Simple and Light Interfaces for C and C++ users

SLLIB − Script-Like C-language library
ユーザーズリファレンスガイド 基本編

Version 1.4.2 日本語版 [改訂中] 2013-05-19

Credits

Software Development:

Chisato Yamauchi

Quality Assurance:

SEC Co., LTD.

Manual Document:

Chisato Yamauchi, Sachimi Fujishima and *SEC Co., LTD.*

Special Thanks:

Daisuke Ishihara, Hajime Baba, Iku Shinohara, Keiichi Matsuzaki, Michitaro Koike, Sergio Pascual and *Yukio Yamamoto*

Web page: `http://www.ir.isas.jaxa.jp/~cyamauch/sli/`

目 次

1 はじめに 12
1.1 SLLIB とは 12
1.2 SLLIB が生まれた理由 15
1.3 SLLIB の開発方針—libc の作法を踏襲する・libc の利点を活かす 15
1.4 C 言語の知識があれば十分!! 16
1.5 エンドユーザにとってのオブジェクト指向とは? 16
1.5.1 オブジェクト指向は特別な事ではない 16
1.5.2 オブジェクト指向の御利益 18
1.5.3 言葉の定義とコードの考え方 19

2 インストール 20
2.1 対応 OS 20
2.2 SLLIB のビルドと設置 20
2.2.1 方法 1—make 一発による方法 20
2.2.2 方法 2—configure と make による方法 21

3 チュートリアル 22
3.1 Hello World 22
3.2 ファイルのオープンと読み込み 23
3.2.1 標準ストリームを使った場合 23
3.2.2 最強の “万能型” ストリームを使った場合 (超オススメ!!) 23
3.2.3 libc の関数との対応関係 25
3.2.4 複雑なバイナリデータのエンディアン変換 26
3.2.5 GNUPLOT との連携 26
3.3 文字列の操作 27
3.3.1 基本 28
3.3.2 1 文字ずつのアクセス 29
3.3.3 ストリームからテキストファイルを読む場合での活用 29
3.3.4 文字列の編集 29
3.3.5 文字列の活用 30
3.3.6 拡張正規表現の活用 (後方参照も OK) 30
3.4 文字列配列の操作 31
3.4.1 いきなり代入 31
3.4.2 デバッグには dprint()!! 32
3.4.3 速攻で execv()，execvp() に渡す 33
3.4.4 全要素に対する文字列編集 33
3.4.5 配列の編集 33
3.4.6 main() の引数を使い易く 34
3.4.7 空白区切りや CSV 形式の文字列を分割して配列に入れる—split() メンバ関数 34
3.4.8 正規表現マッチの結果を格納 35
3.5 連想配列の操作 36
3.5.1 いきなり代入 36
3.5.2 デバッグには dprint()!! 36

3.5.3 全要素に対する文字列編集 37
3.5.4 編集 37
3.5.5 split_keys(), split_values() でデータファイルに簡単アクセス 37
3.6 多次元配列を楽に扱う 38
3.6.1 いきなり代入 (自動リサイズモード：一次元配列〜三次元配列) 39
3.6.2 次元数と配列長の変更 39
3.6.3 各次元ごとの配列長の変更 40
3.6.4 配列に対する演算 40
3.6.5 非自動リサイズモード (画像バッファ向き) 40
3.6.6 高速な一要素ずつのアクセス 41
3.6.7 IDL/Python 風の表現による画像の領域選択や演算 42
3.6.8 配列に対する統計量の計算 43
3.6.9 画像データのコンバイン 43
3.6.10 エンディアンの変換 44

4 SLLIB を使う前に知っておきたい事 45
4.1 NAMESPACE 45
4.2 NULL と 0 45
4.3 const char *, char *const *, const char *const * 46
4.4 参照 46
4.5 オブジェクトのポインタ変数と関数の引数・返り値 47

5 FAQ 48
5.1 コンパイル時のありがちな警告・エラー 48
5.1.1 warning: cannot pass objects of non-POD type 48
5.1.2 error: ‘xxx’ was not declared in this scope 48
5.1.3 error: call of overloaded ‘xxx’ is ambiguous 48
5.1.4 error: invalid conversion from ‘const char*’ to ‘char*’ 48
5.1.5 error: passing ‘xxx’ as ‘yyy’ argument of ‘zzz’ discards qualifiers 49

6 上級者向けの情報 49
6.1 ヒープにオブジェクトを作る場合の注意 49
6.2 ヒープにオブジェクトの配列を作りたい場合 49
6.3 構造体とクラスとの連携 50
6.4 例外の取り扱い, try {} & catch () 50

7 CSTREAMIO クラスとその継承クラスのサマリ 52
7.1 継承クラスのサマリ 52
7.2 基本クラス・継承クラスのメンバ関数実装の全体像 52

8 CSTREAMIO クラスとその継承クラスのリファレンス 54
8.1 CSTREAMIO クラスのメンバ関数 54
8.1.1 open(), openf(), vopenf() 54
8.1.2 close() 57
8.1.3 read(), write() 57

8.1.4 bread() . . . 59
8.1.5 bwrite() . . . 60
8.1.6 rskip() . . . 62
8.1.7 wskip() . . . 63
8.1.8 getchr() . . . 63
8.1.9 getstr() . . . 64
8.1.10 getline() . . . 65
8.1.11 scanf(), vscanf() . . . 66
8.1.12 putchr() . . . 69
8.1.13 putstr() . . . 70
8.1.14 printf(), vprintf() . . . 70
8.1.15 flush() . . . 74
8.1.16 eof(), error(), reseterr() . . . 74
8.1.17 seek() . . . 75
8.1.18 tell() . . . 76
8.1.19 is_seekable() . . . 76
8.2 STDSTREAMIO クラス . . . 78
8.2.1 オブジェクトの作り方 . . . 79
8.2.2 open(), openf(), vopenf() . . . 79
8.2.3 eprintf(), veprintf() . . . 80
8.2.4 eflush() . . . 81
8.2.5 seek(), rewind() . . . 82
8.2.6 tell() . . . 83
8.2.7 content_length() . . . 83
8.3 GZSTREAMIO クラス . . . 84
8.3.1 open(), openf(), vopenf() . . . 84
8.3.2 sync() . . . 87
8.4 BZSTREAMIO クラス . . . 89
8.4.1 open(), openf(), vopenf() . . . 89
8.5 HTTPSTREAMIO クラス . . . 93
8.5.1 open(), openf(), vopenf() . . . 93
8.5.2 content_length() . . . 95
8.5.3 user_agent().assign() . . . 95
8.6 FTPSTREAMIO クラス . . . 96
8.6.1 open(), openf(), vopenf() . . . 97
8.6.2 content_length() . . . 98
8.6.3 username().assign() . . . 99
8.6.4 password().assign() . . . 99
8.7 PIPESTREAMIO クラス . . . 101
8.7.1 open(), openf(), vopenf() . . . 101
8.8 DIGESTSTREAMIO クラス . . . 105
8.8.1 open(), openf(), vopenf() . . . 106
8.8.2 openp(), openpf(), vopenpf() . . . 107
8.8.3 is_write_mode() . . . 110

8.8.4 content_length() . . . 110
8.8.5 user_agent().assign() . . . 111
8.8.6 username().assign() . . . 111
8.8.7 password().assign() . . . 111
8.9 TERMLINEIO クラス . . . 112
8.9.1 open() . . . 113
8.9.2 set_prompt(), setf_prompt(), vsetf_prompt() . . . 114
8.9.3 automate_history() . . . 115
8.9.4 add_history() . . . 116
8.9.5 clear_history() . . . 117
8.9.6 stifle_history() . . . 118
8.9.7 unstifle_history() . . . 119
8.9.8 read_history(), readf_history(), vreadf_history() . . . 120
8.9.9 write_history(), writef_history(), vwritef_history() . . . 121
8.10 TERMSCREENIO クラス . . . 123
8.10.1 open() . . . 123
8.11 INETSTREAMIO クラス . . . 126
8.11.1 open() . . . 126
8.11.2 path() . . . 128
8.11.3 host() . . . 128
8.11.4 サンプルコード . . . 128

9 TSTRING クラス 130
9.1 オブジェクトの作り方—3 つの動作モード . . . 131
9.1.1 通常モード . . . 131
9.1.2 NULL 無しモード . . . 131
9.1.3 固定長バッファモード . . . 131
9.1.4 固定長バッファモードの制限 . . . 132
9.2 メンバ関数の引数の規則性 . . . 132
9.3 メンバ関数一覧 . . . 132
9.4 演算子 . . . 135
9.4.1 [] . . . 135
9.4.2 = . . . 136
9.4.3 += . . . 137
9.4.4 == . . . 137
9.4.5 != . . . 138
9.5 メンバ関数 . . . 139
9.5.1 length() . . . 139
9.5.2 max_length() . . . 140
9.5.3 cstr(), c_str() . . . 140
9.5.4 str_ptr(), str_ptr_cs() . . . 141
9.5.5 cchr() . . . 142
9.5.6 at(), at_cs() . . . 142
9.5.7 update_length() . . . 143

9.5.8 dprint() 144
9.5.9 getstr() 144
9.5.10 copy() 145
9.5.11 swap() 147
9.5.12 init() 148
9.5.13 printf(), vprintf(), assign(), assignf(), vassignf() 149
9.5.14 implode() 151
9.5.15 import_binary() 152
9.5.16 put(), putf(), vputf() 152
9.5.17 strcat(), strncat(), append(), appendf(), vappendf() 154
9.5.18 insert(), insertf(), vinsertf() 156
9.5.19 replace(), replacef(), vreplacef() 157
9.5.20 erase() 160
9.5.21 clean() 161
9.5.22 resize() 161
9.5.23 resizeby() 162
9.5.24 crop() 163
9.5.25 chomp() 164
9.5.26 trim() 164
9.5.27 ltrim() 166
9.5.28 rtrim() 166
9.5.29 strreplace() 167
9.5.30 regreplace() 168
9.5.31 tolower() 171
9.5.32 toupper() 172
9.5.33 expand_tabs() 172
9.5.34 contract_spaces() 174
9.5.35 atoi(), atol(), atoll() 175
9.5.36 atof() 176
9.5.37 strtol(), strtoll() 178
9.5.38 strtoul(), strtoull() 179
9.5.39 strtod() 181
9.5.40 scanf(), vscanf() 182
9.5.41 strcmp(), compare() 183
9.5.42 strncmp(), compare() 185
9.5.43 strcasecmp(), strncasecmp() 186
9.5.44 isalpha(), isalnum(), isdigit(), islower(), isupper(), 他 188
9.5.45 strchr(), find() 189
9.5.46 strstr(), find() 190
9.5.47 strrchr(), rfind() 191
9.5.48 strrstr(), rfind() 192
9.5.49 find_first_of() 193
9.5.50 find_last_of() 195
9.5.51 find_first_not_of() 197

9.5.52 find_last_not_of() 199
9.5.53 strpbrk() 201
9.5.54 strrpbrk() 202
9.5.55 strspn() 203
9.5.56 strrspn() 206
9.5.57 strcspn() 207
9.5.58 strmatch(), fnmatch(), pnmatch() 209
9.5.59 regmatch() 210

10 TARRAY_TSTRING クラス 215

10.1 オブジェクトの作り方 216
10.2 メンバ関数一覧 216
10.3 演算子 218
10.3.1 [] 218
10.3.2 = ... 219
10.3.3 += 220
10.3.4 += 221
10.4 メンバ関数 222
10.4.1 length() 222
10.4.2 cstrarray() 223
10.4.3 cstr(), c_str() 223
10.4.4 at(), at_cs() 224
10.4.5 dprint() 225
10.4.6 copy() 226
10.4.7 swap() 227
10.4.8 init() 228
10.4.9 assign(), assignf(), vassignf() 229
10.4.10 assign(), vassign() 230
10.4.11 explode() 231
10.4.12 split() 232
10.4.13 regassign() 234
10.4.14 put(), putf(), vputf() 236
10.4.15 put(), vput() 237
10.4.16 append(), appendf(), vappendf() 239
10.4.17 append(), vappend() 240
10.4.18 insert(), insertf(), vinsertf() 242
10.4.19 insert(), vinsert() 243
10.4.20 replace(), replacef(), vreplacef() 244
10.4.21 replace(), vreplace() 246
10.4.22 erase() 248
10.4.23 clean() 249
10.4.24 resize() 249
10.4.25 resizeby() 251
10.4.26 crop() 251

10.4.27 chomp() 252
10.4.28 trim() 252
10.4.29 ltrim() 253
10.4.30 rtrim() 254
10.4.31 strreplace() 254
10.4.32 regreplace() 255
10.4.33 tolower() 256
10.4.34 toupper() 257
10.4.35 expand_tabs() 257
10.4.36 contract_spaces() 257
10.4.37 find_elem() 258
10.4.38 rfind_elem() 259
10.4.39 find() 260
10.4.40 rfind() 262
10.4.41 find_matched_str() 264
10.4.42 find_matched_fn() 265
10.4.43 find_matched_pn() 266
10.4.44 regmatch() [Normal 版] 267
10.4.45 regmatch() [Advanced 版] 269

11 ASARRAY_TSTRING クラス 272

11.1 オブジェクトの作り方 273
11.2 メンバ関数一覧 273
11.3 演算子 275
11.3.1 [] 275
11.3.2 = ... 275
11.4 メンバ関数 276
11.4.1 length() 276
11.4.2 cstrarray() 277
11.4.3 cstr(), c_str(), cstrf(), vcstrf() 278
11.4.4 at(), atf() 279
11.4.5 at_cs(), atf_cs() 281
11.4.6 index(), indexf(), vindexf() 282
11.4.7 key() 283
11.4.8 keys() 283
11.4.9 values() 284
11.4.10 dprint() 284
11.4.11 swap() 285
11.4.12 init() 286
11.4.13 assign(), assignf(), vassignf() 287
11.4.14 assign(), vassign() 288
11.4.15 assign_keys() 289
11.4.16 assign_values() 290
11.4.17 split_keys() 291

11.4.18 split_values() 292
11.4.19 append(), appendf(), vappendf() 294
11.4.20 append(), vappend() 295
11.4.21 insert(), insertf(), vinsertf() 297
11.4.22 insert(), vinsert() 298
11.4.23 erase() 299
11.4.24 clean() 301
11.4.25 rename_a_key() 302
11.4.26 chomp() 302
11.4.27 trim() 303
11.4.28 ltrim() 304
11.4.29 rtrim() 304
11.4.30 strreplace() 305
11.4.31 regreplace() 306
11.4.32 tolower() 307
11.4.33 toupper() 308
11.4.34 expand_tabs() 308
11.4.35 contract_spaces() 309

12 MDARRAY_*クラス 310

12.1 オブジェクトの作り方 311
12.1.1 何も引数を指定しない方法 311
12.1.2 配列の大きさを与える方法 312
12.1.3 配列の大きさと初期値を与える方法 312
12.2 数学関数 313
12.3 メンバ関数一覧 314
12.4 演算子 317
12.4.1 [] 317
12.4.2 () 318
12.4.3 = ... 319
12.4.4 = ... 320
12.4.5 += 321
12.4.6 += 321
12.4.7 -= 322
12.4.8 -= 323
12.4.9 *= 324
12.4.10 *= 325
12.4.11 /= 325
12.4.12 /= 326
12.4.13 + ... 327
12.4.14 + ... 327
12.4.15 - 328
12.4.16 - 329
12.4.17 * 329

12.4.18 * 330
12.4.19 / 330
12.4.20 / 331
12.4.21 == 332
12.4.22 != 332
12.5 メンバ関数 333
12.5.1 size_type() 333
12.5.2 bytes() 335
12.5.3 dim_length() 335
12.5.4 length() 336
12.5.5 byte_length() 336
12.5.6 col_length() 337
12.5.7 row_length() 338
12.5.8 layer_length() 338
12.5.9 at(), at_cs() 339
12.5.10 dvalue() 340
12.5.11 lvalue(), llvalue() 341
12.5.12 default_value(), assign_default() 342
12.5.13 auto_resize(), set_auto_resize() 343
12.5.14 rounding(), set_rounding() 343
12.5.15 dprint() 344
12.5.16 carray(), array_ptr() 345
12.5.17 get_elements() 346
12.5.18 put_elements() 347
12.5.19 getdata() 348
12.5.20 putdata() 349
12.5.21 reverse_endian() 350
12.5.22 init() 352
12.5.23 assign() 354
12.5.24 put() 355
12.5.25 swap() 356
12.5.26 move() 357
12.5.27 cpy() 358
12.5.28 insert() 359
12.5.29 crop() 360
12.5.30 erase() 361
12.5.31 resize() 362
12.5.32 resizeby() 364
12.5.33 increase_dim() 365
12.5.34 decrease_dim() 365
12.5.35 swap() 366
12.5.36 convert() 367
12.5.37 ceil() 367
12.5.38 floor() 368

12.5.39 round() 369
12.5.40 trunc() 370
12.5.41 abs() 370
12.5.42 compare() 371
12.5.43 copy() 372
12.5.44 copy() 373
12.5.45 cut() 374
12.5.46 cut() 375
12.5.47 clean() 377
12.5.48 fill() 378
12.5.49 add() 379
12.5.50 multiply() 381
12.5.51 paste() 382
12.5.52 add() 383
12.5.53 subtract() 385
12.5.54 multiply() 386
12.5.55 divide() 387

1 はじめに

1.1 SLLIB とは

SLLIB (えすえるりぶ; Script-Like C-language library) は，**各種スクリプト言語 (Perl・PHP・Python・IDL 等) と同じ感覚で「ストリーム」「文字列」「多次元配列」を扱えるようにする実用的な API を，C 言語に追加する**ライブラリです．日常的に頻出する処理で露見する C 言語の弱点を最小化するので，**C 言語でのコーディング・デバッグの労力を減らして開発効率を改善する事**ができます．

例えば，次のコードを見てください．

```
#include <sli/tarray_tstring.h>
using namespace sli;

int main()
{
    tarray_tstring arr;
    arr[0] = "foo";            /* arr[0] に "foo" を代入 */
    arr[1] = "bar";            /* arr[1] に "bar" を代入 */
```

この例は，文字列`"foo"`と`"bar"`をそれぞれ文字列の配列に代入するコードを，SLLIB を使って書いたものです．みなさんの C 言語のコードとはどこが異なるでしょうか？このコードでは，ポインタ配列や文字列バッファの確保に関する記述が一切無いという事です．なぜ無いのかといえば，**必要なメモリ領域はライブラリ側で自動的に確保・管理している**ので，ユーザのコードでは書く必要が無いからです (もちろん，メモリ領域の開放も自動的に行なわれます)．

次のコードは，ネットワーク経由のストリームと正規表現を使った例です．

```
#include <sli/digeststreamio.h>
#include <sli/tstring.h>
using namespace sli;

int main()
{
    tstring line;
    digeststreamio f_in;
    f_in.open("r", "http://www.foo.bar/data/foo.txt.gz");   /* ファイルを open */
    while ( (line=f_in.getline()) != NULL ) {           /* 1 行ずつ読む */
        line.chomp();                                   /* 改行文字を削除 */
        if ( 0 <= line.regmatch("^[a-zA-Z]",NULL) ) {   /* 正規表現マッチを試行 */
            printf("%s\n",line.cstr());                 /* 表示する */
        }
    }
    f_in.close();                                       /* ファイルを close */
```

この例は，**Web サーバ** http://www.foo.bar/ **に置かれた gzip 圧縮されたテキストファイル** foo.txt.gz を開き，解凍しながらアルファベットから始まる行のみ表示するコードです．スクリプト言語でおなじみの文字列の編集や**正規表現マッチも簡単に行なう事ができます**．

もう 1 つ，今度は多次元配列を扱うコードです．倍精度浮動小数点型の配列の一部を選択し，それ

らに対して 2 で割り，log をとって表示するコードです．

```
#include <sli/mdarray.h>
#include <sli/mdarray_math.h>
using namespace sli;

int main()
{
    const double arr0_src[] = {0.02, 0.2, 2.0,  20.0, 200.0, 2000.0,  1, 2, 3};
    mdarray_double arr0, arr1;
    /* 配列の大きさを設定 (x,y) */
    arr0.resize_2d(3,3);
    arr0.put_elements(arr0_src, 3*3);
    /* arr0 の最初の 2 行を選択して arr1 へ代入 */
    arr1 = arr0.sectionf("*, 0:1");
    /* 2 で割ったすべての要素に対して log10() を計算 */
    arr1 = log10(arr1 / 2);
    for ( size_t j=0 ; j < arr1.length(1) ; j++ ) {
        for ( size_t i=0 ; i < arr1.length(0) ; i++ ) printf("[%g]", arr1(i,j));
        printf("\n");
    }
```

これを実行すると次の結果を得ます．

```
[-2][-1][0]
[1][2][3]
```

IDL や Python のように，"*,0:1"のような表現 (x,y の順で指定) による範囲指定や，全要素に対する演算にオペレータや数学関数を適用する事ができ，配列の選択・演算のためのコードの大幅な簡素化が可能です．さらに，オブジェクト内で 2D，3D 配列用のポインタ配列を自動生成する事も可能で，不必要な手間をかけずに最高のパフォーマンスを狙う事が可能です．

このように，SLLIB は日常的に頻出する用途において C 言語の弱点を補う強力なライブラリで，主に次のような機能を提供します．

- 様々なストリーム (圧縮ファイル，ネットワーク等) を統一的に手軽に扱える API.
- 文字列処理の強化．正規表現，文字列配列，文字列連想配列のための API.
- 多次元配列を手軽に扱える API.

  IDL/Python 風の表現による要素指定，全要素に対するオペレータ・数学関数，統計用関数，2D または 3D 配列のポインタ配列の自動生成 等.

C 言語開発環境の基礎部分の強化により，コーディングとデバッグの労力を減らし，開発効率を改善します．

表 1に SLLIB のストリームに関する機能の全体像を示します．どのストリームでも，基本的な API は全く同じである事がおわかりいただけると思います．

クラス名		stdstreamio	gzstreamio	bzstreamio	httpstreamio	ftpstreamio	pipestreamio	digeststreamio	termlineio	termscreenio	inetstreamio
基本クラスcstreamioのメンバ関数											
`open()`, etc.	§8.1.1	§8.2.2	§8.3.1	§8.4.1	§8.5.1	§8.6.1	§8.7.1	§8.8.1	§8.9.1	§8.10.1	§8.11.1
`close()`	§8.1.2	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`read()`	§8.1.3	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`write()`	§8.1.3	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`bread()`	§8.1.4	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`bwrite()`	§8.1.5	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`rskip()`	§8.1.6	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`wskip()`	§8.1.7	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`getchr()`	§8.1.8	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`getstr()`	§8.1.9	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`getline()`	§8.1.10	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`scanf()`	§8.1.11	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`putchr()`	§8.1.12	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`putstr()`	§8.1.13	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`printf()`	§8.1.14	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`flush()`	§8.1.15	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`eof()`, etc.	§8.1.16	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`seek()`, etc.	§8.1.17	§8.2.5	-	-	-	-	-	⇐	-	-	-
継承クラスの新規メンバ関数											
`eprintf()`		§8.2.3	-	-	-	-	-	-	-	-	-
`eflush()`		§8.2.4	-	-	-	-	-	-	-	-	-
`sync()`		-	§8.3.2	-	-	-	-	-	-	-	-
`content_length()`		§8.2.7	-	-	§8.5.2	§8.6.2	-	§8.8.4	-	-	-
`user_agent.assign()`		-	-	-	§8.5.3	-	-	§8.8.5	-	-	-
`username.assign()`		-	-	-	-	§8.6.3	-	§8.8.6	-	-	-
`password.assign()`		-	-	-	-	§8.6.4	-	§8.8.7	-	-	-
`openp()`, etc.		-	-	-	-	-	-	§8.8.2	-	-	-
`is_write_mode()`		-	-	-	-	-	-	§8.8.3	-	-	-
`set_prompt()`		-	-	-	-	-	-	-	§8.9.2	-	-
`automate_history()`		-	-	-	-	-	-	-	§8.9.3	-	-
`add_history()`		-	-	-	-	-	-	-	§8.9.4	-	-
`clear_history()`		-	-	-	-	-	-	-	§8.9.5	-	-
`stifle_history()`		-	-	-	-	-	-	-	§8.9.6	-	-
`unstifle_history()`		-	-	-	-	-	-	-	§8.9.7	-	-
`read_history()`		-	-	-	-	-	-	-	§8.9.8	-	-
`write_history()`		-	-	-	-	-	-	-	§8.9.9	-	-
`path()`		-	-	-	-	-	-	-	-	-	§8.11.2
`host()`		-	-	-	-	-	-	-	-	-	§8.11.3

表 1: 基本クラス cstreamio とその継承クラスが提供するメンバ関数一覧．記号「⇐」は基本クラスのメンバ関数を継承している事を示す．§ の表記があるものについては，メンバ関数が再定義されているか，追加定義されている事を示す．

1.2 SLLIB が生まれた理由

§1.1で，SLLIB は C 言語の弱点を補うと書きましたが，それは C++標準ライブラリが作られた目的の 1 つでもあります．

さてみなさんは,「`cout << "foo" << endl;`」というコードを見た事があるでしょうか？この書き方は,「`printf("foo");`」を C++標準ライブラリの作法に従って書いたものです．C++標準ライブラリでこのような記法を使えるようにしたのには理由があり，C 言語の弱点を補っているというのも事実です[1]．しかし，ビットシフトの演算子を全く別の意味に再定義するなんて納得いかない，あるいは処理内容が想像しにくい，と思う方は少なくないと思います．

C++標準ライブラリでは，libc(C 言語の標準ライブラリ) には無い多くの新しい API(主にアルゴリズム) を提供する一方で，万人にとって容易に受け入れやすいとは言えない “新しい作法” を API の仕様として導入しました．こういった「作法」の学習が壁となり，C++を敬遠してきた開発者は意外と多いのではないかと思います．

しかし，そんな事のために C++で提供されるコーディングの労力の最小化に役立つ利点 (§1.5で解説) を活用できないとしたら，もったいない事です．実際，§1.1で紹介したスクリプト言語のような API は，C++を使ってはじめて自然な形をとれるようになるのです．

そこで，この「もったいない」状況をなんとかしようという事で，“新しい作法” で API を固めるのではなく，libc の作法を踏襲しつつ C++の利点を活用した便利な基本ライブラリを作ろうと思ったわけです．こうして，チマチマと libc の関数の大部分をまとめなおしていくうちに SLLIB の原形ができ，スクリプト言語風の様々な API を持つライブラリになっていったのです．

1.3 SLLIB の開発方針—libc の作法を踏襲する・libc の利点を活かす

	C++標準ライブラリ	**SLLIB**
機能	広範囲に及ぶ	日常的に頻出する用途に限定
libc 風の関数	排除	最大限取り込む
printf() の記法	排除	最大限利用
変数の型	独自の型が多数	libc で定義されている型のみ使用
演算子	独自性が強い	独自性は最小限
手軽さ	△	○

表 2: C++標準ライブラリと SLLIB とのコンセプトの違い．

§1.2 では，C++標準ライブラリが libc の作法のかなりの部分と決別してしまった，という事を書きました．それに対して，SLLIB は libc の作法を踏襲する路線をとるライブラリです．

SLLIB のコンセプトを C++標準ライブラリの場合と比較してわかりやすくまとめたのが表 2です．このように，SLLIB は機能を頻出する用途に限定し，API を C 言語的作法に近くする事により，ユーザが新しく学ばなければならない事を最小限に抑えています．したがって，SLLIB は **C++を全く知らない方でも手軽に利用できます**．また，演算子の独自性を最小限に抑えているので，C++標準ライブラリのように「知らない人にはコードが読めない」という事態にはなりません．

C 言語的作法の代表例として挙げられるのは，何といっても printf() 関数のフォーマット&可変長引数でしょう．printf() 関数の引数では，豊富なフォーマット指示子による文字列への変換，文字列

[1] とはいえ，google のコーディング規約では，printf() 関数と C++のストリームのどちらにも利点・欠点があると指摘しています．結局，規約では C++標準ライブラリのストリームを使わない事を推奨しています．

の結合などが，非常に短かく簡単に記述できます．printf() はやっぱり最強，というわけで SLLIB では，printf() の記法を様々な API で使えるようにしています．

例えば，§1.1で登場した `f_in.open()` の別のバージョンとして，`openf()` というものもあり，libc の printf() 関数の引数と同様に

```
f_in.openf("r", "http://%s/%s", "www.foo.bar", "data/foo.txt.gz");
```

と書く事もできます．

SLLIB の文字列処理においても，libc の string.h，ctype.h 等でみられる作法や関数名を踏襲した API(例えば，atof()，strchr() 等) を豊富に用意していますので，C 言語プログラマにとって親しみやすいライブラリになっています．

また，SLLIB は，C++標準ライブラリが持たない libc の機能を補完してくれるため，すでに C++ を利用しているユーザにとっても，コードをより洗練する助けになるでしょう．SLLIB は libc の関数の大部分をクラスのメンバ関数で実装していますから，従来の libc + C++標準ライブラリ，すなわち「手続き」と「オブジェクト指向」の混在コードから，SLLIB+ C++標準ライブラリ，すなわち，より純粋な「オブジェクト指向」のコードへ移行する事ができます．

1.4 C 言語の知識があれば十分!!

ここまでの内容で，SLLIB は C++で書かれている事がバレてしまったかもしれません．ここで，「げっ! C++なんて使わないよ!!」と思ったあなた．安心してください．**C++は C の上位互換ですから，これまで通り，C 言語のコードと同じように「`#include <stdio.h>`」「`printf(...);`」等と書けば，C++コンパイラでコンパイルする事ができます**．C++だからといって，C++標準ライブラリの作法「`cout << "foo" << endl;`」に従う必要はありません．

§1.3で述べたように，SLLIB は，C 言語の作法を最大限使えるように作られています．もちろん，C++的な部分についてはマニュアルを C 言語風に工夫する事で，C++未経験な方でも利用方法が容易に理解できるようになっています．ですから，SLLIB は C 言語の知識があれば誰でも簡単に利用できます．ぜひ，スクリプト言語のような手軽さを体験してほしいと思います．

1.5 エンドユーザにとってのオブジェクト指向とは?

