
asctime	関数	指定された構造体で表わされる詳細時間を文字列形式に変換します。
ctime	関数	指定された暦時間を文字列形式の地域時間に変換します。
gmtime	関数	指定された暦時間をグリニッジ標準時間（GMT）で表わされる詳細時間に変換します。
localtime	関数	指定された暦時間を、地域時間で表わされる詳細時間に変換します。

上記のマクロ名はすべて処理系定義です。

本標準インクルードファイル内で定義される関数において使用される以下の時間について説明します。

- (1) 暦時間 : 日付と時間からなる西暦
- (2) 地域時間 : ある特定の時間帯を表現するための時間
- (3) 季節時間 : 一時点に変更される時間
- (4) 詳細時間 : 暦時間を表わすために必要な項目（秒、分、時、日、月、年、曜日通算日数、季節時間採用の有無）を保持している時間

7.14.1 clock 関数

機能

現在までのプロセッサ使用時間を決定します。

呼び出し手順

```
#include <time.h>

clock_t ret;

ret=clock( );
```

パラメタ

No.	名前	型	意味
-	-	-	-

リターン値

型 : clock_t

正常 : 処理系定義の時点から現在までのプロセッサの使用時間

異常 : プロセッサの使用時間が有効でない時 : -1

clock 関数は、処理系定義の時点から現在までのプロセッサの使用時間を処理系定義の方法で計算し、その値をリターン値として返します。

もし、リターン値を秒として扱いたい時は、リターン値を CLK_TCK によって除算することによって秒としての値が求められます。

7.14.2 difftime 関数

機能

二つの暦時間の差を計算します。

呼び出し手順

```
#include <time.h>
time_t time1, time2;
double ret;

ret=difftime(time1, time2);
```

パラメタ

No.	名前	型	意味
1	time1	time_t	もともになる暦時間
2	time2	time_t	差を求めるための暦時間

リターン値

型 : **double**

正常 : 二つの暦時間の差

異常 : -

二つの暦時間、**time1**, **time2** において「**time1-time2**」の値を秒単位で計算してリターン値として返します。

7.14.3 time 関数

機能

現在の暦時間を決定します。

呼び出し手順

```
#include <time.h>
```

```
time_t *timer, ret;
```

```
ret=time(timer);
```

パラメタ

No.	名前	型	意味
1	timer	time_t を指すポインタ	現在の暦時間を設定する記憶域へのポインタ

リターン値

型 : time_t

正常 : 現在の暦時間

異常 : 暦時間が有効でない時 : -1

現在の暦時間を処理系定義の方法で求め、リターン値として返します。また、**timer** が **NULL** でない時は、リターン値と同様の値が **timer** が指す記憶域に設定されます。

7.14.4 asctime 関数

機能

指定された構造体で表わされる詳細時間を、文字列の形式に変換します。

呼び出し手順

```
#include <time.h>

char *ret;

const struct tm *timeptr;

ret=asctime(timeptr);
```

パラメタ

No.	名前	型	意味
1	timeptr	const struct tm 型を指す	文字列形式に変換する詳細時間

リターン値

型 : char 型へのポインタ

正常 : timeptr により指される詳細時間を変換した文字列へのポインタ

異常 : -

timeptr で指定された詳細時間を、以下の文字列の形式に変換します。

” 曜日 月 日 時:分:秒 西暦\n\0 ”

例

```
"Mon Mar 01 16:54:10 1987\n\0"
```

7.14.5 ctime 関数

機能

指定された暦時間を文字列形式の地域時間に変換します。

呼び出し手順

```
#include <time.h>

const time_t *timer;

char *ret;

ret=ctime(timer);
```

パラメタ

No.	名前	型	意味
1	timer	const time_t を指すポインタ	文字列形式の地域時間に変換する暦時

リターン値

型 : char 型へのポインタ

正常 : timer により指される暦時間を地域時間に変換した結果の文字列へのポインタ

異常 : -

変換される文字列の形式については、「7.14.4 asctime 関数」を参照してください。

7.14.6 gmtime 関数

機能

指定された暦時間をグリニッジ標準時間 (GMT) で表わされる詳細時間に変換します。

呼び出し手順

```
#include <time.h>

const time_t *timer;

struct tm *ret;

ret=gmtime(timer);
```

パラメタ

No.	名前	型	意味
1	timer	const time_t を指すポインタ	詳細時間に変換する暦時間

リターン値

型 : struct tm で定義された構造体へのポインタ

正常 : timer に対応する GMT で表わされる詳細時間

異常 : GMT が有効でない時 : NULL

7.14.7 localtime 関数

機能

指定された暦時間を、地域時間で表わされる詳細時間に変換します。

呼び出し手順

```
#include <time.h>

const time_t *timer;

struct tm *ret;

ret=localtime(timer);
```

パラメタ

No.	名前	型	意味
1	timer	const time_t を指すポインタ	詳細時間に変換する暦時間

リターン値

型 : struct tm で定義された構造体へのポインタ

正常 : timer に対応する地域時間で表わされる詳細時間

異常 : -

8. DSP ライブラリ

8.1 概要

`shdsplib.lib` は、SH-DSP 用に開発されたデジタル信号処理 (DSP) ライブラリです。本ライブラリは標準的な DSP 関数を含んでおり、単独または連続的に使用することによって、DSP 演算を行なうことができます。

本ライブラリは以下の関数を含んでいます。

- ・ 高速フーリエ変換
- ・ 窓関数
- ・ フィルタ
- ・ 畳み込みと相関
- ・ その他

本ライブラリ関数は高速フーリエ変換とフィルタを除いてリエントラントです。

本ライブラリを使用するときには、`ensigdsp.h` をプログラムの中でインクルードしてください。さらに、本ライブラリに含まれているフィルタ関数を使用するときには、`filt_ws.h` をインクルードしてください。

本ライブラリを呼び出した際、関数が正常に終了した場合は `EDSP_OK` を、異常があった場合は `EDSP_BAD_ARG` もしくは `EDSP_NO_HEAP` をリターン値として返します。リターン値の詳細については各関数の説明を参照してください。

8.2 データフォーマット

本ライブラリはデータを符号付き 16 ビット固定小数点数として扱います。符号付き 16 ビット固定小数点数は図 8-1(a)に示すように、小数点が最上位ビット (MSB) の右側に固定されたデータフォーマットとなっており、 $-1 \sim 1 \cdot 2^{-15}$ の範囲の値を表現できます。

本ライブラリでは、データの受け渡しは short 型のデータフォーマットを使用します。したがって、C/C++ プログラムから本ライブラリを使用する場合、データを符号付き 16 ビット固定小数点数で表現する必要があります。

(例) $+0.5$ は符号付き 16 ビット固定小数点数で表現すると H'4000 です。したがってライブラリ関数に渡す short 型実引数は H'4000 となります。

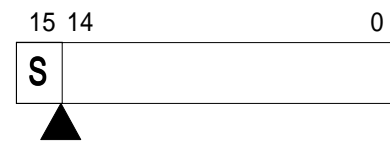
本ライブラリ内部の演算では、符号付き 32 ビット固定小数点数と符号付き 40 ビット固定小数点数も使用します。符号付き 32 ビット固定小数点数は図 8-1(b)に示すデータフォーマットとなっており、 $-1 \sim 1 \cdot 2^{-31}$ の範囲の値を表現できます。符号付き 40 ビット固定小数点数は図 8-1(c)に示すように 8 ビットのガードビットが付加されたデータフォーマットとなっており、 $-2^8 \sim 2^8 \cdot 2^{-31}$ の範囲の値を表現できます。

符号付き 16 ビット固定小数点数の乗算結果は符号付き 32 ビット固定小数点数で保持します。DSP 命令を用いた固定小数点乗算では、H'8000 \times H'8000 の場合だけオーバーフローが発生することに注意してください。また乗算結果の最下位ビット (LSB) は常に 0 になります。乗算結果を次の演算に使用する場合、上位 16 ビットを取り出し、符号付き 16 ビット固定小数点数に変換します。このときアンダーフローや精度低下が発生する可能性があります。

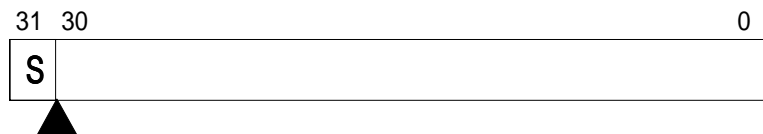
本ライブラリの積和演算では、加算結果を符号付き 40 ビット固定小数点数で保持します。加算のときにオーバーフローが発生しないように注意してください。

演算の際、オーバーフローが発生すると正しい結果が得られません。オーバーフローを防ぐためには、係数や入力データをスケーリングする必要があります。本ライブラリには、スケーリングの機能が組み込まれています。スケーリングの詳細については各関数の説明を参照してください。

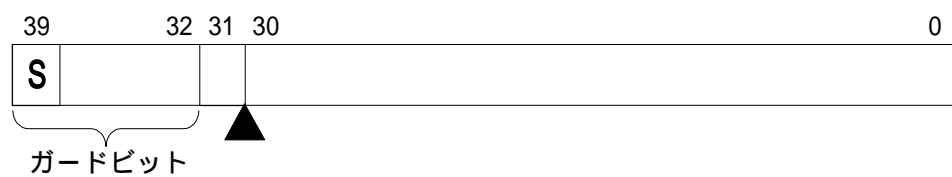
- (a) 符号付き16ビット固定小数点数
($-1 \sim 1 \cdot 2^{-15}$)



- (b) 符号付き32ビット固定小数点数
($-1 \sim 1 \cdot 2^{-31}$)



- (c) 符号付き40ビット固定小数点数
($-2^8 \sim 2^8 \cdot 2^{-31}$)



S : 符号ビット
: 小数点

図8-1 データフォーマット

8.3 効率

本ライブラリ関数は、SH-DSP 上で高速に実行するように最適化しています。

ライブラリを効率よく活用するために、開発するシステムのメモリマップを決める際には、できるだけ以下の 2 つの推奨事項に従ってください。

- ・ プログラムコードセグメントは、1 サイクルでの 32 ビットリードをサポートしているメモリに配置する。
- ・ データセグメントは、1 サイクルでの 16 (または 32) ビットリード・ライトをサポートしているメモリに配置する。

使用するマイコンが、ライブラリコードとデータを配置するのに十分な容量の 32 ビットメモリを内蔵している場合は、その 32 ビットメモリに配置するのが最適です。その他のメモリを使用しなければならない場合は、可能な限り上記の推奨事項に従ってください。

8.4 高速フーリエ変換

8.4.1 概要

8.4.1.1 関数一覧

- ・ **FftComplex** **not-in-place** 複素数 FFT を実行します。
- ・ **FftReal** **not-in-place** 実数 FFT を実行します。
- ・ **IfftComplex** **not-in-place** 複素数逆 FFT を実行します。
- ・ **IfftReal** **not-in-place** 実数逆 FFT を実行します。
- ・ **FftInComplex** **in-place** 複素数 FFT を実行します。
- ・ **FftInReal** **in-place** 実数 FFT を実行します。
- ・ **IfftInComplex** **in-place** 複素数逆 FFT を実行します。
- ・ **IfftInReal** **in-place** 実数逆 FFT を実行します。
- ・ **LogMagnitude** 複素数データを対数絶対値に変換します。
- ・ **InitFft** FFT 回転係数を生成します。
- ・ **FreeFft** FFT 回転係数の格納に使用したメモリを解放します。

not-in-place、**in-place** については「8.4.1.5 FFT 構造」を参照してください。

これらの関数は、ユーザが定義したスケーリングを使って、順方向高速フーリエ変換と逆方向高速フーリエ変換を実行します。

順方向フーリエ変換は以下の式で定義されます。

$$y_n = 2^{-s} \sum_{n=0}^N e^{-2j\pi n/N} \cdot x_n$$

ここで、 s はスケーリングが行なわれるステージの数、 N はデータ数を示しています。

逆方向フーリエ変換は以下の式で定義されます。

$$y_n = 2^{-s} \sum_{n=0}^N e^{2j\pi n/N} \cdot x_n$$

スケーリングについては「8.4.1.4 スケーリング」を参照してください。

8.4.1.2 複素数データ配列フォーマット

FFT および IFFT の複素数データ配列は、実数を X メモリに、虚数を Y メモリに配置します。ただし、実数 FFT の出力データと実数 IFFT の入力データの配置は異なります。実数、虚数を格納する配列をそれぞれ x, y とすると、 $x[0]$ には DC 成分の実数成分が入り、 $y[0]$ には DC 成分の虚数成分ではなく $F_s/2$ 成分の実数成分が入ります（DC 成分と $F_s/2$ 成分はどちらも実数で、虚数成分は 0 です）。

8.4.1.3 実数データ配列フォーマット

FFT および IFFT の実数データ配列フォーマットには、以下の 3 種類があります。

- ・ 単一の配列に格納し、任意のメモリブロックに配置。
- ・ 単一の配列に格納し、X メモリに配置。
- ・ 2 つの配列に分けて格納。それぞれの配列のサイズは $N/2$ で、配列の前半は X メモリに配置し、後半は Y メモリに配置。

FftReal は 1 番目の指定方法のみです。IfftReal、FftInReal および IfftInReal は 2 番目か 3 番目の方法をユーザが選択します。

8.4.1.4 スケーリング

基数 2 の FFT は各ステージで信号強度が倍になり、ピーク信号振幅も倍になります。そのため、高強度信号を変換する際にオーバーフローが発生することがありますが、各ステージで信号を $1/2$ にすることにより（これをスケーリングといいます）オーバーフローを防ぐことができます。しかし、スケーリングしすぎると不要な量子化雑音が発生する可能性があります。

オーバーフローや量子化雑音とスケーリングの最適なバランスは入力信号の特性に大きく依存します。スペクトルが大きなピークを持つ信号はオーバーフローを防ぐために最大スケーリングが必要になりますが、インパルス信号ではスケーリングの必要はほとんどありません。

すべてのステージでスケーリングするのが最も安全な方法です。入力データが強度 2^{30} 未満であれば、この方法でオーバーフローを防ぐことができます。また本ライブラリでは、各ステージごとにスケーリングを行なうかどうかを指定できます。したがって、スケーリング指定を精密に行なうことによって、オーバーフローと量子化雑音の影響を最小限に抑えることができます。

スケーリングの方法を指定するために、各 FFT 関数の引数に `scale` が含まれています。`scale` は最下位ビットから 1 ビットずつが各ステージに対応しています。対応する `scale` のビットが 1 に設定されているすべてのステージで、2 の除算を実行します。

本ライブラリは実行速度を上げるために基数 4 の FFT を使用しています。`scale` は最下位ビットから 2 ビットずつが各ステージに対応しています。どちらか 1 ビットが 1 に設定されていれば、2 の除算を実行します。両方が 1 に設定されていれば 4 の除算を実行します。つまり、2 つの基数 2 の FFT ステージが 1 つの基数 4 の FFT ステージに置き換えられたのと同じことになります。しかし、基数 2 の FFT よりも基数 4 の FFT の方が量子化雑音の発生する可能性があります。

以下に `scale` の例を示します。

- `scale = H'FFFFFFFF` (または `size-1`) はすべての基数 2 の FFT ステージでスケールリングを行いません。すべての入力データの強度が 2^{30} 未満であれば、オーバーフローは発生しません。
- `scale = H'55555555` は 1 つおきの基数 2 の FFT ステージでスケールリングを行いません。
- `scale = 0` はスケールリングを行いません。

これらの `scale` の値は、`ensigdsp.h` で `EFFTALLSCALE (H'FFFFFFFF)`、`EFFTMIDSCALE (H'55555555)`、`EFFTNOSCALE (0)` と定義されています。

8.4.1.5 FFT 構造

本ライブラリの FFT 構造には `not-in-place FFT` と `in-place FFT` の 2 種類があります。

`not-in-place FFT` では、入力データを RAM から取り出し、FFT を実行し、出力結果を RAM のユーザが指定した別の場所に格納します。

一方 `in-place FFT` では、入力データを RAM から取り出し、FFT を実行し、出力結果を RAM の同じ場所に格納します。この方法を用いると FFT の実行時間は増加しますが、使用メモリスペースが削減できます。

入力データを FFT 関数の他にも使用する場合は、`not-in-place FFT` を使用してください。また、メモリスペースを節約したい場合は、`in-place FFT` を使用してください。

8.4.2 各関数の説明

8.4.2.1 FftComplex 関数

(1) 定義

```
int FftComplex (short op_x[ ], short op_y[ ], const short ip_x[ ], const short ip_y[ ],  
                long size, long scale)
```

(2) 引数

op_x[]	出力データの実数成分
op_y[]	出力データの虚数成分
ip_x[]	入力データの実数成分
ip_y[]	入力データの虚数成分
size	FFT のサイズ
scale	スケーリング指定

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none">• size < 4• size が 2 の累乗ではありません• size > max_fft_size

(4) 説明

本関数は複素数高速フーリエ変換を実行します。本関数は **not-in-place** で行ないますので、入力配列と出力配列を別々に用意してください。

(5) 補足事項

1. 複素数データ配列の配置については「8.4.1.2 複素数データ配列フォーマット」を参照してください。
2. 本関数を呼び出す前に `InitFft` を呼び出して、回転係数と `max_fft_size` を初期化してください。
3. スケーリング指定については「8.4.1.4 スケーリング」を参照してください。
4. `scale` は下位 $\log_2(\text{size})$ ビットを使用します。
5. 本関数はリエントラントではありません。

8.4.2.2 FftReal 関数

(1) 定義

```
int FftReal (short op_x[ ], short op_y[ ], const short ip[ ], long size, long scale)
```

(2) 引数

op_x[]	正の出力データの実数成分
op_y[]	正の出力データの虚数成分
ip[]	実数入力データ
size	FFT のサイズ
scale	スケーリング指定

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none">• size < 8• size が 2 の累乗ではありません• size > max_fft_size

(4) 説明

本関数は実数高速フーリエ変換を実行します。

op_x と **op_y** には **size/2** の正の出力データが格納されます。負の出力データは正の出力データの共役複素数です。また、0 と $F_s/2$ での出力データの値は実数なので、 $F_s/2$ の実数出力は **op_y[0]** に格納されます。

本関数は **not-in-place** で行ないますので、入力配列と出力配列を別々に用意してください。

(5) 補足事項

1. 複素数と実数データ配列の配置については「8.4.1.2 複素数データ配列フォーマット」「8.4.1.3 実数データ配列フォーマット」を参照してください。
2. 本関数を呼び出す前に `InitFft` を呼び出して、回転係数と `max_fft_size` を初期化してください。
3. スケーリング指定については「8.4.1.4 スケーリング」を参照してください。
4. `scale` は下位 $\log_2(\text{size})$ ビットを使用します。
5. 本関数はリエントラントではありません。

8.4.2.3 IfftComplex 関数

(1) 定義

```
int IfftComplex (short op_x[ ], short op_y[ ], const short ip_x[ ], const short ip_y[ ],  
                 long size, long scale)
```

(2) 引数

op_x[]	出力データの実数成分
op_y[]	出力データの虚数成分
ip_x[]	入力データの実数成分
ip_y[]	入力データの虚数成分
size	逆 FFT のサイズ
scale	スケーリング指定

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none">• size < 4• size が 2 の累乗ではありません• size > max_fft_size

(4) 説明

本関数は複素数逆高速フーリエ変換を実行します。本関数は **not-in-place** で行ないますので、入力配列と出力配列を別々に用意してください。

(5) 補足事項

1. 複素数データ配列の配置については「8.4.1.2 複素数データ配列フォーマット」を参照してください。
2. 本関数を呼び出す前に **InitFft** を呼び出して、回転係数と **max_fft_size** を初期化してください。
3. スケーリング指定については「8.4.1.4 スケーリング」を参照してください。
4. **scale** は下位 $\log_2(\text{size})$ ビットを使用します。
5. 本関数はリエントラントではありません。