§1.3でも少し出てきましたが，みなさんは「オブジェクト指向」という言葉を聞いた事があるでしょうか? C 言語にスクリプト言語の変数のような実装を実現する場合に活躍するのが「オブジェクト指向」という手法です．世の中の多くの書籍では「モデリング」「メッセージパッシング」のような言葉を使って「ライブラリの開発者向けのオブジェクト指向」を解説しているわけですが，ライブラリを単に使うだけのエンドユーザにとっては「この手法によってコーディングの労力を最小化できる事」がわかれば十分です．

この章では，エンドユーザにとって必要となる知識に絞って，オブジェクト指向のお話をします．みなさんが経験してきた事を元に解説していきますから，怖がらずに読んでください．

1.5.1 オブジェクト指向は特別な事ではない

SLLIB は，オブジェクト指向のライブラリです．オブジェクト指向という言葉を見て,「また難しい事を…」と思われる方もいらっしゃるかもしれませんが，全く心配する必要はありません．なぜなら，オブジェクト指向的な書き方は，C でも FORTRAN でも登場するからです．間違いなくみなさ

	手続き型	**オブジェクト指向型**
概念図	関数(サブルーチン) ⇔ 引数1 / 引数2 / 引数3 → 返り値	**オブジェクト(処理の主人公)** ⇔ 関数(演算子) ⇔ 引数1 / 引数2 → 返り値
C 言語の例	`fprintf(fp,"foo");`	`a += b;`
FORTRAN の例	`call sub(x,y,z)`	`x .gt. y`

表 3: 手続き型とオブジェクト指向型の概念図と，C 言語，FORTRAN にみられるそれぞれの場合の書き方の例．ほとんどの言語は,「手続き型」と「オブジェクト指向型」の両方の書き方を文法として取り入れている．

んもすでにオブジェクト指向的コードを書いた経験があります．

図 3 に「手続き型」と「オブジェクト指向型」の概念図と，C 言語と FORTRAN のコードの例を示します．この例が示すとおり，C 言語にも FORTRAN にも「手続き型」「オブジェクト指向型」の両方の文法が存在します．関数 (サブルーチン) の呼び出しのような「命令主体」の書き方を「手続き型」，それに対して関数を呼び出すという動作は同じものの，その時の「処理の主対象」を文法上特別扱いするのが「オブジェクト指向型」というわけです．概念図の「手続き型」の引数の 1 つを関数の左側に移動させると，オブジェクト指向型と同じ形になりますよね．

図 3 の例を見ていきましょう.「手続き型」の例「`fprintf(fp,"foo");`」「`call sub(x,y,z)`」では，関数 (サブルーチン) の引数のどれかに処理の主対象があるのかもしれませんが，文法上はすべての引数は対等です．一方，図の「オブジェクト指向型」の例「`a += b;`」「`x .gt. y` 」は，常に最も左にいる変数が処理の主対象であり，文法上でも特別扱いされている事がわかります．

このように，処理の主対象 (=オブジェクト) を明確にする「オブジェクト指向型」の書き方を使う頻度が高い言語を「オブジェクト指向言語」，逆に「`fprintf(fp,"foo");`」のような「命令主体」の書き方を使う頻度が高い言語を「手続き言語」と呼んでいるのです．C 言語や FORTRAN は，最終的には関数 (サブルーチン) を作ってそれらを組合せて処理を達成する事になるので,「手続き言語」に分類されます．

では「オブジェクト指向言語」ってどんな言語なのでしょうか？「`+=`」とか「`.gt.`」のような記号や略称ばかり書く事になるのでしょうか？もちろん，そんな事はありません．C++のような「オブジェクト指向言語」では，この「`+=`」とか「`.gt.`」の部分を一般化して,「. 文字列 (引数⋯)」の形で記述できるように言語仕様を拡張し，例えば次のように書けるようにしたのです[2)]．

```
a.add(b);           /* a に b を加算 */
```

これが典型的なオブジェクト指向言語の書き方です.「`a += b;`」の場合と非常に似ている事にお気付きでしょうか．処理の主対象「`a`」を先頭に書き，その後に動詞を書きます．オブジェクト指向言語とは，このように「`+=`」「`.gt.`」の書き方を一般化した「. 文字列 (引数⋯)」の形を主に使ってコードを書いていく言語の事を言います．ですから，オブジェクト指向言語というのは，このような考え方の原点をみてみると特別な言語ではないと言えます．図 3の右側の概念を主に使っていく，たったそれだけの事です．

[2)] 実際にはこのように書けるとは限りません．

スクリプト言語 Perl では「=~」や「.=」といった記号も登場します．みなさんも使った事があるのではないでしょうか．もちろん，これらもオブジェクト指向的な書き方で，左側に処理の主対象がいます．ひょっとすると言語によっては「#=」や「;=」なんて記号も存在するかもしれません．しかし，こういった記号類は言語仕様を知らない人がその意味を理解できるとは限りません．そのように考えると,「a.add(b);」のようなオブジェクト指向言語の書き方は，コードの可読性を確保する上でも優れた方法である事が理解できると思います.

1.5.2 オブジェクト指向の御利益

このように，オブジェクト指向でコードを書くと，処理の主対象が明確になり，コードが読み易くなります．さらに，オブジェクト指向でコードを書くともっと大きなご利益があります.

C の場合,「型」は int や double といった固定バイト長のものしか扱えませんでした．したがって，可変長の文字列や可変長の配列といったものは，必ず，malloc()，realloc()，free() を使って領域を計算・確保・開放しなければならず，どうしても処理内容のわりにコードが大きくなってしまいます．確保した領域は，ポインタ変数を使ってアクセスするので，領域外へのアクセスで SEGV，といった経験はどなたでもお持ちだと思います．最も悩まされるのは，free() の書き忘れによるメモリリークでしょう．もし，可変長の文字列や配列を管理する機構が「型」に備われば，基本的にこういった問題はすべて解決します．まさに，オブジェクト指向はその事を実現できるのです．C++でも,「=」や「+=」といった書き方を一般化して,「. 文字列 (引数…)」の形を使えるように言語仕様を拡張したのはそのためで，§1.5.1で登場した a.add(c); の「add()」の部分をライブラリの開発者が作り込む事により，ユーザは，可変長の文字列や配列をより簡単かつ安全に扱えるようになるのです．例えば，SLLIB を使って 2 つの文字列 "abc" "012" を結合する場合，次のように書けます.

```
tstring str;
str.printf("abc");          /* "abc" を str に代入 */
str.append("012");          /* "012" を str に追加 */
```

ありがたい事に，malloc() や realloc() を使った領域確保の処理は，.printf() や.append() がすべてやってくれるのです．しかも，str が消滅する時は，malloc() 等で確保されていた領域が自動的に開放されるので，メモリリークの心配もありません．したがってユーザは，単に何をやりたいかをオブジェクト str の後に書くだけといった形になり，思考を処理内容に集中できるようになります．もちろん，コードの量も圧倒的に小さくなります.

オブジェクト指向の御利益はそれだけではありません．最大の御利益は，**ユーザが覚えるべき事を最小限にできる**事です．オブジェクト指向の言語では,「継承」というルールによって，ライブラリの開発者が非統一な仕様の API を作れないようになっているのです．SLLIB のストリーム入出力の API でも,「継承」が大活躍しており，ある種類のストリームの API を使えるようになれば，他の種類のストリームも全く同じ方法で使えるようになっています.

ここで，SLLIB を使ったユーザ・コードの例を示します．左側は foo.txt をオープンして表示するコード，右側は foo.txt.gz をオープンして解凍しながら表示するコードです.

```
#include <sli/stdstreamio.h>
using namespace sli;

int main()
{
    stdstreamio f_in;
    int status = -1;
    char buf[256];
    /* ファイルをオープン */
    status = f_in.open("r", "foo.txt");
    if ( status != 0 ) goto quit;
    /* 1 行ずつ読み，表示する */
    while ( f_in.getstr(buf,256) != NULL ) {
        printf("%s",buf);
    }
    f_in.close();
 quit:
    return status;
}
```

```
#include <sli/gzstreamio.h>
using namespace sli;

int main()
{
    gzstreamio f_in;
    int status = -1;
    char buf[256];
    /* ファイルをオープン */
    status = f_in.open("r", "foo.txt.gz");
    if ( status != 0 ) goto quit;
    /* 1 行ずつ読み，表示する */
    while ( f_in.getstr(buf,256) != NULL ) {
        printf("%s",buf);
    }
    f_in.close();
 quit:
    return status;
}
```

左と右とで異なる部分に下線を引いてみました．左右で異なるのは，たったの 3ヶ所だけしかありません．`getstr()` などの API の使い方については全く違いがありません．つまり，SLLIB では，ユーザが標準入出力を使えるようになれば，圧縮ファイルやネットワーク上のファイルも自動的に使えるようになるわけです．

このように，1 つ覚えれば芋蔓式に多くの API が使えるというのが，オブジェクト指向の強みなのです．

1.5.3 言葉の定義とコードの考え方

C++のようなオブジェクト指向の言語では，従来「型」といっていたものを「クラス」と呼び,「変数」といっていたものを「オブジェクト」あるいは「インスタンス」と呼ぶ事になっています．そして，§1.5.2で登場した，`str.append("012");` の「append()」の部分を「メンバ関数」と呼びます．

SLLIB は，オブジェクト指向のライブラリです．したがって，ユーザのみなさんが SLLIB の API を使う場合は，まず処理の主対象であるオブジェクト (インスタンス) を作って，クラス中のメンバ関数 (上の例だと `printf()` や append()) を使ってコードを作っていく事になります．従来の手続きのコードを書く場合は，動詞 (命令) を先に書く事から,「××する，○○に」と考える必要がありましたが，オブジェクト指向の場合はもっと自然に「○○に，××する」と考えて，それをそのままコードに書き下していけばよいのです．

2 インストール

2.1 対応 OS

SLLIB が対応する OS は，Linux，FreeBSD，MacOSX，Solaris，Cygwin です．いずれも，32 ビット版と 64 ビット版の両方をサポートしています．

必要なコンパイラは GCC g++バージョン 3 系列以降，または Intel® C++ Compiler です (筆者は g++ 3.3.2 以降で動作確認を行なっています)．

2.2 SLLIB のビルドと設置

SLLIB のビルドの方法は，二通りあります．次のどちらかを選んでください．

- **方法 1**—make 一発による方法 (§2.2.1)

  一般のプログラマや科学研究者の方は，こちらをお勧めします．

  コンパイラの変更や，ライブラリのパスの変更は可能ですが，基本的にはデフォルトの設定でビルドする事を前提としており，細かな設定には向いていません[3]．

  スタティックライブラリだけをインストールします．共有ライブラリが必要な場合は，方法 2 を使ってください．

- **方法 2**—configure と make による方法 (§2.2.2)

  プロフェッショナルなソフトウェア開発者やハッカーの方，あるいは細かな設定が必要な場合は，こちらをお勧めします．

  configure のオプションで，zlib，bzip2，readline を使わないように設定できます．

  スタティック・共有ライブラリの両方をインストールします．

  サポート外の OS の場合は，こちらの方法で試してみてください．

どちらの方法でも，ビルド・システムが自動的に OS を検出し，コンパイラに適切なオプションを与えてビルドしてくれます．

2.2.1 方法 1—make 一発による方法

ビルドには，zlib，bzlib，ncurses，readline ライブラリが必要ですので，あらかじめインストールしておきます (rpm の名前は，zlib-devel，bzip2-devel，ncurses-devel，readline-devel 等になっている事が多いようです[4])．次に Redhat 系の場合の例を示します．

```
# yum install zlib-devel
# yum install bzip2-devel
# yum install ncurses-devel
# yum install readline-devel
```

アーカイブを展開して，make します．

[3] config.h を編集すると，外部ライブラリの有効/無効を設定できます．
[4] Debian 系の場合，zlib1g-dev，libbz2-dev，ncurses5-dev，libreadline5-dev のようなパッケージ名になっています．

```
$ gzip -dc sllib-x.xx.tar.gz | tar xvf -
$ cd sllib-x.xx
$ make
```

ここで，Intel® C++ Compiler を使いたい場合や，64-bit 用 (または 32-bit 用) のライブラリを作りたい場合は,

```
$ make CXX=icc
```

```
$ make CCFLAGS="-m64"
```

のようにして，コンパイラを変更したり，コンパイラのオプションを追加できます．32-bit OS 上の gcc の場合，SSE2 が有効にならない事があります[5]．その場合は次のように SSE に関係するオプションを与えるとパフォーマンスを改善できます.

```
$ make CCFLAGS="-msse2 -mfpmath=sse"
```

PREFIX の変更が必要な場合も，同様に与える事ができます．例えば，次のようにします.

```
$ make CCFLAGS="-msse2 -mfpmath=sse" PREFIX="/home/guest/local"
```

インストールします.

```
$ su
# make install32
```

この例は 32-bit OS の場合で，64-bit OS の場合は「`make install64`」とします．デフォルトでは「install32」では`/usr/local/lib` に,「install64」では`/usr/local/lib64` に `libsllib.a`がインストールされます．同時に，`/usr/local/include/sli` にヘッダファイル一式がコピーされ，`/usr/local/bin` に C++コンパイラの wrapper スクリプト `s++`がインストールされます.

2.2.2 方法 2—configure と make による方法

すべてのクラスが利用可能な SLLIB をビルドするには，zlib，bzlib，ncurses，readline ライブラリが必要ですので，あらかじめインストールしておきます (rpm の名前は，zlib-devel，bzip2-devel，ncurses-devel，readline-devel 等になっている事が多いようです[6])．次に Redhat 系の場合の例を示します.

```
# yum install zlib-devel
# yum install bzip2-devel
# yum install ncurses-devel
# yum install readline-devel
```

アーカイブを展開して，configure・make します

```
$ gzip -dc sllib-x.xx.tar.gz | tar xvf -
$ cd sllib-x.xx
$ sh configure
$ make
```

`configure` のオプションとして,「`--disable-readline`」「`--disable-bz2lib`」「`--disable-zlib`」

[5] 64-bit OS ではデフォルトで有効になります
[6] Debian 系の場合，zlib1g-dev，libbz2-dev，ncurses5-dev，libreadline5-dev のようなパッケージ名になっています.

が有効です．ただし，これらのオプションを付加した場合，当該ライブラリに依存するクラスは利用できなくなります．

インストールします．

```
$ su
# make install
```

デフォルトでは，/usr/local/lib にライブラリファイルが，/usr/local/include/sli にヘッダファイル一式がコピーされ，/usr/local/bin に C++コンパイラの wrapper スクリプト s++がインストールされます．

64-bit OS の場合，次のようにライブラリのパスを configure で指定する必要があるかもしれません．

```
$ sh configure --libdir='${prefix}/lib64'
```

3 チュートリアル

3.1 Hello World

おなじみ，Hello World です．

```
#include <sli/stdstreamio.h>
using namespace sli;

int main()
{
    stdstreamio sio;                    /* オブジェクトを作る (標準出力用) */
    sio.printf("Hello World\n");        /* 標準出力へ書き出し */
    return 0;
}
```

上記のコードを s++コマンドを使って作りましょう．

```
$ s++ hello.cc
```

この後，テンプレートコードを作成するかどうか聞かれるので「y」と答えると，コードが作成されます．

コンパイルし，実行します．s++を使うと簡単にコンパイルできます．

```
$ s++ hello.cc
g++ -I/usr/local/include -L/usr/local/lib -Wall -O -o hello hello.cc -lsllib -lz
 -lbz2 -lreadline -lcurses
$ ./hello
Hello World
```

このように，s++では，ユーザが出力ファイルを指定しない場合には C++コンパイラの-o オプションを自動的に追加します．

さらに，s++に「/」オプションを指定すると，コンパイル直後にプログラムを実行します．

```
$ s++ hello.cc /
Hello World
```

「/」に後に続けて引数を追加すると，それらがプログラムへ引数として与えられます．

このようにして，s++を使うとスクリプト言語の感覚でユーザ・プログラムが作成できます.

3.2 ファイルのオープンと読み込み

3.2.1 標準ストリームを使った場合

Hello World と同じように stdstreamio クラス (§8.2) を使います．今度の場合は，open() メンバ関数 (§8.1.1) を使ってファイルを読み込み用に開きます.

```
#include <sli/stdstreamio.h>
using namespace sli;

int main()
{
    stdstreamio sio, fin;                 /* オブジェクトを作る */
    char buf[512];
    if ( fin.open("r","foo.txt") < 0 ) {  /* foo.txt を読み込み only で開く */
        エラー処理
    }
    fin.getstr(buf,512);                  /* 最初の 1 行を読み，buf に入れる */
    sio.printf("%s",buf);                 /* 標準出力に出力 */
    fin.close();
    return 0;
}
```

SLLIB では，fopen() のかわりに open() を，fgets() のかわりに getstr() を使います.

さらに，SLLIB では getline() というメンバ関数があります (§8.1.10)．これは改行文字'\n' まで読んでオブジェクト内バッファに格納し，その先頭アドレスを得る関数で，Perl に近い感覚でテキストファイルを処理する事ができます．次の例は，getline() を使ってテキストファイルの内容をすべて表示するコードです (open() のエラー処理は省略).

```
    const char *line;
    fin.open("r","foo.txt");
    while ( (line=fin.getline()) != NULL ) {
        sio.printf("%s",line);
    }
```

なお，getline() で返すバッファ領域は，オブジェクト内部で管理されるのでユーザが勝手に開放してはいけません (free() しようとしてもコンパイルが通りません).

getline() と tstring クラス (§9.5) とを組み合わせると，行単位の文字列の編集や正規表現によるパターンマッチが簡単に行えるので，テキストファイルの解析がたいへん手軽に行なえます．§3.3.3で紹介していますので，そちらもご覧ください.

3.2.2 最強の "万能型" ストリームを使った場合 (超オススメ!!)

次は，ファイルの圧縮解凍・ネットワークやパイプに対応した digeststreamio クラス (§8.8) を使ってみます．**digeststreamio クラスは最強のストリーム入出力用クラスであり，たいへんオススメ度**

が高いです.

```
#include <sli/stdstreamio.h>
#include <sli/digeststreamio.h>
using namespace sli;

int main()
{
    stdstreamio sio;                          /* オブジェクトを作る (標準出力) */
    digeststreamio fin;                       /* オブジェクトを作る (ファイル入力) */
    char buf[512];
    if ( fin.open("r","foo.txt.gz") < 0 ) { /* foo.txt.gz を read only で開く */
        エラー処理
    }
    fin.getstr(buf,512);                      /* 最初の 1 行を読み, buf に入れる */
    sio.printf("%s",buf);                     /* 標準出力に出力 */
    fin.close();
    return 0;
}
```

gzip で圧縮されたファイル `foo.txt.gz` を開き，解凍した内容を `getstr()` で読み込んでいます．解凍が必要かどうかは，ファイルのオープンの時に指定したパス名のサフィックスで自動的に判断してくれます．さらに，ネットワーク経由でのファイルにアクセスも

```
    if ( fin.open("r", "http://www.foo.bar/data/foo.txt.gz") < 0 ) {
```

のようにするだけです．パスの先頭に `http://`があると，Web サーバに接続し，MIME ヘッダの解析結果とパスのサフィックスから，解凍が必要な場合は zlib あるいは bz2lib を利用して読み込みます．

FTP サーバに書き込む例です．ユーザとパスワードを指定する事ができます．省略すると匿名でのアクセスとなります．パスのサフィックスにより，圧縮も行ないます．

```
    fout.open("w", "ftp://username@passwd:ftp.foo.bar/data/foo.txt.gz")
```

`open()` のかわりに `openf()` を使うと，ファイルのパスを printf() 関数の引数と同様に指定できます (他のクラスでも `openf()` が使用できます).

```
   for ( i=0 ; i < N ; i++ ) {
       fin.openf("r", "ftp://foo.ac.jp/file_%d.txt.gz", i);
         :
       fin.close();
   }
```

このように，番号のついたファイルを連続して読む場合や，Web サーバ上の cgi スクリプトの引数を指定する場合に便利です．

stdstreamio クラスの場合と異なり，digeststreamio クラスには `openp()` というメンバ関数があります (§8.8.2)．openp() メンバ関数 () は Perl のような使い方ができる，digeststreamio クラス特有のメンバ関数で，例えば，はじめの例での `fin.open(…)` を

```
    fin.openp("< foo.txt.gz")
```

と書く事ができます．`openp()` は，次のように様々な使い方ができて大変便利です．

パイプで入力する例です.

```
pin.openp("cat /etc/hosts | egrep -v -e '^#' -e '^$' |")
```

パイプで出力する例です.

```
pout.openpf("| %s", pager)
```

3.2.3 libc の関数との対応関係

§3.2.2で登場した openp() や openpf() は digeststreamio クラス特有のメンバ関数ですが, getstr() や getline() などは digeststreamio クラスの場合でも stdstreamio クラスの場合と全く同じ使い方ができるので, 新たに API を学習する必要はありません. しかも, SLLIB のストリーム入出力のメンバ関数は, ほとんどが libc の標準関数に相当するものばかりですから, 抵抗なく使えると思います. 表 4に, libc と SLLIB とのストリーム入出力 API の対応関係を示します.

libc	SLLIB	
fopen(path, mode)	open(mode, path), openf(mode, path_fmt, ...)	§8.1.1
fclose(fp)	close()	§8.1.2
fread(buf, size, nmemb, fp)	read(buf, size)	§8.1.3
fwrite(buf, size, nmemb, fp)	write(buf, size)	§8.1.3
fgetc(fp)	getchr()	§8.1.8
fgets(buf, size, fp)	getstr(buf, size)	§8.1.8
fscanf(fp, format, ...)	scanf(format, ...)	§8.1.11
fputc(ch, fp)	putchr(ch)	§8.1.12
fputs(buf, fp)	putstr(buf)	§8.1.12
fprintf(fp, format, ...)	printf(format, ...)	§8.1.14
fflush(fp)	flush()	§8.1.15

表 4: libc と SLLIB とのストリーム入出力 API の対応関係

libc と比較して, 関数名や若干引数の仕様が異なったりしますが, それは libc の API の不統一性を解消しているためです. 例えば, libc の関数名を調べてみると, 関数名の命名規則が曖昧です. libc には「character」の意味を含む関数として, getchar(), fgetc(), strchr() がありますが,「char」「c」あるいは「chr」のように, 様々な省略形が混在しており, 覚えにくい原因となっていました. こういった部分について, SLLIB では命名規則として

「character」の省略形は「chr」, 「string」の省略形は「str」

のように定め, これに従ってメンバ関数名を決めています. 表 4のメンバ関数のほか, 文字列の API(表 18) 等にもこの規則を適用しているので, 全体として SLLIB のメンバ関数名は覚えやすいはずです.

標準入出力ストリーム (stdstreamio クラス; §8.2) あるいは万能入出力ストリーム (digeststreamio クラス; §8.8) が使えるようになれば, gzip 圧縮入出力ストリーム (gzstreamio クラス; §8.3), bzip2 圧縮入出力ストリーム (bzstreamio クラス; §8.4), http 入力ストリーム (httpstreamio クラス; §8.5), ftp 入出力ストリーム (ftpstreamio クラス; §8.6), パイプ入出力ストリーム (pipestreamio クラス; §8.7), 端末行単位入出力 (termlineio クラス; §8.9) なども全く同じ手法で使う事ができます. これらの入出力ストリームを使いたい場合, #include <sli/...>の部分に使いたいクラス名を指定し, オブジェクトを使いたいクラスで作り, 同じように open() などを使ってファイルなどの読み書きが行えます.

cstreamio クラスと派生クラスについては，§7 に簡単にまとめてあります．そこでは，表 6にどのクラスでどのメンバ関数が利用できるかを示しており，全体像を把握するのに役立つと思います．ぜひご覧ください．

3.2.4 複雑なバイナリデータのエンディアン変換

bread() メンバ関数，bwrite() メンバ関数では，ユーザによって定義されたバイナリデータの構造に従ってエンディアン変換を行ないながらバイナリストリームを読み書きする事ができます．

次の例では，4 バイト整数が 1 個，double 型が 3 個，char 型が 64 個のデータブロックを，2 回読み込んでバッファbuf に保存しています．

```
stdstreamio f_in;
bstream_info binfo[] = { {4,1}, {-8,3}, {1,64}, {0} };
char buf[512];
ssize_t len;
/* ファイルをオープン */
if ( f_in.open("r","binary.dat") < 0 ) {
    エラー処理
}
/* バイナリストリーム (ビッグエンディアン) の読み込み */
len = f_in.bread(buf, binfo, 2, false);
```

bread() の最後の引数で，バイナリデータのエンディアンを指定しています．この例では，ビッグエンディアンを指定しています．

このコードに，データブロックを表わす構造体を定義し，そのポインタ変数に buf のアドレスを代入するコードを追加すれば，各データの要素に簡単にアクセスできます (実際には，binfo 配列はその構造体から作成されるべきでしょう)．

3.2.5 GNUPLOT との連携

計算しながらその結果をリアルタイムで表示したい場合，普通はシェルスクリプトを併用すると思います．

libc の popen() 関数を使うと，C 言語から gnuplot にコマンドを送る事ができました．SLLIB の場合でも，pipestreamio クラスで同じ事ができます．

次のコードは gnuplot とパイプで接続し，gnuplot に描画コマンドを送ってウネウネアニメーションを表示するコードです．

```
#include <sli/pipestreamio.h>
using namespace sli;

int main()
{
    pipestreamio pout;
    int i;

    pout.open("w", "gnuplot");
    pout.printf("set isosamples 48\n");
    for ( i=0 ; i < 1000 ; i++ ) {
        pout.printf("splot sin(%g + sqrt(x*x+y*y))\n", i/10.0);
        pout.flush();
    }
    pout.close();
```

図 1 に実行時の様子を示します．

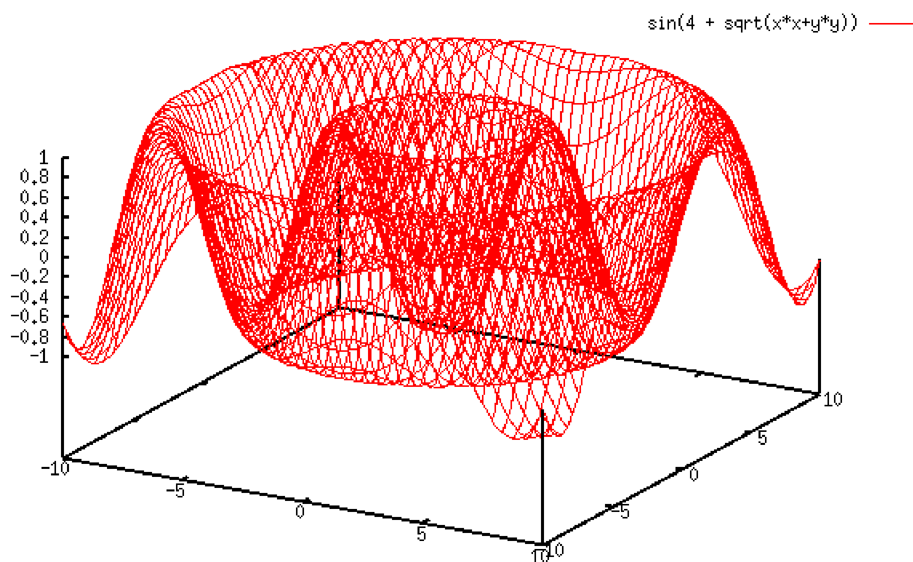


図 1: GNUPLOT でウネウネアニメーション．

3.3 文字列の操作

ここでは，SLLIB を使った文字列の操作について解説します．どのような API が用意されているかを，表 18を眺めてざっと把握しておくと，より読みやすくなると思います．

3.3.1 基本

C での文字列の操作は手間がかかるものです．SLLIB の tstring クラス (§9.5) を使うと，文字列の操作がかなり簡単になります.

まず，最も典型的と思われる使い方を示しましょう.

```
#include <sli/stdstreamio.h>
#include <sli/tstring.h>
using namespace sli;

int main( int argc, char *argv[] )
{
    stdstreamio sio;                              /* 標準出力用のオブジェクト */
    tstring my_str;                               /* 文字列オブジェクトを作る */
    my_str = "Hello World";                       /* "Hello World" を my_str に代入 */
    sio.printf("my_str = '%s'\n", my_str.cstr());        /* 標準出力へ書き出し */
```

このように「=」演算子で，文字列が代入できます．const char *型の時の「=」では文字列定数のアドレスが代入されるのに対し，tstring クラスの「=」ではオブジェクトの内部に自動的にバッファが確保され，そこに文字列全体がコピーされます．内部バッファの先頭アドレスは，cstr() メンバ関数 (§9.5.3) で得る事ができます．もちろん，`cstr()` で得た文字列の終端は`'\0'` で終わるようになっていますから，このようにすぐに `printf()` などに与える事ができます.

そして，やはり printf() は欠かせないでしょう．printf() メンバ関数を使えば，バッファの大きさを気にする事なく，フォーマットされた文字列をオブジェクトに代入できます.

```
    tstring my_str;                               /* オブジェクトを作る */
    my_str.printf("filesize is %d", size);  /* printf() の結果を my_str に代入 */
```

この場合もオブジェクト内部で自動的に必要なバッファが確保されます.

次の例は，`argv[1]` と `argv[2]` とを結合した文字列を作る例です．今度は演算子「=」を使わないで書いてみます.

```
int main( int argc, char *argv[] )
{
       :
       :
    tstring my_str;                                 /* オブジェクトを作る */
    my_str.assign(argv[1]);                         /* argv[1] を my_str に代入 */
    my_str.append(argv[2]);                         /* argv[2] を my_str に追加 */
    sio.printf("my_str = '%s'\n",my_str.cstr());      /* 標準出力へ書き出し */
    return 0;
}
```

append() のような文字列の編集処理においても，オブジェクト `my_str` の内部バッファは自動的に調整されます (オブジェクトが消滅すると，内部バッファの領域も自動的に開放されます)．この例の場合，`argv[1]` と `argv[2]` とを結合した文字列が `my_str` の内部バッファに入ります.

3.3.2 1文字ずつのアクセス

オブジェクト内部のバッファを1文字ずつ読みたい場合には次のように書きます.

```
    for ( i=0 ; i < my_str.length() ; i++ ) {
        sio.printf("[%c]",my_str.cchr(i));
    }
```

`length()` が文字列の長さを返し (§9.5.1), `cchr(i)` が i 文字目のキャラクタコード (int 型) を返します (§9.5.5).

この内部バッファを1文字編集するには,

```
    my_str.at(i) = 'a';
```

あるいは,

```
    my_str[i] = 'a';
```

と書きます (§9.5.6, §9.4.1). この場合, 内部バッファの i 文字目を'a' に変更します. i は任意の値をとる事ができ, 内部バッファの領域が足りない場合は自動的に再確保します. put() メンバ関数 (§9.5.16) でも同じ事ができます.

3.3.3 ストリームからテキストファイルを読む場合での活用

tstring クラスの演算子「=」(§9.4.2) は const char *の入力に対しては const char *型を返すので, ストリームからテキストファイルを扱うコードを短かく書く事ができます.

次のコードは SLLIB でテキストファイルを扱う場合の典型例です.

```
#include <sli/digeststreamio.h>
#include <sli/tstring.h>
    :
    :
    digeststreamio fin;                         /* ストリーム入力用オブジェクト */
    tstring line;                               /* ラインバッファとして利用 */
    if ( fin.open("r", "foo.txt.gz") < 0 ) {    /* read only で開く */
        エラー処理
    }
    while ( (line=fin.getline()) != NULL ) {
        line.trim("\n");                        /* 改行文字を消去 */
        /* 処理いろいろ */
        sio.printf("%s\n", line.cstr());        /* 表示 */
    }
```

この例では, 改行文字の消去に trim() メンバ関数 (§9.5.26) を使っていますが, chomp() メンバ関数 (§9.5.25) を使う方法もあります.

3.3.4 文字列の編集

文字列を編集するためのメンバ関数としては, `append()`, `insert()`, `replace()`, `erase()`, `crop()`, `chomp()`, `trim()`, `toupper()`, `tolower()` などがあり, 追加, 挿入, 置き換えなどが簡単に行なえます (§9.5.17〜).

メンバ関数の中には，関数名の最後の文字が「f」のものが多く存在します．例としては insertf() や replacef() がありますが，これらは printf() と同じように引数を与えて使う事ができます．

```
    my_str.insertf(0, "ID:%d ", id);   /* 位置 0 にフォーマットされた文字列を挿入 */
```

§3.3.3でも登場しましたが，使う場面が多い関数の 1 つに trim() が挙げられるでしょう．

```
    my_str = "  Hello World  \n";
    my_str.trim(" \n");                /* 左右の空白を削除する */
```

このようにすると，オブジェクト my_str には"Hello World" が入ります (中央の空白は残ります)．

§3.3.1で登場した「=」演算子の他，演算子「+=」も利用可能です．「+=」は append() と同じ機能を持つメンバ関数で，2 つの文字列を結合したい場合，

```
    my_str = argv[1];
    my_str += argv[2];
```

と書く事もできます．

3.3.5 文字列の活用

文字列を他の形へ変換するためのメンバ関数としては，atof()，atoi()，scanf() などがあります (§9.5.35〜)．例えば，

```
    double my_value;
    my_value = my_str.atof();
```

とすると，my_str の内部バッファの文字列を実数値に変換できます．

文字列の比較や検索のためのメンバ関数としては，strcmp()，strcasecmp()，strchr()，strstr()，strpbrk()，strspn()，regmatch() などがあります (§9.5.41〜)．

例えば，文字列"sllib"に一致するかどうかを調べる場合，

```
    if ( my_str.strcmp("sllib") == 0 ) {
```

あるいは，

```
    if ( my_str == "sllib" ) {
```

と書けます．大文字と小文字を区別せずに"sllib"と比較する場合，

```
    if ( my_str.strcasecmp("sllib") == 0 ) {
```

のように書けます．

tstring クラスには，豊富なメンバ関数があります．libc の文字列関連の API と，C++標準ライブラリの string クラスのメンバ関数に相当する機能のほとんどが利用できます．詳細はリファレンスを御覧ください．

3.3.6 拡張正規表現の活用 (後方参照も OK)

POSIX 拡張正規表現を使って，文字列の検索，置換が簡単に行なえます (regmatch() メンバ関数; §9.5.59，regreplace() メンバ関数; §9.5.30)．

まず，最も簡単な例です．正規表現を使って，数字が先頭にあるかどうかを調べてみます．

```
stdstreamio sio;
tstring my_str = "123abc";
if ( 0 <= my_str.regmatch("^[0-9]", NULL) ) {
   sio.printf("マッチしました\n");
}
```

regmatch() はマッチした場合，マッチした文字列の位置を返します．

もちろん，後方参照の情報を引き出す事もできます．その場合は，tarray_tstring クラスの regassign() メンバ関数を使います．§3.4.8 に解説がありますので，ご覧ください．

今度は，sed コマンドと同じ感覚で，正規表現を使って置換してみます．

```
stdstreamio sio;
tstring my_url = "http://darts.isas.jaxa.jp/foo/";

my_url.regreplace("([a-z]+://)([^/]+)(.*)", "\\2");
sio.printf("hostname = %s\n", my_url.cstr());
```

後方参照を使って，URL からホスト名「darts.isas.jaxa.jp」を取り出しています．

regreplace() は，第三引数に true をセットすると，1 回目の置換だけでなく，文字列全体について置換を行なうようになります (sed の's/../../g' と同じ)．

3.4 文字列配列の操作

次のように，通常の型と同じように tstring クラスの配列を作る事もできます．

```
tstring my_arr[32];
my_arr[0] = "abc";
```

しかしこの使い方では，配列の編集に自由度がありません．

tarray_tstring クラス (§10) を使うと，文字列配列をもっと柔軟に扱う事ができます．配列のサイズ変更を伴う様々な操作や，すべての文字列要素に対しての文字列編集 (chomp(), trim() 等) や検索 (find(), regmatch() 等) が手短かに行なえます．もちろん，tstring クラスと同様，内部バッファの大きさは自動的に調整されるので，非常にお手軽です．C 言語的なコードとの親和性を考慮して，ポインタ配列からの取り込み，ポインタ配列への変換が簡単に行なえるようになっています．

どのような API が用意されているかを，表 22を眺めてざっと把握しておくと，以降の解説がより読みやすくなると思います．

3.4.1 いきなり代入

メモリ管理は完全自動ですから「最初に 10 個確保して…」のようなコードは必要ありません．オブジェクトを作ったら，即，代入できます．

最も使用頻度が高い代入方法は，次の例のような NULL 終端のポインタ配列の代入と，定数文字列の代入でしょう．

```
#include <sli/stdstreamio.h>
#include <sli/tarray_tstring.h>
using namespace sli;

int main()
{
    stdstreamio sio;
    const char *group1[] = {"sakura", "mizuho", NULL};
    tarray_tstring my_arr = group1;
    size_t i;
    my_arr[2] = "fuji";
    my_arr[3] = "hayabusa";
    for ( i=0 ; i < my_arr.length() ; i++ ) {                  /* 表示 */
        sio.printf("%zu: [%s]\n", i, my_arr[i].cstr());
    }
```

実行結果は次のとおりです.

```
0: [sakura]
1: [mizuho]
2: [fuji]
3: [hayabusa]
```

次のように，オブジェクトを作る時に値を代入する事もできます (最後の NULL を忘れずに).

```
    tarray_tstring my_arr("sakura", "mizuho", NULL);
```

オブジェクトの初期化の方法については，§10.1をご覧ください.

3.4.2 デバッグには dprint()!!

fprintf(stderr, ...); をコードに挟んでデバッグ，というのは日常的に良く行なう事だと思います．しかし配列の場合は全要素を表示するのに必ず for 文などでまわす必要があり，やや面倒です.

こんな時は，dprint() メンバ関数 (§10.4.5) を使うだけで，配列の一覧を標準エラー出力に出力する事ができます．例えば,

```
    tarray_tstring my_arr("nEC", "Fujitsu", "Toshiba", NULL);
    my_arr.dprint();
```

とすると，次のように表示されます.

```
sli::tarray_tstring[obj=0x7fbffff230] = {"nEC", "Fujitsu", "Toshiba"}
```

オブジェクトのアドレスが表示されますが，もちろんこれは環境に依存する値です.

dprint() メンバ関数は，連想配列を扱うクラス asarray_tstring クラス (§11)，mdarray_*クラス (§12) でも使えます (§11.4.10，§12.5.15).

3.4.3 速攻で execv()，execvp() に渡す

cstrarray() メンバ関数 (§10.4.2) を使うと，オブジェクト内部の文字列バッファに対する NULL 終端のポインタ配列を得る事ができるので，libc の `execvp()` などの char *[] 型の引数にそのまま与える事ができます．

```
    tarray_tstring cmd;
      :
    execvp(cmd[0].cstr(), (char *const *)cmd.cstrarray());
```

3.4.4 全要素に対する文字列編集

例えば，次のように `split()` を使って，csv 形式の文字列を分割して配列に格納したとします (`split()` については，この後の §3.4.7で紹介しています)．

```
    const char *line = " Z-80,,  8086 , 6800";
    tarray_tstring my_arr;
    my_arr.split(line, ",", true);
    my_arr.dprint();
```

これを実行すると，次のように表示されます．

```
sli::tarray_tstring[obj=0x7fbffff4d0] = {" Z-80", "", "  8086 ", " 6800"}
```

このような場合，分割後の文字列の左右の空白文字が余計ですね．すべての要素について，この余計な空白文字を消したい場合，

```
    my_arr.trim();
```

と書くだけです．

このように，全要素に対して一気に文字列操作を行なえるメンバ関数は，`trim()` のほか，`chomp()`，`strreplace()`，`regreplace()`，`tolower()`，`toupper()` などがあります (一覧は表 22を参照)．詳細は，リファレンス部の §10.4.27 以降をご覧ください．

3.4.5 配列の編集

§3.4.6 の例では，`erase()` が登場していますが，この他にも `append()`，`insert()`，`replace()` などのメンバ関数があり，tstring クラスの場合と同様，簡単に配列を編集する事ができます (§10.4.16～)．

次の例は，配列に対して要素を挿入・追加するものです．

```
    tarray_tstring arr;
    arr[0] = "SUICA";
    arr.insert(0, "ICOCA", 1);                              /* 挿入 */
    const char *others[] = {"PASPY", "PASMO", NULL};
    arr.append(others);                                     /* 追加 */
```

結果として，4 つの文字列が`"ICOCA"`，`"SUICA"`，`"PASPY"`，`"PASMO"` の順に入った arr オブジェクトができます．

3.4.6 main() の引数を使い易く

main() の引数といえば libc の getopt() 関数ですが，getopt() を使うまでもないような場合に，tarray_tstring クラスは便利かもしれません．

§3.4.1でも示したように，tarray_tstring クラスのオブジェクトへは，char *[] 型の NULL 終端のポインタ配列の全文字列要素を「=」記号一発でコピーする事ができます．コピーした後は，提供されているメンバ関数によって，編集や活用が自在に行なえます．

```
#include <sli/stdstreamio.h>
#include <sli/tarray_tstring.h>
using namespace sli;

int main( int argc, char *argv[] )
{
    size_t i;
    double x = 0, y = 0;
    stdstreamio sio;
    tarray_tstring args = argv;                  /* args で引数を使えるようにする */

    args.erase(0,1);                             /* args[0] を削除 */
    for ( i=0 ; i < args.length() ; i++ ) {      /* 引数を解析・編集 */
        if ( args[i] == "-x" ) {
            x = args[i+1].atof();
            args.erase(i,2);
        }
        else if ( args[i] == "-y" ) {
            y = args[i+1].atof();
            args.erase(i,2);
        }
    }
    for ( i=0 ; i < args.length() ; i++ ) {      /* 解析できなかった引数を表示 */
        sio.printf("%s\n",args[i].cstr());
    }
```

この例に登場する==演算子や.atof()，.cstr() は tstring クラスのメンバ関数です．つまり，args[i] の後には，すべての tstring クラスのメンバ関数が使えるという事です．正規表現を使った処理も楽勝です．

3.4.7 空白区切りや CSV 形式の文字列を分割して配列に入れる—split() メンバ関数

split() メンバ関数により，空白区切りや CSV 形式の 1 行文字列を各要素に分割し，配列として扱えるようになります (§10.4.12)．split() メンバ関数はクォーテーションに囲まれた部分の特殊処理をサポートするなど非常に強力で，文字列を分割する時の動作をユーザの用途によって切り替える事ができます．次のような選択肢があります．

- CSV 形式のように文字列長ゼロの要素を許すかどうか

(許す例: `"abc,,xyz"` → `"abc", "", "xyz"`) (許さない例: `"abc  xyz"` → `"abc", "xyz"`)

- クォーテーションや括弧に囲まれた領域の分割禁止の指定 (デフォルトでは特別扱いしない)

  (例: `"abc[ ] 'x y z'"` → `"abc[ ]", "'x y z'"`)

- エスケープ文字の指定 (例:「\」; デフォルトでは指定なし)

  (例: `"winnt program\\ files"` → `"winnt", "program\\ files"`)

まず，基本形です．この場合は，各種スクリプト言語の split と同じ動作です．

```
const char *line = "abc def  wxyz ";
tarray_tstring my_arr;
my_arr.split(line, " ", false);              /* "abc","def","wxyz" に分割 */
for ( i=0 ; i < my_arr.length() ; i++ ) {   /* 分割した文字列を表示 */
    sio.printf("%s\n",my_arr.cstr(i));
}
```

次は，CSV 形式の場合です．3 番目の引数を `true` にする事で，文字列長ゼロの要素を許可しています．

```
const char *line = "JAN,,MAR,";
tarray_tstring my_arr;
my_arr.split(line, ",", true);               /* "JAN","","MAR","" に分割 */
```

クォーテーションと丸括弧で囲まれた部分は分割しない場合を示します．

```
const char *line = "winnt( ) program\\ files 'mary\\'s music'";
tarray_tstring my_arr;
/* "winnt( )", "program\\ files", "'mary\\'s music'" に分割 */
my_arr.split(line, " ", false, "'()", '\\', false);
```

このように，SLLIB の split() は非常に強力です．

3.4.8 正規表現マッチの結果を格納

ある文字列に対して正規表現マッチした結果を，後方参照の情報を含めて配列に保存する事ができます (regassign() メンバ関数; §10.4.13)．

次の例は，文字列 `"OS = linux"` について，キーワードと値とを取り出しています．

```
stdstreamio sio;
tarray_tstring my_elms;
tstring my_str = "OS = linux";

my_elms.regassign(my_str, "([^ ]+)([ ]*=[ ]*)([^ ]+)");
if ( my_elms.length() == 4 ) {
    sio.printf("keyword=[%s] value=[%s]\n",
               my_elms.cstr(1), my_elms.cstr(3));
}
```

実行すると,「`keyword=[OS] value=[linux]`」と表示されます．`my_elms.cstr(0)` には，`my_pat`

全体についてマッチした部分の文字列が入り，`my_elms.cstr(1)` 以降に，後方参照の部分文字列が入ります.

3.5 連想配列の操作

asarray_tstring クラス (§11) が文字列の連想配列を扱うためのクラスで，tarray_tstring クラス ((§11) と同様に非常に手軽に使う事ができます.

どのような API が用意されているかを，表 23を眺めてざっと把握しておくと，以降の解説がより読みやすくなると思います.

3.5.1 いきなり代入

連想配列の場合もいきなり代入できます (演算子「`[]`」; §11.3.1).

```
#include <sli/asarray_tstring.h>
using namespace sli;
      :
    asarray_tstring my_arr;
    my_arr["JR-EAST"] = "SUICA";
    my_arr["JR-CENTRAL"] = "TOICA";
    my_arr["JR-WEST"] = "ICOCA";
    /* 表示 */
    for ( i=0 ; i < my_arr.length() ; i++ ) {
        const char *key = my_arr.key(i);
        sio.printf("%s: [%s]\n", key, my_arr[key].cstr());
    }
```

実行結果は次のとおりです.

```
JR-EAST: [SUICA]
JR-CENTRAL: [TOICA]
JR-WEST: [ICOCA]
```

3.5.2 デバッグには dprint()!!

tarray_tstring クラスと同様，asarray_tstring クラスでも `dprint()` が使えます.

dprint() メンバ関数 (§11.4.10) を使うだけで，連想配列の一覧を標準エラー出力に出力する事ができます．例えば,

```
    asarray_tstring my_arr("PKG","IDL", "VENDOR","ITT", NULL);
    my_arr.dprint();
```

とすると，次のように表示されます.

```
sli::asarray_tstring[obj=0x7fbffff1f0] = { {"PKG", "IDL"}, {"VENDOR", "ITT"} }
```

オブジェクトのアドレスが表示されますが，もちろんこれは環境に依存する値です.

3.5.3 全要素に対する文字列編集

連想配列の場合も，全要素に対する文字列編集のためのメンバ関数が用意されています．例えば，次のように書くだけで要素値をすべて小文字にする事ができます．

```
    asarray_tstring my_arr("OS","SOLARIS", "VENDOR","Sun", NULL);
    my_arr.tolower();
    my_arr.dprint();
```

次のように表示されます．

```
sli::asarray_tstring[obj=0x7fbffff1f0] = { {"OS", "solaris"}, {"VENDOR", "sun"} }
```

このように，全要素に対して一気に文字列操作を行なえるメンバ関数は，trim()，chomp()，strreplace()，regreplace()，toupper() などがあります (一覧は表 23を参照)．詳細は，リファレンス部の §11.4.26 以降をご覧ください．

3.5.4 編集

連想配列の場合は要素の順番が問題になる事は少ないと思いますが，asarray_tstring クラスでは文字列配列にインデックスを付けるという実装になっているので，要素の順番をユーザが操作する事ができます．tarray_tstring クラスと同様に append()，insert()，erase() といったメンバ関数が利用できます (§11.4.19〜)．

例えば，キー "JR-EAST" の前に 1 つのキーと値のセットを挿入する場合は，

```
    my_arr.insert("JR-EAST", "JR-HOKKAIDO","KITACA");
```

のように書けます．

3.5.5 split_keys()，split_values() でデータファイルに簡単アクセス

例えば，こんなデータファイル (ファイル名は data.txt.gz) があったとします．

```
NAME       FROM TO       DISTANCE LOCOMOTIVE     CARRIAGE
Hokuriku   Ueno Kanazawa 517.4    EF64,EF81      14-series
Akebono    Ueno Aomori   772.8    EF64,EF81      24-series
Cassiopeia Ueno Sapporo  1214.9   EF81,ED79,DD51 E26-series
Hokutosei  Ueno Sapporo  1214.9   EF81,ED79,DD51 24-series
```

data.txt.gz の内容を連想配列に取り込んでみます．この時，連想配列のキー，値それぞれにスプリットが使えます (split_keys() ; §11.4.17，split_values() ; §11.4.18)．

```
#include <sli/stdstreamio.h>
#include <sli/digeststreamio.h>
#include <sli/asarry_tstring.h>
    :
    :
    stdstreamio sio;
    digeststreamio fin;                        /* ストリーム入力用オブジェクト */
    tstring line_buf;                          /* ラインバッファ */
    asarray_tstring vals;
    if ( fin.open("r", "data.txt.gz") < 0 ) {   /* read only で開く */
        エラー処理
    }
    if ( (line_buf=fin.getline()) == NULL ) {   /* カラム名を読む */
        エラー処理
    }
    vals.split_keys(line_buf.cstr(), " \n", false);          /* key をゲット */
    while ( (line_buf=fin.getline()) != NULL ) {
        vals.split_values(line_buf.cstr(), " \n", false);     /* 値をゲット */
        sio.printf("%s: %skm\n", vals["NAME"].cstr(), vals["DISTANCE"].cstr());
    }
```

テキストのデータファイル data.txt.gz を開き，データファイルの 1 行目にあるカラム名を split_keys() メンバ関数で連想配列のキーに割り当てています．2 行目以降のデータ値を split_values() メンバ関数を使って連想配列の値としています．すべての値のうち，NAME と DISTANCE のカラムを標準出力に出力しています．

実行結果は次のとおりです．

```
Hokuriku 517.4km
Akebono 772.8km
Cassiopeia 1214.9km
Hokutosei 1214.9km
```

split_keys()，split_values() は tarray_tstring クラスの場合と同様，ダブルクォーテーションやエスケープ文字の処理に対応しています．詳細はリファレンスをご覧ください．

3.6 多次元配列を楽に扱う

mdarray クラスの継承クラス (§12) を使えば，C 言語の型の一次元配列，多次元配列を手軽に扱う事ができます．IDL/Python 風の表現による要素の範囲指定，配列の全要素に対して演算を行なうための数学関数 (log10() や sin()，cos() など; 表 25) と演算子 (+，-，*，/; §12.4)，統計用関数などが利用可能です．もちろん，ユーザのコードでメモリ確保を行う必要はありません．

mdarray クラスは 2 つの動作モードを持ち，リサイズを自動的に行う「自動リサイズモード」，ユーザの指示によってリサイズを行う「非自動リサイズモード」をユーザは選択する事ができます．この動作モードは，オブジェクトを作る時か init() メンバ関数で初期化する時に選択できます (何も指定しない場合は「自動リサイズモード」になります)．

3.6.1 いきなり代入 (自動リサイズモード：一次元配列〜三次元配列)

mdarray の場合もいきなり代入できます.

```
#include <sli/mdarray.h>
using namespace sli;
      :
    mdarray_double arr;                  /* 8 バイト double で使う */
    arr[0] = 1.57079632679489661923;
    arr[1] = 3.14159265358979323846;
    /* 表示 */
    for ( i=0 ; i < arr.length() ; i++ ) {
        sio.printf("[%f]\n", arr[i]);
    }
```

この例は, mdarray_double クラスを使っていますが, このほか mdarray_float クラス, mdarray_uchar クラス (unsinged char 型), mdarray_short クラス, mdarray_int クラス, mdarray_long クラス, mdarray_llong クラス (long long 型), mdarray_ssize クラス (ssize_t 型), stdint.h で定義されたいくつかの型に対応するクラスがあります[7]

例えば, long 型の配列を扱う場合は,

```
mdarray_long larr;
```

とします.

自動リサイズモードの場合, 二次元配列や三次元配列として使いたい場合も,

```
arr(0,1) = 0.31830988618379067154;
arr(0,0,1) = 1.41421356237309504880;
```

のように書くだけで自動的に二次元配列, 三次元配列へと拡張されます. 二次元, 三次元の場合は,「[]」ではなく「()」を使います (§12.4.1, §12.4.2). それぞれの次元数を n_0, n_1, n_2 とすると, $n_0 \times n_1 \times n_2$ 個分の内部バッファが確保されます. この時, 未定義要素の値はデフォルトでは 0 が代入されますが, そのデフォルト値をユーザが指定する事も可能です.

もし, オブジェクトに登録されている型にかかわらず常に double で読み出ししたい場合は, 次のように dvalue() メンバ関数 (§12.5.10) と assign() メンバ関数 (§12.5.23) を使う事ができます.

```
    double value;
      :
    value = arr.dvalue(x0,y0,z0);
    arr.assign(value, x1,y1,z1);
```

整数型で初期化されたオブジェクトに対する代入や演算はデフォルトでは「小数点以下切捨て」ですが,「四捨五入」とする事もできます. どちらを採用するかは set_rounding() メンバ関数で設定する事ができます.

3.6.2 次元数と配列長の変更

次元数と配列長とを同時に設定可能な `resize_1d()`, `resize_2d()`, `resize_3d()` が, mdarray クラスでのリサイズ変更で最も基本的な方法です (§??〜).

[7] short 型, int 型, long 型, long long 型のバイトサイズは処理系依存です. 1 要素のバイトサイズに厳密性が必要な場合は, mdarray_int16 クラス, mdarray_int32 クラス, mdarray_int64 クラスを使いましょう.

次のように使います.

```
mdarray_double arr;
arr.resize_2d(1024, 768);          /* 要素数 1024x768 の 2D 配列とする */
```

3.6.3 各次元ごとの配列長の変更

各次元ごとのサイズ変更処理のために，resize()，resizeby()，insert()，crop()，erase() などのメンバ関数が用意されています (§12.5.28〜).

例えば，resize() メンバ関数 (§12.5.31) を使うと，次のように次元ごとにリサイズできます.

```
arr.resize(0, 1024);            /* 1 次元目 */
arr.resize(1, 768);             /* 2 次元目 */
```

また,

```
arr.insert(1, 128, 256);
```

とする事で，2 次元目 (次元のインデックスも 0 から始まる) の位置 128 に 256 個の要素を挿入する事ができます．挿入された部分は初期値で初期化されます.

3.6.4 配列に対する演算

演算子「+」「-」「*」「/」「+=」「-=」「*=」「/=」を，mdarray クラスのオブジェクトあるいはスカラー値に対して使う事ができます (§12.4.3〜)．libc の math.h で定義される数学関数のほとんどが，mdarray クラスのオブジェクトあるいはスカラー値に対して使う事ができます (表 25)．例えば，arr=pow(arr,2.0); のような使い方も可能です.

演算に使う複数個のオブジェクトの型が一致している必要はありません．つまり，以下のように演算ができるわけです.

```
mdarray_long larr;
mdarray_double darr;
  :
darr *= 10.0;
darr = log10(darr + larr);
```

この場合，long 型を扱う larr オブジェクト，double 型を扱う darr オブジェクトを作り，darr オブジェクトの全要素を 10.0 倍し，最後に darr に larr を加算した全要素に対して log をとってその結果を darr に代入しています.

この例の最後のように，型が異なる 2 つのオブジェクトの演算結果の型は，通常のスカラー値の場合と同じ型になります.

3.6.5 非自動リサイズモード (画像バッファ向き)

画像バッファのように，自動リサイズが適さないアプリケーションも存在するでしょう．その場合は，初期化時に，第一引数に false を指定します (同時に各次元の初期サイズを指定できます).

次の場合，1 要素が unsigned char(8-bit) の 1920×1080 の画像を 3 枚持たせる場合です.

```
mdarray_uchar arr(false);
arr.resize_3d(1920,1080,3);
```

この場合も，

```
arr(x,y,z) = value;
```

のように各要素にアクセスするわけですが，x，y，z に範囲外の値を指定してもエラーにはなりません．範囲外の場合，値を書き込んだ場合は単に捨てられ，読み込んだ場合は INDEF_UCHAR が返ります．

途中で次元数を変更したい場合は，increase_dim() メンバ関数 (§12.5.33)，decrease_dim() メンバ関数 (§12.5.34) を，バッファサイズを変更したい場合は，resize_1d()，resize_2d()，resize_3d() の他，resize() メンバ関数 (§12.5.31)，resizeby() メンバ関数 (§12.5.32) などを利用します．

なお，自動リサイズ/非自動リサイズの両方を混在させる事は推奨しません．というのは,「=」演算子で配列から配列へ代入された場合には，オブジェクトの属性もコピーされるからです．

3.6.6 高速な一要素ずつのアクセス

§3.6.1や§3.6.5 で紹介した「arr(x,y,z) = ...;」による一要素ずつのアクセスは安全ですが，バッファ境界のチェックが入るため高速ではありません[8]．一要素ずつのアクセスで最高の速度が必要な場合は，オブジェクト内部で自動生成される 2D または 3D 配列用のポインタ配列を利用します．コードは簡単で，次のように array_ptr_2d() または array_ptr_3d() を使います (§??，§??).

```
mdarray_float arr0(false);
arr0.resize_2d(8,4);
float *const *arr0_ptr = arr0.array_ptr_2d(true);
size_t i, j;
for ( i=0 ; i < arr0.row_length() ; i++ ) {       /* Y */
    for ( j=0 ; j < arr0.col_length() ; j++ ) {   /* X */
        arr0_ptr[i][j] = 100 + 10*i + j;
    }
}
arr0.dprint();
```

上記のコードの結果は次のとおりです．

```
sli::mdarray[obj=0xbffff4f0, sz_type=-4, dim=(8,4)] = {
 { 100, 101, 102, 103, 104, 105, 106, 107 },
 { 110, 111, 112, 113, 114, 115, 116, 117 },
 { 120, 121, 122, 123, 124, 125, 126, 127 },
 { 130, 131, 132, 133, 134, 135, 136, 137 }
}
```

array_ptr_2d()，array_ptr_3d() の引数は，ポインタ配列の生成のためのスイッチで，false を指定するとそれまでのポインタ配列を破棄し，内部での生成を停止させる事ができます．

なお，mdarray クラスでは次元数にかかわらず，**内部バッファは常に一次元**ですので，array_ptr() (§12.5.16) を使って内部バッファの先頭アドレスを取得し，プログラマ側でアドレスを計算して要素にアクセスする事もできます．

[8] sin() などの数学関数ほどは遅くありません

3.6.7 IDL/Python 風の表現による画像の領域選択や演算

IDL/Python 風の表現を使って，画像の一部をコピーバッファにコピーし，そのコピーバッファの画像を任意の位置に貼り付ける事が簡単に行なえます．また，加減乗除の画像演算ならメンバ関数を使うだけでできますし，統計用関数と連携する事で複雑な画像演算も可能です (§??〜)．ここでは，§3.6.6 のコードに続く形で，使用例を紹介します．

§3.6.6 の float の配列 `arr0` の一部分を double の配列 `arr1` にコピーします (型も変換します)．

```
    mdarray_double arr1(false);
    arr1 = arr0.sectionf("4:7, *");
    arr1.dprint();
```

上記のコードの結果は次のとおりです．

```
sli::mdarray[obj=0xbffff2d0, sz_type=-8, dim=(4,4)] = {
 { 104, 105, 106, 107 },
 { 114, 115, 116, 117 },
 { 124, 125, 126, 127 },
 { 134, 135, 136, 137 }
}
```

「`arr1 = arr0.sectionf(...);`」の「`=`」では，型が同一の場合は shallow copy が使われます．なお，引数が `"4:7,*"` または `"(4:7,*)"` の場合，**0-indexed** で解釈されます．ただし，`"[…]"` と表記した場合は 1-indexed で解釈され，今の場合は `"[5:8,*]"` とも書く事ができます．

今度は，配列 `arr0` の下半分に配列 `arr1` を貼り付けてみましょう．

```
    arr0.pastef(arr1, "*, 2:3");
    arr0.dprint();
```

結果は次のとおりです．

```
sli::mdarray[obj=0xbffff4f0, sz_type=-4, dim=(8,4)] = {
 { 100, 101, 102, 103, 104, 105, 106, 107 },
 { 110, 111, 112, 113, 114, 115, 116, 117 },
 { 104, 105, 106, 107, 124, 125, 126, 127 },
 { 114, 115, 116, 117, 134, 135, 136, 137 }
}
```

最後の `pastef()` にかわりに `addf()`，`subtractf()`，`multiplyf()`，`dividef()` を使うと加減乗除の画像演算ができます．では続いて，同じ部分を配列 `arr1` で割ってみましょう．

```
    arr0.dividef(arr1, "*, 2:3");
    arr0.dprint();
```

結果は次のとおりです．

```
sli::mdarray[obj=0xbffff4f0, sz_type=-4, dim=(8,4)] = {
 { 100, 101, 102, 103, 104, 105, 106, 107 },
 { 110, 111, 112, 113, 114, 115, 116, 117 },
 { 1, 1, 1, 1, 124, 125, 126, 127 },
 { 1, 1, 1, 1, 134, 135, 136, 137 }
}
```

sectionf() などの「f」が最後につくメンバ関数は，配列の範囲指定のため文字列を printf() 関数の引数と同様に指定できます．変数で範囲を指定する場合は，次のように書く事ができます．

```
    arr1 = arr0.sectionf("%d:%d, %d:%d",
                         x, x + width - 1, y, y + height - 1);
```

3.6.8 配列に対する統計量の計算

SLLIB では，mdarray クラスのオブジェクトに対するモーメントや median などの統計量を求めるための関数が提供されています．SLLIB のヘッダファイル設置場所 (例: /usr/local/include/) の sli/mdarray_statistics.h にそれらのための関数が，inline で (コンパイルされていない状態で) 記述されていますので，内容の確認やコードの流用も容易です．

では，§3.6.7のコードの最後の arr0 の配列の内容について，統計量「平均」「分散」「歪度」「尖度」を計算してみましょう．

```
    /* get mean, variance, skewness, kurtosis */
    mdarray_double moment = md_moment(arr0, false, NULL, NULL);
    moment.dprint();
```

結果は次のとおりです．

```
sli::mdarray[obj=0xbfffee90, sz_type=-8, dim=(4)] = {
 87.125, 2657.919355, -0.967679358, -0.8964442873
}
```

「平均」や「median」専用の関数も提供されており，それらには x 方向，y 方向，z 方向に統計値を求めるためのものがそれぞれ用意されています．例えば，y 方向で median[9] 得る場合，次のようにコードを書きます．

```
    arr1 = md_median_y(arr0);
    arr1.dprint();
```

結果は次のとおりです．

```
sli::mdarray[obj=0xbffff2d0, sz_type=-8, dim=(8,1)] = {
 { 50.5, 51, 51.5, 52, 119, 120, 121, 122 }
}
```

3.6.9 画像データのコンバイン

複数の画像データを平均値や median でコンバインする場合は，3 次元アレイを作ってから z 方向に統計値を求めると良いでしょう．この場合，次のようなコードを書きます．

[9] SLLIB では IRAF の midpt(median の近似値) ではなく，本物の median を高速に算出します．

```
/* 3次元アレイを用意(画像10枚分) */
arr0.resize_3d(1024, 512, 10);

/* 1枚ずつ z の位置を変えて 3次元アレイにペースト */
mdarray_float arr_tmp;
arr_tmp.resize_2d(1024, 512);
for ( i=0 ; i < 10 ; i++ ) {
       :
    /* arr_tmp に代入する処理 */
       :
    arr0.pastef(arr_tmp, "*,*,%d", i);
}

/* z 方向に median コンバイン */
arr1 = md_median_small_z(arr0);
```

コンバインする画像が多い場合は，`md_median_small_z()` のかわりに，`md_median_z()` を使う事ができます．前者は zx 面単位で一時バッファを使うので高速ですが，メモリを多めに使用します．後者は z 方向の 1 本単位で一時バッファを使うため省メモリですが，z 方向が小さい場合には内部での関数呼び出しのオーバヘッドのために良好なパフォーマンスが得られない事があります．

3.6.10 エンディアンの変換

cstreamio クラスの bread() メンバ関数，bwrite() メンバ関数でもエンディアンの変換はできますが，それらがうまく適用できない場合には mdarray の reverse_endian() メンバ関数 (§12.5.21) を使ってエンディアンの処理を行なう事ができます．

```
mdarray_short arr1;
  :
  :
arr1.reverse_endian(false);
```