8.4.2.4 IfftReal 関数

(1) 定義

```
int IfftReal (short op_x[ ], short scratch_y[ ], const short ip_x[ ], const short ip_y[ ],
             long size, long scale, int op_all_x)
```

(2) 引数

op_x[]	実数出力データ
scratch_y[]	スクラッチメモリまたは実数出力データ
ip_x[]	正の入力データの実数成分
ip_y[]	正の入力データの虚数成分
size	逆 FFT のサイズ
scale	スケーリング指定
op_all_x	出力データの配置指定

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • size < 8 • size が 2 の累乗ではありません • size > max_fft_size • op_all_x ≠ 0 または 1

(4) 説明

本関数は実数逆高速フーリエ変換を実行します。

ip_x と ip_y には size/2 の正の入力データを格納してください。負の入力データは正の入力データの共役複素数です。また、0 と $F_s/2$ での入力データの値は実数なので、 $F_s/2$ での実数入力はい `ip_y[0]` に格納してください。

出力データのフォーマットは op_all_x で指定します。op_all_x=1 の場合、全出力データは op_x に格納されます。op_all_x=0 の場合、最初の size/2 の出力データは op_x に格納され、残りの size/2 の出力データは scratch_y に格納されます。

本関数は not-in-place で行ないますので、入力配列と出力配列を別々に用意してください。

(5) 補足事項

1. 複素数と実数データ配列の配置については「8.4.1.2 複素数データ配列フォーマット」「8.4.1.3 実数データ配列フォーマット」を参照してください。
2. `ip_x`、`ip_y` はそれぞれ `size/2` のデータを格納してください。`op_x` は `op_all_x` の値によって、`size` または `size/2` のデータが格納されます。
3. 本関数を呼び出す前に `InitFft` を呼び出して、回転係数と `max_fft_size` を初期化してください。
4. スケーリング指定については「8.4.1.4 スケーリング」を参照してください。
5. `scale` は下位 $\log_2(\text{size})$ ビットを使用します。
6. 本関数はリエントラントではありません。

8.4.2.5 FftInComplex 関数

(1) 定義

```
int FftInComplex (short data_x[ ], short data_y[ ], long size, long scale)
```

(2) 引数

data_x[]	入出力データの実数成分
data_y[]	入出力データの虚数成分
size	FFT のサイズ
scale	スケーリング指定

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • size < 4 • size が 2 の累乗ではありません • size > max_fft_size

(4) 説明

本関数は **in-place** 複素数高速フーリエ変換を実行します。

(5) 補足事項

1. 複素数データ配列の配置については「8.4.1.2 複素数データ配列フォーマット」を参照してください。
2. 本関数を呼び出す前に **InitFft** を呼び出して、回転係数と **max_fft_size** を初期化してください。
3. スケーリング指定については「8.4.1.4 スケーリング」を参照してください。
4. **scale** は下位 $\log_2(\text{size})$ ビットを使用します。
5. 本関数はリエントラントではありません。

8.4.2.6 FftInReal 関数

(1) 定義

```
int FftInReal (short data_x[ ], short data_y[ ], long size, long scale, int ip_all_x)
```

(2) 引数

data_x[]	入力時は実数データ、出力時は正の出力データの実数成分
data_y[]	入力時は実数データまたは未使用、出力時は正の出力データの虚数成分
size	FFT のサイズ
scale	スケーリング指定
ip_all_x	入力データの配置指定

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none">• size < 8• size が 2 の累乗ではありません• size > max_fft_size• ip_all_x ≠ 0 または 1

(4) 説明

本関数は **in-place** 実数高速フーリエ変換を実行します。

入力データのフォーマットは、**ip_all_x** で指定します。**ip_all_x=1** の場合、全入力データは **data_x** から取り出します。**ip_all_x=0** の場合、前半の **size/2** の入力データは **data_x** から、後半の **size/2** の入力データは **data_y** から取り出します。

本関数実行後、**data_x** と **data_y** には **size/2** の正の出力データが格納されます。負の出力データは正の出力データの共役複素数です。また、0 と $F_s/2$ での出力データの値は実数なので、 $F_s/2$ での実数出力は **data_y[0]** に格納されます。

(5) 補足事項

1. 複素数と実数データ配列の配置については「8.4.1.2 複素数データ配列フォーマット」「8.4.1.3 実数データ配列フォーマット」を参照してください。
2. `data_y` は `size/2` のデータを格納します。`data_x` は `ip_all_x` の値によって `size` または `size/2` のデータを格納します。
3. 本関数を呼び出す前に `InitFft` を呼び出して、回転係数と `max_fft_size` を初期化してください。
4. スケーリング指定については「8.4.1.4 スケーリング」を参照してください。
5. `scale` は下位 $\log_2(\text{size})$ ビットを使用します。
6. 本関数はリエントラントではありません。

8.4.2.7 IfftInComplex 関数

(1) 定義

```
int IfftInComplex (short data_x[ ], short data_y[ ], long size, long scale)
```

(2) 引数

data_x[]	入出力データの実数成分
data_y[]	入出力データの虚数成分
size	逆 FFT のサイズ
scale	スケーリング指定

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none">• size < 4• size が 2 の累乗ではありません• size > max_fft_size

(4) 説明

本関数は **in-place** 複素数逆高速フーリエ変換を実行します。

(5) 補足事項

1. 複素数データ配列の配置については「8.4.1.2 複素数データ配列フォーマット」を参照してください。
2. 本関数を呼び出す前に `InitFft` を呼び出して、回転係数と `max_fft_size` を初期化してください。
3. スケーリング指定については「8.4.1.4 スケーリング」を参照してください。
4. `scale` は下位 $\log_2(\text{size})$ ビットを使用します。
5. 本関数はリエントラントではありません。

8.4.2.8 IfftInReal 関数

(1) 定義

```
int IfftInReal (short data_x[ ], short data_y[ ], long size, long scale, int op_all_x)
```

(2) 引数

data_x[]	入力時は正の入力データの実数成分、出力時は実数データ
data_y[]	入力時は正の入力データの虚数成分、出力時は実数データまたは未使用
size	逆 FFT のサイズ
scale	スケーリング指定
op_all_x	出力データの配置指定

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • size < 8 • size が 2 の累乗ではありません • size > max_fft_size • op_all_x ≠ 0 または 1

(4) 説明

本関数は **in-place** 実数逆高速フーリエ変換を実行します。

data_x と **data_y** には **size/2** の正の入力データを格納してください。負の入力データは正の入力データの共役複素数です。また、0 と $F_s/2$ での入力データの値は実数なので、 $F_s/2$ での実数入力 **data_y[0]** に格納してください。

出力データのフォーマットは **op_all_x** で指定します。**op_all_x=1** の場合、全出力データは **data_x** に格納されます。**op_all_x=0** の場合、前半の **size/2** の出力データは **data_x** に格納され、後半の **size/2** の出力データは **data_y** に格納されます。

(5) 補足事項

1. 複素数と実数データ配列の配置については「8.4.1.2 複素数データ配列フォーマット」「8.4.1.3 実数データ配列フォーマット」を参照してください。
2. `data_y` は `size/2` のデータを格納します。`data_x` は、`op_all_x` の値によって `size` または `size/2` のデータが格納されます。
3. 本関数を呼び出す前に `InitFft` を呼び出して、回転係数と `max_fft_size` を初期化してください。
4. スケーリング指定については「8.4.1.4 スケーリング」を参照してください。
5. `scale` は下位 $\log_2(\text{size})$ ビットを使用します。
6. 本関数はリエントラントではありません。

8.4.2.9 LogMagnitude 関数

(1) 定義

```
int LogMagnitude (short output[ ], const short ip_x[ ], const short ip_y[ ],
                  long no_elements, float fscale)
```

(2) 引数

output[] 実数出力 z
 ip_x[] 入力の実数成分 x
 ip_y[] 入力の虚数成分 y
 no_elements 出力データ数 N
 fscale 出力スケール係数

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_elements < 1 • no_elements > 32767 • $fscale \geq 2^{15} / (10 \log_{10} 2^{31})$

(4) 説明

本関数は、複素数入力データの対数絶対値をデシベル単位で計算し、スケール結果を出力配列に書き込みます。

$$z(n) = 10 \text{ fscale} \cdot \log_{10} (x(n)^2 + y(n)^2) \quad 0 \leq n < N$$

(5) 補足事項

1. 複素数データ配列の配置については「8.4.1.2 複素数データ配列フォーマット」を参照してください。

8.4.2.10 InitFft 関数

(1) 定義

```
int InitFft (long max_size)
```

(2) 引数

max_size 必要になる FFT の最大サイズ

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_NO_HEAP	malloc で確保できるメモリスペースが不十分
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none">• max_size < 2• max_size が 2 の累乗ではありません• max_size > 32768

(4) 説明

本関数は FFT 関数で使用する回転係数 (1/4 サイズ) を生成します。回転係数は malloc によって確保されるメモリに格納されます。

回転係数が生成されると max_fft_size グローバル変数が更新されます。max_fft_size は FFT の最大許容サイズを示します。

本関数は最初の FFT 関数を呼び出す前に必ず一度呼び出してください。

(5) 補足事項

1. 回転係数は max_size で指定した変換サイズで生成されます。max_size より小さいサイズの FFT 関数を実行したときも同じ回転係数を使用します。
2. 回転係数のアドレスは内部変数内に格納されています。ここはユーザプログラムでアクセスしないでください。
3. 本関数はリエントラントではありません。

8.4.2.11 FreeFft 関数

(1) 定義

void **FreeFft** (void)

(2) 引数

なし

(3) リターン値

なし

(4) 説明

本関数は回転係数の格納に使用したメモリを解放し、`max_fft_size` グローバル変数を 0 にします。**FreeFft** を実行した後再び **FFT** 関数を実行するときには、その前に必ず **InitFft** を実行してください。

(5) 補足事項

1. 本関数はリエントラントではありません。

8.5 窓関数

8.5.1 概要

8.5.1.1 関数一覧

- ・ **GenBlackman** ブラックマン窓を生成します。
- ・ **GenHamming** ハミング窓を生成します。
- ・ **GenHanning** ハニング窓を生成します。
- ・ **GenTriangle** 三角窓を生成します。

8.5.2 各関数の説明

8.5.2.1 GenBlackman 関数

(1) 定義

int **GenBlackman** (short output[], long win_size)

(2) 引数

output[] 出力データ $w(n)$
win_size 窓サイズ N

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	$\text{win_size} \leq 1$

(4) 説明

本関数はブラックマン窓を生成し、**output** に出力します。実際のデータにこの窓をかけるときは **VectorMult** を使用します。

使用する関数を以下に示します。

$$W(n) = (2^{15} - 1) \left[0.42 - 0.5 \cos\left(\frac{2\pi n}{N}\right) + 0.08 \cos\left(\frac{4\pi n}{N}\right) \right] \quad 0 \leq n < N$$

8.5.2.2 GenHamming 関数

(1) 定義

int **GenHamming** (short output[], long win_size)

(2) 引数

output[] 出力データ $w(n)$
win_size 窓サイズ N

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	$\text{win_size} \leq 1$

(4) 説明

本関数はハミング窓を生成し、**output** に出力します。実際のデータにこの窓をかけるときは **VectorMult** を使用します。

使用する関数を以下に示します。

$$W(n) = (2^{15} - 1) \left[0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right) \right] \quad 0 \leq n < N$$

8.5.2.3 GenHanning 関数

(1) 定義

```
int GenHanning (short output[ ], long win_size)
```

(2) 引数

output[] 出力データ $w(n)$
win_size 窓サイズ N

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	$\text{win_size} \leq 1$

(4) 説明

本関数はハニング窓を生成し、**output** に出力します。実際のデータにこの窓をかけるときは **VectorMult** を使用します。

使用する関数を以下に示します。

$$W(n) = \left(\frac{2^{15} - 1}{2} \right) \left[1 - \cos \left(\frac{2\pi n}{N} \right) \right] \quad 0 \leq n < N$$

8.5.2.4 GenTriangle 関数

(1) 定義

int **GenTriangle** (short output[], long win_size)

(2) 引数

output[] 出力データ $w(n)$
win_size 窓サイズ N

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	$\text{win_size} \leq 1$

(4) 説明

本関数は三角窓を生成し、**output** に出力します。実際のデータにこの窓をかけるときは **VectorMult** を使用します。

使用する関数を以下に示します。

$$W(n) = (2^{15} - 1) \left[1 - \left| \frac{2n - N + 1}{N + 1} \right| \right] \quad 0 \leq n < N$$

8.6. フィルタ

8.6.1 概要

8.6.1.1 関数一覧

- ・ **Fir** 有限インパルス応答フィルタ処理を実行します。
- ・ **Fir1** 単一データ用有限インパルス応答フィルタ処理を実行します。
- ・ **Iir** 無限インパルス応答フィルタ処理を実行します。
- ・ **Iir1** 単一データ用無限インパルス応答フィルタ処理を実行します。
- ・ **Dfir** 倍精度無限インパルス応答フィルタ処理を実行します。
- ・ **Dfir1** 単一データ用倍精度無限インパルス応答フィルタ処理を実行します。
- ・ **Lms** 適応 FIR フィルタ処理を実行します。
- ・ **Lms1** 単一データ用適応 FIR フィルタ処理を実行します。
- ・ **InitFir** FIR フィルタ用に作業領域を割り付けます。
- ・ **InitIir** IIR フィルタ用に作業領域を割り付けます。
- ・ **InitDfir** D FIR フィルタ用に作業領域を割り付けます。
- ・ **InitLms** LMS フィルタ用に作業領域を割り付けます。
- ・ **FreeFir** **InitFir** で割り付けられた作業領域を解放します。
- ・ **FreeIir** **InitIir** で割り付けられた作業領域を解放します。
- ・ **FreeDfir** **InitDfir** で割り付けられた作業領域を解放します。
- ・ **FreeLms** **InitLms** で割り付けられた作業領域を解放します。

フィルタ関数を使用するプログラムでは `filt_ws.h` をインクルードしてください。

8.6.1.2 係数のスケーリング

フィルタ処理を行なうと飽和または量子化雑音が発生する可能性があります。これらはフィルタ係数のスケーリングを行なうことによって最小限に抑えることができます。しかし、飽和と量子化雑音の影響をよく考えてスケーリングを行なわなければなりません。係数が大きすぎると飽和が、小さすぎると量子化雑音が発生する可能性があります。

FIR (有限インパルス応答) フィルタの場合、以下の式が成り立つようにフィルタ係数を設定すれば飽和は起こりません。

$$\begin{aligned} \text{coeff}[i] &\neq H \cdot 8000 \quad (\text{すべての } i \text{ について}) \\ \sum |\text{coeff}| &< 2^{24} \\ \text{res_shift} &= 24 \end{aligned}$$

`coeff` はフィルタ係数、`res_shift` は出力で行なわれる右シフトのビット数です。

しかし、多くの入力信号の場合、もっと小さい `res_shift` の値 (またはもっと大きな `coeff` の値) を使用しても飽和する可能性は少なく、量子化雑音も大幅に削減できます。また

入力値に **H'8000** が含まれている可能性があれば、すべての **coeff** の値は **H'8001 ~ H'7FFF** の範囲になるように設定してください。

IIR（無限インパルス応答）フィルタは再帰的な構造になっています。そのため上述したようなスケーリング方法は適していません。

LMS（最小 2 乗平均）適応フィルタは **FIR** フィルタと同様です。しかし、係数を適応するときに飽和を引き起こす場合があります。その場合は、係数を **H'8000** が含まないように設定してください。

8.6.1.3 作業領域

デジタルフィルタでは、ある処理から次の処理へ保持しておかなければならない情報があります。これらの情報は、最小オーバーヘッドでアクセスすることができるメモリに格納します。本ライブラリでは、**Y-RAM** 領域を作業領域として使用します。作業領域はフィルタ処理を実行する前に **Init** 関数を呼び出して初期化してください。

作業領域メモリはライブラリ関数によってアクセスされます。なお、ユーザプログラムから作業領域を直接アクセスしないでください。

8.6.1.4 メモリの使用

SH-DSP を効率よく使うために、フィルタ係数は **X** メモリに、作業領域は **Y** メモリに配置してください。入出力データは任意のメモリセグメントに配置することができます。

フィルタ係数は **#pragma section** 命令を用いて **X** メモリに配置してください。

各フィルタは **Init** 関数を用いてグローバルバッファから作業領域を割り付けます。グローバルバッファは **Y** メモリに配置します。

8.6.2 各関数の説明

8.6.2.1 Fir 関数

(1) 定義

```
int Fir (short output[ ], const short input[ ], long no_samples, const short coeff[ ],
         long no_coefs, int res_shift, short *workspace)
```

(2) 引数

output[]	出力データ y
input[]	入力データ x
no_samples	入力データの数 N
coeff[]	フィルタ係数 h
no_coefs	係数の数 (フィルタの長さ) K
res_shift	各出力に適用される右シフト
*workspace	作業領域へのポインタ

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_samples < 1 • no_coefs ≤ 2 • res_shift < 0 • res_shift > 25

(4) 説明

本関数は有限インパルス応答 (FIR) フィルタ処理を実行します。最新の入力データは作業領域に保持されます。input をフィルタ処理した結果は output に書き込まれます。

$$y(n) = \left[\sum_{k=0}^{K-1} h(k) x(n-k) \right] \cdot 2^{-\text{res_shift}}$$

積和演算の結果は 39 ビットで保持されます。出力 $y(n)$ は res_shift ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバーフローしたときは正または負の最大値となります。

(5) 補足事項

1. 係数のスケーリングについては「8.6.1.2 係数のスケーリング」を参照してください。
2. 本関数を呼び出す前に **InitFir** を呼び出し、フィルタの作業領域を初期化してください。
3. **output** に **input** と同じ配列を指定した場合、**input** は上書きされます。
4. 本関数はリエントラントではありません。

8.6.2.2 Fir1 関数

(1) 定義

```
int Fir1 (short *output, short input, const short coeff[ ], long no_coefs,
          int res_shift, short *workspace)
```

(2) 引数

*output	出力データ $y(n)$ へのポインタ
input	入力データ $x(n)$
coeff[]	フィルタ係数 h
no_coefs	係数の数 (フィルタの長さ) K
res_shift	各出力に適用される右シフト
*workspace	作業領域へのポインタ

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • $\text{no_coefs} \leq 2$ • $\text{res_shift} < 0$ • $\text{res_shift} > 25$

(4) 説明

本関数は単一データ用に有限インパルス応答 (**FIR**) フィルタ処理を実行します。最新の入力データは作業領域に保持されます。**input** をフィルタ処理した結果は ***output** に書き込まれます。

$$y(n) = \left[\sum_{k=0}^{K-1} h(k) x(n-k) \right] \cdot 2^{-\text{res_shift}}$$

積和演算の結果は 39 ビットで保持されます。出力 $y(n)$ は **res_shift** ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバーフローしたときは正または負の最大値となります。

(5) 補足事項

1. 係数のスケーリングについては「8.6.1.2 係数のスケーリング」を参照してください。
2. 関数を呼び出す前に **InitFir** を呼び出し、フィルタの作業領域を初期化してください。
3. 本関数はリエントラントではありません。

8.6.2.3 IIR 関数

(1) 定義

```
int IIR (short output[ ], const short input[ ], long no_samples, const short coeff[ ],
        long no_sections, short *workspace)
```

(2) 引数

output[] 出力データ y_{k-1}
input[] 入力データ x_0
no_samples 入力データの数 N
coeff[] フィルタ係数
no_sections 2 次フィルタセクションの数 K
*workspace 作業領域へのポインタ

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_samples < 1 • no_sections < 1 • $a_{0k} < 0$ • $a_{0k} > 16$

(4) 説明

本関数は無限インパルス応答 (IIR) フィルタ処理を実行します。

フィルタは、バйкаッドという 2 次フィルタを K 個縦列に接続した構成になっています。各バйкаッドの出力で付加的なスケールリングが行なわれます。フィルタ係数は符号付き 16 ビット固定小数点数で指定します。

各バйкаッドの出力は以下の式で与えられます。

$$d_k(n) = [a_{1k}d_k(n-1) + a_{2k}d_k(n-2) + 2^{15}x(n)] \cdot 2^{-15}$$

$$y_k(n) = [b_{0k}d_k(n) + b_{1k}d_k(n-1) + b_{2k}d_k(n-2)] \cdot 2^{-a_{0k}}$$

k 番目のセクションの入力 $x_k(n)$ は、前のセクションの出力 $y_{k-1}(n)$ です。最初のセクション ($k=0$) の入力 は input から読み込まれます。最後のセクション ($k=K-1$) の出力は output に書き込まれます。

coeff は係数を以下の順序に設定してください。

$a_{00}, a_{10}, a_{20}, b_{00}, b_{10}, b_{20}, a_{01}, a_{11}, a_{21}, b_{01} \dots b_{2K-1}$

a_{0k} 項は **k** 番目のバイカッドの出力で行なわれる右シフトのビット数です。

各バイカッドでは飽和演算を 32 ビットで行ないます。各バイカッドの出力は 15 ビットまたは **a_{0k}** ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバーフローしたときは正または負の最大値となります。

(5) 補足事項

1. 本関数を呼び出す前に **InitIir** を呼び出し、フィルタの作業領域を初期化してください。
2. **output** に **input** と同じ配列を指定した場合、**input** は上書きされます。
3. 本関数はリエントラントではありません。

8.6.2.4 lir1 関数

(1) 定義

```
int lir1 (short *output, short input, const short coeff[ ], long no_sections,
          short *workspace)
```

(2) 引数

*output 出力データ $y_{k-1}(n)$ へのポインタ
input 入力データ $x_0(n)$
coeff[] フィルタ係数
no_sections 2 次フィルタセクションの数 **K**
*workspace 作業領域へのポインタ

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_sections < 1 • $a_{0k} < 0$ • $a_{0k} > 16$

(4) 説明

本関数は単一データ用に無限インパルス応答 (**IIR**) フィルタ処理を実行します。

フィルタは、バイカッドという 2 次フィルタを **K** 個縦列に接続した構成になっています。各バイカッドの出力で付加的なスケールが行なわれます。フィルタ係数は符号付き 16 ビット固定小数点数で指定します。

各バイカッドの出力は以下の式で与えられます。

$$d_k(n) = [a_{1k}d_k(n-1) + a_{2k}d_k(n-2) + 2^{15}x(n)] \cdot 2^{-15}$$

$$y_k(n) = [b_{0k}d_k(n) + b_{1k}d_k(n-1) + b_{2k}d_k(n-2)] \cdot 2^{-a_{0k}}$$

k 番目のセクションの入力 $x_k(n)$ は、前のセクションの出力 $y_{k-1}(n)$ です。最初のセクション (**k=0**) の入力 **input** から読み込まれます。最後のセクション (**k=K-1**) の出力は **output** に書き込まれます。

coeff は係数を以下の順序に設定してください。

$a_{00}, a_{10}, a_{20}, b_{00}, b_{10}, b_{20}, a_{01}, a_{11}, a_{21}, b_{01} \dots b_{2K-1}$

a_{0k} 項は k 番目のバイカッドの出力で行なわれる右シフトのビット数です。

各バイカッドでは飽和演算を 32 ビットで行ないます。各バイカッドの出力は 15 ビットまたは a_{0k} ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバーフローしたときは正または負の最大値となります。

(5) 補足事項

1. 本関数を呼び出す前に **InitIir** を呼び出し、フィルタの作業領域を初期化してください。
2. 本関数はリエントラントではありません。

8.6.2.5 Dllr 関数

(1) 定義

```
int Dllr (short output[ ], const short input[ ], long no_samples, const long coeff[ ],
          long no_sections, long *workspace)
```

(2) 引数

output[] 出力データ y_{k-1}
input[] 入力データ x
no_samples 入力データの数 N
coeff[] フィルタ係数
no_sections 2 次フィルタセクションの数 K
*workspace 作業領域へのポインタ

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_samples < 1 • no_sections < 1 • $a_{0k} < 3$ • $k < K-1$ で $a_{0k} > 32$ • $k = K-1$ で $a_{0k} > 48$

(4) 説明

本関数は倍精度無限インパルス応答フィルタ処理を実行します。

フィルタは、バйкаッドという 2 次フィルタを K 個縦列に接続した構成になっています。各バйкаッドの出力で付加的なスケーリングが行なわれます。フィルタ係数は符号付き 32 ビット固定小数点数で指定します。

各バйкаッドの出力は、以下の方程式で与えられます。

$$d_k(n) = [a_{1k}d_k(n-1) + a_{2k}d_k(n-2) + 2^{29}x(n)] \cdot 2^{-31}$$

$$y_k(n) = [b_{0k}d_k(n) + b_{1k}d_k(n-1) + b_{2k}d_k(n-2)] \cdot 2^{-a_{0k}} \cdot 2^2$$

k 番目のセクションの入力 $x_k(n)$ は、前のセクションの出力 $y_{k-1}(n)$ です。最初のセクション ($k=0$) の入力は、input を 16 ビット左シフトした値が読み込まれます。最後のセクション ($k=K-1$) の出力は output に書き込まれます。

coeff は係数を以下の順序に設定してください。

$a_{00}, a_{10}, a_{20}, b_{00}, b_{10}, b_{20}, a_{01}, a_{11}, a_{21}, b_{01} \dots b_{2K-1}$

a_{0k} 項は k 番目のバイカッドの出力で行なわれる右シフトのビット数です。

DLir は、フィルタ係数を 16 ビット値ではなく、32 ビット値で指定するという点で **Iir** と異なっています。積和演算の結果は 64 ビットで保持されます。中間ステージの出力は、 a_{0k} ビット右シフトした結果の下位 32 ビットが取り出されます。オーバーフローしたときは正または負の最大値となります。最終ステージでは、 a_{0K-1} ビット右シフトした結果の下位 16 ビットが取り出されます。なお、オーバーフローしたときは正または負の最大値となります。

(5) 補足事項

1. 本関数を呼び出す前に **InitDLir** を呼び出し、フィルタの作業領域を初期化してください。
2. 遅延ノード $d_k(n)$ は、30 ビットの値に丸められ、オーバーフローしたときは正または負の最大値となります。
3. **DLir** は符号付き 32 ビット固定小数点数で係数を指定して使用してください。このとき、 a_{0k} は $k < K-1$ のときは 31、 $k=K-1$ のときは 47 に設定してください。
4. **DLir** より **Iir** の方が実行速度は速いので、倍精度計算の必要がなければ **Iir** を使用してください。
5. **output** に **input** と同じ配列を指定した場合、**input** は上書きされます。
6. 本関数はリエントラントではありません。

8.6.2.6 Dllr1 関数

(1) 定義

```
int Dllr1 (short *output, const short input, const long coeff[ ], long no_sections,
           long *workspace)
```

(2) 引数

*output 出力データ $y_{k-1}(n)$ へのポインタ
input 入力データ $x_0(n)$
coeff[] フィルタ係数
no_sections 2 次フィルタセクションの数 **K**
*workspace 作業領域へのポインタ

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_sections < 1 • $a_{0k} < 3$ • $k < K-1$ で $a_{0k} > 32$ • $k = K-1$ で $a_{0k} > 48$

(4) 説明

本関数は単一データ用に倍精度無限インパルス応答フィルタ処理を実行します。

フィルタは、バイカッドという 2 次フィルタを **K** 個縦列に接続した構成になっています。各バイカッドの出力で付加的なスケーリングが行なわれます。フィルタ係数は符号付き 32 ビット固定小数点数で指定します。

各バイカッドの出力は、以下の方程式で与えられます。

$$d_k(n) = [a_{1k}d_k(n-1) + a_{2k}d_k(n-2) + 2^{29}x(n)] \cdot 2^{-31}$$

$$y_k(n) = [b_{0k}d_k(n) + b_{1k}d_k(n-1) + b_{2k}d_k(n-2)] \cdot 2^{-a_{0k}} \cdot 2^2$$

k 番目のセクションの入力 $x_k(n)$ は、前のセクションの出力 $y_{k-1}(n)$ です。最初のセクション ($k=0$) への入力、input を 16 ビット左シフトした値が読み込まれます。最後のセクション ($k=K-1$) からの出力は output に書き込まれます。

coeff は係数を以下の順序に設定してください。

$a_{00}, a_{10}, a_{20}, b_{00}, b_{10}, b_{20}, a_{01}, a_{11}, a_{21}, b_{01} \dots b_{2K-1}$

a_{0k} 項は k 番目のバイカッドの出力で行なわれる右シフトのビット数です。

Diir1 は、フィルタ係数を 16 ビット値ではなく、32 ビット値で指定するという点で **Iir** と異なっています。積和演算の結果は 64 ビットで保持されます。中間ステージの出力は、 a_{0k} ビット右シフトした結果の下位 32 ビットが取り出されます。オーバーフローしたときは正または負の最大値となります。最終ステージでは、 a_{0K-1} ビット右シフトした結果の下位 16 ビットが取り出されます。なお、オーバーフローしたときは正または負の最大値となります。

(5) 補足事項

1. 本関数を呼び出す前に **InitDIir** を呼び出し、フィルタの作業領域を初期化してください。
2. 遅延ノード $d_k(n)$ は、30 ビットの値に丸められ、オーバーフローしたときは正または負の最大値となります。
3. **DIir1** は符号付き 32 ビット固定小数点数で係数を指定して使用してください。このとき、 a_{0k} は $k < K-1$ のときは 31、 $k=K-1$ のときは 47 に設定してください。
4. **DIir1** より **Iir1** の方が実行速度は速いので、倍精度計算の必要がなければ **Iir1** を使用してください。
5. 本関数はリエントラントではありません。

8.6.2.7 Lms 関数

(1) 定義

```
int Lms (short output[ ], const short input[ ], const short ref_output[ ],
         long no_samples, short coeff[ ], long no_coefs, int res_shift,
         short conv_fact, short *workspace)
```

(2) 引数

output[] 出力データ y
input[] 入力データ x
ref_output[] 所望の出力値 d
no_samples 入力データの数 N
coeff[] 適応フィルタ係数 h
no_coefs 係数の数 K
res_shift 各出力に適用される右シフト
conv_fact 収束係数 2μ
*workspace 作業領域へのポインタ

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_samples < 1 • no_coefs ≤ 2 • res_shift < 0 • res_shift > 25

(4) 説明

本関数は最小 2 乗平均アルゴリズム (LMS) を使って、実数適応 FIR フィルタ処理を実行します。

FIR フィルタは以下の式で定義されます。

$$y(n) = \left[\sum_{k=0}^{K-1} h_n(k) x(n-k) \right] \cdot 2^{-\text{res_shift}}$$

積和演算の結果は 39 ビットで保持されます。出力 $y(n)$ は res_shift ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバフローしたときは正ま

たは負の最大値となります。

フィルタ係数の更新は **Widrow-Hoff** アルゴリズムを使用します。

$$h_{n+1}(k) = h_n(k) + 2\mu e(n)x(n-k)$$

ここで $e(n)$ は所望する信号と実際の出力の誤差です。

$$e(n) = d(n) - y(n)$$

$2\mu e(n)x(n-k)$ の計算では、16 ビット × 16 ビットの乗算を 2 回行ないます。どちらの乗算結果とも上位 16 ビットが保持され、オーバーフローしたときは正または負の最大値となります。更新した係数の値が **H'8000** になると、積和演算でオーバーフローが発生する可能性があります。係数の値は **H'8001** ~ **H'7FFF** の範囲内になるように設定してください。

(5) 補足事項

1. 係数のスケーリングについては「8.6.1.2 係数のスケーリング」を参照してください。
係数は **LMS** フィルタによって適応させるので、最も安全なスケーリングは係数を 256 個未満にし、**res_shift** を 24 に設定する方法です。
2. **conv_fact** は通常正に設定してください。また **H'8000** には設定しないでください。
3. 本関数を呼び出す前に **InitLms** を呼び出し、フィルタを初期化してください。
4. **output** に **input** または **ref_output** と同じ配列を指定した場合、**input** または **ref_output** は上書きされます。
5. 本関数はリエントラントではありません。

8.6.2.8 Lms1 関数

(1) 定義

```
int Lms1 (short *output, short input, short ref_output, short coeff[ ], long no_coefs,
           int res_shift, short conv_fact, short *workspace)
```

(2) 引数

*output	出力データ $y(n)$ へのポインタ
input	入力データ $x(n)$
ref_output	所望の出力値 $d(n)$
coeff[]	適応フィルタ係数 h
no_coefs	係数の数 K
res_shift	各出力に適用される右シフト
conv_fact	収束係数 2μ
*workspace	作業領域へのポインタ

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • $no_coefs \leq 2$ • $res_shift < 0$ • $res_shift > 25$

(4) 説明

本関数は最小 2 乗平均アルゴリズム (LMS) を使って、単一データ用に実数適応 FIR フィルタ処理を実行します。

FIR フィルタは以下の式で定義されます。

$$y(n) = \left[\sum_{k=0}^{K-1} h_n(k) x(n-k) \right] \cdot 2^{-res_shift}$$

積和演算の結果は 39 ビットで保持されます。出力 $y(n)$ は res_shift ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバフローしたときは正または負の最大値となります。

フィルタ係数の更新は **Widrow-Hoff** アルゴリズムを使用します。

$$h_{n+1}(k) = h_n(k) + 2\mu e(n) x(n-k)$$

ここで $e(n)$ は所望する信号と実際の出力の誤差です。

$$e(n) = d(n) - y(n)$$

$2\mu e(n)x(n-k)$ の計算では、16 ビット \times 16 ビットの乗算を 2 回行ないます。どちらの乗算でも、上位 16 ビットが保持され、オーバーフローしたときは正または負の最大値となります。更新した係数の値が $H'8000$ になると、積和演算でオーバーフローが発生する可能性があります。係数の値は $H'8001 \sim H'7FFF$ の範囲内になるように設定してください。

(5) 補足事項

1. 係数のスケーリングについては「6.1.2 係数のスケーリング」を参照してください。
係数は LMS フィルタによって適応させるので、最も安全なスケーリングは係数を 256 個未満にし、`res_shift` を 24 に設定する方法です。
2. `conv_fact` は通常正に設定してください。また $H'8000$ には設定しないでください。
3. 本関数を呼び出す前に `InitLms` を呼び出し、フィルタを初期化してください。
4. 本関数はリエントラントではありません。

8.6.2.9 InitFir 関数

(1) 定義

```
int InitFir (short **workspace, long no_coeffs)
```

(2) 引数

****workspace** 作業領域へのポインタへのポインタ
no_coeffs 係数の数 **K**

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_NO_HEAP	workspace の使用できるメモリスペースが不十分
EDSP_BAD_ARG	$\text{no_coeffs} \leq 2$

(4) 説明

本関数は **Fir** と **Fir1** で使用する作業領域を割り付けます。すでに入力されているデータは **0** に初期化されます。

(5) 補足事項

1. **Fir**、**Fir1**、**Lms** および **Lms1** だけが **InitFir** で割り付けられた作業領域を操作することができます。ユーザプログラムから作業領域を直接アクセスしないでください。
2. **filt_ws.h** をプログラムの中でインクルードしてください。
3. 本関数はリエントラントではありません。

8.6.2.10 Initlir 関数

(1) 定義

```
int Initlir (short **workspace, long no_sections)
```

(2) 引数

****workspace** 作業領域へのポインタへのポインタ

no_sections 2 次フィルタセクションの数 **K**

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_NO_HEAP	workspace の使用できるメモリスペースが不十分
EDSP_BAD_ARG	no_sections < 1

(4) 説明

本関数は **Iir** と **Iir1** で使用する作業領域を割り付けます。すでに入力されているデータは **0** に初期化されます。

(5) 補足事項

1. **Iir** と **Iir1** だけが **Initlir** で割り付けられた作業領域を操作することができます。ユーザプログラムから作業領域を直接アクセスしないでください。
2. **filt_ws.h** をプログラムの中でインクルードしてください。
3. 本関数はリエントラントではありません。

8.6.2.11 InitDlir 関数

(1) 定義

```
int InitDlir (long **workspace, long no_sections)
```

(2) 引数

****workspace** 作業領域へのポインタへのポインタ

no_sections 2 次フィルタセクションの数 **K**

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_NO_HEAP	workspace の使用できるメモリスペースが不十分
EDSP_BAD_ARG	no_sections < 1

(4) 説明

本関数は **Dlir** と **Dlir1** で使用する作業領域を割り付けます。すでに入力されているデータは **0** に初期化されます。

(5) 補足事項

1. **Dlir** と **Dlir1** だけが **InitDlir** で割り付けられた作業領域を操作することができます。
2. **filt_ws.h** をプログラムの中でインクルードしてください。
3. 本関数はリエントラントではありません。

8.6.2.12 InitLms 関数

(1) 定義

```
int InitLms (short **workspace, long no_coeffs)
```

(2) 引数

****workspace** 作業領域へのポインタへのポインタ

no_coeffs 係数の数 **K**

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_NO_HEAP	workspace の使用できるメモリスペースが不十分
EDSP_BAD_ARG	$\text{no_coeffs} \leq 2$

(4) 説明

本関数は **Lms** と **Lms1** で使用する作業領域を割り付けます。すでに入力されているデータは **0** に初期化されます。

(5) 補足事項

1. **Fir**、**Fir1**、**Lms** および **Lms1** だけが **InitLms** で割り付けられた作業領域を操作することができます。ユーザプログラムから作業領域を直接アクセスしないでください。
2. **filt_ws.h** をプログラムの中でインクルードしてください。
3. 本関数はリエントラントではありません。

8.6.2.13 FreeFir 関数

(1) 定義

```
int FreeFir (short **workspace, long no_coeffs)
```

(2) 引数

****workspace** 作業領域へのポインタへのポインタ
no_coeffs 係数の数 **K**

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	$\text{no_coeffs} \leq 2$

(4) 説明

本関数は **InitFir** で割り付けられた作業領域を解放します。

(5) 補足事項

1. 本関数はリエントラントではありません。

8.6.2.14 Freilir 関数

(1) 定義

int **Freilir** (short **workspace, long no_sections)

(2) 引数

**workspace 作業領域へのポインタへのポインタ

no_sections 2 次フィルタセクションの数 **K**

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	no_sections < 1

(4) 説明

本関数は **InitIir** で割り付けられた作業領域を解放します。

(5) 補足事項

1. 本関数はリエントラントではありません。

8.6.2.15 FreeDlir 関数

(1) 定義

```
int FreeDlir (long **workspace, long no_sections)
```

(2) 引数

****workspace** 作業領域へのポインタへのポインタ

no_sections 2 次フィルタセクションの数 **K**

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	$\text{no_section} \leq 2$

(4) 説明

本関数は **InitDlir** で割り付けられた作業領域メモリを解放します。

(5) 補足事項

1. 本関数はリエントラントではありません。

8.6.2.16 FreeLms 関数

(1) 定義

```
int FreeLms (short **workspace, long no_coeffs)
```

(2) 引数

****workspace** 作業領域へのポインタへのポインタ
no_coeffs 係数の数 **K**

(3) リターン値

int 型	
<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	no_coeffs < 1

(4) 説明

本関数は **InitLms** で割り付けられた作業領域メモリを解放します。

(5) 補足事項

1. 本関数はリエントラントではありません。

8.7 畳み込みと相関

8.7.1 概要

8.7.1.1 関数一覧

- ・ **ConvComplete** 2つの配列の完全な畳み込みを計算します。
- ・ **ConvCyclic** 2つの配列の周期的な畳み込みを計算します。
- ・ **ConvPartial** 2つの配列の部分的な畳み込みを計算します。
- ・ **Correlate** 2つの配列の相関を計算します。
- ・ **CorrCyclic** 2つの配列の周期的な相関を計算します。