`reverse_endian()` の 1 つめの引数で，ファイルに保存すべきデータがリトルエンディアンなら `true` を，そうでないなら `false` を指定します．例えば，天文学で利用される FITS ファイルの場合はビッグエンディアンですので，`false` を指定します．例では省略されていますが，追加の引数を指定すると配列の一部分のみエンディアンを変換できます．詳細はリファレンスをご覧ください．

4 SLLIB を使う前に知っておきたい事

SLLIB は C++のライブラリですので，ユーザのみなさんも C++コンパイラを使う事になります．C++は基本的には C の上位互換ですから，これまでの C の場合と同様にコードを書く事ができますので，ご安心ください．

ただ，全く難しくない事ですが，SLLIB を使う前にちょっとだけ知っておきたい事があります．それは，C++の拡張にともなって微妙に C との互換性が失われている部分と，難易度が低くて便利な C++の拡張機能などです．この章ではそれらを解説します．

4.1 NAMESPACE

C++で導入されたものに，namespace があります．これは，別々の人が同じ名前の関数や型を作ってしまうと困った事になる，という問題を回避するもので，要するにカテゴリ名のようなものです．SLLIB では,「sli」という namespace をつけており，例えば何かクラスを使う場合，正式には「`sli::stdstreamio sio;`」のように先頭に `sli::`をつけて使います．

しかし，SLLIB を中心的に使っていく，という事であれば，いちいち `sli::`を書きたくないものです．その場合，

```
using namespace sli;
```

と書けば，それ以降では `sli::`を省略する事ができます．このマニュアルの使用例では，`sli::`を省略しています．C++が初めての方は,「`#include <sli/...>`」したら「`using namespace sli;`」する，と覚えておけば良いでしょう．

4.2 NULL と 0

多くの処理系では，C の場合は

```
define NULL ((void*)0)
```

と定義されています．しかし，C++の場合,

```
define NULL (0)
```

と定義されます．

C++で `NULL` が `0` となっている理由は，C++の場合，ポインタ変数の型のチェックが C よりも厳格になっているからです．例えば，2 つのポインタ変数 `char *ptr0; void *ptr1;` があり,

```
ptr0 = ptr1;
```

はエラーになります．しかし，`0` だけは「どこでもないアドレス」と定義しているので，`ptr0 = 0;` はエラーになりません．そういうわけで，C++では `NULL` は `0` になっているのです．

さて，C++では，クラスの持つメンバ関数は，同名でも引数が異なる場合があります．例えば,

```
int foo( int a );
int foo( char *p );
```

のような場合です．ここで，`hoge.foo(NULL)` あるいは `hoge.foo(0)` とすると，一体どちらの関数を使いたいのかコンパイラが理解できないという事態になります．この場合，`NULL` や `0` を使う場合にはキャストして，型を明示的に示す事が必要です．すなわち,

```
hoge.foo((char *)NULL);
hoge.foo((int)0);
```

のようにしなければなりません．

C++では,「NULL **や** 0 **を使う時はキャストする**」と覚えておけば間違いないでしょう．あるいは，NULL を捨ててしまって,「**いつも 0 をキャストして使う**」方法もあります．

4.3 const char *，char *const *, const char *const *

「const」は，C 言語の範囲なのですが，きちんと使っていない人を意外と多く見かけます．特に，char *p や char **pp といったポインタ変数では関数の仕様を明確にさせるために，const は重要です．

SLLIB でも，関数の引数や返り値に，

```
const char *p0
char *const *p1
const char *const *p2
```

といった表現がでてきます．これらはそれぞれ何が const なのか解説しておきます．

p0 の場合，p0[0] = c; のような変更が禁止されます．p0++; といった演算は禁止されません．

p1 の場合，p1[0][0] = c; のような変更は禁止されませんが，p1[0] = p; のような変更が禁止されます．

p2 の場合，p2[0][0] = c; のような変更も，p2[0] = p; のような変更も禁止されます．

このように，const を使う事で，関数の仕様が明確になります．

また，関数以外の場合，例えば，

```
const char *name = "My Name";
```

のように const を使っていれば，変更禁止の領域を誤って書き換える事もありません．

ユーザのみなさんは，今日から C でも C++でも，ちゃんと const を使いましょう．

4.4 参照

参照は，C++で導入されたポインタ型に似た新しい型ですが，実は**ポインタ型よりも単純な事しかできません**．したがって，**使い方も簡単**です．

例えば，int a; という変数の参照を alias_of_a という名前で作る場合，

```
int &alias_of_a = a;
```

とします．参照は「**エイリアス (別名)**」とも呼ばれる事もありますが，まさしくそのように振舞います．ファイルシステムで言えば，**シンボリックリンクのようなもの**です．例えば，

```
alias_of_a = 10;
```

とすると，a に 10 が代入されますし，

```
int b = alias_of_a;
```

とすると，b に a の値が代入されます．また，

```
int *p = &alias_of_a;
```

とすると，p に a のアドレスが代入されます．

このように，参照は変数の「別名」を提供するだけの簡単な仕組みで，ポインタ変数のように「*」がたくさんついて考え込む事もありませんし，NULL が入る事もありません．NULL が入らないのは，

```
int &alias_of_a;
```

のような，御本尊が存在しない参照を作る事は禁止されているからです．

SLLIB の場合，構造体やクラスのオブジェクトをメンバ関数の引数や返り値にする場合に，参照を使っています．参照を使う理由は，参照の実装はやはりアドレスのコピーですから，構造体やク

ラスのオブジェクトを参照で引数にすると効率が良く，かつ「引数や返り値が配列とはみなさない/みなされない」事がはっきりするからです (例えば，関数の引数が foo *p; の場合，p[n] にアクセスされるかされないのかはっきりしない)．もうおわかりかと思いますが，参照はアドレス渡しですから，関数の引数に使った場合，参照されているオブジェクトは書き換わる可能性があります．ただし，const が引数につく場合は書き換わる事がありません．例えば，

```
int strcmp( const tstring &str, size_t pos2 = 0 );
```

とあれば，

```
    tstring foo;
    if ( hoge.strcmp(foo) == 0 ) {
```

のように使う事ができ，foo は strcmp() によって書き換わる事はありません (SLLIB では，参照の引数のほとんどに const がついています)．

参照は，ポインタとは異なって単純な事しかできないので，上記のように，関数の引数の受け渡しなど限られた事にしか使えず，無くても困らないものです．ですから，ユーザのみなさんは，特に率先して参照を使う必要はありません．ただ，参照の引数がでてきたら，何もつけずに，そのままオブジェクトを引数に書くだけ，と覚えておけば十分でしょう．

4.5 オブジェクトのポインタ変数と関数の引数・返り値

オブジェクトのポインタ変数は，C 言語の型と同じように作る事ができます．例を示します．

```
    tstring foo = "abc";
    tstring *str_ptr = &foo;
    printf("%s\n", str_ptr->cstr());
```

ポインタ変数からメンバ関数を呼び出す場合は,「.」ではなく,「->」を使います．これは構造体の時のルールと同じです．

関数の引数として使う場合も同じです．

```
int my_function( const tstring *str_ptr )
{
    printf("%s\n", str_ptr->cstr());
```

ただし，関数の引数として使う場合には，引数のオブジェクトが書き換わらない場合は，その事を示す「const」を忘れずにつけましょう．

C++では特定の場合を除いて，ポインタ変数のかわりに参照 (§4.4) を使う事もできます．参照の場合は NULL が入る事がないので，その点では安全性が高いと言えます．次に例を示します．

```
int my_function( const tstring &str_ref )
{
    printf("%s\n", str_ref.cstr());
```

このように，参照の場合は「->cstr()」ではなく「.cstr()」を使います．

オブジェクトを関数の返り値にする事もできます．

```
tstring my_function( int a )
{
    tstring ret;
    :
    :
    return ret;
}
```

ただしこの場合は，動作速度の点でやや不利ですので，速度を求める場合はポインタ変数の引数か参照の引数で返すようにします．

関数の引数に，ポインタ変数か参照を使うかは，いくつかの議論があります．例えば，google の規約では,「関数内でオブジェクトを書き換える場合 (つまり const ではない場合) は，ポインタ変数を引数に使ってください」としています．筆者としては，読者自身でルールを作ってそれに従えば良いと考えます．

5 FAQ

5.1 コンパイル時のありがちな警告・エラー

5.1.1 warning: cannot pass objects of non-POD type

可変引数にオブジェクトそのものを与えている場合，コンパイル時にこの警告が出ます．そのまま実行するとたいてい SEGV します．

```
    tstring foo = "abc";
    printf("%s\n",foo);         /* .cstr() を忘れている */
```

.cstr() などを追加しましょう．

5.1.2 error: ‘xxx’ was not declared in this scope

「#include <sli/...>」あるいは「using namespace sli;」を書き忘れていませんか？

5.1.3 error: call of overloaded ‘xxx’ is ambiguous

ユーザ・プログラム側の引数の型と，ライブラリ側の引数の型とのマッチングが完全でなく，ユーザ・プログラムがどのメンバ関数を使おうとしているかが判定できない場合に出ます．

特に，NULL または 0 を引数に与えた場合には要注意です．C++では NULL はゼロそのものだという事を忘れないようにしましょう．NULL と 0 については，§4.2もご覧ください．

このエラーが出た場合は，引数をキャストしてみましょう．

5.1.4 error: invalid conversion from ‘const char*’ to ‘char*’

次のようなコードを書くと，このエラーが出ます．

```
    tstring foo = "abc";
    char *p = foo.cstr();       /* const を忘れている */
```

2 行目は,「const char *p = ... 」としなければなりません．また，強引にキャストしてはいけません．

5.1.5 error: passing ‘xxx’ as ‘yyy’ argument of ‘zzz’ discards qualifiers

次のようなコードを書くと，このエラーが出ます．

```
int my_function( const tstring &my_arg )
{
    my_arg.append("abc");
```

関数の内部でオブジェクト foo を改変しようとしていますが，関数の引数では「const」属性をつけているので，エラーになります．

6 上級者向けの情報

6.1 ヒープにオブジェクトを作る場合の注意

これまでに示したすべてのコードの例では，クラスのオブジェクトをスタックに作っていました．ほとんどのケースでは，それで十分なはずですが，どうしてもオブジェクトをヒープに作りたい場合もあるでしょう．

オブジェクトをヒープに作りたい場合には注意が必要です．その場合には，malloc() 関数，realloc() 関数を使ってオブジェクトを作る事はできません．必ず，new 演算子を使う必要があります．そして，new 演算子で作ったオブジェクトは必ず delete 演算子で消去しなければなりません．

new と delete を使ってオブジェクトを作成・消去する例を示します．

```
    tstring *mystr_ptr = new tstring;             /* オブジェクトの作成 */
    *mystr_ptr = "abc";                           /* 代入 */
    mystr_ptr->append("xyz");                     /* 追加 */
    printf("%s\n", mystr_ptr->cstr());            /* 表示 */
    delete mystr_ptr;                             /* オブジェクトの消去 */
```

「new tstring(64);」のように，コンストラクタの引数を与える事もできます．

当然，new の後の delete を忘れると，メモリリークを引き起こします．したがって，new と delete の使用は最小限にすべきです．

6.2 ヒープにオブジェクトの配列を作りたい場合

ヒープにオブジェクトの配列を作りたい場合にも，malloc() 関数，realloc() 関数を使う事はできません．SLLIB 上級編マニュアルで取り扱っている tarray テンプレートクラス，asarray テンプレートクラス，あるいは STL の vector テンプレートクラス等を使います．

次の例は，任意のクラスの連想配列を作る事ができる asarray テンプレートクラスを使うものです．asarray_tstring クラスのオブジェクトを連想配列で管理しています．

```
#include <sli/asarray_tstring.h>
#include <sli/asarray.h>
    :
    :
    asarray<asarray_tstring> my_config;
    my_config["TEST"]["HOST"] = "www.foo.com";
    printf("%s\n", my_config["TEST"]["HOST"].cstr());
```

6.3 構造体とクラスとの連携

構造体のメンバ定義にクラスを使う事も可能です．例えば,

```
struct mytag {
  int id;
  tstring name;
};
    :
    :
    struct mytag foo;
    foo.id = 0;
    foo.name = "gates";
    sio.printf("%s\n",foo.name.cstr());
```

のように使う事ができます．あるいは,

```
    struct mytag foo_arr[256];
```

のように配列として使う事もできます.

このように，スタックから構造体のオブジェクトを作る場合には，何の注意も必要ありません．しかし，ヒープからオブジェクトを作る場合には，§6.1，§6.2 で紹介した方法で行なう必要があります.

6.4 例外の取り扱い，try {} & catch ()

SLLIB では「メモリ確保の失敗」とごく一部の場合を除き，例外を出しません．したがって，この部分については，本格的なコーディングを目指さないのであれば，読み飛ばしてくださってかまいません.

try{} と catch(){} は，C++で導入された「例外」を扱うための構文です．SLLIB は,「**メモリ確保失敗**」などのユーザが与えた引数と無関係な**致命的な問題**が生じた場合には,「例外」を発生させます[10)]．この「例外」は，ユーザのコードでは，try{} と catch(){} で捉える事ができます．SLLIB の場合は必ず err_rec 型のメッセージを伴った例外が発生するので，これを捉えたい場合は,

[10)] §6.1で紹介した new 演算子の場合，メモリの割り当てに失敗すると bad_alloc 例外を発生します．bad_alloc 例外については，google 等で調べてみてください.

```
    try {
        /* 配列のバッファサイズを変更 */
        array.resize(0,very_big_size);
        return_status = 0;
    }
    catch ( err_rec msg ) {
        sio.eprintf("[EXCEPTION] function=[%s::%s] message=[%s]\n",
                    msg.class_name, msg.func_name, msg.message);
        return_status = -1;
    }
```

のようにします．なお，`try{}` と `catch( err_rec msg ){}` を使っていない場合に例外が発生すると，abort() 関数が呼ばれ，プログラムは終了します．

通常，このような致命的なエラーが発生した場合というのは，それ以上プログラムを続行できないケースがほとんどだと思います．したがって，例外発生時に abort() 関数が呼ばれる仕様で問題なければ，`try{}` と `catch( err_rec msg ){}` を使う必要はありません．

7 CSTREAMIO クラスとその継承クラスのサマリ

cstreamio クラスは，libc の `stdio.h` とそっくりのファイル入出力 API を提供する抽象基本クラス (基本的なメンバ関数の仕様が策定されるクラス) です．cstreamio クラスのメンバ関数の仕様を理解すれば，一から API を再学習する事なく継承クラスでサポートされる様々なストリームが扱えるようになります．

表 5に示すように，様々なストリームに対応した継承クラスがあり，継承クラスではそれぞれのストリームに特化したメンバ関数が追加定義されています．このセクションでは，CSTREAMIO クラスとその継承クラスについて，全体を主に表でまとめています．

使用例については，チュートリアルの §3.2か，リファレンス (§8.1以降) の EXAMPLE をご覧ください．

7.1 継承クラスのサマリ

表 5 に，cstreamio クラスの継承クラス一覧を示します．

	クラス名	機能
§8.2	stdstreamio	標準入出力・標準エラー出力・標準ファイル入出力 (`stdio.h` に相当)
§8.3	gzstreamio	gzip 圧縮・伸長に対応したファイル入出力
§8.4	bzstreamio	bzip2 圧縮・伸長に対応したファイル入出力
§8.5	httpstreamio	http サーバからの入力 (download)
§8.6	ftpstreamio	ftp サーバから入力 (download)・ftp サーバへ出力 (upload)
§8.7	pipestreamio	パイプから入力・パイプへの出力
§8.8	digeststreamio	{std, gz, bz, http, ftp}streamio をパス名によって自動的に切り替えて利用
§8.9	termlineio	便利なコマンド入力 (readline への wrapper)
§8.10	termscreenio	端末スクリーンへの入力・出力 (ページャ・エディタを起動)
§8.11	inetstreamio	シーケンシャル・1or2 ウェイ接続用の低レベルインターネットクライアント

表 5: ユーザが実際に利用する，cstreamio クラスの継承クラス一覧．

上記の各継承クラスに共通するメンバ関数については，§8.1以降，各継承クラスで追加定義されたメンバ関数については，§8.2以降で解説しています．それらの全体像は表 6 をご覧ください．

7.2 基本クラス・継承クラスのメンバ関数実装の全体像

表 6 に，各クラスにおいてどのメンバ関数が使えるか，あるいはどのように実装されているかの一覧を示します．表の上半分のメンバ関数は，抽象基本クラス cstreamio で策定されたものを示しており，表の下半分のメンバ関数は継承クラスにおいて追加定義されたものです．

クラス名		stdstreamio	gzstreamio	bzstreamio	httpstreamio	ftpstreamio	pipestreamio	digeststreamio	termlineio	termscreenio	inetstreamio
基本クラスcstreamioのメンバ関数											
`open()`, etc.	§8.1.1	§8.2.2	§8.3.1	§8.4.1	§8.5.1	§8.6.1	§8.7.1	§8.8.1	§8.9.1	§8.10.1	§8.11.1
`close()`	§8.1.2	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`read()`	§8.1.3	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`write()`	§8.1.3	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`bread()`	§8.1.4	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`bwrite()`	§8.1.5	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`rskip()`	§8.1.6	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`wskip()`	§8.1.7	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`getchr()`	§8.1.8	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`getstr()`	§8.1.9	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`getline()`	§8.1.10	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`scanf()`	§8.1.11	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`putchr()`	§8.1.12	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`putstr()`	§8.1.13	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`printf()`	§8.1.14	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`flush()`	§8.1.15	⇐	⇐	⇐	-	⇐	⇐	⇐	⇐	⇐	⇐
`eof()`, etc.	§8.1.16	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐
`seek()`, etc.	§8.1.17	§8.2.5	-	-	-	-	-	⇐	-	-	-
継承クラスの新規メンバ関数											
`eprintf()`		§8.2.3	-	-	-	-	-	-	-	-	-
`eflush()`		§8.2.4	-	-	-	-	-	-	-	-	-
`sync()`		-	§8.3.2	-	-	-	-	-	-	-	-
`content_length()`		§8.2.7	-	-	§8.5.2	§8.6.2	-	§8.8.4	-	-	-
`user_agent.assign()`		-	-	-	§8.5.3	-	-	§8.8.5	-	-	-
`username.assign()`		-	-	-	-	§8.6.3	-	§8.8.6	-	-	-
`password.assign()`		-	-	-	-	§8.6.4	-	§8.8.7	-	-	-
`openp()`, etc.		-	-	-	-	-	-	§8.8.2	-	-	-
`is_write_mode()`		-	-	-	-	-	-	§8.8.3	-	-	-
`set_prompt()`		-	-	-	-	-	-	-	§8.9.2	-	-
`automate_history()`		-	-	-	-	-	-	-	§8.9.3	-	-
`add_history()`		-	-	-	-	-	-	-	§8.9.4	-	-
`clear_history()`		-	-	-	-	-	-	-	§8.9.5	-	-
`stifle_history()`		-	-	-	-	-	-	-	§8.9.6	-	-
`unstifle_history()`		-	-	-	-	-	-	-	§8.9.7	-	-
`read_history()`		-	-	-	-	-	-	-	§8.9.8	-	-
`write_history()`		-	-	-	-	-	-	-	§8.9.9	-	-
`path()`		-	-	-	-	-	-	-	-	-	§8.11.2
`host()`		-	-	-	-	-	-	-	-	-	§8.11.3

表 6: 基本クラス cstreamio とその継承クラスが提供するメンバ関数一覧．記号「⇐」は基本クラスのメンバ関数を継承している事を示す．§ の表記があるものについては，メンバ関数が再定義されているか，追加定義されている事を示す．

8 CSTREAMIO クラスとその継承クラスのリファレンス

8.1 CSTREAMIO クラスのメンバ関数

表 7 はメンバ関数一覧です．libc と同じ機能を持つメンバ関数については，対応を示します．

	cstreamio クラス	機能	対応する libc の関数
§8.1.1	open(), openf(), vopenf()	ストリームを開く	fopen()
§8.1.2	close()	ストリームを閉じる	fclose()
§8.1.3	read()	バイナリストリームの入力	fread()
§8.1.3	write()	バイナリストリームの出力	fwrite()
§8.1.4	bread()	バイナリストリームの入力 (endian 変換有)	—
§8.1.5	bwrite()	バイナリストリームの出力 (endian 変換有)	—
§8.1.6	rskip()	ストリームのシーク (可能な場合) または読み飛ばし	—
§8.1.7	wskip()	ストリームのシーク (可能な場合) または n バイトの空白キャラクタの書き込み	—
§8.1.8	getchr()	1 文字の入力	fgetc()
§8.1.9	getstr()	文字列の入力	fgets()
§8.1.10	getline()	1 行の入力	—
§8.1.11	scanf()	入力を書式で変換・引数に代入	fscanf()
§8.1.12	putchr()	1 文字の出力	fputc()
§8.1.13	putstr()	文字列の出力	fputs()
§8.1.14	printf()	引数の値を書式変換後に出力	fprintf()
§8.1.15	flush()	バッファの内容を強制出力	fflush()
§8.1.16	eof(), error(), reseterr()	ストリームステータスのチェックとリセット	feof(), etc.
§8.1.17	seek()	ストリームの位置の変更	fseek()
§8.1.17	rewind()	ストリーム位置を先頭に変更	rewind()
§8.1.18	tell()	ストリーム位置表示子の値	ftell()
§8.1.19	is_seekable()	シーク可能かどうかを調べる	-

表 7: cstreamio クラスが提供するメンバ関数一覧.

8.1.1 open(), openf(), vopenf()

NAME

open(), openf(), vopenf() — ストリームを開く

SYNOPSIS

```
int open( const char *mode );  ........................................... 1
int open( const char *mode, int fd );  ................................... 2
int open( const char *mode, cstreamio &sref );  .......................... 3
int open( const char *mode, const char *path );  ......................... 4
```

```
int openf( const char *mode, const char *path_fmt, ... );  ................. 5
int vopenf( const char *mode, const char *path_fmt, va_list ap );  ....... 6
```

DESCRIPTION

path あるいは path_fmt で示されたファイルを開くか，fd で指定されたディスクリプタ，あるいは sref で指定された cstreamio クラスの継承クラスのオブジェクトをストリームに結びつけます．mode については，メンバ関数 1 およびメンバ関数 2 の場合，読み込みの時は"r"を，書き込みの時は"w"を指定します．メンバ関数 3〜6 の場合も基本的には同様ですが，継承クラスによって利用できる値が異なりますので，各継承クラスでの解説を参照してください．

メンバ関数 3 は，すでに cstreamio クラスの継承クラスのオブジェクトでオープンしたストリームがあった時，ストリームの途中から gzip や bzip2 の圧縮がかかったストリームに変更になるような場合に使います (EXAMPLE-2 を参照)．

メンバ関数 5 およびメンバ関数 6 では，path_fmt 以降の引数を libc の printf() および vprintf() のそれと同様に指定可能です．printf() および vprintf() の書式については，§8.1.14 の解説を参照してください．

PARAMETER

[I]	mode	ファイルを開くモード
[I]	fd	ファイルディスクリプタ
[I]	sref	cstreamio クラスの継承クラスのオブジェクト
[I]	path	ファイル名
[I]	path_fmt	ファイル名のフォーマット指定
[I]	...	ファイル名の各要素データ
[I]	ap	ファイル名の全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	ファイルが存在しない等の理由で，ストリームのオープンに失敗した場合
	:	指定した mode が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	指定した mode と fd の関係が正しくない等の理由で，オープンに失敗した場合 (メンバ関数 2)
	:	ストリームが，指定のモードでのアクセスを許されていない等の理由で，オープンに失敗した場合
	:	path_fmt のパスを示す文字列が PATH_MAX を超えている場合
	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合

EXCEPTION

標準入出力のファイル・ディスクリプタの複製に失敗した場合

EXAMPLE-1

次のコードは，ディレクトリ *directory* に存在するファイル *file.txt* を読み込みモードで開きます．

```
stdstreamio f_in;
```

```
if (f_in.openf("r", "%s/%s", "directory", "file.txt") < 0) {
    エラー処理
}

f_in.close();
```

EXAMPLE-2

次のコードは，1 行のテキストヘッダと gzip 圧縮によるバイナリデータからなるファイル *complex_file.dat* を読み，標準出力します．

```
stdstreamio f_in_text;
gzstreamio f_in_gz;
char c_buf[256];

if (f_in_text.open("r", "complex_file.dat") < 0) {
    エラー処理
}

/* ヘッダ部を読み，表示する */
printf("ヘッダ: %s", f_in_text.getline());

/* 圧縮されたデータを読む */
if (f_in_gz.open("r", f_in_text) < 0) {
    エラー処理
}

/* 圧縮されたデータを 1 行ずつ読み込み表示する */
while (f_in_gz.getstr(c_buf, 256) != NULL) {
    printf("%s",c_buf);
}

f_in_gz.close();
f_in_text.close();
```

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用して下さい．

メンバ関数 3〜6 については，実際に使用するクラスの解説を参照してください．各クラスの解説は次に示すとおりです．

§8.2.2	`stdstreamio::open()`	§8.3.1	`gzstreamio::open()`
§8.4.1	`bzstreamio::open()`	§8.5.1	`httpstreamio::open()`
§8.6.1	`ftpstreamio::open()`	§8.7.1	`pipestreamio::open()`
§8.8.1	`digeststreamio::open()`	§8.9.1	`termlineio::open()`
§8.10.1	`termscreenio::open()`	§8.11.1	`inetstreamio::open()`

8.1.2 close()

NAME

close() — ストリームを閉じる

SYNOPSIS

```
int close();
```

DESCRIPTION

`open()` で開いたストリームを閉じます.

RETURN VALUE

0	:	正常終了
0 以外の値	:	エラー

EXAMPLE

次のコードは，ディレクトリ *directory* に存在する読み込みモードで開かれたファイル *file.txt* を閉じます.

```
stdstreamio f_in;

if (f_in.openf("r", "%s/%s", "directory", "file.txt") < 0) {
    エラー処理
}

f_in.close();
```

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください.

8.1.3 read(), write()

NAME

read(), write() — ストリームの入出力

SYNOPSIS

```
ssize_t read( void *buf, size_t size );
ssize_t write( const void *buf, size_t size );
```

DESCRIPTION

`read()` は，`open()` で開いたストリームから `size` バイトのデータを読み込み，`buf` で与えられたバッファに格納します.

`write()` は，`buf` で指定されたバッファから得た `size` バイトのデータを，`open()` で開いたストリームに書き込みます.

PARAMETER

[I]	`buf`	データの格納バッファ(`read()` の場合)
[O]	`buf`	データの格納バッファ(`write()` の場合)
[I]	`size`	データの個数

([I]：入力，[O]：出力)

RETURN VALUE

正の値	:	読み書きに成功したバイト数
0	:	読み込むストリームの EOF に達した場合 (`read()`)
	:	指定した `size` が 0 の場合 (`read()`)
負の値 (エラー)	:	指定した `buf` が不適切である場合
	:	ストリームが開かれていない場合 (`read()`)
	:	入出力ストリームのオープンモードが不整合の場合
	:	読み込むストリームが，本メンバ関数を使用する以前に異常であった場合 (`read()`)
	:	上記以外の動作環境上の理由によって，読み書きに失敗した場合 ⟦例えば，ファイルディスクリプタによって参照されるストリームに十分な空きがない場合が該当します.⟧

EXAMPLE

次のコードは，ディレクトリ *directory* に存在する読み込みモードで開かれたファイル *file.txt* から 512 バイトのデータ を読み込み，`buf` で与えられたバッファに格納します.

```
stdstreamio f_in;
char c_buf[1024];

if (f_in.openf("r", "%s/%s", "directory", "file.txt") < 0) {
    エラー処理
}

if (f_in.read(c_buf, 512) < 0) {
    エラー処理
}

printf("%s", c_buf);

f_in.close();
```

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください.

`buf` で指定されたバッファのサイズが，`size` より小さい場合には，実行しているプログラムが異常終了する，または強制終了される事があります.

8.1.4 bread()

NAME

bread() — バイナリストリームの入力

SYNOPSIS

```
ssize_t bread( void *buf, ssize_t sz_type, size_t n,
               bool little_endian );  ......................................... 1
ssize_t bread( void *buf, const bstream_info binfo[], size_t n,
               bool little_endian );  ......................................... 2
```

DESCRIPTION

bread() は，open() で開いたストリームからバイナリデータを読み込み，buf で与えられたバッファに格納します．ストリームデータのエンディアン指定 little_endian と処理系に応じて，バイトオーダーを変換します．

メンバ関数 1 の場合，|sz_type| バイトの整数または浮動小数点値を n 個読み込みます．浮動小数点値の場合は，sz_type に負の数を与えます．

メンバ関数 2 の場合，バイナリデータの構造を bstream_info 構造体の配列で与える事ができるので，複雑なデータにも対応できます．配列 bstream_info で定義されるデータブロックを，n 回読み込みます．

bstream_info 構造体の定義は次のとおりです．

```
typedef struct {
    ssize_t sz_type;
    ssize_t length;
} bstream_info;
```

メンバ sz_type に 1 要素のバイト数とデータ種別 (負の場合は浮動小数点値) を与え，length にはその個数を与えます．定義の終端を示すために，配列の最後の sz_type には 0 をセットしてください．

PARAMETER

[O]	buf	データの格納バッファ(bread() の場合)
[I]	sz_type	データの 1 要素のバイト数とデータ種別 (負の値：浮動小数点値の場合, 正の値：整数の場合)
[I]	binfo	バイナリデータの 1 データブロックの構造定義
[I]	n	データまたはデータブロックの個数
[I]	little_endian	ストリームデータのエンディアン指定 (true：リトルエンディアン, false：ビッグエンディアン)

([I]：入力，[O]：出力)

RETURN VALUE

正の値	:	読み込みに成功したバイト数
0	:	読み込むストリームの EOF に達した場合
	:	引数により 0 バイトの読み込みが指定された場合
負の値 (エラー)	:	指定した buf が不適切である場合
	:	ストリームが開かれていない場合
	:	入出力ストリームのオープンモードが不整合の場合
	:	読み込むストリームが，本メンバ関数を使用する以前に異常であった場合
	:	上記以外の動作環境上の理由で読み書きに失敗した場合 [[例えば，ファイルディスクリプタによって参照されるストリームに異常が生じた場合が該当します.]]

EXAMPLE

次のコードは，ディレクトリ *directory* に存在する，バイナリファイル *file.dat* から 4 バイトのデータを読み込み，`ui_buf` で与えられたバッファに格納します．そして，`ui_buf` の内容を 16 進数で標準出力します．この場合，バイナリファイル *file.dat* はリトルエンディアンで記載されています.

```
stdstreamio f_in;
unsigned int ui_buf;

if (f_in.openf("r", "%s/%s", "directory", "file.dat") < 0) {
    エラー処理
}

if (f_in.bread(&ui_buf, sizeof(ui_buf), 1, true) < 0) {
    エラー処理
}

printf("%X\n", ui_buf);

f_in.close();
```

メンバ関数 2 の使用例については，§8.1.5の EXAMPLE を参照してください.

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください.

`buf` で指定されたバッファのサイズが，引数による指定に対して不十分な場合には，実行しているプログラムが異常終了する，または強制終了される事があります.