これらの関数を使用する際は、2つの入力配列のうち1つはXメモリに、もう1つはYメモリに配置してください。出力配列はどのメモリに配置してもかまいません。

8.7.2 各関数の説明

8.7.2.1 ConvComplete 関数

(1) 定義

```
int ConvComplete (short output[ ], const short ip_x[ ], const short ip_y[ ],  
                  long x_size, long y_size, int res_shift)
```

(2) 引数

output[]	出力 z
ip_x[]	入力 x
ip_y[]	入力 y
x_size	ip_x のサイズ X
y_size	ip_y のサイズ Y
res_shift	各出力に適用される右シフト

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none">• x_size < 1• y_size < 1• res_shift < 0• res_shift > 25

(4) 説明

本関数は 2 つの入力配列 **x,y** を完全に畳み込み、結果を出力配列 **z** に書き出します。

$$z(m) = \left[\sum_{i=0}^{X-1} x(i)y(m-i) \right] \cdot 2^{-\text{res_shift}} \quad 0 \leq m < X+Y-1$$

入力配列外のデータは 0 として読み込まれます。

(5) 補足事項

1. 出力配列サイズは **X+Y-1** 以上に設定してください。

8.7.2.2 ConvCyclic 関数

(1) 定義

```
int ConvCyclic (short output[ ], const short ip_x[ ], const short ip_y[ ],
                 long size, int res_shift)
```

(2) 引数

output[]	出力 z
ip_x[]	入力 x
ip_y[]	入力 y
size	配列のサイズ N
res_shift	各出力に適用される右シフト

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • size < 1 • res_shift < 0 • res_shift > 25

(4) 説明

本関数は 2 つの入力配列 **x**, **y** を周期的に畳み込み、結果を出力配列 **z** に書き出します。

$$z(m) = \left[\sum_{i=0}^{N-1} x(i) y(|m-i+N|_N) \right] \cdot 2^{-\text{res_shift}} \quad 0 \leq m < N$$

ここで、 $|i|_N$ は剰余 ($i \% N$) を意味します。

8.7.2.3 ConvPartial 関数

(1) 定義

```
int ConvPartial (short output[ ], const short ip_x[ ], const short ip_y[ ], long x_size,  
                  long y_size, int res_shift)
```

(2) 引数

output[]	出力 z
ip_x[]	入力 x
ip_y[]	入力 y
x_size	ip_x のサイズ X
y_size	ip_y のサイズ Y
res_shift	各出力に適用される右シフト

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none">• x_size < 1• y_size < 1• res_shift < 0• res_shift > 25

(4) 説明

本関数は 2 つの入力配列 **x**, **y** を畳み込み、結果を出力配列 **z** に書き出します。入力配列外のデータから引き出された出力は含まれていません。

$$z(m) = \left[\sum_{i=0}^{A-1} a(i) b(m + A - 1 - i) \right] \cdot 2^{-\text{res_shift}} \quad 0 \leq m \leq |A - B|$$

ただし、配列の個数は **a** < **b** で、**A** は **a** のサイズ、**B** は **b** のサイズです。

(5) 補足事項

1. 出力配列サイズは $|X - Y| + 1$ 以上に設定してください。
2. 入力配列外のデータは 0 として読み込まれます。

8.7.2.4 Correlate 関数

(1) 定義

```
int Correlate (short output[ ], const short ip_x[ ], const short ip_y[ ], long x_size,
               long y_size, long no_corr, int x_is_larger, int res_shift)
```

(2) 引数

output[]	出力 z
ip_x[]	入力 x
ip_y[]	入力 y
x_size	ip_x のサイズ X
y_size	ip_y のサイズ Y
no_corr	計算する相関の数 M
x_is_larger	X=Y のときの配列指定
res_shift	各出力に適用される右シフト

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • x_size < 1 • y_size < 1 • no_corr < 1 • res_shift < 0 • res_shift > 25 • x_is_larger ≠ 0 または 1

(4) 説明

本関数は 2 つの入力配列 **x,y** の相関を求め、結果を出力配列 **z** に書き出します。以下の式では配列の個数は **a > b** で、**A** は **a** のサイズとします。**X=Y** の場合は、**x_is_larger=1** とすると **x** を **a** とし、**x_is_larger=0** とすると **x** を **b** とします。

$$z(m) = \left[\sum_{i=0}^{A-1} a(i)b(i+m) \right] \cdot 2^{-\text{res_shift}} \quad 0 \leq m < M$$

A < X + M となっても差し支えありません。この場合、入力配列外のデータは **0** を使用します。

(5) 補足事項

1. `res_shift = 0` は通常の整数計算に、`res_shift = 15` は小数計算に相当します。

8.7.2.5 CorrCyclic 関数

(1) 定義

```
int CorrCyclic (short output[ ], const short ip_x[ ], const short ip_y[ ],
                long size, int reverse, int res_shift)
```

(2) 引数

output[]	出力 z
ip_x[]	入力 x
ip_y[]	入力 y
size	配列のサイズ N
reverse	反転フラグ
res_shift	各出力に適用される右シフト

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • size < 1 • res_shift < 0 • res_shift > 25 • reverse ≠ 0 または 1

(4) 説明

本関数は周期的に配列 x, y の相関を求め、結果を出力配列 z に書き出します。

$$z(m) = \left[\sum_{i=0}^{N-1} x(i) y(|i+m|_N) \right] \cdot 2^{-\text{res_shift}} \quad 0 \leq m < N$$

ここで、 $|i|_N$ は剰余 ($i \% N$) を意味します。reverse=1 の場合、出力のデータは反転され、実際の計算は以下のようになります。

$$z(m) = \left[\sum_{i=0}^{N-1} y(i) x(|i+m|_N) \right] \cdot 2^{-\text{res_shift}} \quad 0 \leq m < N$$

8.8 その他

8.8.1 概要

8.8.1.1 関数一覧

・ Limit	H'8000 のデータを H'8001 に置き換えます。
・ CopyXtoY	配列を X メモリから Y メモリにコピーします。
・ CopyYtoX	配列を Y メモリから X メモリにコピーします。
・ CopyToX	配列を指定した場所から X メモリにコピーします。
・ CopyToY	配列を指定した場所から Y メモリにコピーします。
・ CopyFromX	配列を X メモリから指定した場所にコピーします。
・ CopyFromY	配列を Y メモリから指定した場所にコピーします。
・ GenGWnoise	白色ガウス雑音を生成します。
・ MatrixMult	2 つのマトリックスの乗算をします。
・ VectorMult	2 つのデータの乗算をします。
・ MsPower	2 乗平均強度を求めます。
・ Mean	平均を求めます。
・ Variance	平均と偏差を求めます。
・ MaxI	整数配列の最大値を求めます。
・ MinI	整数配列の最小値を求めます。
・ PeakI	整数配列の最大絶対値を求めます。

8.8.2 各関数の説明

8.8.2.1 Limit 関数

(1) 定義

```
int Limit (short data_xy[ ], long no_elements, int data_is_x)
```

(2) 引数

data_xy[] データ配列
no_elements データ数
data_is_x データ配置指定

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_elements < 1 • data_is_x ≠ 0 または 1

(4) 説明

本関数は、値が **H'8000** の入力データを **H'8001** に置き換えます。これにより、**DSP** 命令の固定小数点乗算の際にオーバーフローが発生しないようにします。ただし、この処理を行っても積和演算の加算でオーバーフローが発生する可能性はあります。

(5) 補足事項

1. **data_is_x=1** のときはデータは **X** メモリに、**data_is_x=0** のときはデータは **Y** メモリに配置してください。

8.8.2.2 CopyXtoY 関数

(1) 定義

```
int CopyXtoY (short op_y[ ], const short ip_x[ ], long n)
```

(2) 引数

op_y[]	出力配列
ip_x[]	入力配列
n	データ数

(3) リターン値

int 型	
<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	$n < 1$

(4) 説明

本関数は配列を **ip_x** から **op_y** へコピーします。

(5) 補足事項

2. **ip_x** は X メモリに、**op_y** は Y メモリに配置してください。

8.8.2.3 CopyYtoX 関数

(1) 定義

```
int CopyYtoX (short op_x[ ], const short ip_y[ ], long n)
```

(2) 引数

op_x[]	出力配列
ip_y[]	入力配列
n	データ数

(3) リターン値

int 型	
<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	$n < 1$

(4) 説明

本関数は配列を **ip_y** から **op_x** へコピーします。

(5) 補足事項

1. **op_x** は X メモリに、**ip_y** は Y メモリに配置してください。

8.8.2.4 CopyToX 関数

(1) 定義

```
int CopyToX (short op_x[ ], const short input[ ], long n)
```

(2) 引数

op_x[]	出力配列
input[]	入力配列
n	データ数

(3) リターン値

int 型	
<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	$n < 1$

(4) 説明

本関数は配列 **input** を **op_x** へコピーします。

(5) 補足事項

1. **op_x** は X メモリに、**input** は任意のメモリに配置してください。

8.8.2.5 CopyToY 関数

(1) 定義

```
int CopyToY (short op_y[ ], const short input[ ], long n)
```

(2) 引数

op_y[]	出力配列
input[]	入力配列
n	データ数

(3) リターン値

int 型	
<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	$n < 1$

(4) 説明

本関数は配列 **input** を **op_y** へコピーします。

(5) 補足事項

1. **op_y** は Y メモリに、**input** は任意のメモリに配置してください。

8.8.2.6 CopyFromX 関数

(1) 定義

int **CopyFromX** (short output[], const short ip_x[], long n)

(2) 引数

output[]	出力配列
ip_x[]	入力配列
n	データ数

(3) リターン値

int 型	
<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	$n < 1$

(4) 説明

本関数は配列 **ip_x** を **output** へコピーします。

(5) 補足事項

1. **ip_x** は X メモリに、**output** は任意のメモリに配置してください。

8.8.2.7 CopyFromY 関数

(1) 定義

```
int CopyFromY (short output[ ], const short ip_y[ ], long n)
```

(2) 引数

output[]	出力配列
ip_y[]	入力配列
n	データ数

(3) リターン値

int 型	
値	意味
EDSP_OK	成功
EDSP_BAD_ARG	$n < 1$

(4) 説明

本関数は配列 **ip_y** を **output** へコピーします。

(5) 補足事項

1. **ip_y** は Y メモリに、**output** は任意のメモリに配置してください。

8.8.2.8 GenGWnoise 関数

(1) 定義

int **GenGWnoise** (short output[], long no_samples, float variance)

(2) 引数

output[] 白色雑音データの出力
no_samples 出力データ数
variance ノイズ分布の偏差 σ^2

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です ・ no_samples < 1 ・ variance ≤ 0.0

(4) 説明

本関数は平均が 0 で、ユーザが指定した偏差をもつ白色ガウス雑音を生成します。出力データは 2 つ 1 組で生成されます。1 組の出力データを生成するために **rand** 関数を使用し、 x の 2 乗合計が 1 未満になる組が求まるまで -1 ~ 1 の間で 1 組の乱数 γ_1 、 γ_2 を生成します。そして、1 組の出力データ o_1 、 o_2 が以下の式で計算されます。

$$o_1 = \sigma \gamma_1 \sqrt{-2 \ln(x)/x}$$
$$o_2 = \sigma \gamma_2 \sqrt{-2 \ln(x)/x}$$

(5) 補足事項

1. データ数を奇数に設定した場合、最後の組の 2 番目のデータは破棄されます。
2. 本関数が呼び出している標準ライブラリの **rand** 関数はリエントラントではないので、生成される乱数 γ_1 、 γ_2 の順番が常に同じになるとは限りません。しかし、生成される白色雑音 o_1 、 o_2 の特性に影響を及ぼすことはありません。
3. 本関数は浮動小数点演算を使用しています。浮動小数点演算は処理速度が遅くなるので、本関数は評価用として使うことをおすすめします。

8.8.2.9 MatrixMult 関数

(1) 定義

```
int MatrixMult (void *op_matrix, const void *ip_x, const void *ip_y, long m, long n,
                 long p, int x_first, int res_shift)
```

(2) 引数

*op_matrix	出力の第一データへのポインタ
*ip_x	入力 x の第一データへのポインタ
*ip_y	入力 y の第一データへのポインタ
m	マトリックス 1 の行数
n	マトリックス 1 の列数、マトリックス 2 の行数
p	マトリックス 2 の列数
x_first	マトリックス乗算の順番指定
res_shift	各出力に適用される右シフト

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • m, n, または $p < 1$ • $res_shift < 0$ • $res_shift > 25$ • $x_first \neq 0$ または 1

(4) 説明

本関数は 2 つのマトリックス x, y の乗算を行ない、結果を `op_matrix` に配置します。

$x_first=1$ の場合、 $x \cdot y$ を計算します。このとき、`ip_x` は $m \times n$ 、`ip_y` は $n \times p$ 、`op_matrix` は $m \times p$ となります。

$x_first=0$ の場合、 $y \cdot x$ を計算します。このとき、`ip_y` は $m \times n$ 、`ip_x` は $n \times p$ 、`op_matrix` は $m \times p$ となります。

積和演算の結果は 39 ビットで保持されます。出力 $y(n)$ は `res_shift` ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバフローしたときは正または負の最大値となります。

各マトリックスは通常の C 様式（行優先順）で配置されます。

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & a_{10} & a_{11} \end{pmatrix}$$

（5）補足事項

1. 任意の配列サイズを指定できるようにするために、配列パラメタは `void*` で指定します。これらのパラメタは `short` 変数を指すようにしてください。
2. 入力配列 `ip_x, ip_y` と出力配列 `op_matrix` は別々に用意してください。

8.8.2.10 VectorMult 関数

(1) 定義

```
int VectorMult (short output[ ], const short ip_x[ ], const short ip_y[ ],
                 long no_elements, int res_shift)
```

(2) 引数

output[]	出力
ip_x[]	入力 1
ip_y[]	入力 2
no_elements	データ数
res_shift	各出力に適用される右シフト

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_elements < 1 • res_shift < 0 • res_shift > 16

(4) 説明

本関数は ip_x, ip_y から 1 つずつデータを取り出して乗算を行ない、結果を output に配置します。

(5) 補足事項

1. 出力は res_shift ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバーフローしたときは正または負の最大値となります。
2. 本関数はデータの乗算を行ないません。内積を計算する場合は m と p を 1 に設定して **MatrixMult** を使用してください。

8.8.2.11 MsPower 関数

(1) 定義

int **MsPower** (long *output, const short input[], long no_elements, int src_is_x)

(2) 引数

*output 出力へのポインタ
input[] 入力 x
no_elements データ数 N
src_is_x データ配置指定

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です ・ no_elements < 1 ・ src_is_x ≠ 0 または 1

(4) 説明

本関数は入力データの平均 2 乗値を求めます。

$$\text{平均2乗値} = \frac{1}{N} \sum_{i=0}^{N-1} x(i)^2$$

(5) 補足事項

1. 除算結果は最も近い整数値に丸められます。
2. 演算の結果は 63 ビットで保持されます。no_elements が 2^{32} 以上の場合、オーバーフローが発生することがあります。
3. src_is_x=1 のときはデータは X メモリに、src_is_x=0 のときはデータは Y メモリに配置してください。

8.8.2.12 Mean 関数

(1) 定義

```
int Mean (short *mean, const short input[ ], long no_elements, int src_is_x)
```

(2) 引数

*mean input の平均 \bar{x} へのポインタ
input[] 入力 x
no_elements データ数 N
src_is_x データ配置指定

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_elements < 1 • src_is_x ≠ 0 または 1

(4) 説明

本関数は入力データの平均値を求めます。

$$\bar{x} = \frac{1}{N} \sum_{i=0}^{N-1} x(i)$$

(5) 補足事項

1. 除算結果は最も近い整数値に丸められます。
2. 演算結果は 32 ビットで保持されます。no_elements が $2^{16}-1$ よりも大きい場合、オーバフローが発生することがあります。
3. src_is_x=1 のときはデータは X メモリに、src_is_x=0 のときはデータは Y メモリに配置してください。

8.8.2.13 Variance 関数

(1) 定義

```
int Variance (long *variance, short *mean, const short input[ ], long no_elements,
               int src_is_x)
```

(2) 引数

*variance 入力の偏差 σ^2 へのポインタ
 *mean データの平均 \bar{x} へのポインタ
 input[] 入力 x
 no_elements データ数 N
 src_is_x データ配置指定

(3) リターン値

int 型

値	意味
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_elements < 1 • src_is_x ≠ 0 または 1

(4) 説明

本関数は **input** の平均と偏差を求めます。

$$\bar{x} = \frac{1}{N} \sum_{i=0}^{N-1} x(i)$$

$$\sigma^2 = \frac{1}{N} \sum_{i=0}^{N-1} x(i)^2 - \bar{x}^2$$

(5) 補足事項

1. 除算結果は最も近い整数値に丸められます。
2. \bar{x} は 32 ビットで保持されます。また、オーバーフローのチェックはしません。
 no_elements が $2^{16}-1$ よりも大きい場合、オーバーフローが発生することがあります。
3. σ^2 は 63 ビットで保持されます。オーバーフローのチェックはしません。
4. src_is_x=1 のときはデータは X メモリに、src_is_x=0 のときはデータは Y メモリに配置してください。

8.8.2.14 MaxI 関数

(1) 定義

```
int MaxI (short **max_ptr, short input[], long no_elements, int src_is_x)
```

(2) 引数

****max_ptr** 最大データへのポインタへのポインタ
input[] 入力
no_elements データ数
src_is_x データ配置指定

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_elements < 1 • src_is_x ≠ 0 または 1

(4) 説明

本関数は配列 **input** の最大値を検索して、そのアドレスを **max_ptr** に返します。

(5) 補足事項

1. 複数のデータが同じ最大値をもつ場合、**input** の先頭に最も近いデータのアドレスが返されます。
2. **src_is_x=1** のときはデータは X メモリに、**src_is_x=0** のときはデータは Y メモリに配置してください。

8.8.2.15 Mini 関数

(1) 定義

```
int Mini (short **min_ptr, short input[ ], long no_elements, int src_is_x)
```

(2) 引数

****min_ptr** 最小データへのポインタへのポインタ
input[] 入力
no_elements データ数
src_is_x データ配置指定

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none">• no_elements < 1• src_is_x ≠ 0 または 1

(4) 説明

本関数は配列 **input** の最小値を検索して、そのアドレスを **min_ptr** に返します。

(5) 補足事項

1. 複数のデータが同じ最小値をもつ場合、**input** の先頭に最も近いデータのアドレスが返されます。
2. **src_is_x=1** のときはデータは X メモリに、**src_is_x=0** のときはデータは Y メモリに配置してください。

8.8.2.16 PeakI 関数

(1) 定義

```
int PeakI (short **peak_ptr, short input[ ], long no_elements, int src_is_x)
```