8.1.5 bwrite()

NAME

bwrite() — バイナリストリームの出力

SYNOPSIS

```
ssize_t bwrite( const void *buf, ssize_t sz_type, size_t n,
                bool little_endian );  ........................................ 1
ssize_t bwrite( const void *buf, const bstream_info binfo[], size_t n,
                bool little_endian );  ........................................ 2
```

DESCRIPTION

bwrite() は，open() で開いたストリームに buf で与えられたバイナリデータを書き込み，ストリームデータのエンディアン指定 little_endian と処理系に応じて，バイトオーダーを変換します.

メンバ関数 1 の場合，|sz_type| バイトの整数または浮動小数点値を n 個書き込みます．浮動小数点値の場合は，sz_type に負の数を与えます.

メンバ関数 2 の場合，バイナリデータの構造を bstream_info 構造体の配列で与える事ができるので，複雑なデータにも対応できます．配列 bstream_info で定義されるデータブロックを，n 回書き込みます.

bstream_info 構造体の定義は次のとおりです.

```
typedef struct {
    ssize_t sz_type;
    ssize_t length;
} bstream_info;
```

メンバ sz_type に 1 要素のバイト数とデータ種別 (負の場合は浮動小数点値) を与え，length にはその個数を与えます．定義の終端を示すために，配列の最後の sz_type には 0 をセットしてください.

PARAMETER

[I]	buf	データの格納バッファ
[I]	sz_type	データの 1 要素のバイト数とデータ種別 (負の値：浮動小数点値の場合, 正の値：整数の場合)
[I]	binfo	バイナリデータの 1 データブロックの構造定義
[I]	n	データまたはデータブロックの個数
[I]	little_endian	ストリームデータのエンディアン指定 (true：リトルエンディアン, false：ビッグエンディアン)

([I]：入力，[O]：出力)

RETURN VALUE

正の値	:	読み書きに成功したバイト数
0	:	引数により 0 バイトの書き込みが指定された場合
負の値 (エラー)	:	指定した buf が不適切である場合
	:	ストリームが開かれていない場合
	:	入出力ストリームのオープンモードが不整合の場合
	:	書き込むストリームが，本メンバ関数を使用する以前に異常であった場合
	:	上記以外の動作環境上の理由で読み書きに失敗した場合 [[例えば，ファイルディスクリプタによって参照されるストリームに十分な空きがない場合が該当します.]]

EXCEPTION

エンディアン変換用の一時バッファの確保に失敗した場合

EXAMPLE

次のコードは，ディレクトリ *directory* に存在する，バイナリファイル *file.dat* へ，`binfo` で定義されるデータブロックを 1 回書き込みます．1 データブロックは，double 型が 3 個，char 型が 32 個と定義しています．この場合，バイナリファイル *file.dat* はビッグエンディアンで記載されています．

```
stdstreamio f_out;
bstream_info binfo[] = { {-8,3}, {1,32}, {0} };
char buffer[256];
  :
buffer にデータを登録
  :
if (f_out.openf("w", "%s/%s", "directory", "file.dat") < 0) {
    エラー処理
}

if (f_out.bread(buffer, binfo, 1, false) < 0) {
    エラー処理
}

f_out.close();
```

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください．

`buf` で指定されたバッファのサイズが，引数による指定に対して不十分な場合には，実行しているプログラムが異常終了する，または強制終了される事があります．

8.1.6 rskip()

NAME

rskip() — ストリームのシーク (可能な場合) または読み飛ばし

SYNOPSIS

`ssize_t rskip( size_t n );`

DESCRIPTION

`rskip()` は，`open()` で開かれた読み込み可能なストリームに対して，可能であれば `n` バイトの順方向のシークを行ないます．シークが不可能な場合は，`n` バイトの読み飛ばしを行ないます．

PARAMETER

[I] `n` シークまたは読み飛ばすバイト数

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	シークまたは読み飛ばしたバイト数
負の値	:	エラーが発生した場合

EXCEPTION

読み飛ばしのための一時領域の確保に失敗した場合

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください．

8.1.7 wskip()

NAME

wskip() — ストリームのシーク (可能な場合) または n バイトの空白キャラクタの書き込み

SYNOPSIS

```
ssize_t wskip( size_t n, int ch );
```

DESCRIPTION

`wskip()` は，`open()` で開かれた書き込み可能なストリームに対して，可能であれば `n` バイトの順方向のシークを行ないます．シークが不可能な場合は，空白キャラクタ `ch` を `n` バイトを書き込みます．

PARAMETER

[I]	`n`	シークまたは空白データを書き込むバイト数
[I]	`ch`	書き込む場合の空白キャラクタ

([I] : 入力，[O] : 出力)

RETURN VALUE

非負の値	:	シークまたは書き込んだバイト数
負の値	:	エラーが発生した場合

EXCEPTION

空白データのための一時領域の確保に失敗した場合

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください．

8.1.8 getchr()

NAME

getchr() — 文字の入力

SYNOPSIS

```
int getchr();
```

DESCRIPTION

`getchr()` は，`open()` で開いたストリームから 1 文字読み込み，int 型で返します.

RETURN VALUE

非負の値	:	読み込んだ unsigned char 型の文字を int 型にキャストした値
EOF	:	ストリームの終端に達した場合 (`getchr()`)
EOF (エラー)	:	ストリームが開かれていない場合
	:	入出力ストリームのオープンモードが不整合の場合

EXAMPLE

次のコードは，ディレクトリ *directory* に存在する読み込みモードで開かれたファイル *file.txt* から 1 文字を読み込み，標準出力します.

```
stdstreamio f_in;
int i_chr;

if (f_in.openf("r", "%s/%s", "directory", "file.txt") < 0) {
    エラー処理
}

if ((i_chr = f_in.getchr()) == EOF) {
    printf("EOF\n");
}
else {
    printf("%c\n", i_chr);
}

f_in.close();
```

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください.

8.1.9 getstr()

NAME

getstr() — 文字列の入力

SYNOPSIS

`char *getstr( char *s, size_t size );`

DESCRIPTION

`getstr()` は，`open()` で開いたストリームから最大で `size`-1 個の文字を読み込み，`s` で指定されるバッファに格納します．読み込みは EOF または改行文字を読み込んだ後で停止します．読み込まれた改行文字も `s` で指定されるバッファに格納されます.

PARAMETER

[O]	`s`	読み込んだ文字の格納バッファ
[I]	`size`	読み込む文字数

([I]：入力，[O]：出力)

RETURN VALUE

NULL 以外の値	:	格納バッファのアドレス
NULL	:	ストリームの終端に達した場合
NULL(エラー)	:	指定した `buf` または `size` が不適切である場合
	:	ストリームが開かれていない場合
	:	入出力ストリームのオープンモードが不整合の場合

EXCEPTION

データ読み込み用のバッファの確保に失敗した場合

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください．

8.1.10 getline()

NAME

getline() — 文字と文字列の入力

SYNOPSIS

```
const char *getline(); ........................................ 1
const char *getline( size_t nchars ); ......................... 2
```

DESCRIPTION

`open()` で開いたストリームから改行文字までの文字列をオブジェクト内部のバッファに読み込み，その内部バッファのアドレスを返します．このアドレスは，次にこのオブジェクトのメンバ関数が呼ばれると無効になります．文字数を制限したい場合，`nchars` を指定します．この場合，改行文字が現れるか，`nchars` 文字目に達するまで読み込みます．

PARAMETER

[I]	`nchars`	読み込む文字数の制限値

([I]：入力，[O]：出力)

RETURN VALUE

NULL 以外の値	:	内部バッファのアドレス
NULL	:	ストリームの終端に達した場合
NULL(エラー)	:	ストリームが開かれていない場合
	:	入出力ストリームのオープンモードが不整合の場合

EXCEPTION

データ読み込み用バッファの確保に失敗した場合

EXAMPLE

次のコードは，ディレクトリ *directory* に存在する読み込みモードで開かれたファイル *file.txt* から 1 行ずつ読み込んで，標準出力します．

```
stdstreamio f_in;
const char *line_ptr;

if (f_in.openf("r", "%s/%s", "directory", "file.txt") < 0) {
    エラー処理
}

while ((line_ptr = f_in.getline()) != NULL) {
    printf("%s", line_ptr);
}

f_in.close();
```

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください．

8.1.11 scanf(), vscanf()

NAME

scanf(), vscanf() — 書式付き入力変換

SYNOPSIS

`int scanf( const char *format, ... );` 1
`int vscanf( const char *format, va_list ap );` 2
`int scanf( size_t nchars, const char *format, ... );` 3
`int vscanf( size_t nchars, const char *format, va_list ap );` 4

DESCRIPTION

`open()` で開いたストリームから `format` の変換指定に従って読み込み，`format` 以降の引数に格納します．

`format` の変換指定に応じて変換した結果を，メンバ関数 1 およびメンバ関数 3 では各要素データに，メンバ関数 2 およびメンバ関数 4 では可変長引数のリスト `ap` に読み込みます．

入力変換の対象となる文字列バッファは `getline()` メンバ関数 (§8.1.10) で返されるものを利用します．すなわち，`nchars` を指定しない場合，入力変換は必ず行単位で行なわれます．`format` 中の `%`に続く変換文字とその機能は次の表のとおりです．

変換文字	変換の内容	引数の型
hhd	入力を符号付き 10 進数に変換する	char 型
hd	入力を符号付き 10 進数に変換する	short 型
d	入力を符号付き 10 進数に変換する	int 型
ld	入力を符号付き 10 進数に変換する	long 型
lld	入力を符号付き 10 進数に変換する	long long 型
zd	入力を符号付き 10 進数に変換する	ssize_t 型
hhu	入力を符号なし 10 進数に変換する	unsigned char 型
hu	入力を符号なし 10 進数に変換する	unsigned short 型
u	入力を符号なし 10 進数に変換する	unsigned int 型
lu	入力を符号なし 10 進数に変換する	unsigned long 型
llu	入力を符号なし 10 進数に変換する	unsigned long long 型
zu	入力を符号なし 10 進数に変換する	size_t 型
hho	入力を符号なし 8 進数に変換する	(unsigned) char 型
ho	入力を符号なし 8 進数に変換する	(unsigned) short 型
o	入力を符号なし 8 進数に変換する	(unsigned) int 型
lo	入力を符号なし 8 進数に変換する	(unsigned) long 型
llo	入力を符号なし 8 進数に変換する	(unsigned) long long 型
zo	入力を符号なし 8 進数に変換する	size_t 型, ssize_t 型
hhx, hhX	入力を符号なし 16 進数に変換する	(unsigned) char 型
hx, hX	入力を符号なし 16 進数に変換する	(unsigned) short 型
x, X	入力を符号なし 16 進数に変換する	(unsigned) int 型
lx, lX	入力を符号なし 16 進数に変換する	(unsigned) long 型
llx, llX	入力を符号なし 16 進数に変換する	(unsigned) long long 型
zx, zX	入力を符号なし 16 進数に変換する	size_t 型, ssize_t 型
c	「最大フィールド幅」(デフォルトは 1) で指定された幅の文字列を引数に格納する	char *型
s	ホワイトスペースではない文字で構成された文字列を引数に格納する	char *型
f	入力を符号付き浮動小数点実数に変換する	float 型
lf	入力を符号付き浮動小数点実数に変換する	double 型
e, E	入力を符号付き浮動小数点実数に変換する	float 型
le, lE	入力を符号付き浮動小数点実数に変換する	double 型
g, G	入力を符号付き浮動小数点実数に変換する	float 型
lg, lG	入力を符号付き浮動小数点実数に変換する	double 型
a, A	入力を符号付き浮動小数点実数に変換する	float 型
la, lA	入力を符号付き浮動小数点実数に変換する	double 型
p	入力を void*ポインタに変換する	void*型
n	入力からここまでに消費された文字数を int*ポインタ引数が指す整数に保存する	int*型

さらに，% と変換文字との間に，最大フィールド幅を指定する事ができます．

```
f_in.scanf("%__f",si_value);
             ↑
            [m]
```

オプション	意味	例
m (桁数記入)	最大フィールド幅を指定する．この値に達するか，一致しない文字が見つかるまで文字を読み込む．	.scanf("%10d···

PARAMETER

[I]	format	読み込みフォーマット指定
[O]	...	書き込み先となる各要素データ
[O]	ap	書き込み先となる可変長引数のリスト
[I]	nchars	読み込む文字数の制限値

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	読み込みと変換が成功した入力要素の個数
EOF(エラー)	:	ストリームが開かれていない場合
	:	入出力ストリームのオープンモードが不整合の場合
	:	読み込まれたバイト列が有効な数字でない場合
	:	引数が十分でない，または format が NULL である場合
	:	メモリ不足である場合
	:	format で指定された整数型に変換した際，対応する整数型に格納できるサイズを超えている場合

EXCEPTION

データ読み込み用バッファの確保に失敗した場合

EXAMPLE

次のコードは，ディレクトリ *directory* に存在する読み込みモードで開かれたファイル *file.txt* からスペース区切りで数値をフォーマットに従って読み込み，i_YY,i_MM および i_DD に格納します．そして，格納した値を指定したフォーマットに従って，標準出力します．なお，本例において *directory/file.txt* は，YYYY␣MM␣DD のフォーマットで年月日が記されているものとします．

```
stdstreamio f_in;
int i_YY = 0, i_MM = 0, i_DD = 0;

if (f_in.openf("r", "%s/%s", "directory", "file.txt") < 0) {
    エラー処理
}

if (f_in.scanf("%d %d %d", &i_YY, &i_MM, &i_DD) < 0) {
    エラー処理
}
```

```
    printf("%04d / %02d / %02d\n", i_YY, i_MM, i_DD);

    f_in.close();
```

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください．

`format` に"%s"を指定する場合，格納バッファの大きさ以上の文字列が入力されるとバッファオーバーランが発生します．この問題は，下記のように最大フィールド幅を指定して読み込む事で，回避できます．その際に読み込まれなかった文字あるいは改行文字をバッファに残したくない場合は，下記のように改行文字までの文字列を空読みした後，改行文字を空読みしてください．

```
    stdstreamio f_in;
    char c_buf[20];

    f_in.scanf("%19s%*[^\n]",c_buf);
    f_in.getchr();
```

8.1.12 putchr()

NAME

putchr() — 文字の出力

SYNOPSIS

`int putchr( int c );`

DESCRIPTION

`putchr()` は，文字 `c` を `open()` で開いたストリームに書き込みます．

PARAMETER

[I] c 書き込む文字

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	正常終了，書き込まれた unsigned char 型の文字を int 型にキャストした値
EOF(エラー)	:	入出力ストリームのオープンモードが不整合の場合
	:	上記以外の動作環境上の理由によって，書き込みに失敗した場合 〚例えば，ファイルディスクリプタによって参照されるストリームに十分な空きがない場合が該当します.〛

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください．

8.1.13 putstr()

NAME

putstr() — 文字列の出力

SYNOPSIS

`int putstr( const char *s );`

DESCRIPTION

`putstr()` は，文字列 `s` を `open()` で開いたストリームに書き込みます．

PARAMETER

[I] `s` 書き込む文字列

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	書き込みに成功した場合
EOF(エラー)	:	入出力ストリームのオープンモードが不整合の場合
	:	上記以外の動作環境上の理由によって，書き込みに失敗した場合 [[例えば，ファイルディスクリプタによって参照されるストリームに十分な空きがない場合が該当します.]]

EXAMPLE

次のコードは，ディレクトリ *directory* に存在する書き込みモードで開かれたファイル *file.txt* に文字列 `c_sentence` を書き込みます.

```
stdstreamio f_out;
const char *c_sentence = "This is an example code for the USER";

if (f_out.openf("w", "%s/%s", "directory", "file.txt") < 0) {
    エラー処理
}

if (f_out.putstr(c_sentence) < 0) {
    エラー処理
}

f_out.close();
```

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください.

8.1.14 printf(), vprintf()

NAME

printf(), vprintf() — 整形された書式へ変換する関数

SYNOPSIS

```
int printf( const char *format, ... );
int vprintf( const char *format, va_list ap );
```

DESCRIPTION

open() で開いたストリームに，format 以降の引数に格納されている内容を，format の変換指定に従って書き込みます．

printf() は，各要素データを，vprintf() は，可変長引数のリスト ap を format の変換指定に応じて変換します．

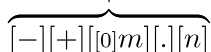
format 中の % に続く変換文字とその機能は次の表のとおりです．なお，% 自体を出力したい場合は %% とします．

変換文字	変換の内容	引数の型
hhd	引数を符号付き 10 進数に変換する	char 型
hd	引数を符号付き 10 進数に変換する	short 型
d	引数を符号付き 10 進数に変換する	int 型
ld	引数を符号付き 10 進数に変換する	long 型
lld	引数を符号付き 10 進数に変換する	long long 型
zd	引数を符号付き 10 進数に変換する	ssize_t 型
hhu	引数を符号なし 10 進数に変換する	unsigned char 型
hu	引数を符号なし 10 進数に変換する	unsigned short 型
u	引数を符号なし 10 進数に変換する	unsigned int 型
lu	引数を符号なし 10 進数に変換する	unsigned long 型
llu	引数を符号なし 10 進数に変換する	unsigned long long 型
zu	引数を符号なし 10 進数に変換する	size_t 型
hho	引数を符号なし 8 進数に変換する	(unsigned) char 型
ho	引数を符号なし 8 進数に変換する	(unsigned) short 型
o	引数を符号なし 8 進数に変換する	(unsigned) int 型
lo	引数を符号なし 8 進数に変換する	(unsigned) long 型
llo	引数を符号なし 8 進数に変換する	(unsigned) long long 型
zo	引数を符号なし 8 進数に変換する	size_t 型, ssize_t 型
hhx, hhX	引数を符号なし 16 進数に変換する	(unsigned) char 型
hx, hX	引数を符号なし 16 進数に変換する	(unsigned) short 型
x, X	引数を符号なし 16 進数に変換する	(unsigned) int 型
lx, lX	引数を符号なし 16 進数に変換する	(unsigned) long 型
llx, llX	引数を符号なし 16 進数に変換する	(unsigned) long long 型
zx, zX	引数を符号なし 16 進数に変換する	size_t 型, ssize_t 型
c	引数を 1 文字として出力する	int 型
s	引数によって指し示される文字列を出力する．出力されるのは null 文字の前の文字までか，指示された最大の文字数分	const char *型
f	引数を float か double として受けとり [-] ○○○．○○○の形の 10 進数に変換する	浮動小数点型
e，E	引数を float か double として受けとり [-] ○．○○○○○ e[±] ○○の形の 10 進数に変換する	浮動小数点型
g, G	%e あるいは %f による変換の文字数の少ない方の変換をとる	浮動小数点型
a, A	引数を float か double として受けとり [-]0x ○．○○○○○ p[±] ○○の形の 16 進数に変換する	浮動小数点型
p	引数を void*ポインタとして受けとり 16 進数で出力する	void*型
n	これまでに出力された文字数を int*ポインタ引数が指す整数に保存する	int*型

さらに，% と変換文字との間に，次のオプション指示子を入れると，より詳細な書式指定ができます.

```
sio.printf("%__f…\n",x);
```

↑



オプション	意味	例
− (マイナス符号)	変換された引数を左詰めで出力する. 1234______	.printf("%-6d…
+	符号付き変換された数値の前に，常に符号 (+ か -) をつける	.printf("%+6d…
m (桁数記入)	確保すべき領域幅を最小でも m 字の広さにとる. 領域幅に空白が生じる時は，左側にブランクが入る．0 をつけるとゼロパディングする. ______1234	.printf("%10d… .printf("%010d…
. (ピリオド)	フィールド幅と文字数または小数点以下の桁数との区切り	.printf("%10.5f…
n (桁数記入)	f 指定の場合は，小数点以下の桁数. e，E，g，G 指定の場合は，精度. 文字列の場合はその取り幅 (文字の頭から)	.printf("%10.5f… .printf("%.15g… .printf("%10.5s…

PARAMETER

[I]	format	書き込みフォーマット指定
[I]	...	書き込む各要素データ
[I]	ap	書き込む可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	書き込まれた文字数
負の値 (エラー)	:	動作環境上の理由によって，書き込みに失敗した場合

EXCEPTION

データ書き込み用バッファの確保に失敗した場合

EXAMPLE

次のコードは，文字列 c_sentence を%s 形式でストリーム (標準出力) に書き込みます.

```
stdstreamio s_io;
const char *c_sentence = "This is an example code for the USER";

if (s_io.printf("%s\n", c_sentence) < 0) {
    エラー処理
}
```

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください.

8.1.15 flush()

NAME

flush() — ストリームの内容を強制的に出力

SYNOPSIS

```
int flush();
```

DESCRIPTION

flash() は，open() で開いたストリーム内のバッファに格納されているすべてのデータを強制的に書き込みます.

RETURN VALUE

0	:	正常終了
EOF(エラー)	:	ストリームが開かれていない場合
	:	入出力ストリームのオープンモードが不整合の場合
	:	上記以外の動作環境上の理由によって，強制出力に失敗した場合 ⟦例えば，ファイルディスクリプタによって参照されるストリームに十分な空きがない場合が該当します.⟧

EXAMPLE

次のコードは，文字列 c_sentence を%s 形式で標準出力ストリームに書き出した後，バッファの強制出力をしています.

```
stdstreamio s_io;
const char *c_sentence = "This is an example code for the USER";

if (s_io.printf("%s\n", c_sentence) < 0) {
    エラー処理
}

if (s_io.flush() < 0) {
    エラー処理
}
```

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください.

8.1.16 eof(), error(), reseterr()

NAME

eof(), error(), reseterr() — ストリームステータスのチェックとリセット

SYNOPSIS

```
int eof();
int error();
cstreamio &reseterr();
```

DESCRIPTION

`eof()` は，`open()` で開いたストリームにおいて EOF 指示子がセットされていれば，0 以外の数値を返します.

`error()` は，`open()` で開いたストリームにおいてエラー指示子がセットされていれば，0 以外の数値を返します.

`reseterr()` は，EOF 指示子とエラー指示子とをリセットし，自身の参照を返します.

これらのメンバ関数は，libc の `feof()`，`ferror()`，`clearerr()` と同様の動作をします.

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください.

8.1.17 seek()

NAME

seek(), rewind() — ストリームの位置を変更

SYNOPSIS

```
int seek( long offset, int whence );
int rewind();
```

DESCRIPTION

`seek()` は，`open()` で開いたストリームにおいて，ストリーム位置表示子をバイト単位で設定します．設定される位置は `whence` で指定された位置に `offset` バイトを加える事によって得られます.

`rewind()` は，`open()` で開いたストリームにおいて，ストリーム位置表示子をファイルの先頭にセットします.

PARAMETER

[I]	`offset`	ストリーム位置表示子のオフセット
[I]	`whence`	ストリーム位置表示子の基準位置 (`SEEK_SET`：ストリームの先頭, `SEEK_CUR`：現在の位置表示子，`SEEK_END`：ストリームの末尾)

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	ストリームがシークできない場合
	:	`whence` 引数が不適切である場合
	:	上記以外の動作環境上の理由によって，処理に失敗した場合 ⟦例えばカーネルのメモリが足りない等の場合が該当します.⟧

EXAMPLE

次のコードは，ディレクトリ *directory* に存在する読み込みモードで開かれたファイル *file.txt* の書き込み操作位置をファイルの先頭から 40 バイトに変更します.

```
stdstreamio f_in;

if (f_in.openf("r", "%s/%s", "directory", "file.txt") < 0) {
    エラー処理
}

if (f_in.seek(40, SEEK_SET) < 0) {
    エラー処理
}

f_in.close();
```

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください．

8.1.18 tell()

NAME

tell() — ファイル位置表示子の現時点での値を返す

SYNOPSIS

`long tell();`

DESCRIPTION

`tell()` は，`open()` で開いたストリームにおける，ストリーム位置表示子の現時点での値を返します．

RETURN VALUE

0 以上の値	:	現在のオフセット
負の値 (エラー)	:	ストリームがシークできない場合
	:	`whence` 引数が不適切である場合
	:	上記以外の動作環境上の理由によって，処理に失敗した場合 [[例えばカーネルのメモリが足りない等の場合が該当します.]]

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください．

8.1.19 is_seekable()

NAME

is_seekable() — ストリームがシーク可能かどうかを調べる

SYNOPSIS

`bool is_seekable();`

DESCRIPTION

`is_seekable()` は，`open()` で開いたストリームが，ディスクシーク可能かどうかを返します.

RETURN VALUE

true : ディスクシーク可能な場合
false : ディスクシーク不可能な場合

WARNING

cstreamio クラスは抽象クラスなので，ユーザが直接利用する事はできません．表 5 に示したクラスのメンバ関数として使用してください.

8.2 STDSTREAMIO クラス

stdstreamio クラスは，libc の printf() や fopen() と同様に，標準入出力，標準エラー出力および通常のファイル入出力を扱うためのクラスです．cstreamio を継承しているので，§8.1のメンバ関数は基本的にすべて利用可能ですが，open(const char *mode, cstreamio &sref) だけは利用できません．標準入出力，標準エラー出力に関しては，open()，close() をしなくても使用可能です．

stdstreamio クラスを使う場合は,「#include <sli/stdstreamio.h>」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は,「using namespace sli;」もコードに書いてください．表 8 はメンバ関数一覧です．表 8 の「対応する libc の関数」には，各メンバ関数と同じ機能を持つ libc の関数との対応を示しています．

	stdstreamio クラス	機能	対応する libc の関数
§8.2.2	open(), openf(), vopenf()	ストリームを開く	fopen()
§8.1.2	close()	ストリームを閉じる	fclose()
§8.1.3	read()	バイナリストリームの入力	fread()
§8.1.3	write()	バイナリストリームの出力	fwrite()
§8.1.4	bread()	バイナリストリームの入力 (endian 変換有)	—
§8.1.5	bwrite()	バイナリストリームの出力 (endian 変換有)	—
§8.1.6	rskip()	ストリームのシーク (可能な場合) または読み飛ばし	—
§8.1.7	wskip()	ストリームのシーク (可能な場合) または n バイトの空白キャラクタの書き込み	—
§8.1.8	getchr()	1 文字の入力	fgetc()
§8.1.9	getstr()	文字列の入力	fgets()
§8.1.10	getline()	1 行の入力	—
§8.1.11	scanf()	入力を書式で変換・引数に代入	fscanf()
§8.1.12	putchr()	1 文字の出力	fputc()
§8.1.13	putstr()	文字列の出力	fputs()
§8.1.14	printf()	引数の値を書式変換後に出力	fprintf()
§8.1.15	flush()	バッファの内容を強制出力	fflush()
§8.1.16	eof(), error(), reseterr()	ストリームステータスのチェックとリセット	feof(), etc.
§8.2.3	eprintf(...)	標準エラー出力への出力	fprintf(stderr, ...)
§8.2.4	eflush()	標準エラー出力の flush	fflush(stderr)
§8.2.5	seek()	ストリームの位置の変更	fseek()
§8.2.5	rewind()	ストリーム位置を先頭に変更	rewind()
§8.2.6	tell()	ストリーム位置表示子の値	ftell()
§8.1.19	is_seekable()	シーク可能かどうかを調べる	-

表 8: stdstreamio クラスで利用可能なメンバ関数一覧.

stdstreamio クラスで再定義されたメンバ関数および追加されたメンバ関数の使用法について，以下に記述します.

8.2.1 オブジェクトの作り方

stdstreamio クラスでは，次の例のようにオブジェクトを作る時に何も引数を与えないと，オブジェクト生成時の出力先は標準出力になります．

```
#include <sli/stdstreamio.h>
using namespace sli;

int main()
{
    stdstreamio sio;
    sio.printf("Hello\n");
```

オブジェクトを作る時に，次のようにフラグを与えると，オブジェクト生成時の出力先を標準エラー出力に切り替える事ができます．

```
    stdstreamio eout(true);
    eout.printf("This is STDERR\n");
```

8.2.2 open(), openf(), vopenf()

NAME

open(), openf(), vopenf() — ストリームを開く

SYNOPSIS

```
int open( const char *mode ); .............................................. 1
int open( const char *mode, int fd ); ...................................... 2
int open( const char *mode, const char *path ); ............................ 3
int openf( const char *mode, const char *path_fmt, ... ); ................. 4
int vopenf( const char *mode, const char *path_fmt, va_list ap ); ........ 5
```

DESCRIPTION

`path` あるいは `path_fmt` で示されたファイルを開くか，`fd` で指定されたディスクリプタをストリームに結びつけます．`path` あるいは `path_fmt` が `NULL` の場合は，標準入力または標準出力を使用します．

`mode` については，メンバ関数 1 およびメンバ関数 2 の場合，読み込みの時は`"r"`を，書き込みの時は`"w"`を指定します．また，メンバ関数 3〜5 の場合は，`"r"`，`"r+"`，`"w"`，`"w+"`，`"a"`または`"a+"`のいずれかを指定します．指定可能な `mode` の詳細は，次の表のとおりです．

mode	処理	ファイルが存在しない場合
"r"	読み込みのみ	異常終了
"r+"	読み込みおよび書き込み	異常終了
"w"	書き込みのみ	ファイルの新規作成
"w+"	読み込みおよび書き込み	ファイルの新規作成
"a"	追加書き込みのみ	ファイルの新規作成
"a+"	読み込みおよび追加書き込み	ファイルの新規作成

メンバ関数 4 およびメンバ関数 5 の詳細は，§8.1.1 の解説を参照してください．

PARAMETER

[I]	`mode`	ファイルを開くモード
[I]	`fd`	ファイルディスクリプタ
[I]	`path`	ファイル名
[I]	`path_fmt`	ファイル名のフォーマット指定
[I]	`...`	ファイル名の各要素データ
[I]	`ap`	ファイル名の可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	ファイルが存在しない等の理由で，ストリームのオープンに失敗した場合
	:	指定した `mode` が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	ストリームが，指定のモードでのアクセスを許されていない等の理由で，オープンに失敗した場合
	:	`path_fmt` のパスを示す文字列が `PATH_MAX` を超えている場合
	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合

EXCEPTION

標準入出力および標準エラー出力のファイル・ディスクリプタの複製に失敗した場合

EXAMPLE

次のコードは，ディレクトリ *directory* に存在するファイル *file.txt* を読み込みモードで開きます.

```
stdstreamio f_in;

if (f_in.openf("r", "%s/%s", "directory", "file.txt") < 0) {
    エラー処理
}

f_in.close();
```

8.2.3 eprintf(), veprintf()

NAME

eprintf(), veprintf() — 整形された書式へ変換し，標準エラー出力へ出力

SYNOPSIS

```
int eprintf( const char *format, ... );
int veprintf( const char *format, va_list ap );
```

DESCRIPTION

標準エラー出力ストリームに，format 以降の引数に格納されている内容を，format の変換指定に従って書き込みます.

eprintf() は，各要素データを，veprintf() は，可変長引数のリスト ap を format の変換指定に応じて変換します.

format 中の %に続く変換文字とその機能は §8.1.14の表のとおりです．なお，%自体を出力したい場合は %% とします.

PARAMETER

[I]	format	書き込みフォーマット指定
[I]	...	書き込む各要素データ
[I]	ap	書き込む可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	書き込まれた文字数
負の値 (エラー)	:	動作環境上の理由によって，書き込みに失敗した場合

EXAMPLE

次のコードは，文字列 c_error を%s 形式で標準エラー出力に書き込みます.

```
stdstreamio s_io;
const char *c_error = "This example code doesn't work exactly";

if (s_io.eprintf("%s\n", c_error) < 0) {
    エラー処理
}
```

8.2.4 eflush()

NAME

eflush() — 標準エラー出力の内容を強制的に出力

SYNOPSIS

```
int eflush();
```

DESCRIPTION

eflush() は，標準エラー出力ストリーム内のバッファに格納されているすべてのデータを強制的に書き込みます.

RETURN VALUE

0	:	正常終了
EOF(エラー)	:	動作環境上の理由によって，強制出力に失敗した場合

EXAMPLE

次のコードは，文字列 c_error を%s 形式で標準エラー出力ストリームに書き出した後，バッファの強制出力をしています.

```
stdstreamio s_io;
const char *c_error = "This example code doesn't work exactly";

if (s_io.eprintf("%s\n", c_error) < 0) {
    エラー処理
}

if (s_io.eflush() < 0) {
    エラー処理
}
```

8.2.5 seek(), rewind()

NAME

seek(), rewind() — ストリームの位置を変更

SYNOPSIS

```
int seek( long offset, int whence );
int rewind();
```

DESCRIPTION

`seek()` は，`open()` で開いたストリームにおいて，ストリーム位置表示子をバイト単位で設定します．設定される位置は `whence` で指定された位置に `offset` バイトを加える事によって得られます．

`rewind()` は，`open()` で開いたストリームにおいて，ストリーム位置表示子をファイルの先頭にセットします．

PARAMETER

[I]	`offset`	ストリーム位置表示子のオフセット
[I]	`whence`	ストリーム位置表示子の基準位置 (`SEEK_SET`：ストリームの先頭, `SEEK_CUR`：現在の位置表示子，`SEEK_END`：ストリームの末尾)

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	ストリームがシークできない場合
	:	`whence` 引数が不適切である場合
	:	上記以外の動作環境上の理由によって，処理に失敗した場合 ⟦例えばカーネルのメモリが足りない等の場合が該当します.⟧

EXAMPLE

次のコードは，ディレクトリ *directory* に存在する読み込みモードで開かれたファイル *file.txt* の書き込み操作位置をファイルの先頭から 40 バイトに変更します．

```
stdstreamio f_in;
```

```
    if (f_in.openf("r", "%s/%s", "directory", "file.txt") < 0) {
        エラー処理
    }

    if (f_in.seek(40, SEEK_SET) < 0) {
        エラー処理
    }

    f_in.close();
```

8.2.6 tell()

NAME

tell() — ファイル位置表示子の現時点での値を返す

SYNOPSIS

`long tell();`

DESCRIPTION

`tell()` は，`open()` で開いたストリームにおける，ストリーム位置表示子の現時点での値を返します.

RETURN VALUE

0 以上の値	:	現在のオフセット
負の値 (エラー)	:	ストリームがシークできない場合
	:	`whence` 引数が不適切である場合
	:	上記以外の動作環境上の理由によって，処理に失敗した場合 [[例えばカーネルのメモリが足りない等の場合が該当します.]]

8.2.7 content_length()

NAME

content_length() — ストリームの長さを調べる

SYNOPSIS

`long long content_length() const;`

DESCRIPTION

`open()` で開いたストリームのバイト長を返します.

RETURN VALUE

非負の値	:	ストリームのバイト長
負の値 (エラー)	:	ストリームのバイト長が得られない場合

8.3 GZSTREAMIO クラス

gzstreamio クラスは，gzip 圧縮・伸長を行ないながら標準入出力と通常のファイル入出力を扱うためのクラスです．cstreamio を継承しているので，§8.1のメンバ関数はすべて利用可能です．gzstreamio クラスでは，必ず `open()`，`close()` を使う必要があります．

gzstreamio クラスを使う場合は,「`#include <sli/gzstreamio.h>`」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は,「`using namespace sli;`」もコードに書いてください．

表 9 はメンバ関数一覧です．表 9の「対応する libc の関数」には，各メンバ関数と同じ機能を持つ libc の関数との対応を示しています．

	gzstreamio クラス	機能	対応する libc の関数
§8.3.1	`open()`, `openf()`, `vopenf()`	ストリームを開く	`fopen()`
§8.1.2	`close()`	ストリームを閉じる	`fclose()`
§8.1.3	`read()`	バイナリストリームの入力	`fread()`
§8.1.3	`write()`	バイナリストリームの出力	`fwrite()`
§8.1.4	`bread()`	バイナリストリームの入力 (endian 変換有)	—
§8.1.5	`bwrite()`	バイナリストリームの出力 (endian 変換有)	—
§8.1.6	`rskip()`	ストリームの読み飛ばし	—
§8.1.7	`wskip()`	n バイトの空白キャラクタの書き込み	—
§8.1.8	`getchr()`	1 文字の入力	`fgetc()`
§8.1.9	`getstr()`	文字列の入力	`fgets()`
§8.1.10	`getline()`	1 行の入力	—
§8.1.11	`scanf()`	入力を書式で変換・引数に代入	`fscanf()`
§8.1.12	`putchr()`	1 文字の出力	`fputc()`
§8.1.13	`putstr()`	文字列の出力	`fputs()`
§8.1.14	`printf()`	引数の値を書式変換後に出力	`fprintf()`
§8.1.15	`flush()`	バッファの内容を強制出力	`fflush()`
§8.1.16	`eof()`, `error()`, `reseterr()`	ストリームステータスのチェックとリセット	`feof()`, etc.
§8.3.2	`sync()`	バッファの内容を強制出力	`fflush()`

表 9: gzstreamio クラスで利用可能なメンバ関数一覧.

gzstreamio クラスで再定義されたメンバ関数および追加されたメンバ関数の使用法について，以下に記述します．

8.3.1 open(), openf(), vopenf()

NAME

open(), openf(), vopenf() — ストリームを開く

SYNOPSIS

```
int open( const char *mode );  ............................................ 1
int open( const char *mode, int fd );  .................................... 2
int open( const char *mode, cstreamio &sref );  ........................... 3
int open( const char *mode, const char *path );  .......................... 4
int openf( const char *mode, const char *path_fmt, ... );  ................ 5
```

`int vopenf( const char *mode, const char *path_fmt, va_list ap );` 6

DESCRIPTION

`path` あるいは `path_fmt` で示された gzip フォーマットのファイルを開くか，`fd` で指定された gzip フォーマットのファイルディスクリプタ，あるいは `sref` で指定された cstreamio クラスの継承クラスのオブジェクトを gzip フォーマットのストリームに結びつけます．`path` あるいは `path_fmt` が `NULL` の場合は，標準入力または標準出力を使用します．

メンバ関数 1，2，4，5 または 6 の `mode` については，読み込みの時は`"rb"`を，書き込みの時は`"wb"`を指定します．このように，`"b"`を指定すると，強制的にバイナリモードにする事ができます．バイナリファイルを扱っている際に`"b"`を指定しないと，画像ファイルが壊れる等の問題が生じる可能性があるため，指定する事を強く推奨します．メンバ関数 3 の `mode` については，読み込みの時は`"r"`を，書き込みの時は`"w"`を指定して下さい．

書き込み時には，`mode` に圧縮レベルや圧縮手法を指定する事が可能です．この場合は，mode には”wb6f”のように `[fopen のモード] [圧縮レベル [圧縮手法]]` の形式で設定する事もできます．圧縮レベルと圧縮手法は次の表のとおりです．速度と圧縮率の間をとったデフォルトの圧縮レベル値には 6 が設定されています．

指定文字	圧縮レベル
0	圧縮なし
1	処理速度重視
⋮	
9	圧縮効率重視

指定文字	圧縮手法
f	フィルタデータ
h	ハフマン法だけを使った圧縮 (単語など同じバイナリ集合が多いデータ向き)
R	ランレンクス・エンコード (画像のような連続するバイナリが続くデータ向き)

メンバ関数 1 は，標準入出力を gzip フォーマットのファイルにリダイレクトするような場合に使います (EXAMPLE を参照).

メンバ関数 3 は，すでに cstreamio クラスの継承クラスのオブジェクトでオープンしたストリームがあった時，ストリームの途中から gzip の圧縮がかかったストリームに変更になるような場合に使います (§8.1.1の EXAMPLE-2 を参照).

メンバ関数 5 およびメンバ関数 6 の `path_fmt` 以降の引数の詳細は，§8.1.1の解説を参照してください．

PARAMETER

[I]	`mode`	ファイルを開くモード
[I]	`fd`	ファイルディスクリプタ
[I]	`sref`	cstreamio クラスの継承クラスのオブジェクト
[I]	`path`	ファイル名
[I]	`path_fmt`	ファイル名のフォーマット指定
[I]	`...`	ファイル名の各要素データ
[I]	`ap`	ファイル名の全要素データ

([I] : 入力，[O] : 出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	ファイルが存在しない等の理由で，ストリームのオープンに失敗した場合
	:	指定した mode が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	指定した mode と fd の関係が正しくない等の理由で，オープンに失敗した場合 (メンバ関数 2)
	:	ストリームが，指定のモードでのアクセスを許されていない等の理由で，オープンに失敗した場合
	:	path_fmt のパスを示す文字列が PATH_MAX を超えている場合
	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合
	:	サポートしていない圧縮メソッドを指定している等の理由で，ストリームのオープンに失敗した場合
	:	予期せずストリームの終端に達した場合

EXCEPTION

標準入出力のファイル・ディスクリプタの複製に失敗した場合 (メンバ関数 1,4)
圧縮アルゴリズム用の一時バッファの確保に失敗した場合 (メンバ関数 3)
圧縮アルゴリズムの初期化に失敗した場合 (メンバ関数 3)
出力バッファの確保に失敗した場合 (メンバ関数 3)

EXAMPLE

次のコードは，標準入力を gzip フォーマットのファイル *file.txt.gz* にリダイレクトし，読み込みモードで開き，その内容を標準出力します．このコードの実行ファイルを *gzstreamio_open* とする場合は，下記のように実行して下さい．

```
$ ./gzstreamio_open < file.txt.gz
```

```
#include <sli/gzstreamio.h>
using namespace sli;

int main()
{
    gzstreamio s_io;
    const char *line_ptr;

    if (s_io.open("rb") < 0) {
         エラー処理
    }

    while ((line_ptr = s_io.getline()) != NULL) {
        printf("%s", line_ptr);
    }

    s_io.close();

    return 0;

}
```

WARNING

gzip フォーマットでないファイルをオープンする事もできますが，書き込みおよび読み込みは意図した処理を行なわない可能性があります.

8.3.2 sync()

NAME

sync() — 内容を強制的に出力し，出力をバイト境界で終わるように整える

SYNOPSIS

`int sync();`

DESCRIPTION

`flush()` の場合 (§8.1.15)，出力が必ずしもバイト境界で終端しているとは限りません．`sync()` は，`flush()` を呼び，さらに出力をバイト境界で終えるように整えます．当然ですが，あまりこのメンバ関数を頻繁に使うと，圧縮率が低下します．ほとんどの用途では，`sync()` メンバ関数を必要とする事はないでしょう.

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	gzip 圧縮に失敗した場合

EXAMPLE

次のコードは，文字列 `c_sentence` を gzip 圧縮したバイナリデータをディレクトリ *directory* に存在するファイル *file.txt.gz* に書き出します.

```
gzstreamio f_out;
const char *c_sentence = "This is an example code for the USER";

if (f_out.openf("wb", "%s/%s", "directory", "file.txt.gz") < 0) {
    エラー処理
}

if (f_out.printf("%s", c_sentence) < 0) {
    エラー処理
}

if (f_out.sync() < 0) {
    エラー処理
}

f_out.close();
```

8.4 BZSTREAMIO クラス

bzstreamio クラスは，bzip2 圧縮・伸長を行ないながら標準入出力と通常のファイル入出力を扱うためのクラスです．cstreamio を継承しているので，§8.1のメンバ関数はすべて利用可能です．bzstreamio クラスでは，必ず `open()`，`close()` を使う必要があります．

bzstreamio クラスを使う場合は，「`#include <sli/bzstreamio.h>`」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は，「`using namespace sli;`」もコードに書いてください．

表 10 はメンバ関数一覧です．表 10 の「対応する libc の関数」には，各メンバ関数と同じ機能を持つ libc の関数との対応を示しています．

bzip2 は，処理速度の点で gzip よりも劣っていますが，より高い圧縮率のデータ圧縮アルゴリズムです．

	bzstreamio クラス	機能	対応する libc の関数
§8.4.1	`open()`, `openf()`, `vopenf()`	ストリームを開く	`fopen()`
§8.1.2	`close()`	ストリームを閉じる	`fclose()`
§8.1.3	`read()`	バイナリストリームの入力	`fread()`
§8.1.3	`write()`	バイナリストリームの出力	`fwrite()`
§8.1.4	`bread()`	バイナリストリームの入力 (endian 変換有)	—
§8.1.5	`bwrite()`	バイナリストリームの出力 (endian 変換有)	—
§8.1.6	`rskip()`	ストリームの読み飛ばし	—
§8.1.7	`wskip()`	n バイトの空白キャラクタの書き込み	—
§8.1.8	`getchr()`	1 文字の入力	`fgetc()`
§8.1.9	`getstr()`	文字列の入力	`fgets()`
§8.1.10	`getline()`	1 行の入力	—
§8.1.11	`scanf()`	入力を書式で変換・引数に代入	`fscanf()`
§8.1.12	`putchr()`	1 文字の出力	`fputc()`
§8.1.13	`putstr()`	文字列の出力	`fputs()`
§8.1.14	`printf()`	引数の値を書式変換後に出力	`fprintf()`
§8.1.15	`flush()`	バッファの内容を強制出力	`fflush()`
§8.1.16	`eof()`, `error()`, `reseterr()`	ストリームステータスのチェックとリセット	`feof()`, etc.

表 10: bzstreamio クラスで利用可能なメンバ関数一覧．

bzstreamio クラスで再定義されたメンバ関数および追加されたメンバ関数の使用法について，以下に記述します．

8.4.1 open(), openf(), vopenf()

NAME

open(), openf(), vopenf() — ストリームを開く

SYNOPSIS

```
int open( const char *mode );  ............................................ 1
int open( const char *mode, int fd );  .................................... 2
int open( const char *mode, cstreamio &sref );  ........................... 3
int open( const char *mode, const char *path );  .......................... 4
```

```
int openf( const char *mode, const char *path_fmt, ... );  ................. 5
int vopenf( const char *mode, const char *path_fmt, va_list ap );  ....... 6
```

DESCRIPTION

path あるいは path_fmt で示された bzip2 フォーマットのファイルを開くか，fd で指定された bzip2 フォーマットのファイルディスクリプタ，あるいは sref で指定された cstreamio クラスの継承クラスのオブジェクトを bzip2 フォーマットのストリームに結びつけます．path あるいは path_fmt が NULL の場合は，標準入力または標準出力を使用します．

メンバ関数 1，2，4，5 または 6 の mode については，読み込みの時は"rb"を，書き込みの時は"wb"を指定します．このように，"b"を指定すると，強制的にバイナリモードにする事ができます．バイナリファイルを扱っている際に"b"を指定しないと，画像ファイルがこわれる等の問題が生じる可能性があるため，指定する事を強く推奨します．メンバ関数 3 の mode については，読み込みの時は"r"を，書き込みの時は"w"を指定して下さい．

メンバ関数 1 は，標準入出力を bzip2 フォーマットのファイルにリダイレクトするような場合に使います (EXAMPLE-1 を参照)．

メンバ関数 3 は，すでに cstreamio クラスの継承クラスのオブジェクトでオープンしたストリームがあった時，ストリームの途中から bzip2 の圧縮がかかったストリームに変更になるような場合に使います (§8.1.1の EXAMPLE-2 を参照)．

メンバ関数 5 およびメンバ関数 6 の path_fmt 以降の引数の詳細は，§8.1.1の解説を参照してください．

PARAMETER

[I]	mode	ファイルを開くモード
[I]	fd	ファイルディスクリプタ
[I]	sref	cstreamio クラスの継承クラスのオブジェクト
[I]	path	ファイル名
[I]	path_fmt	ファイル名のフォーマット指定
[I]	...	ファイル名の各要素データ
[I]	ap	ファイル名の全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	ファイルが存在しない等の理由で，ストリームのオープンに失敗した場合
	:	指定した mode が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	指定した mode と fd の関係が正しくない等の理由で，オープンに失敗した場合 (メンバ関数 2)
	:	ストリームが，指定のモードでのアクセスを許されていない等の理由で，オープンに失敗した場合
	:	path_fmt のパスを示す文字列が PATH_MAX を超えている場合
	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合

EXCEPTION

標準入出力のファイル・ディスクリプタの複製に失敗した場合 (メンバ関数 1,4)
圧縮アルゴリズム用の一時バッファの確保に失敗した場合 (メンバ関数 3)
圧縮アルゴリズムの初期化に失敗した場合 (メンバ関数 3)
出力バッファの確保に失敗した場合 (メンバ関数 3)

EXAMPLE-1

次のコードは，標準入力を bzip2 フォーマットのファイル *file.txt.bz2* にリダイレクトし，読み込みモードで開き，その内容を標準出力します．このコードの実行ファイルを *bzstreamio_open* とする場合は，下記のように実行して下さい．

```
$ ./bzstreamio_open < file.txt.bz2
```

```
#include <sli/bzstreamio.h>
using namespace sli;

int main()
{
    bzstreamio s_io;
    const char *line_ptr;

    if (s_io.open("rb") < 0) {
        エラー処理
    }

    while ((line_ptr = s_io.getline()) != NULL) {
        printf("%s", line_ptr);
    }

    s_io.close();

    return 0;
}
```

EXAMPLE-2

次のコードは，ディレクトリ *directory* に存在する bzip2 フォーマットのファイル *file.txt.bz2* を読み込み及びバイナリーモードで開き，その内容を標準出力します．

```
bzstreamio f_in;
const char *line_ptr;

if (f_in.open("rb", "directory/file.txt.bz2") < 0) {
    エラー処理
}

while ((line_ptr = f_in.getline()) != NULL) {
```

```
        printf("%s", line_ptr);
    }

    f_in.close();
```

WARNING

bzip2 フォーマットでないファイルをオープンする事もできますが，書き込みおよび読み込みは意図した処理を行なわない可能性があります.

8.5 HTTPSTREAMIO クラス

httpstreamio クラスは，http サーバの GET メソッドを使い，http サーバからストリーム入力を行なうクラスです．gzip 伸長または bzip2 伸長を行ないながら入力する事もできます．

cstreamio を継承していますが，§8.1のメンバ関数の一部は利用できません．httpstreamio クラスの利用可能なメンバ関数は表 11のメンバ関数一覧を参照して下さい．httpstreamio クラスでは，必ず `open()`，`close()` を使う必要があります．`open()` メンバ関数では，`mode` には`"r"`か`"r%"`を，`path` に `http://`から始まる URL を指定します．プロキシサーバを介した接続はサポートしていません．

httpstreamio クラスを使う場合は,「`#include <sli/httpstreamio.h>`」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は,「`using namespace sli;`」もコードに書いてください．

表 11 の「対応する libc の関数」には，各メンバ関数と同じ機能を持つ libc の関数との対応を示しています．

	httpstreamio クラス	機能	対応する libc の関数
§8.5.1	`open()`, `openf()`, `vopenf()`	ストリームを開く	`fopen()`
§8.1.2	`close()`	ストリームを閉じる	`fclose()`
§8.1.3	`read()`	バイナリストリームの入力	`fread()`
§8.1.4	`bread()`	バイナリストリームの入力 (endian 変換有)	—
§8.1.6	`rskip()`	ストリームの読み飛ばし	—
§8.1.8	`getchr()`	1 文字の入力	`fgetc()`
§8.1.9	`getstr()`	文字列の入力	`fgets()`
§8.1.10	`getline()`	1 行の入力	—
§8.1.11	`scanf()`	入力を書式で変換・引数に代入	`fscanf()`
§8.1.16	`eof()`, `error()`, `reseterr()`	ストリームステータスのチェックとリセット	`feof()`, etc.
§8.5.2	`content_length()`	ストリームの長さを調べる	—
§8.5.3	`user_agent().assign()`	ユーザエージェントを設定	—

表 11: httpstreamio クラスで利用可能なメンバ関数一覧.

httpstreamio クラスで再定義されたメンバ関数および追加されたメンバ関数の使用法について，以下に記述します．

8.5.1 open(), openf(), vopenf()

NAME

open(), openf(), vopenf() — ストリームを開く

SYNOPSIS

```
int open( const char *mode, const char *path );
int openf( const char *mode, const char *path_fmt, ... );
int vopenf( const char *mode, const char *path_fmt, va_list ap );
```

DESCRIPTION

`path` あるいは `path_fmt` で示された URL を開きます．`mode` には`"r"`または`"r%"`を指定します．`mode` に`"r%"`が指定された場合，必要に応じて gzip または bzip2 で圧縮されたストリームを読

み出し時に伸長します．gzip・bzip2 のどちらを用いるかは，サーバの返すヘッダ情報 (MIME) と path あるいは path_fmt のファイル名のサフィックス (「.gz」か「.bz2」か) で決まります．

openf() および vopenf() の path_fmt 以降の引数の詳細は，§8.1.1の解説を参照してください．

PARAMETER

[I]	mode	URL を開くモード
[I]	path	URL のパス
[I]	path_fmt	URL のフォーマット指定
[I]	...	URL の各要素データ
[I]	ap	URL の全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	指定した URL が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	mode が指定されていない等の理由で，ストリームのオープンに失敗した場合
	:	指定した mode が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	ストリームが，指定のモードでのアクセスを許されていない等の理由で，オープンに失敗した場合
	:	path_fmt のパスを示す文字列が PATH_MAX を超えている場合
	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合
	:	gzip または bzip2 フォーマットのストリームのオープンに失敗した場合

EXCEPTION

メモリの割り当てに失敗した場合

EXAMPLE

次のコードは，URL *http://www.jaxa.jp/*を読み込みモードで開き，リンクされているソースコード (html) の内容を標準出力します．

```
httpstreamio net_in;
const char *line_ptr;

if (net_in.open("r", "http://www.jaxa.jp/") < 0) {
    エラー処理
}

while ((line_ptr = net_in.getline()) != NULL) {
    printf("%s", line_ptr);
}

net_in.close();
```

8.5.2 content_length()

NAME

content_length() — ストリームの長さを調べる

SYNOPSIS

```
long long content_length() const;
```

DESCRIPTION

open() で開いたストリームのバイト長を返します．圧縮されたストリームの場合，圧縮された状態でのバイト長を返します.

RETURN VALUE

非負の値	:	ストリームのバイト長
負の値 (エラー)	:	MIME ヘッダに Content-Length の情報が存在しない場合

EXAMPLE

次のコードは，URL *http://www.jaxa.jp/*を読み込みモードで開き, そのストリームのバイト長を標準出力します.

```
httpstreamio net_in;
long long l_ret;

if (net_in.open("r", "http://www.jaxa.jp/") < 0) {
    エラー処理
}

if ((l_ret = net_in.content_length()) < 0) {
    printf("No information about stream byte size\n");
}
else {
    printf("Stream Byte Size = %lld \n", l_ret);
}

net_in.close();
```

8.5.3 user_agent().assign()

NAME

user_agent().assign() — ユーザエージェントを設定

SYNOPSIS

```
tstring &user_agent().assign( const char *uagent );
tstring &user_agent().assignf( const char *uagent_fmt, ... );
```

DESCRIPTION

Web サーバとの接続時に送信されるユーザエージェントを設定します．未設定の場合,「ホスト名 SLLIB-x.x::httpstreamio」が送信されます.

8.6 FTPSTREAMIO クラス

ftpstreamio クラスは，ftp サーバに接続し，パッシブモードで ftp サーバからストリーム入力，あるいは ftp サーバへストリーム出力を行なうクラスです．gzip 伸長・圧縮または bzip2 伸長・圧縮を行ないながら入力・出力する事もできます．

cstreamio を継承しているので，§8.1のメンバ関数は基本的にすべて利用可能ですが，`open( const char *mode, cstreamio &sref )` だけは利用できません．ftpstreamio クラスでは，必ず `open()`，`close()` を使う必要があります．`open()` メンバ関数では，`mode` には`"r"`, `"r%"`, `"w"` または`"w%"`のいずれか，`path` に `ftp://`から始まる URL を指定します．プロキシサーバを介した接続はサポートしていません．

ftpstreamio クラスを使う場合は,「`#include <sli/ftpstreamio.h>`」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は,「`using namespace sli;`」もコードに書いてください．

表 12 はメンバ関数一覧です．表 12 の「対応する libc の関数」には，各メンバ関数と同じ機能を持つ libc の関数との対応を示しています．

	ftpstreamio クラス	機能	対応する libc の関数
§8.6.1	`open()`, `openf()`, `vopenf()`	ストリームを開く	`fopen()`
§8.1.2	`close()`	ストリームを閉じる	`fclose()`
§8.1.3	`read()`	バイナリストリームの入力	`fread()`
§8.1.3	`write()`	バイナリストリームの出力	`fwrite()`
§8.1.4	`bread()`	バイナリストリームの入力 (endian 変換有)	—
§8.1.5	`bwrite()`	バイナリストリームの出力 (endian 変換有)	—
§8.1.6	`rskip()`	ストリームの読み飛ばし	—
§8.1.7	`wskip()`	n バイトの空白キャラクタの書き込み	—
§8.1.8	`getchr()`	1 文字の入力	`fgetc()`
§8.1.9	`getstr()`	文字列の入力	`fgets()`
§8.1.10	`getline()`	1 行の入力	—
§8.1.11	`scanf()`	入力を書式で変換・引数に代入	`fscanf()`
§8.1.12	`putchr()`	1 文字の出力	`fputc()`
§8.1.13	`putstr()`	文字列の出力	`fputs()`
§8.1.14	`printf()`	引数の値を書式変換後に出力	`fprintf()`
§8.1.15	`flush()`	バッファの内容を強制出力	`fflush()`
§8.1.16	`eof()`, `error()`, `reseterr()`	ストリームステータスのチェックとリセット	`feof()`, etc.
§8.6.2	`content_length()`	ストリームの長さを調べる	—
§8.6.3	`username().assign()`	FTP ユーザの名前を設定	—
§8.6.4	`password().assign()`	FTP ユーザのパスワードを設定	—

表 12: ftpstreamio クラスで利用可能なメンバ関数一覧.

httpstreamio クラスで再定義されたメンバ関数および追加されたメンバ関数の使用法について，以下に記述します．

8.6.1 open(), openf(), vopenf()

NAME

open(), openf(), vopenf() — ストリームを開く

SYNOPSIS

```
int open( const char *mode, const char *path );
int openf( const char *mode, const char *path_fmt, ... );
int vopenf( const char *mode, const char *path_fmt, va_list ap );
```

DESCRIPTION

path あるいは path_fmt で示された URL を開きます．path は必ず ftp://で始まらなくてはなりません．ユーザ名とパスワードを path に含める事ができます．その場合は，ftp://ユーザ名:パスワード@ftp.com/··· の書式で設定します．path にユーザ名とパスワードが含まれない場合は，username().assign()，password().assign() (§8.6.3 〜) でも設定できます．ユーザ名とパスワードが設定されていない場合は，匿名で接続します．

mode には，"r"，"r%"，"w"または"w%"のいずれかを指定します．mode に"r%"が指定された場合，必要に応じて gzip または bzip2 で圧縮されたストリームを読み出し時に伸長します．mode に"w%"が指定された場合，必要に応じて gzip または bzip2 で圧縮してから ftp サーバに送信します．gzip･bzip2 のどちらを用いるかは，path のファイル名のサフィックス (「.gz」か「.bz2」か) で決まります．

openf() および vopenf() の path_fmt 以降の引数の詳細は，§8.1.1の解説を参照してください．

PARAMETER

[I]	mode	URL を開くモード
[I]	path	URL のパス
[I]	path_fmt	URL のフォーマット指定
[I]	...	URL の各要素データ
[I]	ap	URL の全要素データ

([I] : 入力，[O] : 出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	URL が指定されていない等の理由で，ストリームのオープンに失敗した場合
	:	指定した URL が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	指定した `mode` が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	`path_fmt` のパスを示す文字列が `PATH_MAX` を超えている場合
	:	`mode` が指定されていない等の理由で，ストリームのオープンに失敗した場合
	:	ストリームが，指定のモードでのアクセスを許されていない等の理由で，オープンに失敗した場合
	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合
	:	ftp サーバとの接続に失敗した等の理由で，ストリームのオープンに失敗した場合
	:	gzip または bzip2 フォーマットのストリームのオープンに失敗した場合

EXCEPTION

メモリの割り当てに失敗した場合

EXAMPLE

次のコードは，URL *ftp://ftp.boof.com/file.txt* を読み込みモードで開き，その内容を標準出力します．なお，左記の URL は実在のものではありませんので，このままでは open() は異常終了します．URL を実在するものに設定する事で正常に動作します．

```
ftpstreamio f_in;
const char *line_ptr;

if (f_in.open("r", "ftp://ftp.boof.com/file.txt") < 0) {
    エラー処理
}

while ((line_ptr = f_in.getline()) != NULL) {
    printf("%s", line_ptr);
}

f_in.close();
```

8.6.2 content_length()

NAME

content_length() — ストリームの長さを調べる

SYNOPSIS

```
long long content_length() const;
```

DESCRIPTION

open() の読み込みモード指定で開いたストリームの場合，ストリームのバイト長を返します．open() で開いた，圧縮されたストリームの場合は，圧縮された状態でのバイト長を返します．

RETURN VALUE

非負の値	:	ストリームのバイト長
負の値 (エラー)	:	ストリームがオープンされていない場合

EXAMPLE

次のコードは，URL *ftp://ftp.boof.com/file.txt* を読み込みモードで開き, そのストリームのバイト長を標準出力します．なお，左記の URL は実在のものではありませんので，このままでは open() は異常終了します．URL を実在するものに設定する事で正常に動作します．

```
ftpstreamio f_in;
long long l_ret;

if (f_in.open("r", "ftp://ftp.boof.com/file.txt") < 0) {
    エラー処理
}

if ((l_ret = f_in.content_length()) < 0) {
    printf("No information about stream byte size\n");
}
else {
    printf("Stream Byte Size = %lld \n", l_ret);
}

f_in.close();
```

8.6.3 username().assign()

NAME

username().assign() — FTP ユーザの名前を設定

SYNOPSIS

```
tstring &username().assign( const char *user );
```

DESCRIPTION

デフォルト状態では open() メンバ関数 (§8.6.1) で FTP サーバに anonymous でログインしますが，このメンバ関数を使ってユーザ名を設定できます．

8.6.4 password().assign()

NAME

password().assign() — FTP ユーザのパスワードを設定

SYNOPSIS

```
tstring &password().assign( const char *pass );
```

DESCRIPTION

username().assign() で設定した FTP ユーザのパスワードを設定します.

8.7 PIPESTREAMIO クラス

pipestreamio クラスは，ファイルを実行し，その実行したプロセスとパイプでつなぎ，ストリームを入力または出力するためのクラスです．cstreamio を継承しているので，§8.1のメンバ関数は基本的にすべて利用可能ですが，`open( const char *mode, cstreamio &sref )` だけは利用できません．pipestreamio クラスでは，必ず `open()`，`close()` を使う必要があります．`open()` メンバ関数では，`mode` に`"r"`か`"w"`を，`path` にコマンドを指定します．mode が`"r"`の時は実行したプロセスからの入力，`"w"`の時は実行したプロセスへの出力を意味します．`path` にはコマンドへのオプションやパイプやリダイレクトの記号 (|,<,>) を含む事ができます．libc の `execvp()` と同様に `char *const argv[]` の引数でコマンドの指定も可能な `open()` メンバ関数も用意されています．

pipestreamio クラスを使う場合は,「`#include <sli/pipestreamio.h>`」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は,「`using namespace sli;`」もコードに書いてください．

表 13 の「対応する libc の関数」には，各メンバ関数と同じ機能を持つ libc の関数との対応を示しています．

	pipestreamio クラス	機能	対応する libc の関数
§8.7.1	`open()`, `openf()`, `vopenf()`	ストリームを開く	`fopen()`
§8.1.2	`close()`	ストリームを閉じる	`fclose()`
§8.1.3	`read()`	バイナリストリームの入力	`fread()`
§8.1.3	`write()`	バイナリストリームの出力	`fwrite()`
§8.1.4	`bread()`	バイナリストリームの入力 (endian 変換有)	—
§8.1.5	`bwrite()`	バイナリストリームの出力 (endian 変換有)	—
§8.1.6	`rskip()`	ストリームの読み飛ばし	—
§8.1.7	`wskip()`	n バイトの空白キャラクタの書き込み	—
§8.1.8	`getchr()`	1 文字の入力	`fgetc()`
§8.1.9	`getstr()`	文字列の入力	`fgets()`
§8.1.10	`getline()`	1 行の入力	—
§8.1.11	`scanf()`	入力を書式で変換・引数に代入	`fscanf()`
§8.1.12	`putchr()`	1 文字の出力	`fputc()`
§8.1.13	`putstr()`	文字列の出力	`fputs()`
§8.1.14	`printf()`	引数の値を書式変換後に出力	`fprintf()`
§8.1.15	`flush()`	バッファの内容を強制出力	`fflush()`
§8.1.16	`eof()`, `error()`, `reseterr()`	ストリームステータスのチェックとリセット	`feof()`, etc.