(2) 引数

****peak_ptr** 最大絶対値データへのポインタへのポインタ
input[] 入力
no_elements データ数
src_is_x データ配置指定

(3) リターン値

int 型

<u>値</u>	<u>意味</u>
EDSP_OK	成功
EDSP_BAD_ARG	以下のいずれかの場合です <ul style="list-style-type: none"> • no_elements < 1 • src_is_x ≠ 0 または 1

(4) 説明

本関数は配列 **input** の最大絶対値を検索して、そのアドレスを **peak_ptr** に返します。

(5) 補足事項

1. 複数のデータが同じ最大絶対値をもつ場合、**input** の先頭に最も近いデータのアドレスが返されます。
2. **src_is_x=1** のときはデータは X メモリに、**src_is_x=0** のときはデータは Y メモリに配置してください。

付録

付録 A. コンパイラが規定する言語仕様とライブラリ関数仕様

A.1 言語仕様

(1) 翻訳

表 A-1 翻訳の仕様

項番	項目	本コンパイラの仕様
1	エラー検出時のエラー情報	「第 4 章 エラーメッセージ」を参照。

(2) 環境

表 A-2 環境の仕様

項番	項目	本コンパイラの仕様
1	main 関数への実引数の意味	規定しません。
2	対話的入出力装置の構成	規定しません。

(3) 識別子

表 A-3 識別子の仕様

項番	項目	本コンパイラの仕様
1	外部結合とならない識別子(内部名)の有効文字数	外部 / 内部名ともに 250 文字までが有効です。
2	外部結合となる識別子(外部名)の有効文字数	
3	外部結合となる識別子(外部名)の大文字小文字の区別	大文字小文字を区別します。

【注】

250 文字目までが同じで、251 文字目以降が異なっている二つの識別子は、同じ識別子とみなされ
ます。

(a) longabcde...ab;(250 文字が a、251 文字目が b)

(b) longabcde...ac;(250 文字が a、251 文字目が c)

(a)と(b)の二つの識別子は、250 文字目までが一致しているので、同じ識別子とみなします。

(4) 文字

表 A-4 文字の仕様

項番	項目	本コンパイラの仕様
1	ソース文字集合および実行環境文字集合の要素	ソース文字集合、実行環境文字集合ともに ASCII 文字集合です。 ただし、ソースプログラムのコメント内と文字列内にはホスト環境の日本語コードを記述できます。
2	多バイト文字のコード化で使用するシフト状態	シフト状態はサポートしていません。
3	プログラム実行時の文字集合の文字のビット数	ビット数は 8 ビットです。
4	文字定数内、文字列内のソース文字集合と実行環境文字集合の文字との対応付け	同じ ASCII 文字に対応します。
5	言語で規定していない文字や拡張表記を含む整数文字定数の値	言語で規定する以外の文字、拡張表記はサポートしていません。
6	2 文字以上の文字を含む文字定数または 2 文字以上の多バイト文字を含む広角文字定数の値	文字定数は上位 4 文字を有効とします(C コンパイル)。文字定数は上位 1 文字を有効とします(C++コンパイル)。広角文字定数はサポートしていません。また、1 文字より多く指定した場合はウォーニングエラーを出力します。
7	多バイト文字を広角文字に変換するために使用される locale の仕様	locale はサポートしていません。
8	単なる char 型が signed char 型、unsigned char 型のどちらと同じ値の範囲を持つか	signed char 型と同じ値の範囲を持ちます。

(5) 整数

表 A-5 整数の仕様

項番	項目	本コンパイラの仕様
1	整数型の表現方法とその値	表 A-6 に示します。 (負の数は 2 の補数で表現します)
2	整数の値がより短いサイズの符号付き整数型あるいは、符号付き char 型で表現できない値に変換されたときの値 (結果の値が変換先の型で表現できない場合)	整数の値の下位 2 バイトあるいは下位 1 バイトが変換後の値となります。
3	符号付き整数に対するビットごとの演算の結果	符号付きの値とみなします。
4	整数除算における余りの符号	被除数の符号と同符号になります。
5	負の値を持つ符号付き汎整数型の右シフトの結果	符号ビットを保持します。

表 A-6 整数型とその値の範囲

項番	型	値の範囲	データサイズ
1	char (signed char)	- 128 ~ 127	1 バイト
2	unsigned char	0 ~ 255	1 バイト
3	short	- 32768 ~ 32767	2 バイト
4	unsigned short	0 ~ 65535	2 バイト
5	int	- 2147483648 ~ 2147483647	4 バイト
6	unsigned int	0 ~ 4294967295	4 バイト
7	long	- 2147483648 ~ 2147483647	4 バイト
8	unsigned long	0 ~ 4294967295	4 バイト

【注】

括弧内の型指定子は省略可能です。また、型ならび順序は規定されません。

(6) 浮動小数点数

表 A-7 浮動小数点数の仕様

項番	項目	本コンパイラの仕様
1	浮動小数点型の表現方法とその値	浮動小数点数型には、float 型、double 型、long double 型があります。浮動小数点数型の内部表現や変換仕様、演算仕様等の性質は、「A.3 浮動小数点数の仕様」で説明します。表 A-8 に、浮動小数点数型の表現可能な値の限界値を示します。
2	整数を本来の値に正確に表現することができない浮動小数点数に変換したときの切り捨て方向	
3	浮動小数点数をより狭い浮動小数点数に変換したときの切り捨てまたは丸めの方法	

表 A-8 浮動小数点数の限界値

項番	項目	限界値	
		10 進数表現 ^{*1}	16 進数表現
1	float 型の最大値	3.4028235677973364e+38f (3.4028234663852886e+38f)	7f7fffff
2	float 型の正の最小値	7.0064923216240862e-46f (1.4012984643248171e-45f)	00000001
3	double 型 ^{*2} 、long double 型の最大値	1.7976931348623158e+308 (1.7976931348623157e+308)	7fefffffffffffff
4	double 型 ^{*2} 、long double 型の正の最小値	4.9406564584124655e-324 (4.9406564584124654e-324)	0000000000000001

^{*1} : 10 進表現の限界値は 0 または無限大にならない限界値です。また、()内は理論値を表示します。

^{*2} : -double = float オプションが指定されている場合、double 型は float 型と同じ値となります。

-fpu=single オプションが指定されている場合、double、long double 型は float 型と同じ値になります。-fpu=double オプションが指定されている場合、float 型は double 型と同じ値になります。

(7) 配列とポインタ

表 A-9 配列とポインタの仕様

項番	項目	本コンパイラの仕様
1	配列の大きさの最大値を保持するために必要な整数の型(size_t)	unsigned long 型
2	ポインタ型から整数型への変換 (ポインタ型のサイズ 整数型のサイズ)	ポインタ型の下位バイトの値になります。
3	ポインタ型から整数型への変換 (ポインタ型のサイズ < 整数型のサイズ)	符号拡張します。
4	整数型からポインタ型への変換 (整数型のサイズ ポインタ型のサイズ)	整数型の下位バイトの値になります。
5	整数型からポインタ型への変換 (整数型のサイズ < ポインタ型のサイズ)	符号拡張します。
6	同じ配列内のメンバへのポインタ間の差を保持するために必要な整数の型(ptrdiff_t)	int 型

(8) レジスタ

表 A-10 レジスタの仕様

項番	項目	本コンパイラの仕様
1	レジスタに割り付けることができるレジスタ変数の最大数	7 個
2	レジスタに割り付けることができるレジスタ変数の型	char, unsigned char, signed char, bool, short, unsigned short, int, unsigned int, long, unsigned long, float, ポインタ

(9) 構造体、共用体、列挙型、ビットフィールド

表 A-11 構造体、共用体、列挙型、ビットフィールドの仕様

項番	項目	本コンパイラの仕様
1	異なる型のメンバでアクセスされる共用型メンバの参照	参照はできますが、値は保証しません。
2	構造体メンバの境界調整	構造体メンバ中のデータサイズの最大値が境界調整数になります。表 A-6「整数型とその値の範囲」を参照してください。 ^{*1}
3	単なる int 型のビットフィールドの符号	signed int 型とします。
4	int 型のサイズ内のビットフィールドの割り付け順序	上位ビットから割り付けます。 ^{*2}
5	int 型のサイズ内にビットフィールドが割り付けられるとき、次に割り付けるビットフィールドのサイズが int 型内の残っているサイズを越えたときの割り付け方	次の int 型の領域に割り付けます。 ^{*2}
6	ビットフィールドで許される型指定子	char, unsigned char, signed char, short, unsigned short, int, unsigned int, long, unsigned long 型
7	列挙型の値を表現する整数値	int 型

*1：構造体メンバの割り付け方の詳細については「第 2 章 C/C++プログラミング 2.2.2(2)構造体/クラス型」を参照してください

*2：ビットフィールドの割り付け方の詳細については「第 2 章 C/C++プログラミング 2.2.2(3)ビットフィールド」を参照してください。

(10) 修飾子

表 A-12 修飾子の仕様

項番	項目	本コンパイラの仕様
1	volatile データへのアクセスの種類	規定しません。

(11) 宣言

表 A-13 宣言の仕様

項番	項目	本コンパイラの仕様
1	基本型を修飾する型(ポインタ型、配列型、関数型)の数	16 個まで指定できます。

(a) 基本型を修飾する型の数の数え方の例を以下に示します。

(i) `int a;` `a` は `int` 型(基本型)であり、基本型を修飾する宣言子の数は 0 個です。

(ii) `char *f();` `f` は `char` 型(基本型)へのポインタ型を返す関数型であり、基本型を修飾する宣言子の数は 2 個です。

(1 2) 文

表 A-14 文の仕様

項番	項目	本コンパイラの仕様
1	一つの switch 文中で宣言できる case ラベルの数	511 個まで指定できます。

(1 3) プリプロセッサ

表 A-15 プリプロセッサの仕様

項番	項目	本コンパイラの仕様
1	条件コンパイルの定数式内の単一文字の文字定数と実行環境文字集合の対応	プリプロセッサ文の文字定数と実行環境文字集合は一致します。
2	インクルードファイルの読み込み方法	「<」、「>」で囲まれたファイルは include オプションで指定されたディレクトリから読み込みます。 複数ディレクトリを指定した場合は指定した順番に検索します。 ファイルが見つからない場合、環境変数 SHC_INC が指定するディレクトリ、システムディレクトリ(SHC_LIB)の順序で各ディレクトリを検索します。
3	二重引用符で囲まれたインクルードファイルのサポート有無	サポートします。インクルードファイルを現ディレクトリから読み込みます。現ディレクトリにない場合は、前項 2 の規則に従ってファイルを読み込みます。
4	#define 文の実引数の文字列が空白文字のとき展開された後の文字列の空白文字	空白文字列は、空白文字 1 文字として展開されます。
5	#pragma 文の動作	#pragma interrupt #pragma section #pragma inline #pragma inline_asm #pragma abs16 #pragma gbr_base #pragma gbr_base1 #pragma noregsave #pragma noregalloc #pragma regsave #pragma global_register #pragma pack1 #pragma unpack をサポートしています。 ^{*1}
6	__DATE__, __TIME__ の値	コンパイル開始時ホストマシンのタイマに基づく値が設定されます。

*1 : #pragma の仕様については「第 2 章 C/C++プログラミング 2.3 拡張機能」を参照してください。

A.2 ライブラリ関数仕様

C 言語仕様で規定されていない処理系定義のライブラリ関数仕様を以下に示します。

(1) stddef.h

表 A-16 stddef.h の仕様

項番	項目	本コンパイラの仕様
1	マクロ NULL の値	void 型へのポインタ型の値 0 です。(C コンパイル) 値 0 です。(C++コンパイル)
2	ptrdiff_t の内容	int 型

(2) assert.h

表 A-17 assert.h の仕様

項番	項目	本コンパイラの仕様
1	assert 関数が出力する情報と終了動作	出力情報の形式を(a)に示します。情報を出力した後 abort 関数を呼び出して終了します。

(a) assert (式)において、式の値が 0 のとき以下のメッセージを出力します。

Assertion failed : <式> File <ファイル名>,Line <行番号>

(3) ctype.h

表 A-18 ctype.h の仕様

項番	項目	本コンパイラの仕様
1	isalnum 関数、isalpha 関数、iscntrl 関数、islower 関数、isprint 関数、isupper 関数で検査される文字集合	unsigned char 型で表現できる文字集合です。検査の結果真となる文字を表 A-19 に示します。

表 A-19 真となる文字の集合

項番	関数名	真となる文字
1	isalnum	'0' ~ '9', 'A' ~ 'Z', 'a' ~ 'z'
2	isalpha	'A' ~ 'Z', 'a' ~ 'z'
3	iscntrl	'\x00' ~ '\x1f', '\x7f'
4	islower	'a' ~ 'z'
5	isprint	'\x20' ~ '\x7E'
6	isupper	'A' ~ 'Z'

(4) math.h

表 A-20 math.h の仕様

項番	項目	本コンパイラの仕様
1	数学関数の入力パラメタ値が範囲を越えたときの数学関数が返す値	非数を返します。非数の形式については「A.3 浮動小数点の仕様」を参照ください。
2	数学関数でアンダフローエラーが発生したときマクロ「ERANGE」の値が「errno」に設定されるかどうか	設定しません。
3	fmod 関数で第 2 実引数が 0 の場合の動作	非数を返し、定義域エラーとなります。

math.h には、ライブラリのエラー番号の値を示すマクロ ENUM、ERANGE が定義されています。

(5) setjmp.h

表 A-21 setjmp.h の仕様

項番	項目	本コンパイラの仕様
1	setjmp 関数の呼び出しが許されるプログラムの文脈	setjmp()または、ver = setjmp()の形式で、単独の文や、if 文、while 文、do 文、for 文の条件を示す式あるいは、switch 文、return 文の式に指定したときに保証されます。
2	setjmp_a(),longjmp_a()の仕様	CPU が SH4 のときに、浮動小数点拡張レジスタも含めた環境の退避/回復を行います。

(6) stdio.h

表 A-22 stdio.h の仕様

項番	項目	本コンパイラの仕様
1	入力テキストの最終の行が終了を示す改行文字を必要とするかどうか	規定しません。低水準インタフェースルーチンの仕様によります。
2	改行文字の直前に書き出された空白文字は、読み込み時に読み込まれるかどうか	
3	バイナリファイルに書かれたデータに付加されるヌル文字の数	
4	追加モード時のファイル位置指定子の初期値	
5	テキストファイルへの出力によってそれ以降のファイルのデータが失われるかどうか	
6	ファイルのバッファリングの仕様	
7	長さ 0 のファイルが存在するかどうか	
8	正当なファイル名の構成規則	
9	同時に同じファイルをオープンできるかどうか	
10	fprintf 関数における%p 書式変換の出力形式	16 進数出力となります。
11	fscanf 関数における%p 書式変換の入力形式 fscanf 関数での変換文字「-」の意味	16 進数入力となります。 先頭、最後あるいは「^」の直後でない場合、直前の文字と直後の範囲を示します。
12	fgetpos, ftell 関数で設定される errno の値	fgetpos 関数はサポートしていません。 ftell 関数については規定しません。低水準インタフェースルーチンの仕様によります。
13	perror 関数が生成するメッセージの出力形式	メッセージの出力形式を(a)に示します。
14	calloc、malloc、realloc 関数でサイズが 0 の時の動作	0 バイトの領域を割り付けます。

(a) perror 関数の出力形式は、

<文字列>:<error に設定したエラー番号に対応するエラーメッセージ>
となります。

(b) printf 関数、fprintf 関数等で浮動小数点数の無限大および非数を表示するときの形式を表 A-23 に示します。

表 A-23 無限大および非数の表示形式

項番	項目	本コンパイラの仕様
1	正の無限大	++++++
2	負の無限大	-----
3	非数	*****

(7) string.h

表 A-24 string.h の仕様

項番	項目	本コンパイラの仕様
1	memcmp 関数、strcmp 関数、strncmp 関数の処理において返される値の符号	符号付きとして扱います。
2	strerror 関数が返すエラーメッセージの内容	「第 4 章 エラーメッセージ 4.2 標準ライブラリのエラーメッセージ」を参照してください。

(8) errno.h

表 A-25 errno.h の仕様

項番	項目	本コンパイラの仕様
1	errno	int 型変数、ライブラリ関数においてエラーが発生したときにエラー番号が設定される。
2	ERANGE	「第 4 章 エラーメッセージ 4.2 標準ライブラリのエラーメッセージ一覧」を参照してください。
3	EDOM	
4	EDIV	
5	ESTRN	
6	PTRERR	
7	ECBASE	
8	ETLN	
9	EEXP	
10	EEXPN	
11	EFLOATO	
12	EFLOATU	
13	EDBLO	
14	EDBLU	
15	ELDBLO	
16	ELDBLU	
17	NOTOPN	
18	EBADF	
19	ECSPEC	

(9) サポートしていないライブラリ

本コンパイラでサポートしていないライブラリを表 A-26 に示します。ただし、**signal.h**、**time.h** についてはヘッダファイル自体をサポートしていません。

表 A-26 サポートしていないライブラリ

項番	ヘッダファイル	ライブラリ名
1	signal.h	signal, raise
2	stdio.h	remove, rename, tmpfile, tmpnam
3	stdlib.h	getenv, system
4	time.h	clock, difftime, time, asctime, ctime, gmtime, localtime

内部表現の各構成要素の意味を以下に示します。

(i) 符号部

浮動小数点数の符号を示します。0 のとき正、1 のとき負を示します。

(ii) 指数部

浮動小数点数の指数を 2 のべき乗で示します。

(iii) 仮数部

浮動小数点数の有効数字に対応するデータです。

(c) 表現する値の種類

浮動小数点数は、通常の実数値のほかに、無限大等の値も表現することができます。浮動小数点数が表現する値の種類を以下に示します。

(i) 正規化数

指数部が 0 または全ビット 1 ではない場合です。通常の実数値を表現します。

(ii) 非正規化数

指数部が 0 で、仮数部が 0 でない場合です。絶対値の小さな実数値を表現します。

(iii) ゼロ

指数部および仮数部が 0 の場合です。値 0.0 を表現します。

(iv) 無限大

指数部が全ビット 1 で仮数部が 0 の場合です。無限大を表現します。

(v) 非数

指数部が全ビット 1 で仮数部が 0 でない場合です。「0.0/0.0」、「 / 」、「 - 」等、結果が数値または無限大に対応しない演算の結果として得られます。

【注】

非正規化数は、正規化数で表現できない範囲の絶対値の小さな浮動小数点数を表現しますが、正規化数に比較して有効桁数が少なくなっています。したがって、演算の結果、あるいは途中結果が非正規化数となる場合、結果の有効桁数は保証されませんので注意してください。

CPU が SH4 の場合、-denormalization = off のとき非正規化数は 0 として扱い、-denormalization = on のとき非正規化数は非正規化数のまま扱います。

表 A-27 浮動小数点数の表現する値の種類

指数部 仮数部	0	0でも全ビット1でもない	全ビット1
0	0	正規化数	無限大
0以外	非正規化数		非数

(2) float 型

float 型の内部表現は、1 ビットの符号部、8 ビットの指数部、23 ビットの仮数部からなります。

(i) 正規化数

符号部は、0(正)または1(負)で、値の符号を示します。

指数部は、 $1 \sim 254(2^8-2)$ の値をとります。実際の指数は、この値から 127 を引いた値で、その範囲は-126 ~ 127 です。

仮数部は、 $0 \sim 2^{23}-1$ の値をとります。実際の仮数は、 2^{23} のビットを 1 と仮定し、その直後に小数点があるものとして解釈します。

正規化数の表現する値は、

$$(-1)^{(\text{符号部})} \times 2^{(\text{指数部})-127} \times (1 + (\text{仮数部}) \times 2^{-23})$$

となります。

例

[illegible]

符号： -

指数：10000000₍₂₎-127 = 1 ₍₂₎は2進数を表わします。

仮数： $1.11_{(2)}=1.75$

值 : $-1.75 \times 2^1 = -3.5$

(ii) 非正規化数

符号部は 0(正)または 1(負)で値の符号を示します。

指数部は 0 で、実際の引数は-126 になります。

仮数部は、 $1 \sim 2^{23}-1$ で、実際の仮数は、 2^{23} のビットを 0 と仮定し、その直後に小数点があるものとして解釈します。

非正規化数を表現する値は、

$$(-1)^{(\text{符号部})} \times 2^{(\text{指数部})-126} \times ((\text{仮数部}) \times 2^{-23})$$

例

3130	23 22	0
0	0000000000	110000000000000000000000

符号： +

指数： $0_{(2)}-126 = -126$

仮数： $0.11_{(2)}=0.75$ $_{(2)}$ は 2 進数を表わします。

値： 0.75×2^{-126}

(iii) ゼロ

符号部は 0(正)または 1(負)で、それぞれ +0.0、-0.0 を示します。

指数部、仮数部はともに 0 です。

+0.0、-0.0 は、ともに値としては 0.0 を示します。ゼロの符号による、各演算での機能の違いについては「A.3 (4)浮動小数点演算の仕様」を参照してください。

(iv) 無限大

符号部は 0(正)または 1(負)で、それぞれ+、- を示します。

指数部は $255(2^8-1)$ です。

仮数部は 0 です。

(v) 非数

指数部は $255(2^8-1)$ です。

仮数部は 0 以外の値です。

【注】

CPU が SH2E、SH3E、SH4 の場合、仮数部の最上位ビットが 0 の非数を qNaN、仮数部の最上位ビットが 1 の非数を sNaN と呼びます。

その他の仮数フィールドの値、および符号部については規定していません。

(iii) ゼロ

符号部が 0(正)または 1(負)で、それぞれ+0.0、-0.0 を示します。

指数部、仮数部は、ともに 0 です。

+0.0、-0.0 は、ともに値としては 0.0 を示します。ゼロの符号による、各演算での機能の違いについては「A.3 (4)浮動小数点演算の仕様」を参照してください。

(iv) 無限大

符号部は 0(正)または 1(負)で、それぞれ+、- を示します。

指数部は $2047(2^{11}-1)$ です。

仮数部は 0 です。

(v) 非数

指数部は $2047(2^{11}-1)$ です。

仮数部は 0 以外の値です。

【注】

CPU が SH2E,SH3E,SH4 の場合、仮数部の最上位ビットが 0 の非数を qNaN、仮数部の最上位ビットが 1 の非数を sNaN と呼びます。

その他の仮数フィールドの値、および符号部については規定していません。

(4) 浮動小数点演算の仕様

本項では、C 言語の機能として実現されている浮動小数点数の四則演算、およびコンパイル時やライブラリの処理で生じる浮動小数点数の 10 進表現と内部表現の間の変換の仕様について解説します。

(a) 四則演算の仕様

(i) 結果の値の丸め方

浮動小数点数の四則演算の結果の正確な値が、内部表現の仮数の有効数字を越えた場合は、以下の規則に従って丸めを行います。

(ア) 結果の値は、その値を近似する二つの浮動小数点数の内部表現のうち、近い方に向かって丸められます。

(イ) 結果の値が、その値を近似する二つの浮動小数点数のちょうど中央になる場合は、仮数の最後の桁が 0 となる方向に丸められます。

(ウ) CPU が SH2E、SH3E の場合、有効数字を超える部分を切り捨てます。

(エ) CPU が SH4 の場合、-round = nearest オプションが指定されているとき、有効数字を超える部分を四捨五入し、-round = zero オプションが指定されているとき、有効数字を超える部分を切り捨てます。

(ii) オーバフロー、アンダフロー、無効演算の時の処置

実行時のオーバフロー、アンダフロー、無効演算に対しては、以下の処置を行います。

- (ア) オーバフローの場合は、結果の符号に従って正または負の無限大になります。
- (イ) アンダフローの場合は、結果の符号に従って正または負のゼロになります。
- (ウ) 無効演算は、符号が逆の無限大を加算した場合、符号が同じ無限大を減算した場合、ゼロと無限大を乗算した場合、ゼロをゼロで、あるいは無限大を無限大で除算した場合に生じます。これらの場合、結果は非数になります。
- (エ) 浮動小数点数から整数へ変換したときにオーバフローが生じた場合、結果の値は保障されません。

【注】

定数式に関しては、コンパイル時に演算を行います。この時にオーバフロー、アンダフロー、無効演算を検出した場合は、ウォーニングレベルのエラーになります。

(iii) 特殊値の演算に関する注意事項

以下、特殊な値（ゼロ、無限大、非数）の演算に関する注意事項を述べます。

- (ア) 正のゼロと負のゼロの和は正のゼロとなります。
- (イ) 同符号のゼロの差は正のゼロになります。
- (ウ) 被演算子の一方あるいは両方に非数を含む演算の結果は、常に非数になります。
- (エ) 比較演算においては、正のゼロと負のゼロは等しいものとして扱われます。
- (オ) 被演算子の一方あるいは両方が非数であるような比較演算、等値演算の結果は、「!=」については常に真、その他は常に偽となります。

(b) 10 進表現と内部表現の間の変換

本項ではソースプログラム上の浮動小数点数と内部表現の間の変換、あるいはライブラリ関数による ASCII 文字列による浮動小数点数の 10 進表現と内部表現の間の変換の仕様について解説します。

(i) 10 進表現から内部表現に変換する場合、まず 10 進表現を 10 進表現の正規形に変換します。10 進表現の正規形は、「 $\pm M \times 10^{\pm N}$ 」の形式で、M、N の範囲は以下の通りです。

(ア) float 型の正規形

0 M 10^9-1

0 N 99

(イ) double 型、long double 型の正規形

0 M $10^{17}-1$

0 N 999

正規形に変換できない 10 進表現については、オーバフロー、またはアンダフローになります。また、10 進表現が、正規形よりも、多くの有効数字を含んでいる場合は下位の桁は切り捨てます。これらの場合、コンパイル時にはウォーニングレベルのエラーになり、実行時には対応するエラーの番号を変数 `errno` に設定します。

また、正規形に変換するためには、もとの 10 進表現の ASCII 文字列としての長さが 511 文字以下でなければなりません。そうでない場合、コンパイル時にはエラーになり、実行時には対応するエラーの番号を変数 `errno` に設定します。

内部表現から 10 進表現に変換する場合には、一度 10 進表現の正規形に変換してから、指定した書式に従って ASCII 文字列に変換します。

(ii) 10 進表現の正規形と内部表現の間の変換

10 進表現の正規形と内部表現の間の変換は、指数が大きい時や小さい時には、処理の効率上、誤差を回避することができません。以下に、正確な変換ができる範囲とその範囲外の場合の誤差の限界値について解説します。

(イ) 正確な変換ができる範囲

以下に示す指数の範囲の浮動小数点数については、「(a)(i)結果の値の丸め方」に示す丸めが正確に行われます。この範囲ではオーバフロー、アンダフローは生じません。

(1) float 型の場合：0 M 10^9-1 、0 N 13

(2) double 型、long double 型の場合：0 M $10^{17}-1$ 、0 N 27

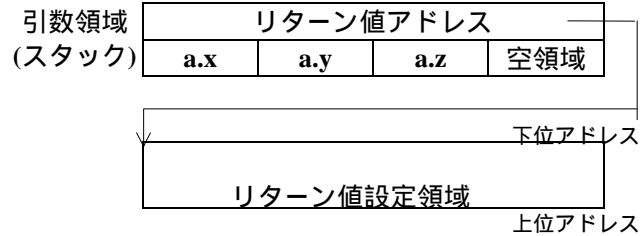
(ロ) 誤差の限界値

(イ) で示す範囲に入っていない値を変換する場合の誤差と正確な丸めを行った時の誤差の差は、有効数字の最小位桁の 0.47 倍を超えません。

また、(イ) で示した範囲を超えている場合、変換の際にオーバフローやアンダフローが生じる場合があります。この場合、コンパイル時にはウォーニングレベルのエラーとなり、実行時には対応するエラーの番号を変数 `errno` に設定します。

例 5 . 関数の返す型が 4 バイトをこえる場合またはクラスの場合、引数領域の直前にリターン値アドレスを設定します。また、クラスのサイズが 4 の倍数バイトでないとき、空領域が生じます。

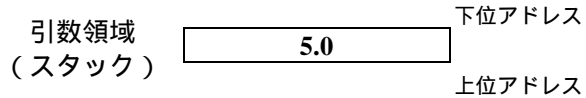
```
struct s{char x,y,z;}a;
double f(struct s);
:
f(a);
```



例 6 . CPU が SH2E、SH3E の場合、float 型の引数は FPU レジスタに割り付きます。

```
int f(char,float,short,float,double);
:
f(1,2.0,3,4.0,5.0);
```

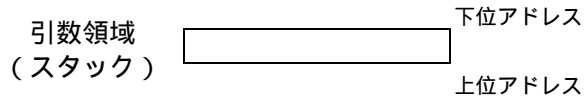
R 4	保証しない	1	F R 4	2.0
R 5	保証しない	3	F R 5	4.0
R 6			F R 6	
R 7			F R 7	
			F R 8	
			F R 9	
			F R 10	
			F R 11	



例 7 . CPU が SH4 かつ-fpu オプション指定なしの場合、float/double 型の引数は FPU レジスタに割り付きます。

```
int f(char,float,double,float,short);
:
f(1,2.0, 4.0,5.0,3);
```

R 4	保証しない	1	F R 4(DR4)	2.0
R 5	保証しない	3	F R 5	5.0
R 6			F R 6(DR6)	4.0
R 7			F R 7	
			F R 8(DR8)	
			F R 9	
			F R 10(DR10)	
			F R 11	



付録 C. レジスタとスタック領域の使用法

コンパイラのレジスタ、スタック領域の使用法を示します。

関数内でのレジスタ、スタック領域はすべてコンパイラが操作しますので、ユーザが特にこの領域の使用法に留意する必要はありません。

レジスタとスタック領域の使用法を図 C-1 に示します。

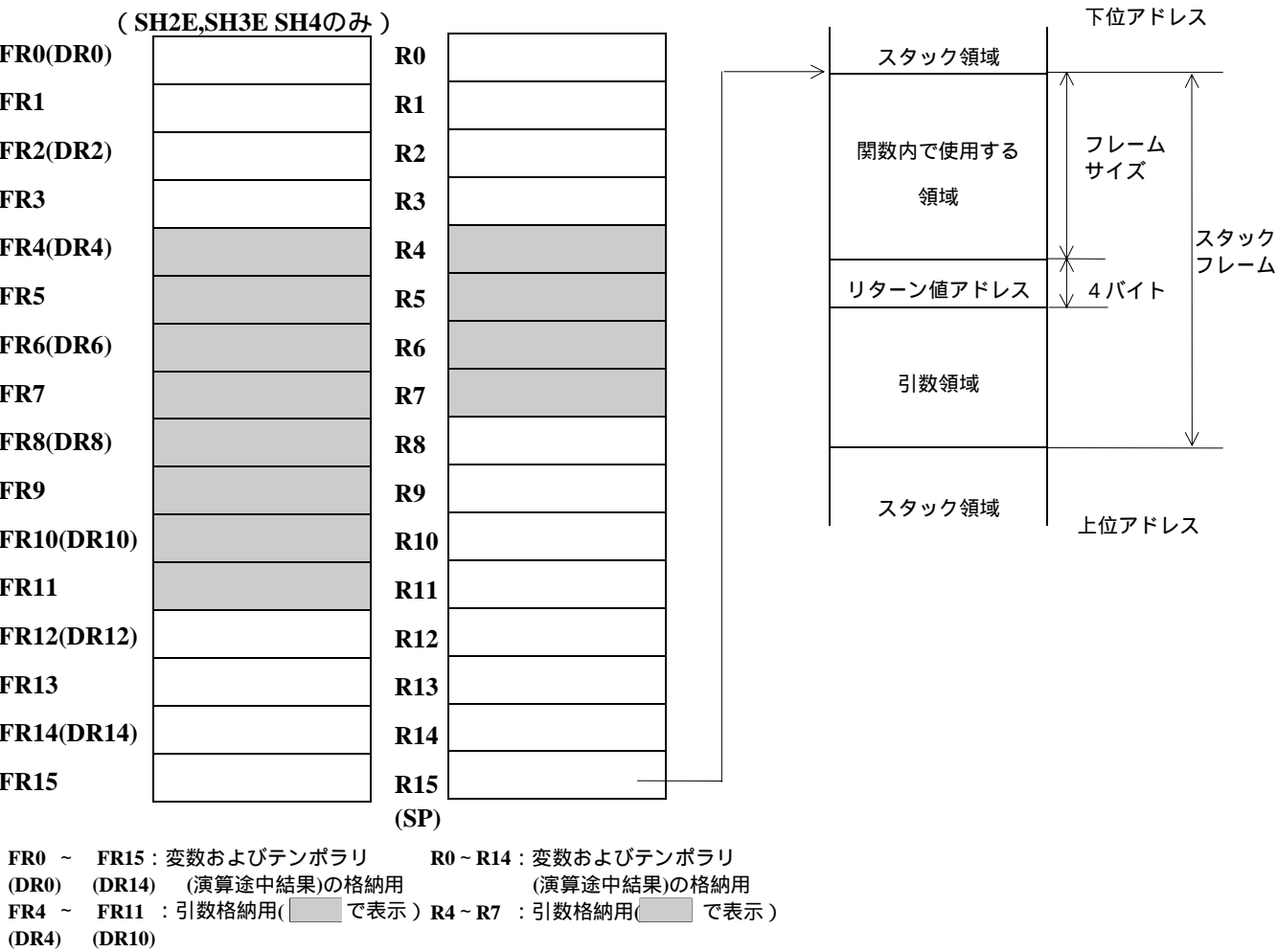


図 C-1 レジスタとスタック領域の使用法

付録 D. 終了処理関数の作成例

D.1 終了処理の登録と実行(onexit)ルーチンの作成例

終了処理の登録を行うライブラリ `onexit` 関数の作成例を示します。

`onexit` 関数では、引数として渡された関数のアドレスを終了処理テーブルに登録します。登録された関数の個数が限界値（ここでは、登録できる関数の個数を 32 個とします）を超えた場合、あるいは、同じ関数が二度以上登録された場合はリターン値として `NULL` を返します。そうでなければ `NULL` 以外の値（この場合は、関数を登録したアドレス）を返します。

以下にプログラム例を示します。

例

```
#include <stdlib.h>

typedef void *onexit_t;

int _onexit_count = 0;
onexit_t(*_onexit_buf[32])(void);

extern onexit_t onexit(onexit_t (*)(void));

onexit_t onexit(f)
onexit_t (*f)(void);
{
    int i;

    for( i = 0; i < _onexit_count; i++ )
        if( _onexit_buf[i] == f)          /*既に登録されていないかチェック*/
            return NULL;
    if( _onexit_count == 32)                /*登録数の限界値チェック*/
        return NULL;
    else {
        _onexit_buf[ _onexit_count ] = f; /*関数のアドレスを登録*/
        _onexit_count++;
        return &_onexit_buf[ _onexit_count - 1 ];
    }
}
```

D.2 プログラムの終了(exit)ルーチンの作成例

プログラムの終了処理を行うライブラリ `exit` 関数の作成例を示します。プログラムの終了処理は、ユーザシステムによって異なりますので、以下のプログラム例を参考にユーザシステムの仕様に従った終了処理を作成してください。

`exit` 関数は、引数として渡されたプログラムの終了コードに従って C プログラムの終了処理を行い、プログラム起動時の環境に戻ります。ここでは、終了コードを外部変数に設定して、`main` 関数を呼び出す直前に `setjmp` 関数で退避した環境に戻ることで実現します。

以下にプログラム例を示します。

例

```
#include <setjmp.h>
#include <stddef.h>

typedef void *onexit_t;
extern int _onexit_count;
extern onexit_t (*_onexit_buf[32])(void);

extern jmp_buf _init_env;
extern int _exit_code;

extern void _CLOSEALL();
extern void exit(int);

void exit( code )
int code;
{
    int i;
    _exit_code = code;          /*_exit_code にリターンコードを設定*/

    for (i = _onexit_count-1; i > 0; i--)
        (*_onexit_buf[i])();    /*onexit 関数で登録した関数を順次実行*/

    _CLOSEALL();                /*オープンした関数をすべてクローズ*/

    longjmp(_init_env, 1);      /*setjmp で退避した環境にリターン*/
}
```

【注】

上記関数で、プログラム実行前の環境に戻るためには、次の関数「callmain」を作成し、初期化ルーチン「init」から関数「main」を呼び出す代わりに関数「callmain」を呼び出してください。

```
#include <setjmp.h>
jmp_buf _init_env;
int      _exit_code;

void callmain()
{
    /*setjmpを用いて現在の環境を退避し、main関数を呼び出します。*/
    /*exit関数からのリターン時には処理を終了します。*/

    if(!setjmp(_init_env))
        _exit_code = main();
}
```

D.3 異常終了 (abort) ルーチンの作成例

異常終了の場合は、ご使用になっているユーザシステムの仕様に従ってプログラムを異常終了させる処理を行ってください。

以下、標準出力装置にメッセージを出力したあと、ファイルをクローズしてから無限ループしてリセットを待つプログラム例を示します。

例

```
#include <stdio.h>

extern void abort();
extern void _CLOSEALL();

void abort()
{
    printf("program is abort !!\n"); /*メッセージの出力*/
    _CLOSEALL();                     /*ファイルのクローズ*/
    while(1);                         /*無限ループ*/
}
```

付録 E. 低水準インタフェースルーチンの作成例

```

/*****
/*
/*          lowsorc.c:
/* - - - - -
/*          S Hシリーズ シミュレータ・デバッガ インタフェースルーチン
/*          - 標準入出力(stdin, stdout, stderr)だけをサポートしています -
/* *****/
#include <string.h>

/* ファイル番号 */

#define STDIN 0          /* 標準入力      (コンソール) */
#define STDOUT 1        /* 標準出力      (コンソール) */
#define STDERR 2        /* 標準エラー出力(コンソール) */

#define FLMIN 0          /* 最小のファイル番号 */
#define FLMAX 3          /* ファイル数の最大値 */

/* ファイルのフラグ */

#define O_RDONLY 0x0001  /* 読み込み専用 */
#define O_WRONLY 0x0002  /* 書き込み専用 */
#define O_RDWR 0x0004    /* 読み書き両用 */

/* 特殊文字コード */

#define CR 0x0d          /* 復帰 */
#define LF 0x0a          /* 改行 */

/* sbrk で管理する領域サイズ */

#define HEAPSIZ 1024

/*****
/* 参照関数の宣言:
/* シミュレータ・デバッガでコンソールへの文字入出力を行うアセンブリプログラムの参照
/* *****/

extern void charput(char);          /* 一文字入力処理 */
extern char charget(void);          /* 一文字出力処理 */
/*****
/* 静的変数の定義:
/* 低水準インタフェースルーチンで使用する静的変数の定義
/* *****/

char flmod[FLMAX];                  /* オープンしたファイルのモード設定場所 */

static union {
    long dummy;                    /* 4 バイト境界にするためのダミー */
    char heap[HEAPSIZ];            /* sbrk で管理する領域の宣言 */
} heap_area;

static char *brk=(char*)&heap_area; /* sbrk で割り付けた領域の最終アドレス