表 13: pipestreamio クラスで利用可能なメンバ関数一覧.

pipestreamio クラスで再定義されたメンバ関数および追加されたメンバ関数の使用法について，以下に記述します.

8.7.1 open(), openf(), vopenf()

NAME

open(), openf(), vopenf() — ストリームを開く

SYNOPSIS

```
int open( const char *mode );  ............................................. 1
int open( const char *mode, int fd );  ..................................... 2
int open( const char *mode, const char *path );  ........................... 3
int open( const char *mode, const char *const argv[] );  ................... 4
int openf( const char *mode, const char *path_fmt, ... );  ................. 5
int vopenf( const char *mode, const char *path_fmt, va_list ap );  ....... 6
```

DESCRIPTION

メンバ関数3〜6は，path，argvあるいはpath_fmtで示されたコマンドを実行し，パイプで接続して，ストリームを開きます．パイプは，単方向のプロセス間通信チャンネルを提供し，「読み出し側」と「書き込み側」があります．パイプの書き込み側で書き込まれたデータは，パイプの読み出し側から読み出す事ができます．modeが"r" の時は，指定したコマンドを実行し，そのコマンドの標準出力からの出力を，オブジェクトの読み出し用メンバ関数で読めるようにパイプ接続します(EXAMPLE-1およびEXAMPLE-2を参照)．modeが"w"の時は，指定したコマンドを実行し，オブジェクトの書き出し用メンバ関数で書き出した結果を，そのコマンドの標準入力から読めるようにパイプ接続します(EXAMPLE-3を参照)．

pathあるいはpath_fmtが設定された場合,「/bin/sh -c コマンド」の形で実行しますので，pathあるいはpath_fmtにはパイプやリダイレクトの記号(|,<,>)を含む事ができます．pathあるいはpath_fmtがNULLの場合は，標準入力または標準出力を使用します．

argv[]の各要素には,「実行ファイルのパス名 引数1 引数2 ··· NULL」の順で設定して下さい．argv[]の最後はNULLになっている必要があります．

メンバ関数1およびメンバ関数2は，コマンドを実行し，パイプで接続して，ストリームをオープンする事はできません．これらのメンバ関数の詳細は，§8.1.1の解説を参照してください．

メンバ関数5およびメンバ関数6のpath_fmt以降の引数の詳細は，§8.1.1の解説を参照してください．

PARAMETER

[I]	mode	ストリームを開くモード
[I]	fd	ファイルディスクリプタ
[I]	path	コマンド
[I]	path_fmt	コマンドのフォーマット指定
[I]	...	コマンドの各要素データ
[I]	ap	コマンドの全要素データ
[I]	argv[]	コマンドの全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	コマンドが指定されていない等の理由で，ストリームのオープンに失敗した場合
	:	指定した mode が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	ストリームが，指定のモードでのアクセスを許されていない等の理由で，オープンに失敗した場合
	:	path_fmt のパスを示す文字列が PATH_MAX を超えている場合
	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合

EXCEPTION

読み出し用および書き込み用のファイル・ディスクリプタの生成に失敗した場合
プロセスの生成に失敗した場合

EXAMPLE-1

次のコードは，関数 fprintf の man ページを整形し，書き込みモードで開かれた出力ファイルにその内容を書き込みます.

```
stdstreamio  f_out;
pipestreamio p_in;
const char  *argv[4] = { "man", "fprintf", NULL };
const char  *line_ptr;

f_out.openf("w", "%s", "file.txt");

if (p_in.open("r", argv) < 0) {
    エラー処理
}

while ((line_ptr = p_in.getline()) != NULL) {
    f_out.printf("%s",line_ptr);
}

p_in.close();
f_out.close();
```

EXAMPLE-2

次のコードは，ディレクトリ *directory* に存在するファイル *file.txt* の内容を 2 番目のフィールドをキーとして行単位で並べ替え，標準出力します.

```
pipestreamio p_in;
const char *line_ptr;

if (p_in.open("r", "cat directory/file.txt | sort -k 2") < 0) {
    エラー処理
```

```
    }

    while ((line_ptr = p_in.getline()) != NULL) {
        printf("%s",line_ptr);
    }

    p_in.close();
```

EXAMPLE-3

次のコードは，ディレクトリ *directory* に存在するファイル *file.txt* の内容を 1 行ずつ読み込みます．読み込まれたそれぞれの内容に対して，文字列 *pattern* に一致する部分を含む行を標準出力します．

```
    stdstreamio f_in;
    pipestreamio p_out;
    const char *line_ptr;

    if (f_in.open("r", "directory/file.txt") < 0) {
        エラー処理
    }

    if (p_out.open("w", "grep pattern") < 0) {
        エラー処理
    }

    while ((line_ptr = f_in.getline()) != NULL) {
        p_out.printf("%s",line_ptr);
    }

    p_out.close();
    f_in.close();
```

8.8 DIGESTSTREAMIO クラス

digeststreamio クラスは，open() メンバ関数の path 引数で示された URL またはファイルを，必要に応じて gzip または bzip2 圧縮・伸長を行ないながらストリーム入出力を扱うためのクラスです．さらに，このクラスで追加された openp() メンバ関数では，それらに加えてパイプを使ってコマンドから入力・コマンドへ出力する事もできます．従って，digeststreamio クラスは cstreamio の継承クラスの中での万能型クラスといえます．サーバの返すヘッダ情報 (MIME) と open() に与えられた path のファイル名のサフィックスで，圧縮・伸長が必要かどうかを判断します．cstreamio を継承しているので，§8.1のメンバ関数は基本的にすべて利用可能ですが，open(const char *mode, cstreamio &sref) だけは利用できません．digeststreamio クラスでは，必ず open()，close() を使う必要があります．open() メンバ関数の mode には必ず"r"か"w"を指定します．http サーバへの出力は，現在は未対応です．プロキシサーバを介した接続はサポートしていません．

digeststreamio クラスを使う場合は，「#include <sli/digeststreamio.h>」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は，「using namespace sli;」もコードに書いてください．

表 14 はメンバ関数一覧です．表 14の「対応する libc の関数」には，各メンバ関数と同じ機能を持つ libc の関数との対応を示しています．

	digeststreamio クラス	機能	対応する libc の関数
§8.8.1	open(), openf(), vopenf()	ストリームを開く	fopen()
§8.8.2	openp(), openpf(), openpf()	ストリームを開く (perl ライク)	—
§8.1.2	close()	ストリームを閉じる	fclose()
§8.1.3	read()	バイナリストリームの入力	fread()
§8.1.3	write()	バイナリストリームの出力	fwrite()
§8.1.4	bread()	バイナリストリームの入力 (endian 変換有)	—
§8.1.5	bwrite()	バイナリストリームの出力 (endian 変換有)	—
§8.1.6	rskip()	ストリームのシーク (可能な場合) または読み飛ばし	—
§8.1.7	wskip()	ストリームのシーク (可能な場合) または n バイトの空白キャラクタの書き込み	—
§8.1.8	getchr()	1 文字の入力	fgetc()
§8.1.9	getstr()	文字列の入力	fgets()
§8.1.10	getline()	1 行の入力	—
§8.1.11	scanf()	入力を書式で変換・引数に代入	fscanf()
§8.1.12	putchr()	1 文字の出力	fputc()
§8.1.13	putstr()	文字列の出力	fputs()
§8.1.14	printf()	引数の値を書式変換後に出力	fprintf()
§8.1.15	flush()	バッファの内容を強制出力	fflush()
§8.1.16	eof(), error(), reseterr()	ストリームステータスのチェックとリセット	feof(), etc.

表 14: digeststreamio クラスで利用可能なメンバ関数一覧.

digeststreamio クラスで再定義されたメンバ関数および追加されたメンバ関数の使用法について，以下に記述します．

8.8.1 open(), openf(), vopenf()

NAME

open(), openf(), vopenf() — ストリームを開く

SYNOPSIS

```
int open( const char *mode ); ........................................... 1
int open( const char *mode, int fd ); ................................... 2
int open( const char *mode, const char *path ); ......................... 3
int openf( const char *mode, const char *path_fmt, ... ); ............... 4
int vopenf( const char *mode, const char *path_fmt, va_list ap ); ....... 5
```

DESCRIPTION

path あるいは path_fmt で示されたファイルまたは URL をオープンします．URL は，file://，http://，ftp:// で始まる文字列を指定する事ができます．path あるいは path_fmt が NULL の場合は，標準入力または標準出力を使用します．

mode が"r"の時は入力，"w"の時は出力を行ないます．メンバ関数 1 およびメンバ関数 2 は，コマンドを実行し，パイプで接続して，ストリームをオープンすることは出来ません．これらのメンバ関数の詳細は，§8.1.1の解説を参照してください．

メンバ関数 4 およびメンバ関数 5 の path_fmt 以降の引数の詳細は，§8.1.1の解説を参照してください．

PARAMETER

[I]	mode	ファイルまたは URL を開くモード
[I]	fd	ファイルディスクリプタ
[I]	path	ファイルまたは URL
[I]	path_fmt	ファイルまたは URL のフォーマット指定
[I]	...	ファイルまたは URL の各要素データ
[I]	ap	ファイルまたは URL の全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	指定した mode が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	指定した mode と fd の関係が正しくない等の理由で，オープンに失敗した場合 (メンバ関数 2)
	:	ストリームが，指定のモードでのアクセスを許されていない等の理由で，オープンに失敗した場合
	:	path_fmt のパスを示す文字列が PATH_MAX を超えている場合
	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合
	:	ファイルまたは URL のオープンに失敗した場合

EXCEPTION

オブジェクトの生成に失敗した場合

EXAMPLE

次のコードは，まず URL *http://www.jaxa.jp/*を読み込みモードで開きます．そのストリームの内容を gzip 圧縮したバイナリデータをディレクトリ *directory* に存在するファイル *file.gz* に書き出します.

```
digeststreamio net_in;
digeststreamio f_out;
const char *line_ptr;

if (net_in.open("r", "http://www.jaxa.jp/") < 0) {
    エラー処理
}

if (f_out.openf("w", "%s/%s", "directory", "file.txt.gz") < 0) {
    エラー処理
}

while ((line_ptr = net_in.getline()) != NULL) {
    if (f_out.printf("%s", line_ptr) < 0) {
        エラー処理
    }
}

f_out.close();
net_in.close();
```

8.8.2 openp(), openpf(), vopenpf()

NAME

openp(), openpf(), vopenpf() — ストリームを開く (Perl ライク)

SYNOPSIS

```
int openp( const char *path );
int openpf( const char *path_fmt, ... );
int vopenpf( const char *path_fmt, va_list ap );
```

DESCRIPTION

これらのメンバ関数は，スクリプト言語 Perl の `open()` に似た関数で，`path` あるいは `path_fmt` がファイルや URL を示している場合はそれを開きます．また，コマンドを示している場合はそれを実行し，パイプで接続して，ストリームを開きます.

読み込みモードか書き込みモードかは，`path` あるいは `path_fmt` がファイルや URL を示している場合は，`"< infile.txt"` や `"> outfile.txt"` のように「<」あるいは「>」をつけて表現します.「<」がある場合は読み込みモードとして (EXAMPLE-1 参照),「>」がある場合は書き込みモードとしてオープンします.「<」「>」の指定が無い場合は，読み込みモードでオープンします．圧縮されたファイルが指定された場合，自動的に圧縮・伸長します.

path あるいは path_fmt がコマンドを示している場合，"command |" や "| command" のように,「|」を path の後か前につけて表現します．「|」が path の後につけられている場合は，指定したコマンドを実行し，そのコマンドの標準出力からの出力を，オブジェクトの読み出し用メンバ関数で読めるようにパイプ接続します (EXAMPLE-2 参照).「|」が path の前につけられている場合は，指定したコマンドを実行し，オブジェクトの書き出し用メンバ関数で書き出した結果を，そのコマンドの標準入力から読めるようにパイプ接続します (EXAMPLE-3 参照) path あるいは path_fmt がコマンドの場合,「/bin/sh -c コマンド」の形で実行しますので，path にはパイプやリダイレクトの記号 (|,<,>) を含む事ができます (EXAMPLE-2 参照).

path あるいは path_fmt が NULL の場合は，標準入力を使用します.

openpf() および vopenpf() の path_fmt 以降の引数の詳細は，§8.1.1の解説を参照してください.

PARAMETER

[I]	path	ファイル，URL またはコマンド
[I]	path_fmt	ファイル，URL またはコマンドのフォーマット指定
[I]	...	ファイル，URL またはコマンドの各要素データ
[I]	ap	ファイル，URL またはコマンドの全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合
	:	path_fmt のパスを示す文字列が PATH_MAX を超えている場合
	:	ファイル，URL またはコマンドのオープンに失敗した場合

EXCEPTION

オブジェクトの生成に失敗した場合

EXAMPLE-1

サーバ名 server のポート番号 port のパス path に，http プロトコルで読み込み専用で接続し，そのストリームの内容を標準出力に書き出します.

```
digeststreamio net_in;
digeststreamio s_io;
const char *line_ptr;

if (net_in.openpf("< http://%s:%d%s",server,port,path) < 0) {
    エラー処理
}

if (s_io.open("w") < 0) {
    エラー処理
}

while ((line_ptr = net_in.getline()) != NULL) {
```

```
        if (s_io.printf("%s", line_ptr) < 0) {
            エラー処理
        }
    }

    s_io.close();
    net_in.close();
```

EXAMPLE-2

次のコードは，ディレクトリ *directory* に存在するファイル *file.txt* の内容を 2 番目のフィールドをキーとして行単位で並べ替え，オープン処理によって標準出力します.
openp() の引数は，"command |"の形式で指定されており，1 番目の「|」はパイプを，2 番目の「|」は，引数がコマンドである事を示しています.
このコードは，§8.7.1の EXAMPLE-2 と同じ事をするものですが，コマンド command の後に「|」をつけているところが異なります.

```
    digeststreamio p_in;
    const char *line_ptr;

    if (p_in.openp("cat directory/file.txt | sort -k 2 |") < 0) {
        エラー処理
    }

    while ((line_ptr = p_in.getline()) != NULL) {
        printf("%s",line_ptr);
    }

    p_in.close();
```

EXAMPLE-3

次のコードは，ディレクトリ *directory* に存在するファイル *file.txt* の内容を 1 行ずつ読み込んで，less コマンドで画面に表示します.

```
    stdstreamio f_in;
    digeststreamio p_out;
    const char *line_ptr;

    if (f_in.open("r", "directory/file.txt") < 0) {
        エラー処理
    }

    if (p_out.openp("| less") < 0) {
        エラー処理
    }
```

```
while ((line_ptr = f_in.getline()) != NULL) {
    p_out.printf("%s",line_ptr);
}

p_out.close();
f_in.close();
```

8.8.3 is_write_mode()

NAME

is_write_mode() — 書き込みモードでオープンされたかを調べる

SYNOPSIS

`bool is_write_mode() const;`

DESCRIPTION

書き込みモードでオープンされたストリームの場合は true，そうでない場合は false を返します．

8.8.4 content_length()

NAME

content_length() — ストリームの長さを調べる

SYNOPSIS

`long long content_length() const;`

DESCRIPTION

`open()` で開いたストリームが，ストリームのバイト長に関する情報を持つ場合，そのバイト長を返します．圧縮されたストリームの場合，圧縮された状態でのバイト長を返します．

RETURN VALUE

非負の値	:	ストリームのバイト長
負の値 (エラー)	:	ストリームのバイト長に関する情報が存在しない場合

EXAMPLE

次のコードは，URL *http://www.jaxa.jp/*を読み込みモードで開き, そのストリームのバイト長を標準出力します．

```
digeststreamio net_in;
long long l_ret;

if (net_in.open("r", "http://www.jaxa.jp/") < 0) {
    エラー処理
}

if ((l_ret = net_in.content_length()) < 0) {
    printf("No information about stream byte size\n");
```

```
    }
    else {
        printf("Stream Byte Size = %lld \n", l_ret);
    }

    net_in.close();
```

8.8.5 user_agent().assign()

NAME

user_agent().assign() — ユーザエージェントを設定

SYNOPSIS

```
tstring &user_agent().assign( const char *uagent );
tstring &user_agent().assignf( const char *uagent_fmt, ... );
```

DESCRIPTION

Web サーバとの接続時に送信されるユーザエージェントを設定します．未設定の場合,「ホスト名 SLLIB-x.x::httpstreamio」が送信されます.

8.8.6 username().assign()

NAME

username().assign() — FTP ユーザの名前を設定

SYNOPSIS

```
tstring &username().assign( const char *user );
```

DESCRIPTION

`open()` メンバ関数で FTP サーバに対して接続する時，デフォルトでは anonymous でログインしますが，このメンバ関数を使ってユーザ名を設定できます.

8.8.7 password().assign()

NAME

password().assign() — FTP ユーザのパスワードを設定

SYNOPSIS

```
tstring &password().assign( const char *pass );
```

DESCRIPTION

`username().assign()` で設定した FTP ユーザのパスワードを設定します.

8.9 TERMLINEIO クラス

termlineio クラスは，GNU readline ライブラリを利用したコマンド入力を支援するためのクラスです．コマンド入力において，カーソルキーやヒストリ機能などをサポートします．cstreamio を継承しているので，§8.1のメンバ関数はすべて利用可能です (GNU readline の API を知る必要はありません)．termlineio クラスでは，必ず open()，close() を使う必要があります．open() メンバ関数では，mode に"r"か"w"を指定します．mode が"r"の時は行単位のコマンドの入力，"w"の時はページャへの出力となります．ページャは環境変数 PAGER で指定されたものが起動されます．

termlineio クラスを使う場合は,「#include <sli/termlineio.h>」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は,「using namespace sli;」もコードに書いてください．

表 15 はメンバ関数一覧です．libc と同じ機能を持つメンバ関数については，対応を示します．

	termlineio クラス	機能	対応する libc の関数
§8.9.1	open(), openf(), vopenf()	ストリームを開く	fopen()
§8.1.2	close()	ストリームを閉じる	fclose()
§8.1.3	read()	バイナリストリームの入力	fread()
§8.1.3	write()	バイナリストリームの出力	fwrite()
§8.1.4	bread()	バイナリストリームの入力 (endian 変換有)	—
§8.1.5	bwrite()	バイナリストリームの出力 (endian 変換有)	—
§8.1.6	rskip()	ストリームの読み飛ばし	—
§8.1.7	wskip()	n バイトの空白キャラクタの書き込み	—
§8.1.8	getchr()	1 文字の入力	fgetc()
§8.1.9	getstr()	文字列の入力	fgets()
§8.1.10	getline()	1 行の入力	—
§8.1.11	scanf()	入力を書式で変換・引数に代入	fscanf()
§8.1.12	putchr()	1 文字の出力	fputc()
§8.1.13	putstr()	文字列の出力	fputs()
§8.1.14	printf()	引数の値を書式変換後に出力	fprintf()
§8.1.15	flush()	バッファの内容を強制出力	fflush()
§8.1.16	eof(), error(), reseterr()	ストリームステータスのチェックとリセット	feof(), etc.
§8.9.2	set_prompt()	プロンプトを設定	—
§8.9.3	automate_history()	ヒストリを自動保存するかどうかを設定	—
§8.9.4	add_history()	ヒストリ・バッファにコマンド履歴を追加	—
§8.9.5	clear_history()	ヒストリ・バッファの初期化	—
§8.9.6	stifle_history()	ヒストリ・バッファの件数を制限	—
§8.9.7	unstifle_history()	ヒストリ・バッファの件数制限を解除	—
§8.9.8	read_history()	ヒストリのファイルからの読み込み	—
§8.9.9	write_history()	ヒストリのファイルへの書き出し	—

表 15: termlineio クラスで利用可能なメンバ関数一覧.

termlineio クラスで再定義されたメンバ関数および追加されたメンバ関数の使用法について，以下に記述します.

8.9.1 open()

NAME

open(), openf(), vopenf() — ストリームを開く

SYNOPSIS

```
int open( const char *mode );  ........................................... 1
int open( const char *mode, int fd );  ................................... 2
int open( const char *mode, const char *path );  ......................... 3
int open( const char *mode, const char *const argv[] );  ................. 4
int openf( const char *mode, const char *path_fmt, ... );  ............... 5
int vopenf( const char *mode, const char *path_fmt, va_list ap );  ....... 6
```

DESCRIPTION

mode が"r"，"r+"の時は GNU readline を用いたターミナルからの入力を行ないます．path には，ヒストリを保存したファイルを指定できます．"r"指定では close() 時にヒストリ・バッファの内容をファイルに保存しませんが，"r+"指定の時はヒストリを close() 時にファイルに保存します．

mode が"w"の時はページャへの出力を行ないます．ページャは，path か argv で指定しますが，指定されない場合は環境変数 PAGER で指定されたものが起動されます．

path あるいは path_fmt が設定された場合,「/bin/sh -c コマンド」の形で実行しますので，path あるいは path_fmt にはパイプやリダイレクトの記号 (|,<,>) を含む事ができます．

argv[] の各要素には,「実行ファイルのパス名 引数 1 引数 2 ··· NULL」の順で設定して下さい．argv[] の最後は NULL になっている必要があります．

メンバ関数 1 およびメンバ関数 2 は，コマンドを実行し，パイプで接続して，ストリームをオープンする事はできません．これらのメンバ関数の詳細は，§8.1.1の解説を参照してください．

メンバ関数 5 およびメンバ関数 6 の path_fmt 以降の引数の詳細は，§8.1.1の解説を参照してください．

PARAMETER

[I]	mode	ファイルまたはページャを開くモード
[I]	fd	ファイルディスクリプタ
[I]	path	ファイルまたはページャ
[I]	argv[]	ファイルまたはページャの全要素データ
[I]	path_fmt	ファイルまたはページャのフォーマット指定
[I]	...	ファイルまたはページャの各要素データ
[I]	ap	ファイルまたはページャの全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	オープンモードが設定されていない場合
	:	ストリームのオープンに失敗した場合
	:	ストリームが，指定のモードでのアクセスを許されていない等の理由で，オープンに失敗した場合
	:	指定した `mode` と `fd` の関係が正しくない等の理由で，オープンに失敗した場合 (メンバ関数 2)
	:	`path_fmt` のパスを示す文字列が `PATH_MAX` を超えている場合
	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合

EXCEPTION

データ読み込み用のバッファの確保に失敗した場合 (メンバ関数 1，3，5，6)
プロセスの生成に失敗した場合 (メンバ関数 2 以外)
読み込み用および書き込み用のファイル・ディスクリプタの生成に失敗した場合 (メンバ関数 2 以外)
ファイルまたはページャの各要素データがフォーマット指定に合わない場合 (メンバ関数 5，6)

EXAMPLE

次のコードは，ファイル *command_history.txt* を読み書きモードで開き，ヒストリ・バッファに読み取ります．そして，コマンドラインから入力された 1 行をヒストリ・バッファに読み取り，ファイル *command_history.txt* に書き込みを行います．

```
termlineio t_in;

if ( t_in.open("r+", "command_history.txt") < 0 ) {
    エラー処理
}

t_in.getline();

t_in.close();
```

8.9.2 set_prompt(), setf_prompt(), vsetf_prompt()

NAME

set_prompt(), setf_prompt(), vsetf_prompt() — プロンプトを設定

SYNOPSIS

```
termlineio &set_prompt( const char *prompt );  ............................. 1
termlineio &setf_prompt( const char *prompt_fmt, ... );  .................. 2
termlineio &vsetf_prompt( const char *prompt_fmt, va_list ap );  ......... 3
```

DESCRIPTION

コマンド入力の時に表示するプロンプトを設定します．

メンバ関数 1 は `prompt` をプロンプトに設定します．

メンバ関数 2 およびメンバ関数 3 のは format の変換指定に応じて変換した文字列を，プロンプトに設定します．format 以降の引数の詳細は，§8.1.14の解説を参照してください．

PARAMETER

[I]	prompt	プロンプト
[I]	prompt_fmt	プロンプトのフォーマット指定
[I]	...	プロンプトの各要素データ
[I]	ap	プロンプトの全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

プロンプト設定用バッファの確保に失敗した場合
プロンプトの各要素データがプロンプトのフォーマット指定に合わない場合 (メンバ関数 2，3)

EXAMPLE

プロンプトに表示する文字列を設定します．このコードを実行時に，"prompt> "がプロンプトに設定され，1 行のコマンドが入力できる事を確認します．

```
termlineio t_in;
const char *cmd;

if ( t_in.open("r") < 0 ) {
    エラー処理
}

cmd = t_in.set_prompt("prompt> ").getline();
printf("Your command is '%s'\n", cmd);

t_in.close();
```

8.9.3 automate_history()

NAME

automate_history() — ヒストリを自動保存するかどうかを設定

SYNOPSIS

```
termlineio &automate_history( bool tf );
```

DESCRIPTION

ヒストリ自動保存フラグ tf が true の場合，空白でない入力行を常にヒストリ・バッファに加えていきます．tf が false の場合，ヒストリ・バッファに登録しません．本メンバ関数によって入力行をヒストリ・バッファに登録しないとした後に，入力行をヒストリ・バッファに登録するには add_history() メンバ関数 (§8.9.4) を使う必要があります．
なお，tf のデフォルトの値は true です．

PARAMETER

[I] `tf` ヒストリ自動保存フラグ (true/false)
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

このコードは，ヒストリ自動保存フラグを false に設定します．次に，コマンドラインを 3 回読み取ります．その間に上矢印キーを押すと，入力したコマンドが登録されておらず，読み込んだファイル *command_history.txt* に記載されている文字列のみが登録されている事が確認できます．

```
termlineio t_in;
int        i_count = 0;

if ( t_in.open("r", "command_history.txt") < 0 )  {
    エラー処理
}

/* ヒストリを自動保存しない */
t_in.automate_history( false );

for (i_count = 0 ; i_count < 3 ; i_count++){
    t_in.getline();
}

t_in.close();
```

8.9.4 add_history()

NAME

add_history() — ヒストリ・バッファにコマンドを追加

SYNOPSIS

```
termlineio &add_history( const char *line );
```

DESCRIPTION

コマンド `line` をヒストリ・バッファに加えます．`automate_history()` メンバ関数 (§8.9.3) でヒストリの自動保存しないと設定した場合に利用します．

PARAMETER

[I] `line` コマンド名
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

コマンド複写のためのバッファの確保に失敗した場合
コマンドをヒストリ・バッファへの格納に失敗した場合

EXAMPLE

このコードは，ヒストリ自動保存フラグを false に設定し，入力されたコマンドを system() 関数で実行します．コマンドが正常終了した場合のみ，`add_history()` メンバ関数を用いてヒストリ・バッファに登録します．
ヒストリ・バッファの内容は *command_history.txt* に保存されます．

```
termlineio t_in;
const char *cmd;

if ( t_in.open("r+", "command_history.txt") < 0 )  {
    エラー処理
}

/* ヒストリを自動保存しない */
t_in.automate_history( false );

/* Ctrl-D が押されるまで，コマンドを受け付ける */
while ( (cmd=t_in.getline()) != NULL ) {
    /* system() を用いてコマンドを実行し，正常終了時のみヒストリを保存 */
    if ( system(cmd) == 0 ) {
        t_in.add_history(cmd);
    }
}

t_in.close();
```

8.9.5 clear_history()

NAME

clear_history() — ヒストリ・バッファの初期化

SYNOPSIS

`termlineio &clear_history();`

DESCRIPTION

ヒストリ・バッファの内容をすべて削除します．

RETURN VALUE

自身の参照

EXAMPLE

このコードは，登録されているヒストリを全て消去します．最初にコマンドラインからの入力

を 3 回読み取り，ヒストリ・バッファに登録します．"check history> "が出力された時，上矢印キーを押すとヒストリ・バッファが全て消えている事が確認できます．

```
termlineio t_in;
int        i_count = 0;

if ( t_in.open("r") < 0 )  {
    エラー処理
}

for (i_count = 0 ; i_count < 3 ; i_count++){
   t_in.getline();
}

t_in.clear_history();
t_in.set_prompt("check history >").getline();

t_in.close();
```

8.9.6 stifle_history()

NAME

stifle_history() — ヒストリ・バッファの件数を制限

SYNOPSIS

`termlineio &stifle_history( int num_lines );`

DESCRIPTION

ヒストリ･バッファの件数を最大で num_lines 件に制限します．この制限は，unstifle_history() メンバ関数 (§8.9.7) で解除できます．

PARAMETER

[I] num_lines ヒストリ・バッファの件数

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

このコードは，登録できるヒストリ:バッファを制限します．登録できる最大のヒストリ・バッファを 3 件に設定します．それからコマンドラインを 5 回読み取り，ヒストリ・バッファにコマンドを登録します．"check history> "が出力された時，上矢印キーを押すと最初に登録された 2 件のヒストリが消えた事が確認できます．

```
termlineio t_in;
int        i_count = 0;
```

```
if ( t_in.open("r") < 0 )  {
    エラー処理
}

/* ヒストリ・バッファを 3 件に制限 */
t_in.stifle_history(3);

for (i_count = 0 ; i_count < 5 ; i_count++){
   t_in.getline();
}

t_in.set_prompt("check history >").getline();

t_in.close();
```

8.9.7 unstifle_history()

NAME

unstifle_history() — ヒストリ・バッファの件数制限を解除

SYNOPSIS

`termlineio &unstifle_history();`

DESCRIPTION

stifle_history()(§8.9.6) で制限したヒストリ・バッファの件数制限を解除します.