```

```

/*****
/*  open:ファイルのオープン                                     */
/*      リターン値:ファイル番号(成功)                           */
/*      -1          (失敗)                                     */
*****/
int open(char *name,          /* ファイル名          */
          int mode)          /* ファイルのモード */
{
    /* ファイル名に従ってモードをチェックし、ファイル番号を返す */

    if(strcmp(name,"stdin")==0){ /* 標準入力ファイル */
        if((mode&O_RDONLY)==0)
            return -1;
        flmod[STDIN]=mode;
        return STDIN;
    }

    else if(strcmp(name,"stdout")==0){ /* 標準出力ファイル */
        if((mode&O_WRONLY)==0)
            return -1;
        flmod[STDOUT]=mode;
        return STDOUT;
    }

    else if(strcmp(name,"stderr")==0){ /* 標準エラー出力ファイル */
        if((mode&O_WRONLY)==0)
            return -1;
        flmod[STDERR]=mode;
        return STDERR;
    }

    else
        return -1; /* エラー */
}

/*****
/*  close:ファイルのクローズ                                     */
/*      リターン値:0          (成功)                           */
/*      -1          (失敗)                                     */
*****/
int close(int fileno)          /* ファイル番号 */
{
    if(fileno<FLMIN || FLMAX<fileno) /* ファイル番号の範囲チェック */
        return -1;

    flmod[fileno]=0; /* ファイルのモードリセット */
    return 0;
}

```

```

/*****
/* read:データの読み込み
/*      リターン値：実際に読み込んだ文字数（成功）
/*      -1                （失敗）
/*****
int read(int  fileno,          /* ファイル番号 */
         char *buf,          /* 転送先バッファアドレス */
         unsigned int  count) /* 読み込み文字数 */
{
    unsigned int i;
    /* ファイル名に従ってモードをチェックし、一文字ずつ入力してバッファに格納 */

    if(flmod[fileno]&O_RDONLY || flmod[fileno]&O_RDWR){
        for(i=count; i>0; i--){
            *buf=charget();
            if(*buf==CR)          /* 改行文字の置き換え */
                *buf=LF;
            buf++;
        }
        return count;
    }
    else
        return -1;
}
/*****
/* write:データの書き出し
/*      リターン値：実際に書き出した文字数（成功）
/*      -1                （失敗）
/*****
int write(int  fileno,          /* ファイル番号 */
          char *buf,          /* 転送元バッファアドレス */
          unsinged int  count) /* 書き出し文字数 */
{
    unsigned int i;
    char c;

    /* ファイル名に従ってモードをチェックし、一文字ずつ出力 */

    if(flmod[fileno]&O_WRONLY || flmod[fileno]&O_RDWR){
        for(i=count; i>0; i--){
            c=*buf++;
            charput(c);
        }
        return count;
    }
    else
        return -1;
}
/*****
/* lseek:ファイルの読み込み／書き出し位置の設定
/*      リターン値：読み込み／書き出し位置のファイル先頭からのオフセット（成功）
/*      -1                （失敗）
/*      （コンソール入出力では、lseek はサポートしていません）
/*****
long lseek(int  fileno,          /* ファイル番号 */
           long offset,          /* 読み込み／書き出し位置 */
           int  base)           /* オフセットの起点 */
{

```

```

    return -1;
}
/*****
/*  sbrk:データの書き出し                                     */
/*      リターン値：割り付けた領域の先頭アドレス（成功）         */
/*      -1                                     （失敗）         */
*****/
char *sbrk(unsigned long size)    /* 割り付ける領域のサイズ */
{
    char *p ;

    if (brk+size>heap_area.heap+HEAPSIZE)    /* 空き領域のチェック */
        return (char *)-1 ;

    p=brk ;                                /* 領域の割り付け */
    brk += size ;                          /* 最終アドレスの更新 */
    return p ;
}
;-----
;                                lowlvl.src                                |
;-----
;      SH-series simulator debugger interface routine                    |
;      -Input/output one character-                                       |
;-----
        .EXPORT      _charput
        .EXPORT      _charget
SIM_IO:    .EQU          H'0080          ;Specifies TRAP_ADDRESS

        .SECTION     P, CODE, ALIGN=4

;-----
; _charput: One character output                                         |
;      C program interface: charput(char)                               |
;-----

_charput:
        MOV.L        O_PAR,R0          ; Sets output buffer address to R0
        MOV.B        R4,@R0           ; Sets output charcter to buffer
        MOV.L        #O_PAR,R1        ; Sets parameter block address to R1
        MOV.L        #H'01220000,R0    ; Specifies function code (PUTC)
        MOV.W        #SIM_IO,R2       ; Sets system call address to R2
        JSR          @R2
        NOP
        RTS
        NOP

        .ALIGN       4
O_PAR:    .DATA.L     OUT_BUF          ; Parameter block

;-----
; _charget: One character input                                         |
;      C program interface: char charget(void)                         |
;-----

        .ALIGN       4
_charget:
        MOV.L        #I_PAR,R1        ; Sets parameter block address to R1
        MOV.L        #H'01210000,R0    ; Specifies function code (GETC)
        MOV.W        #SIM_IO,R2       ; Sets system call address to R2
        JSR          @R2
        NOP

```

```

        MOV.L    I_PAR,R0        ; Sets input buffer address to R0
        MOV.B    @R0,R0         ; Returns input data
        RTS
        NOP

I_PAR:   .ALIGN    4              ; Parameter block
        .DATA.L   IN_BUF

;-----
;               I/O buffer definition
;-----

        .SECTION   B,DATA,ALIGN=4

OUT_BUF: .RES.L    1              ; Output buffer
IN_BUF:  .RES.L    1              ; Input buffer

        .END

```

付録 F. ASCII コード一覧表

表 F-1 ASCII コード一覧表

パリティビット					b₈								
					b₇								
					b₆								
					b₅								
b₄	b₃	b₂	b₁	MSB LSB	0	1	2	3	4	5	6	7	
				0	NUL	DC₀	SP	0	@	P	`	p	
				1	SOM	X-ON	!	1	A	Q	a	q	
				2	EOA	TAPE	"	2	B	R	b	r	
				3	EOM	X-OFF	#	3	C	S	c	s	
				4	EOT	TAPE	\$	4	D	T	d	t	
				5	WRU	ERRO R	%	5	E	U	e	u	
				6	RU	SYNC	&	6	F	V	f	v	
				7	BELL	LEM	'	7	G	W	g	w	
				8	FE₀	CAN	(8	H	X	h	x	
				9	TAB	S₁)	9	I	Y	i	y	
				A	LF	EOF	*	:	J	Z	j	z	
				B	VT	ESC	+	;	K	[k	{	
				C	FF	S₄	,	<	L	¥	l	 	
				D	CR	S₅	-	=	M]	m	}	
				E	S₀	S₆	·	>	N	^	n	~	
				F	S₁	S₇	/	?	O	-	o	RUB OUT	

付録 G. エンコード規則

本コンパイラでは、C++言語仕様の関数、演算子の多重定義機能を実現するため、シンボル名をユニークにするためにエンコードを行っています。そのエンコード規則を以下に示します。

エンコードの対象になる識別子は、C リンケージ指定されていない C++プログラム中の関数および、静的データメンバです。

関数エンコード名=_(関数名情報)_(クラス名情報)F (修飾子情報) (引数型情報)

静的メンバエンコード名=_(静的データメンバ名)_(クラス名情報)

(1)関数名情報

関数名には、ユーザ宣言関数名または、多重定義演算子関数の場合は下記に示す表の文字列がエンコード名に埋め込まれます。

表 G-1 演算子のエンコード

演算子	エンコード	演算子	エンコード
operator () (関数呼び出し)	_ _cl	operator [] (添字付け)	_ _vc
operator new	_ _nw	operator delete	_ _dl
operator new[]	_ _nwvc	operator delete[]	_ _dlvc
operator T (変換関数)	_ _op<T の型情報>	operator *	_ _ml
operator /	_ _dv	operator %	_ _md
operator +	_ _pl	operator -	_ _mi
operator <<	_ _ls	operator >>	_ _rs
operator ==	_ _eq	operator !=	_ _ne
operator <	_ _lt	operator >	_ _gt
operator <=	_ _le	operator >=	_ _ge
operator &	_ _ad	operator 	_ _or
operator ^	_ _er	operator &&	_ _aa
operator 	_ _oo	operator !	_ _nt
operator ~	_ _co	operator ++	_ _pp
operator --	_ _mm	operator =	_ _as
operator ->	_ _rf	operator +=	_ _apl
operator -=	_ _ami	operator *=	_ _amu
operator %=	_ _amd	operator <<=	_ _als
operator >>=	_ _ars	operator &=	_ _aad

演算子	エンコード	演算子	エンコード
operator =	_ _aor	operator ^=	_ _aer
operator ,	_ _cm	operator ->*	_ _rm

(2)クラス情報

クラス情報には、クラス名文字数とクラス名文字列がエンコードに埋め込まれます。

また、入れ子クラスの場合には、Q(入れ子レベル)__(最外側クラス情報)...(最内側クラス情報)がエンコードに埋め込まれます。

(3)修飾子情報

修飾子情報には、**const,volatile,signed,unsigned** の情報が下記に示す表の文字列がエンコードとして埋め込まれます。

修飾子	エンコード
const	C
volatile	V
unsigned	U
signed / - (signed char 以外は、 signed の有無に関わらず S はエンコード文字列に含まれません)	S

(4)引数型情報

引数型情報には、基本型、クラス型、派生型がエンコード文字列に埋め込まれます。

型	エンコード
void	v
char	c
short	s
int	i
long	l
float	f
double	d
long double	r
...	e

クラス名	(クラス名文字列数)(クラス名文字列)
型	エンコード
ポインタ	P
リファレンス	R
配列	A(要素数)_
メンバへのポインタ	M(クラス名文字列数)(クラス名文字列)
n 番目引数と同一型するとき	Tn(n は引数の n 番目を意味)

付録 H. 実行時ルーチン命名規則

実行時ルーチンの関数名の命名規則を以下に示します。

H.1 整数演算、浮動小数点演算、符号変換、ビットフィールド関数の命名規則

`_`[演算名][サイズ][符号][r][p][nm]

[サイズ] :b . . . 1 バイト

 :w . . . 2 バイト

 :l . . . 4 バイト

 :s . . . 4 バイト[単精度浮動小数点]

 :d . . . 8 バイト[倍精度浮動小数点]

[符号] :s . . . 符号付き

 :u . . . 符号なし

[r] :_subdr,_divdr のみ。それぞれ_subd,_divd とパラメタのスタックプッシュ順序
 が異なるときのみ

[p] :ペリフェラル時のみ付与。

[nm] :ノーマスク。ペリフェラルで割り込みノーマスク時のみ付与。

例外 :_muli

【注】[符号]は整数演算のみ付与

H.2 変換関数命名規則

`_`[サイズ]to[サイズ]

[サイズ] :i . . . 符号付き 4 バイト

 :u . . . 符号なし 4 バイト

 :s . . . 単精度浮動小数点

 :d . . . 倍精度浮動小数点

H.3 シフト関数の命名規則

`_[sta_]sft[方向][符号][ビット数]`

`[sta_]` :b . . . ビット数の付く場合のみ付与

`[方向]` :l . . . 左シフト

:r . . . 右シフト

`[符号]*1` :l . . . 論理シフト

:a . . . 算術シフト

`[ビット数]*2` :b . . . 0~31

【注】*1:`[符号]`は`[方向]`がrのときのみ付与

*2:`[ビット数]`は`[sta_]`があるときのみ付与

H.4 その他の関数の命名規則

領域移動、文字列比較、文字列コピーは特例。

付録I 割り込みハンドラ

SH3、SH3E、SH4用の割り込みハンドラの例を以下に示します。

```

;*****;
; Interrupt Starter Routine
;*****;

        .SECTION      inthandl, CODE, ALIGN=4
        .ORG          H'600
        .EXPORT_      _int_start
        .EXPORT_      _int_term

__int_start:
        STC.L          SSR, @-R15          ; save ssr
        STC.L          SPC, @-R15          ; save spc
;
        MOV.L          R8, @-R15           ; save work register
        ADD             # -4, R15           ; sr stack area
        MOV.L          R0, @-R15           ; save work register
        MOV.L          R1, @-R15           ; save work register
        MOV.L          INTEVT, R0          ; set INTEVT address to r0
        MOV.L          @R0, R0             ; set exception code to r0
        CMP/EQ #0, R0                      ; if INTEVT <> 0 then
        BF             label              ; branch to label:
;
        MOV.L          @R15+, R1           ; restore work register
        MOV.L          @R15+, R0           ; restore work register
        ADD             #16, R15           ;
        RTE             ;
        NOP             ;
label: MOV.L          R2, @-R15             ; save work register
;
        MOV.L          INTEVT, R0          ; set INTEVT address to r0
        MOV.L          @R0, R1             ; set exception code to r1
        MOVA           vcttbl, R0          ; set vector table address to r0
        SHLR2          R1                  ; 3bits shift-right exception code
        SHLR           R1
        ADD             # -(h'1c0>>3), R1 ; exception code - h'1c0
        MOV.L          @(R0, R1), R8       ; set interrupt function addr to r8

```

```

;

        MOVA        imasktbl, R0            ; set interrupt mask table addr to r8
        SHLR2       R1                      ; 2bits shift-right exception code
        MOV.B       @(R0, R1), R1          ; set interrupt mask to r1
        EXTU.B      R1, R1

;

        STC         SR, R0                  ; save sr to r0
        LDC         R0, SSR                  ; set current status to ssr
        MOV.L       IMASKclr, R2           ; set IMASK clear data to r1
        AND         R2, R0                  ; clear interrupt mask
        OR          R1, R0                  ; set interrupt mask
        MOV.L       RBBLclr, R1            ; set RB, BL clear data to r1
        AND         R1, R0                  ; (RB = BL = 0)
        MOV.L       R0, @(12, R15)         ; push sr

;

        MOVA        _ _int_term, R0        ; set _ _int_term addr to spc
        LDC.L       R0, SPC

;

        MOV.L       @R15+, R2              ; restore work register
        MOV.L       @R15+, R1              ; restore work register
        MOV.L       @R15+, R0              ; restore work register
        LDC.L       @R15+, SR              ; restore sr
        JMP         @R8                    ; jump to interrupt function
        MOV.L       @R15+, R8              ; restore work register

;

;*****;
; SH-3 Interrupt Terminator Routine
;*****;

        .ALIGN 4

_ _int_term:
        LDC.L       @R15+, SPC              ; load spc
        LDC.L       @R15+, SSR              ; load ssr
        RTE                     ; rte
        NOP

;

```

```
.ALIGN 4
RBLClr: .DATA.LH'4FFFFFFF
IMASKClr: .DATA.LH'FFFFFF0F
INTEVT: .DATA.LH'FFFFFFD8
;
vcttbl: ; Interrupt Vector Table
;
        .DATA.LH'00000000 ; NMI
        .DATA.LH'00000000 ; IRL = 0
        .DATA.LH'00000000 ; IRL = 1
;
        :
RES.L    26
;
        :
        .DATA.LH'00000000 ; RCVI
;
imasktbl: ; Interrupt Mask Table
        .DATA.BH'F0 ; NMI
        .DATA.BH'F0 ; IRL = 0
        .DATA.BH'E0 ; IRL = 1
;
        :
RES.B    26
;
        :
        .DATA.BH'00 ; RCVI
.END
```

[注] imasktblの内蔵周辺モジュールからの割り込み優先順位は、割り込みレベル設定レジスタ A ~ B (IPRA ~ B)で設定した優先順位と同じにしてください。

付録 J. リエントラントライブラリ

以下にリエントラントライブラリー一覧表を掲載します。表中、で示した関数は、`_errno` 変数を設定しますので、プログラム中で `_errno` を参照していなければリエントラントに実行できます。

表 J-1 リエントラントライブラリー一覧 (1)
リエントラント欄 :リエントラント x :ノリエントラント :_errnoを設定

No.	標準 インクルードファイル		関数名	リエントラント	No.	標準 インクルードファイル		関数名	リエントラント
1	stddef.h	1	offsetof		4	math.h	16	acos	
2	assert.h	2	assert	x			17	asin	
3	ctype.h	3	isalnum				18	atan	
		4	isalpha				19	atan2	
		5	isctrl				20	cos	
		6	isdigit				21	sin	
		7	isgraph				22	tan	
		8	islower				23	cosh	
		9	isprint				24	sinh	
		10	ispunct				25	tanh	
		11	isspace				26	exp	
		12	isupper				27	fexp	
		13	isxdigit				28	ldexp	
		14	tolower				29	log	
		15	toupper				30	log10	

表 J-1 リエントラントライブラリー一覧 (2)

No.	標準 インクルードファイル		関数名	リエントラント	No.	標準 インクルードファイル		関数名	リエントラント
4	math.h	31	modf		7	stdio.h	61	fputs	×
		32	pow				62	getc	×
		33	sqrt				63	getchar	×
		34	ceil				64	gets	×
		35	fabs				65	putc	×
		36	floor				66	putchar	×
		37	fmod				67	puts	×
5	setjmp.h	38	setjmp				68	ungetc	×
		39	longjmp				69	fread	×
6	stdarg.h	40	va_start				70	fwrite	×
		41	va_arg				71	fseek	×
		42	va_end				72	ftell	×
7	stdio.h	43	fclose	×			73	rewind	×
		44	fflush	×			74	clearerr	×
		45	fopen	×			75	feof	×
		46	freopen	×			76	ferror	×
		47	setbuf	×			77	perror	×
		48	setvbuf	×	8	stdlib.h	78	atof	
		49	fprintf	×			79	atoi	
		50	fscanf	×			80	atol	
		51	printf	×			81	strtod	
		52	scanf	×			82	strtol	
		53	sprintf				83	rand	×
		54	sscanf				84	srand	×
		55	vfprintf	×			85	calloc	×
		56	vprintf	×			86	free	×
		57	vsprintf				87	malloc	×
		58	fgetc	×			88	realloc	×
		59	fgets	×			89	bsearch	
		60	fputc	×			90	qsort	

表 J-1 リエントラントライブラリー一覧 (3)

No.	標準 インクルードファイル		関数名	リエントラント
8	stdlib.h	91	abs	
		92	div	
		93	labs	
		94	ldiv	
9	string.h	95	memcpy	
		96	strcpy	
		97	strncpy	
		98	strcat	
		99	strncat	
		100	memcmp	
		101	strcmp	
		102	strncmp	
9	string.h	103	memchr	
		104	strchr	
		105	strcspn	
		106	strpbrk	
		107	strrchr	
		108	strspn	
		109	strstr	
		110	strtok	×
		111	memset	
		112	strerror	
		113	strlen	
		114	memmove	

付録 K. 索引

K.1 日本語索引

ア行

アセンブラ埋め込みインライン展開	94
アセンブリプログラムとの結合	62
後処理データ領域	47
アンダフロー	502
位置指示子	220
インクルードファイル	7,213
インクルードファイルの読み込み方法	490
インターナルレベルメッセージ	143
ウォーニングレベルメッセージ	143
エラー指示子	220
エラーメッセージ	143
エラーメッセージ一覧	143
エラーレベルメッセージ	143
オブジェクト情報	31,34
オブジェクトプログラムの構造	46
オプション	7
オプション一覧	7
オプションの指定方法	6
オプションの組み合わせ	27
オーバフロー	502

カ行

外部名	63
外部名の相互参照方法	63
拡張機能	73
仮数部	497
仮想関数表	53,55
仮想関数表へのポインタ	53,55,56

仮想関数表領域	47
仮想基底クラスへのポインタ	54,55
ガードビット	219
空クラス	55,56
環境の仕様	485
環境変数	37
関数のインライン展開	93
関数の呼び出し	64
関数メンバへのポインタ	50,51
基数	219
起動方法	5
基本型	50
境界調整数	49
共用体型	51
組み込み関数	77
組み込み関数の使用方法	77
クラス型	51
グローバルベースレジスタ	78
グローバル変数のレジスタ割り付け	99
限界値	43
言語仕様	485
コーディング上の注意事項	102
構造体型	51
構造体型、共用体、列挙型、ビットフィールドの仕様	489
高速フーリエ変換	407
効率	406
コマンド指定情報	37
コンパイルリストの見方	31
コンパイラの環境変数	37
コンパイラの限界値	43
サ行	
サブコマンドファイル	10,20
識別子の仕様	485
シグナル	275
シグナル番号	275

指数部	497
システム組み込みの概要	109
実行環境の設定	118
実行時ルーチン	111
自動インライン展開	10,21
修飾子の仕様	489
終了処理関数の作成例	507
初期化データ領域	47,112
初期処理データ領域	47
除算器	20
書式	299
処理系定義	220
シンボルテーブルエントリ数	43,44
スカラ型	50
スケーリング	408,431
スタック切り換え指定	74,75
スタックフレーム	65
スタックポインタ	64
スタック領域	47,114,116
スタック領域の使用法	114
ステータスレジスタ	75,77
ストリーム入出力	216
正規化	219
正規化数	497
整数型とその値の範囲	487
整数の仕様	486
静的領域の割り付け	110
積和演算	80
セクション	46,47,110
セクション(#pragma)	90
セクション(オプション)	9,17
セクション切り替え機能	90
セクションの初期化	121
セクションの初期化ルーチンの例	123
セクション名	17
ゼロ拡張	57
宣言の仕様	489

ソースリスト情報	31
----------------	----

タ行

畳み込みと相関	457
ダミー領域	55,56
単精度浮動小数点ライブラリ	91
定義域エラー	245
低水準インタフェースルーチン	130
低水準インタフェースルーチンの作成例	510
定数領域	47,112
テキストファイル	218
データの内部表現	49
データフォーマット	402
データメンバへのポインタ型	50
デバッグ情報	7,8,14,105
統計情報	8,15,36
動的領域	114
動的領域の割り付け	114
トラップ命令リターン指定	74,75
トラブル発生時の対処方法	105

ナ行

内部表現	49,496
内部ラベル	43,44
2 バイトアドレス変数の指定	96
日本語	9,19,93
ヌル文字	217
ノード	21

ハ行

バイナリファイル	218
配列型	51
配列とポインタの仕様	488
範囲エラー	245

引数.....	67
引数の型変換.....	68
引数の割り付け領域.....	69
引数割り付けの具体例.....	504
非正規化数.....	497
非数.....	497
ビットフィールド.....	57,61,489
ヒープ領域.....	47,117
評価順序.....	102
標準インクルードファイル.....	213
標準入出力ファイル.....	218
標準ライブラリとの対応.....	28
標準ライブラリのエラーメッセージ.....	179
ファイルアクセスモード.....	219
ファイル拡張子.....	7
ファイル終了指示子.....	220
ファイルポインタ.....	216
ファイル名の付け方.....	7
フィルタ.....	431
フィールド幅.....	301
フェータルレベルメッセージ.....	143
符号拡張.....	57
符号部.....	497
浮動小数点数の限界値.....	488
浮動小数点数の仕様.....	487,496
ブリプロセッサの仕様.....	490
フレームサイズ.....	116
プログラム開発上のトラブル対処方法.....	105
プログラム作成上の注意事項.....	102
プログラムの構成例.....	118,124
プログラムの実行方式.....	45
プログラム領域.....	47
文の仕様.....	490
ベクタテーブルの設定.....	119
ベクタベースレジスタ.....	77
変換文字.....	302
ポジションインディペンデントコード.....	9,18

翻訳の仕様	485
マ行	
マクロ	214
マクロ名	214
マクロ名の定義	9,16
窓関数	426
丸め	12,26,219,487,501
未初期化データ領域	47
無限大	497
無効演算	502
メモリ領域の割り付け	110
文字の種類	226
文字の仕様	486
文字列内の日本語記述	9,19,93
文字列の共有	9,18
ヤ行	
予約語	131
ラ行	
ライブラリ	28,213
ライブラリ関数のエラーメッセージ	179
ライブラリ関数の実行環境の設定	124
ライブラリ関数の初期設定	126
ライブラリ関数仕様	211,491
リスト	8,15,31
リターンアドレス格納レジスタ	67
リターンコード	217
リターン値	65
リターン値の設定場所	71
リトルエンディアン	10,21,59
リファレンス型	50
レジスタ	65

レジスタとスタック領域の使用法	506
レジスタ退避・回復の制御	98
レジスタの仕様	488
レジスタ保証規則	65
列挙型	489

ワ行

割り込み関数	73
割り込み関数の使用方法	73

K.2 英語索引

A

abort 関数.....	357
abort ルーチン (異常終了関数).....	509
abs 関数	365
abs16(オプション).....	11,23
abs16(pragma 指定)	96
acos 関数.....	247
all(サブオプション)	11,23
align16(オプション).....	10,22
ASCII コード.....	515
asctime 関数	396
asin 関数	248
asmcode(サブオプション).....	9,16
assert マクロ	224
assert.h(標準ヘッダファイル)	223,491
atan 関数	249
atan2 関数	250
atof 関数	344
atoi 関数	345
atol 関数	346

B

bss(サブオプション).....	9,17
big(サブオプション).....	10,21
big endian	59
bool 型	50
branch(サブオプション).....	189,191
browser(サブオプション)	11,25
bsearch 関数.....	362
BUFSIZ	283

C

c(サブオプション).....	12,27
calloc 関数	353
case(オプション)	12,26
ceil 関数	265
char 型	50,59
CHAR_BIT	243
CHAR_MAX.....	243
CHAR_MIN.....	243
clearerr 関数	338
CLK_TCK.....	392
clock 関数	393
close ルーチン(低水準インタフェースルーチン).....	135,511
code(オプション).....	9,16
comment(オプション).....	9,19
const(サブオプション).....	9,17,18
const 型	104
ConvComplete 関数	458
ConvCyclic 関数	459
ConvPartial 関数	460
CopyFromX 関数	470
CopyFromY 関数	471
CopyToX 関数	468
CopyToY 関数	469
CopyXtoY 関数.....	466
CopyYtoX 関数.....	467
CorrCyclic 関数.....	463
Correlate 関数	461
cos 関数.....	251
cosh 関数	254
cpp(サブオプション)	12,27
cpu(オプション).....	8,12
cpu(サブオプション)	10,20
ctime 関数.....	397
ctype.h(標準ヘッダファイル)	225,491
C_\$VTBL	47,112

C/C++プログラムの実行方式.....	45
----------------------	----

D

data(サブオプション).....	9,17,18
DBL_DIG	241
DBL_EXP_DIG	242
DBL_MANT_DIG	241
DBL_MAX	240
DBL_MAX_EXP	241
DBL_MAX_10_EXP	241
DBL_MIN	240
DBL_MIN_EXP.....	241
DBL_MIN_10_EXP.....	241
DBL_NEG_EPS.....	242
DBL_NEG_EPS_EXP.....	242
DBL_POS_EPS.....	242
DBL_POS_EPS_EXP.....	242
debug(オプション).....	8,14
define(オプション).....	9,16
denormalization(オプション).....	11,26
difftime 関数	394
DIir 関数.....	441
DIir1 関数.....	443
div 関数.....	366
division(オプション).....	10,20
div_t	342
double(オプション).....	10,23
double(サブオプション).....	11,25
double 型	23,50
D_END_.....	47,112
D_INIT_	47,112

E

ecpp(オプション)	11,25
EDOM	244
elf(オプション)	189,194
ELF/DWARF フォーマット	185
endian(オプション)	10,21
ensigdsp.h(ヘッダファイル)	403
enum 型	50
EOF	283
ERANGE	244
errno	127,222,494
errno.h(標準ヘッダファイル)	494
euc(オプション)	9,19,93
euc(サブオプション)	11,23
exit 関数	358
exit ルーチン(プログラム終了関数)	508
expansion(サブオプション)	8,15
exp 関数	257
extern “C”	62

F

fabs 関数	266
fclose 関数	291
feof 関数	339
ferror 関数	340
fflush 関数	292
FFT 構造	409
FftComplex 関数	410
FftInComplex 関数	417
FftInReal 関数	418
FftReal 関数	412
fgetc 関数	319
fgets 関数	320
FILE	283
FILE 型	129

FILE 構造体.....	216
filt_ws.h(ヘッダファイル).....	403,431
Fir 関数.....	433
Fir1 関数.....	435
float(サブオプション).....	10,23
float.h(標準ヘッダファイル)	240
float 型	50,59
floor 関数.....	267
FLT_DIG	241
FLT_EXP_DIG.....	242
FLT_GUARD.....	240
FLT_MANT_DIG	241
FLT_MAX.....	240
FLT_MAX_EXP.....	241
FLT_MAX_10_EXP.....	241
FLT_MIN.....	240
FLT_MIN_EXP	241
FLT_MIN_10_EXP.....	241
FLT_NEG_EPS	242
FLT_NEG_EPS_EXP	242
FLT_NORMALIZE	240
FLT_POS_EPS	242
FLT_POS_EPS_EXP	242
FLT_RADIX	240
FLT_ROUNDS	240
fmod 関数	268
fopen 関数	293
fprintf 関数.....	299
fpu(オプション).....	11,25
fputc 関数	321
fputs 関数	322
fread 関数.....	330
free 関数	354
FreeDlir 関数.....	455
FreeFir 関数.....	453
FreeFft 関数.....	425
FreeIir 関数.....	454

FreeLms 関数.....	456
freopen 関数.....	295
frexp 関数.....	258
fscanf 関数.....	307
fseek 関数	334
ftell 関数	336
fwrite 関数.....	332

G

GBR(グローバルベースレジスタ).....	78,86
gbr_base(pragma 指定)	97
gbr_base1(pragma 指定)	97
GBR ベース変数の指定	97
GenBlackman 関数.....	427
GenGWnoise 関数	472
GenHamming 関数.....	428
GenHanning 関数.....	429
GenTriangle 関数	430
getc 関数	323
getchar 関数	324
getenv 関数.....	359
gmtime 関数	398
gets 関数	325
goptimize(オプション).....	12,27

H

help(オプション)	9,17
HUGE_VAL.....	244

I

IEEE	496
IfftComplex 関数	414
IfftInComplex 関数	420
IfftInReal 関数.....	421
IfftReal 関数.....	415

ifthen(サブオプション)	12,26
Iir 関数	437
Iir1 関数	439
include(オプション)	9,17
include(サブオプション)	8,15
information(オプション)	189,195
InitDIir 関数	451
InitFir 関数	449
InitFft 関数	424
InitIir 関数	450
InitLms 関数	452
inline(オプション)	10,21
inline(pragma 指定)	93
inline_asm(pragma 指定)	94
int 型	50,59
interrupt(pragma 指定)	74
INT_MAX	243
INT_MIN	243
isalnum 関数	227
isalpha 関数	228
isctrl 関数	229
isdigit 関数	230
isgraph 関数	231
islower 関数	232
isprint 関数	233
ispunct 関数	234
isspace 関数	235
isupper 関数	236
isxdigit 関数	237
J	
jmp_buf	269

L

labs 関数	367
lang(オプション)	12,27
latin1(オプション)	11,25
LDBL_DIG	241
LDBL_EXP_DIG	242
LDBL_MANT_DIG	241
LDBL_MAX	240
LDBL_MAX_EXP	241
LDBL_MAX_10_EXP	241
LDBL_MIN	240
LDBL_MIN_EXP	241
LDBL_MIN_10_EXP	241
LDBL_NEG_EPS	242
LDBL_NEG_EPS_EXP	242
LDBL_POS_EPS	242
LDBL_POS_EPS_EXP	242
ldexp 関数	259
ldiv 関数	368
ldiv_t	342
length(サブオプション)	8,15
Limit 関数	465
limits.h(標準ヘッダファイル)	243
listfile(オプション)	8,15
little(サブオプション)	10,21
little endian	21,59
Lms 関数	445
Lms1 関数	447
localtime 関数	399
log 関数	260
log10 関数	261
LogMagnitude 関数	423
long 型	50,59
long double 型	50
longjmp 関数	272
LONG_MAX	243

LONG_MIN.....	243
loop(オプション).....	11,23
loop(サブオプション).....	8,13
lseek ルーチン(低水準インタフェースルーチン)	138,512
L_tmpnam.....	283

M

machinecode(サブオプション)	9,16
machine.h(標準ヘッダファイル).....	77,90
macsave(オプション).....	10,22,65
malloc 関数.....	355
math.h(標準ヘッダファイル)	91,244,492
mathf.h(標準ヘッダファイル).....	91
MatrixMult 関数.....	473
MaxI 関数.....	479
Mean 関数	477
memchr 関数.....	380
memcmp 関数	377
memcpy 関数.....	372
memset 関数.....	389
message(オプション)	10,22
MinI 関数	480
modf 関数	262
MsPower 関数.....	476

N

near(サブオプション).....	12,26
nestinline(オプション).....	11,24
NDEBUG.....	223
noalign16(オプション).....	10,22
noloop(オプション).....	11,23
nomessage(オプション)	10,22
nooptimize(オプション).....	189,192
nortnext(オプション).....	11,24
novolatile(オプション).....	12,26

NULL..... 222

O

object(サブオプション)..... 8,15

objectfile(オプション)..... 8,16

off(サブオプション)..... 11,25

on(サブオプション)..... 11,25

onexit 関数..... 360

onexit ルーチン(終了処理関数)..... 507

onexit_t..... 342

open ルーチン(低水準インタフェースルーチン)..... 133,511

optimize(オプション)..... 8,13,189,190

outcode(オプション)..... 11,23

P

pack(オプション)..... 12,27

pack1(pragma 指定)..... 100

PeakI 関数..... 481

peripheral(サブオプション)..... 10,20

perror 関数..... 341

pic(オプション)..... 9,18

pow 関数..... 263

PR レジスタ..... 67

pragma..... 73,490

preinclude(オプション)..... 10,21

preprocessor(オプション)..... 11,25

printf 関数..... 311

program(サブオプション)..... 9,17

ptrdiff_t..... 222

ptrdiff_t 型..... 488,491

putc 関数..... 326

putchar 関数..... 327

puts 関数..... 328

Q

qsort 関数 364

R

raise 関数 277

RAM 112,114

rand 関数 351

RAND_MAX 342

read ルーチン(低水準インタフェースルーチン) 136,512

realloc 関数 356

regsave(pragma 指定) 98

remove 関数 287

rename 関数 288

register(サブオプション) 189,191

rewind 関数 337

ROM 112,114

ROM(リンケージエディタのサブコマンド) 114

round(オプション) 12,26

rtnext(オプション) 11,24

run(サブオプション) 11,23

S

safe(サブオプション) 189,191

same_code(サブオプション) 189,191

samecode_forbid(オプション) 189,193

samesize(オプション) 189,192

sbrk ルーチン(低水準インタフェースルーチン) 140,513

scanf 関数 312

SCHAR_MAX 243

SCHAR_MIN 243

section(オプション) 9,17

section(pragma 指定) 90

SEEK_CUR 283

SEEK_END 283

SEEK_SET.....	283
setbuf 関数	296
setjmp.h(標準ヘッダファイル).....	269,492
setjmp 関数	271
setvbuf 関数	297
sh1(サブオプション).....	8,12
sh2(サブオプション).....	8,12
sh2e(サブオプション).....	8,12
sh3(サブオプション).....	8,12
sh3e(サブオプション).....	8,12
sh4(サブオプション).....	8,12
SHC_INC	37
SHC_LIB.....	37
SHC_TMP.....	37
SHCPU	38
shift(サブオプション).....	8,13
short 型	50,59
show(オプション).....	8,15
SHRT_MAX	243
SHRT_MIN.....	243
SIGABRT	273
SIGFPE	273
SIGILL	273
SIGINT	273
signal 関数.....	275
signal.h(標準ヘッダファイル)	273
SIGSEGV	273
SIGTERM.....	273
SIG_DFL	273
SIG_ERR.....	273
SIG_IGN	273
sin 関数	252
single(サブオプション).....	11,25
sinh 関数.....	255
size(オプション).....	8,13
size_t	222
sjis(オプション).....	9,19,93

sjis(サブオプション).....	11,23
smachine.h(標準ヘッダファイル)	77,90
source(サブオプション)	8,15
SP(スタックポインタ).....	65,69,72,75,118,119
sp(スタック切り替え指定).....	75
speed(オプション).....	8,13
speed(サブオプション).....	189,191
sprintf 関数	313
sqrt 関数	264
SR(ステータスレジスタ)	75,77
srand 関数	352
sscanf 関数	314
start(リンケージエディタのサブコマンド).....	114
statistics(サブオプション).....	8,15
stdarg.h(標準ヘッダファイル)	278
stddef.h(標準ヘッダファイル).....	222,491
stderr	283
stdin	283
stdio.h(標準ヘッダファイル).....	283,493
stdlib.h(標準ヘッダファイル).....	342
stdout	283
strcat 関数	375
strchr 関数	381
strcmp 関数	378
strcpy 関数	373
strcspn 関数	382
strerror 関数	390
string(オプション)	9,18
string.h(標準ヘッダファイル)	369,494
string_unify(サブオプション).....	189,190
strlen 関数	391
strncat 関数	376
strncmp 関数	379
strncpy 関数	374
strpbrk 関数	383
strrchr 関数	384
strspn 関数	385

strstr 関数.....	386
strtod 関数.....	347
strtok 関数.....	387
strtol 関数.....	349
struct(サブオプション)	8,13
subcommand(オプション).....	10,20,190,195
switch(サブオプション).....	8,13
symbol_delete(サブオプション)	189,191
symbol_forbid(オプション).....	189,193
sysrof(オプション)	189,194
SYSROF フォーマット	185
sysrofplus(オプション).....	189,194
SYSROF PLUS フォーマット	185
SYS_OPEN	283
system 関数	361

T

table(サブオプション)	12,26
tan 関数	253
tanh 関数	256
time 関数	395
time.h(標準ヘッダファイル).....	392
time_t.....	392
tm.....	392
tmpfile 関数.....	289
tmpnam 関数.....	290
TMP_MAX	283
tn(トラップ命令リターン指定).....	74
tolower 関数	238
toupper 関数	239
TRAPA 命令	74,78

U

UCHAR_MAX.....	243
UINT_MAX	243

ULONG_MAX.....	243
umachine.h(標準ヘッダファイル).....	77,90
ungetc 関数.....	329
unpack(pragma 指定)	100
unsigned.....	50,59
USHRT_MAX.....	243

V

Variance 関数	478
va_arg マクロ	281
va_end マクロ.....	282
va_list.....	278
va_start マクロ	280
vfprintf 関数.....	315
VBR(ベクタベースレジスタ)	77
VEC_TBL(ベクタテーブル)	119,125
VectorMult 関数	475
volatile 型	489
volatile(オプション).....	12,26
vprintf 関数	317
vsprintf 関数	318

W

width(サブオプション).....	8,15
write ルーチン(低水準インタフェースルーチン).....	137,512

Z

zero(サブオプション).....	12,26
--------------------	-------

—

__call_end.....	119
__call_init.....	119
__CLOSEALL	129,130

_ _DATE_ _	490
_ _INIT	120,125
_ _INITLIB	126
_ _INITSCT	121,126
_ _INIT_IOLIB.....	128
_ _INIT_LOWLEVEL	127
_ _INIT_OTHERLIB	129
_ _IOFBF.....	283
_ _IOLBF	283
_ _IONBF	283
_ _TIME_ _	490

記号

.LINE	28
\$G0.....	97
\$G1.....	97