RETURN VALUE

自身の参照

EXAMPLE

このコードは，ヒストリ・バッファの制限を解除します．最初に §8.9.6の `EXAMPLE` と同様の処理を行ないます．その後，ヒストリ・バッファの制限を解除し，再度ヒストリ・バッファにコマンドを登録します．`"check history2> "`が出力された時，上矢印キーを押すと 8 件のコマンドヒストリを確認できます.

```
termlineio t_in;
int        i_count = 0;

if ( t_in.open("r") < 0 )  {
    エラー処理
}

/* ヒストリ・バッファを 3 件に制限 */
t_in.stifle_history(3);
```

```
    for ( i_count = 0 ; i_count < 5 ; i_count++){
       t_in.getline();
    }

    t_in.set_prompt("check history1 >").getline();
    t_in.set_prompt("");

    /* ヒストリ・バッファの制限を解除 */
    t_in.unstifle_history();

    for ( i_count = 0 ; i_count < 5 ; i_count++){
       t_in.getline();
    }

    t_in.set_prompt("check history2 >").getline();

    t_in.close();
```

8.9.8 read_history(), readf_history(), vreadf_history()

NAME

read_history(), readf_history(), vreadf_history() — ヒストリのファイルからの読み込み

SYNOPSIS

`read_history( const char *path );` 1
`readf_history( const char *path_fmt, ... );` 2
`vreadf_history( const char *path_fmt, va_list ap );` 3

DESCRIPTION

`path` で指定されたファイルを読み，ファイルに記載された内容をヒストリ・バッファに追加します．

メンバ関数 2 およびメンバ関数 3 のは `format` の変換指定に応じて変換し，読み込むファイル名を作成します．`format` 以降の引数の詳細は，§8.1.14の解説を参照してください．

PARAMETER

[I]	`path`	ファイル名
[I]	`path_fmt`	ファイル名のフォーマット指定
[I]	`...`	ファイル名の各要素データ
[I]	`ap`	ファイル名の全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
0 以外 (エラー)	:	指定したファイル等が存在しない等の理由によって，ヒストリ・バッファへの追加に失敗した場合

EXCEPTION

ファイルパスの各要素データがフォーマット指定に合わない場合 (メンバ関数 2，3)

EXAMPLE

このコードは，ファイル *temporary_file.txt* から読み込んだコマンドをヒストリ・バッファに追加します．"push ↑ button >"のプロンプトが出力された時，上矢印キーを押すとヒストリ・バッファに *temporary_file.txt* の内容が保存されている事を確認できます．前提条件として，あらかじめ *command_history.txt*， *temporary_file.txt* には文字列を記載しておいてください．

```
termlineio t_in;

if (t_in.open("r", "command_history.txt") < 0) {
    エラー処理
}

if (t_in.read_history("temporary_file.txt") != 0) {
    エラー処理
}
else {
    t_in.set_prompt("push ↑ button >").getline();
}

t_in.close();
```

8.9.9 write_history(), writef_history(), vwritef_history()

NAME

write_history(), writef_history(), vwritef_history() — ヒストリのファイルへの書き出し

SYNOPSIS

`int write_history( const char *path );` 1
`int writef_history( const char *path_fmt, ... );` 2
`int vwritef_history( const char *path_fmt, va_list ap );` 3

DESCRIPTION

ヒストリ・バッファの内容を `path` で指定されたファイルに書き出します．ファイルが存在しなかった場合，新たにファイルを作成します．指定したファイルが既にある場合は上書きをします．ファイル名に NULL を指定した場合は，`~/.history` に保存します．

メンバ関数 2 およびメンバ関数 3 のは `format` の変換指定に応じて変換し，書き込むファイル名を作成します．`format` 以降の引数の詳細は，§8.1.14の解説を参照してください．

PARAMETER

[I]	`path`	ファイル名
[I]	`path_fmt`	ファイル名のフォーマット指定
[I]	`...`	ファイル名の各要素データ
[I]	`ap`	ファイル名の全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
0 以外 (エラー)	:	指定したファイル等に実行権がない等の理由によって，ファイルへの書き込みに失敗した場合

EXCEPTION

ファイルパスの各要素データがフォーマット指定に合わない場合 (メンバ関数 2，3)

EXAMPLE

このコードは，ヒストリ・バッファの内容をファイル *new_file.txt* に書き込みます．*new_file.txt* にはヒストリ・バッファに読み込んだファイル *command_history.txt* の内容が書き込まれます．前提条件として，あらかじめファイル *command_history.txt* には文字列を記載しておいてください．

```
termlineio t_in;

if (t_in.open("r", "command_history.txt") < 0) {
    エラー処理
}

if (t_in.write_history("new_file.txt") != 0) {
    エラー処理
}

t_in.close();
```

8.10 TERMSCREENIO クラス

termscreenio クラスは，ターミナル上でエディタ経由での入力 (一時ファイルを利用)，ページャへの出力を行なうためのクラスです．内部で環境変数 EDITOR と PAGER を参照します．cstreamio を継承しているので，§8.1のメンバ関数はすべて利用可能です．termscreenio クラスでは，必ず open()，close() を使う必要があります．

termscreenio クラスを使う場合は,「#include <sli/termscreenio.h>」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は,「using namespace sli;」もコードに書いてください．

表 16 はメンバ関数一覧です．libc と同じ機能を持つメンバ関数については，対応を示します．

	termscreenio クラス	機能	対応する libc の関数
§8.10.1	open(), openf(), vopenf()	ストリームを開く	fopen()
§8.1.2	close()	ストリームを閉じる	fclose()
§8.1.3	read()	バイナリストリームの入力	fread()
§8.1.3	write()	バイナリストリームの出力	fwrite()
§8.1.4	bread()	バイナリストリームの入力 (endian 変換有)	—
§8.1.5	bwrite()	バイナリストリームの出力 (endian 変換有)	—
§8.1.6	rskip()	ストリームの読み飛ばし	—
§8.1.7	wskip()	n バイトの空白キャラクタの書き込み	—
§8.1.8	getchr()	1 文字の入力	fgetc()
§8.1.9	getstr()	文字列の入力	fgets()
§8.1.10	getline()	1 行の入力	—
§8.1.11	scanf()	入力を書式で変換・引数に代入	fscanf()
§8.1.12	putchr()	1 文字の出力	fputc()
§8.1.13	putstr()	文字列の出力	fputs()
§8.1.14	printf()	引数の値を書式変換後に出力	fprintf()
§8.1.15	flush()	バッファの内容を強制出力	fflush()
§8.1.16	eof(), error(), reseterr()	ストリームステータスのチェックとリセット	feof(), etc.

表 16: termscreenio クラスで利用可能なメンバ関数一覧.

termscreenio クラスで再定義されたメンバ関数および追加されたメンバ関数の使用法について，以下に記述します.

8.10.1 open()

NAME

open() — ストリームを開く

SYNOPSIS

```
int open( const char *mode );  ................................................ 1
int open( const char *mode, int fd );  ........................................ 2
int open( const char *mode, const char *path );  ............................ 3
int open( const char *mode, const char *const argv[] );  .................... 4
int openf( const char *mode, const char *path_fmt, ... );  ................. 5
```

```
int vopenf( const char *mode, const char *path_fmt, va_list ap );
```
.... 6

DESCRIPTION

modeが"r"の場合は，一時ファイルを作成してエディタを起動し，エディタを終了すると編集した一時ファイルをストリームとして開きます．エディタはpathかargvで指定する事ができますが，指定が無い場合は，環境変数EDITORで示されたエディタを起動します．環境変数を設定していない場合は，viを起動します．

modeが"w"の場合は，ページャに対して，ストリームを開きます．ページャはpathかargvで指定する事ができますが，指定が無い場合は，環境変数PAGERで示されたページャを起動します．環境変数を設定していない場合は，moreを起動します．

argv[] の各要素には,「実行ファイルのパス名 引数1 引数2 ··· NULL」の順で設定して下さい．argv[] の最後はNULLになっている必要があります．

メンバ関数1およびメンバ関数2は，エディタやページャを介して，ストリームをオープンする事はできません．これらのメンバ関数の詳細は，§8.1.1の解説を参照してください．

メンバ関数5およびメンバ関数6のpath_fmt以降の引数の詳細は，§8.1.1の解説を参照してください．

PARAMETER

[I]	mode	エディタまたはページャを開くモード
[I]	fd	ファイルディスクリプタ
[I]	path	エディタまたはページャ
[I]	argv[]	エディタまたはページャの全要素データ
[I]	path_fmt	エディタまたはページャのフォーマット指定
[I]	...	エディタまたはページャの各要素データ
[I]	ap	エディタまたはページャの全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	オープンモードが設定されていない場合
	:	ストリームのオープンに失敗した場合
	:	path_fmtのパスを示す文字列がPATH_MAXを超えている場合
	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合

EXCEPTION

データ読み込み用バッファの確保に失敗した場合 (メンバ関数1，3，5，6)
プロセスの生成に失敗した場合 (メンバ関数2以外)
読み込み用および書き込み用のファイル・ディスクリプタの生成に失敗した場合 (メンバ関数2以外)
エディタまたはページャの各要素データがフォーマット指定に合わない場合 (メンバ関数5，6)

EXAMPLE

次のコードは，バイナリモードで起動したエディタ *vi* から入力を行ない，入力した最初の1行を標準出力に表示します．

```
termscreenio t_in;

if (t_in.open("r", "vi -b") < 0) {
    エラー処理
}

printf("%s\n", t_in.getline());

t_in.close();
```

8.11 INETSTREAMIO クラス

inetstreamio クラスは，シーケンシャル・1 ウェイまたは 2 ウェイ接続用の低レベルインターネットクライアントです．cstreamio を継承しているので，§8.1のメンバ関数はすべて利用可能です．このクラスを利用して，httpstreamio クラス (§8.5) や ftpstreamio クラス (§8.6) が実装されています．

inetstreamio クラスでは，必ず `open()`，`close()` を使う必要があります．open() メンバ関数では，`mode` に`"r+"`，`path` に`"http://www.jaxa.jp/"`のような URL を指定します．URL により，ポート番号とホスト名を解決し，サーバに接続します．プロキシサーバを介した接続はサポートしていません．

inetstreamio クラスを使う場合は,「`#include <sli/inetstreamio.h>`」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は,「`using namespace sli;`」もコードに書いてください．

表 17 はメンバ関数一覧です．libc と同じ機能を持つメンバ関数については，対応を示します．

	inetstreamio クラス	機能	対応する libc の関数
§8.11.1	`open()`, `openf()`, `vopenf()`	ストリームを開く	`fopen()`
§8.1.2	`close()`	ストリームを閉じる	`fclose()`
§8.1.3	`read()`	バイナリストリームの入力	`fread()`
§8.1.3	`write()`	バイナリストリームの出力	`fwrite()`
§8.1.4	`bread()`	バイナリストリームの入力 (endian 変換有)	—
§8.1.5	`bwrite()`	バイナリストリームの出力 (endian 変換有)	—
§8.1.6	`rskip()`	ストリームの読み飛ばし	—
§8.1.7	`wskip()`	n バイトの空白キャラクタの書き込み	—
§8.1.8	`getchr()`	1 文字の入力	`fgetc()`
§8.1.9	`getstr()`	文字列の入力	`fgets()`
§8.1.10	`getline()`	1 行の入力	—
§8.1.11	`scanf()`	入力を書式で変換・引数に代入	`fscanf()`
§8.1.12	`putchr()`	1 文字の出力	`fputc()`
§8.1.13	`putstr()`	文字列の出力	`fputs()`
§8.1.14	`printf()`	引数の値を書式変換後に出力	`fprintf()`
§8.1.15	`flush()`	バッファの内容を強制出力	`fflush()`
§8.1.16	`eof()`, `error()`, `reseterr()`	ストリームステータスのチェックとリセット	`feof()`, etc.
§8.11.2	`path()`	URL で指定されたパスを返す	—
§8.11.3	`host()`	URL で指定されたホスト名を返す	—

表 17: inetstreamio クラスで利用可能なメンバ関数一覧.

inetstreamio クラスで再定義されたメンバ関数および追加されたメンバ関数の使用法について，以下に記述します．

8.11.1 open()

NAME

open() — ストリームを開く

SYNOPSIS

```
int open( const char *mode, const char *path );  ............................ 1
int openf( const char *mode, const char *path_fmt, ... );  ................. 2
int vopenf( const char *mode, const char *path_fmt, va_list ap );  ........ 3
```

DESCRIPTION

path あるいは path_fmt で示された URL を開きます．path あるいは path_fmt が NULL の場合は，標準入力または標準出力を使用します．

mode には，"r"，"r+"，"w"または"w+"のいずれかを指定します．"r"，"w"はそれぞれ読み込み，書き込みのみの 1 ウェイで接続します．"r+"，"w+"はそれぞれ読み書きが可能な 2 ウェイ接続です．"r+"，"w+"はどちらも動作は同じです．

メンバ関数 2 およびメンバ関数 3 の path_fmt 以降の引数の詳細は，§8.1.1の解説を参照してください．

PARAMETER

[I]	mode	URL を開くモード
[I]	path	URL のパス
[I]	path_fmt	URL のフォーマット指定
[I]	...	URL の各要素データ
[I]	ap	URL の全要素データ

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	URL が指定されていない等の理由で，ストリームのオープンに失敗した場合
	:	指定した URL が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	path_fmt のパスを示す文字列が PATH_MAX を超えている場合
	:	mode が指定されていない等の理由で，ストリームのオープンに失敗した場合
	:	指定した mode が適切でない等の理由で，ストリームのオープンに失敗した場合
	:	ストリームが，指定のモードでのアクセスを許されていない等の理由で，オープンに失敗した場合
	:	すでに，本節に示すメンバ関数によって，ストリームがオープンされていた場合

EXCEPTION

メモリの割り当てに失敗した場合
ソケット通信に失敗した場合
与えられた mode でソケットが open 出来なかった場合
URL の各要素データがフォーマット指定に合わない場合 (メンバ関数 2，3)

EXAMPLE

EXAMPLE は §8.11.4を参照してください．

8.11.2 path()

NAME

path() — URL で指定されたパスを返す

SYNOPSIS

```
const char *path();
```

DESCRIPTION

open() メンバ関数 (§8.11.1) で指定した `path` から，パス部を抜き出した文字列を返します.

RETURN VALUE

URL のパス

EXAMPLE

EXAMPLE は §8.11.4を参照してください.

8.11.3 host()

NAME

host() — URL で指定されたホスト名を返す

SYNOPSIS

```
const char *host();
```

DESCRIPTION

open() メンバ関数 (§8.11.1) で指定した `path` から，ホスト名を抜き出した文字列を返します.

RETURN VALUE

URL で指定されたホスト名

EXAMPLE

EXAMPLE は §8.11.4を参照してください.

8.11.4 サンプルコード

簡単な HTTP クライアントの例です.

```
#include <sli/stdstreamio.h>
#include <sli/inetstreamio.h>
using namespace sli;

int main()
{
    int status = -1;
    stdstreamio sio;
    inetstreamio isio;
    const char *line_ptr;
    /* http サーバに接続 */
    if ( isio.open("r+","http://www.jaxa.jp/") < 0 ) {
        sio.eprintf("[ERROR] isio.open() failed\n");
        goto quit;
    }
    /* http サーバにリクエストを送信 */
    isio.printf("GET %s HTTP/1.0\r\n",isio.path());
    isio.printf("User-Agent: My Program\r\n");
    isio.printf("Host: %s\r\n",isio.host());
    isio.printf("Connection: close\r\n");
    isio.printf("\r\n");
    isio.flush();
    /* http サーバからのデータを受信 */
    while ( (line_ptr=isio.getline()) != NULL ) {
        sio.printf("%s",line_ptr);
    }
    /* 接続を終了 */
    isio.close();
    status = 0;
quit:
    return status;
}
```

9 TSTRING クラス

tstring クラスは，Perl，PHP，Ruby のようなスクリプト言語にみられる文字列処理や，libc の stdio.h，string.h，strings.h，stdlib.h，ctype.h で提供される関数とそっくりの文字列処理が行える API を提供します．

1 文字ずつ任意の位置に文字をセットするメンバ関数から，POSIX 拡張正規表現が使えるメンバ関数まで豊富な API を用意しました．tstring クラスのメンバ関数の引数の順番は，規則性が徹底しているのでたいへん覚えやすくなっています．

文字列バッファとそのサイズは内部で自動管理されるため，文字列を編集する場合にユーザはバッファを作る必要もありませんし，バッファの大きさを気にする必要もありません．例えば，次のようにオブジェクトを作った直後に文字列を代入できます．

```
tstring my_str;                       /* オブジェクトを作る */
my_str.printf("Hello World");         /* "Hello World" を my_str に代入 */
sio.printf("%s\n",my_str.cstr());     /* my_str の内容を標準出力へ出力 */
```

tstring クラスを使う場合は,「#include <sli/tstring.h>」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は,「using namespace sli;」もコードに書いてください．

次に実際に近い使用例を示します．

```
#include <sli/stdstreamio.h>
#include <sli/tstring.h>
using namespace sli;

int main()
{
    stdstreamio sio;
    stdstreamio fin;
    const char *line;

    fin.open("r","infile.txt");
    while ( (line=fin.getline()) != NULL ) {
        tstring str0;
        /* 両端のスペース，タブ，改行文字を除去して str0 へ格納 */
        str0.assign(line).trim(" \t\n");
        /* 1 文字以上あれば… */
        if ( 0 < str0.length() ) {
            if ( str0.cchr(0) == '#' ) {
                /* コメントを表示 */
                sio.printf("%s\n",str0.cstr());
            }
            else {
                /* 整数値に変換 */
                int n, a=0, b=0;
                n = str0.scanf("%d %d",&a,&b);
                sio.printf("n=%d  a=%d b=%d\n",n,a,b);
            }
        }
    }
    fin.close();

    return 0;
}
```

この例では，`infile.txt` をオープンし，#で始まる行はコメントとしてそのまま表示，空白・タブ・改行文字以外が存在する場合は，フォーマット入力して値を取り込み，それを表示します．

なお本章では，この例でも示したように `cstreamio` クラス (§8) を使って EXAMPLE を説明します．

9.1 オブジェクトの作り方—3 つの動作モード

tstring クラスのオブジェクトは，3 つの動作モードがあります．これらの動作モードを，目的に応じて使い分けてください．なお，動作モードはオブジェクトを作る時のみ決める事ができます．

9.1.1 通常モード

オブジェクトを作る時に何も指定しない場合，例えば，

```
    tstring my_str;
```

とした場合，オブジェクトを作った状態では内部の文字列はバッファを持っておらず，cstr() メンバ関数 (§9.5.3) の返り値は `NULL` です．この使い方を，**通常モード**と呼ぶ事にします．しかし，文字列を改変するためのメンバ関数 (ただし `init()` を除く) を使った場合，必ず何らかの文字列を保つようになります (最短では`""`を持つ)．文字列を持った状態から,「空っぽ」つまり `NULL` にしたい場合は，init() メンバ関数 (§9.5.12) を使うか，演算子「`=`」(§9.4.2) で `NULL` を代入します．

通常モードでは，次のように初期値を与える事もできます．

```
    tstring my_str("Hello");
```

9.1.2 NULL 無しモード

内部の文字列が `NULL` の状態になると困る場合があるかもしれません．その場合は，オブジェクトを作る時に，

```
    tstring my_str(true);
```

のように，フラグ `true` をセットします．このようにすると，オブジェクトは必ず何らかの文字列を保つようになり，cstr() メンバ関数 (§9.5.3) で `NULL` が返る事はありません．この使い方を，**NULL 無しモード**と呼ぶ事にします．

9.1.3 固定長バッファモード

もう 1 つの使い方は，**固定長バッファモード**で，扱いたい最大の文字列の長さを引数に指定し，

```
    tstring my_str(64);
```

のようにオブジェクトを作る方法です．この例は，オブジェクトは最大 64 文字の文字列を扱える事を意味します．このモードでは，最初に指定した文字列長までの文字列しか扱う事ができませんが，文字列の編集のためのメンバ関数は，メモリの再確保を最低限度に抑え，コードも高速動作する設計になっています．固定長バッファモードでは，cstr() メンバ関数 (§9.5.3) で `NULL` が返る事はありません．

9.1.4 固定長バッファモードの制限

通常モードや NULL 無しモードでは，オブジェクト内の文字列を改変するためのメンバ関数の引数に，自身のオブジェクトの参照や cstr() のアドレスを与える事ができますが，固定長バッファモードでは動作速度を最優先しているため，そのような使い方はできません．

9.2 メンバ関数の引数の規則性

オブジェクト内部の文字列に対する文字の位置と個数とが引数の指定にある場合，必ず引数の先頭側に指定するようになっています．例えば，strtol() メンバ関数 (§9.5.37) の場合，

```
long strtol( int base, size_t *endpos ) const;
long strtol( size_t pos, int base, size_t *endpos ) const;
long strtol( size_t pos, size_t n, int base, size_t *endpos ) const;
```

のように，オブジェクト内の文字列の位置と長さを示す pos，n が左側から登場します．

つまり，左側の引数はオブジェクトに対する指定で，右側の引数はオブジェクト以外に対する指定というわけです．

9.3 メンバ関数一覧

表 18 はメンバ関数一覧です．libc と同じ機能を持つメンバ関数については，対応を示します．

	メンバ関数名	機能	対応する libc の関数
§9.4.1	`[]`	指定した位置の文字の参照	—
§9.4.2	`=`	文字列を代入	—
§9.4.3	`+=`	文字列の追加	—
§9.4.4	`==`	文字列の比較	—
§9.4.5	`!=`	文字列の比較	—
§9.5.1	`length()`	文字列の長さ	`strlen()`
§9.5.2	`max_length()`	文字列の最大の長さ	—
§9.5.3	`cstr()`, `c_str()`	文字列の先頭アドレス (読み取り専用)	—
§9.5.4	`str_ptr()`, `str_ptr_cs()`	文字列の先頭アドレス	—
§9.5.5	`cchr()`	指定した位置の文字の読み取り	—
§9.5.6	`at()`, `at_cs()`	指定した位置の文字の参照	—
§9.5.7	`update_length()`	内部管理情報の更新 (固定長バッファモード時に，バッファに時下書きした場合に使用)	—
§9.5.8	`dprint()`	オブジェクト情報を標準エラー出力へ出力	—
§9.5.9	`getstr()`	(部分) 文字列を外部バッファにコピー	—
§9.5.10	`copy()`	(部分) 文字列を外部オブジェクトにコピー	`strdup()`
§9.5.11	`swap()`	オブジェクトのスワップ	—
§9.5.12	`init()`	オブジェクトの完全初期化	—
§9.5.13	`printf()`, `assignf()`	オブジェクトの初期化	`sprintf()`
§9.5.14	`implode()`	文字列配列を区切り文字で結合した文字列をセット	—
§9.5.15	`import_binary()`	バイナリデータの取り込み	—
§9.5.16	`put()`, `putf()`	任意の位置へ文字または文字列をセット	—
§9.5.17	`strcat()`, `append()`	文字または文字列の追加	`strcat()`
§9.5.17	`strncat()`, `append()`	文字または文字列の追加	`strncat()`
§9.5.18	`insert()`, `insertf()`	文字または文字列の挿入	—
§9.5.19	`replace()`, `replacef()`	文字列の置換	—
§9.5.20	`erase()`	文字列の消去	—
§9.5.21	`clean()`	既存の文字列全体を任意の文字でパディング	—
§9.5.22	`resize()`	文字列の長さを変更	—
§9.5.23	`resizeby()`	文字列の長さを相対的に変更	—
§9.5.24	`crop()`	文字列の切り抜き	—
§9.5.25	`chomp()`	改行文字の除去	—
§9.5.26	`trim()`	文字列の両端スペースの除去	—
§9.5.27	`ltrim()`	文字列の左端スペースの除去	—
§9.5.28	`rtrim()`	文字列の右端スペースの除去	—
§9.5.29	`strreplace()`	文字列の検索と置換	—
§9.5.30	`regreplace()`	拡張正規表現でマッチした部分の置換を行なう	—
§9.5.31	`tolower()`	大文字を小文字に変換	`tolower()`
§9.5.32	`toupper()`	小文字を大文字に変換	`toupper()`
§9.5.33	`expand_tabs()`	TAB 文字を空白文字に置換	—
§9.5.34	`contract_spaces()`	空白文字を TAB 文字に置換	—

表 18: tstring クラスで利用可能なメンバ関数一覧 (次ページに続く).

	メンバ関数名	機能	対応する libc の関数
§9.5.35	atoi()	整数値に変換	atoi()
§9.5.35	atol()	整数値に変換	atol()
§9.5.35	atoll()	整数値に変換	atoll()
§9.5.36	atof()	実数値に変換	atof()
§9.5.37	strtol()	整数値に変換	strtol()
§9.5.37	strtoll()	整数値に変換	strtoll()
§9.5.38	strtoul()	符号なし整数値に変換	strtoul()
§9.5.38	strtoull()	符号なし整数値に変換	strtoull()
§9.5.39	strtod()	実数値に変換	strtod()
§9.5.40	scanf()	書式付き入力変換	sscanf()
§9.5.41	strcmp(), compare()	文字列の比較	strcmp()
§9.5.42	strncmp(), compare()	部分的に文字列を比較	strncmp()
§9.5.43	strcasecmp()	文字列の比較 (大文字・小文字区別なし)	strcasecmp()
§9.5.43	strncasecmp()	文字列の比較 (大文字・小文字区別なし)	strncasecmp()
§9.5.44	isalnum()	英字または数字かを調べる	isalnum()
§9.5.44	isalpha()	アルファベットかどうか調べる	isalpha()
§9.5.44	iscntrl()	制御文字かどうかを調べる	iscntrl()
§9.5.44	isdigit()	数字 (0〜9) かどうかを調べる	isdigit()
§9.5.44	isgraph()	表示可能な文字かどうかを調べる	isgraph()
§9.5.44	islower()	小文字かどうかを調べる	islower()
§9.5.44	isprint()	表示可能な文字かどうかを調べる (空白を含む)	isprint()
§9.5.44	ispunct()	表示可能な文字かどうかを調べる (空白と英数字を除く)	ispunct()
§9.5.44	isspace()	空白文字かどうかを調べる	isspace()
§9.5.44	isupper()	大文字かどうかを調べる	isupper()
§9.5.44	isxdigit()	16 進数での数字かどうかを調べる	isxdigit()
§9.5.45	strchr(), find()	左側からの文字を検索	strchr()
§9.5.46	strstr(), find()	左側からの文字列を検索	strstr()
§9.5.47	strrchr(), rfind()	右側からの文字を検索	strrchr()
§9.5.48	strrstr(), rfind()	右側からの文字列を検索	—
§9.5.49	find_first_of()	左側から文字セット中の文字を検出	strpbrk()
§9.5.50	find_last_of()	右側から文字セット中の文字を検出	—
§9.5.51	find_first_not_of()	左側から文字セットに含まれない文字を検出	—
§9.5.52	find_last_not_of()	右側から文字セットに含まれない文字を検出	—
§9.5.53	strpbrk()	左側から文字セット中の文字を検出	strpbrk()
§9.5.54	strrpbrk()	右側から文字セット中の文字を検出	—
§9.5.55	strspn()	左側から文字セット中の文字が続く長さを調べる	strspn()
§9.5.56	strrspn()	右側から文字セット中の文字が続く長さを調べる	—
§9.5.57	strcspn()	左側から文字セットに含まれない文字が続く長さを調べる	strcspn()
§9.5.58	strmatch()	シェルライクな文字列マッチを試行	fnmatch()
§9.5.59	regmatch()	拡張正規表現による文字列マッチを試行	regexec()

表 18: tstring クラスで利用可能なメンバ関数一覧 (続き).

9.4 演算子

演算子の濫用はコードの可読性を下げる原因になりかねないので，演算子は最低限のものしか用意していません．

9.4.1 []

NAME

[] — 指定した位置の文字の参照

SYNOPSIS

`unsigned char &operator[]( size_t pos );` 1
`const unsigned char &operator[]( size_t pos ) const;` 2

DESCRIPTION

[] で指定した位置の文字の参照を返します．

メンバ関数 1 は読み書き両用で at() と同じ動作を，メンバ関数 2 は読み取り専用で at_cs() と同じ動作をします．

pos に文字列長以上の値を指定した場合，メンバ関数 1 では文字列の長さが自動拡張されますが，メンバ関数 2 では例外が発生します．

メンバ関数 1・メンバ関数 2 のどちらが使われるかは，オブジェクトの「const」属性の有無で自動的に決まります.「const」属性が無い場合にはメンバ関数 1 が，有る場合にはメンバ関数 2 が自動的に選択されます．

at()，at_cs() についての詳細は §9.5.6を参照してください．

PARAMETER

[I] pos 文字列の位置
([I]：入力，[O]：出力)

RETURN VALUE

指定した位置にある文字の参照

EXCEPTION

固定長バッファモードで pos に最大文字列長以上の値を指定した場合 (メンバ関数 1)
pos に文字列長以上の値を指定した場合 (メンバ関数 2)

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列の 6 番目の文字を読み取り，その結果を標準出力します．
次に my_str の 9 文字目の位置に文字 X を書き込み，その結果を標準出力します．

```
stdstreamio   sio;
tstring       my_str = "abcdefgh";
unsigned char c_read;

c_read = my_str[6];
```

```
sio.printf("%c\n", c_read);

my_str[9] = 'X';
sio.printf("%s\n", my_str.cstr());
```

実行結果

```
g
abcdefgh␣X
```
(␣ は空白文字を意味しています．)

9.4.2 =

NAME

= — 文字列を代入

SYNOPSIS

`tstring &operator=(const tstring &obj);` 1
`const char *operator=(const char *str);` 2

DESCRIPTION

演算子の右側 (引数) で指定したオブジェクト・文字列を自身に代入します.

PARAMETER

[I] `obj` tstring クラスのオブジェクト
[I] `str` 文字列のアドレス

RETURN VALUE

自身の参照 (メンバ関数 1)
内部バッファのアドレス (メンバ関数 2)

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合 (メンバ関数 1)

EXAMPLE

次のコードは，オブジェクト `my_str` が持つ文字列に文字列 *Hello SLLIB User !*を代入し，その結果を標準出力します．`c_str()` に関しては §9.5.3の解説を参照してください.

```
stdstreamio sio;
tstring     my_str;

my_str = "Hello SLLIB User !";

sio.printf("%s\n", my_str.c_str());
```

実行結果

```
Hello SLLIB User !
```

9.4.3 +=

NAME

+= — 文字列の追加

SYNOPSIS

`tstring &operator+=(const tstring &obj);` 1
`const char *operator+=(const char *str);` 2

DESCRIPTION

自身の文字列に，演算子の右側 (引数) で指定した文字列の追加を行います．

PARAMETER

[I] `obj` tstring クラスのオブジェクト
[I] `str` 文字列のアドレス

RETURN VALUE

自身の参照 (メンバ関数 1)
内部バッファのアドレス (メンバ関数 2)

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列と文字列 c_sentence の内容を連結し，その結果を標準出力します．`c_str()` に関しては §9.5.3の解説を参照してください．

```
stdstreamio sio;
tstring     my_str      = "User ID : ";
const char  *c_sentence = "1234";

my_str += c_sentence;

sio.printf("%s\n", my_str.c_str());
```

実行結果

```
User ID : 1234
```

9.4.4 ==

NAME

== — 文字列の比較

SYNOPSIS

`bool operator==(const tstring &obj) const;`
`bool operator==(const char *str) const;`

DESCRIPTION

自身の文字列と，演算子の右側 (引数) で指定した文字列が一致しているかどうかの比較を行い，その結果を返します．

PARAMETER

[I] `obj` tstring クラスのオブジェクト
[I] `str` 文字列のアドレス

RETURN VALUE

true : 文字列が一致した場合
false : 文字列が不一致である場合

EXAMPLE

次のコードは，オブジェクト `my_str` のが持つ文字列と文字列 *User ID : 1234* とを比較し，その結果を標準出力します.

```
stdstreamio sio;
tstring     my_str = "User ID : 1234";

if (my_str == "User ID : 1234") {
    sio.printf("This character string is corresponding.\n");
}
else {
    sio.printf("This character string is not corresponding.\n");
}
```

実行結果

```
This character string is corresponding.
```

9.4.5 !=

NAME

!= — 文字列の比較

SYNOPSIS

```
bool operator!=(const tstring &obj) const;
bool operator!=(const char *str) const;
```

DESCRIPTION

自身の文字列と，演算子の右側 (引数) で指定した文字列が不一致かどうかの比較を行い，その結果を返します.

PARAMETER

[I] `obj` tstring クラスのオブジェクト
[I] `str` 文字列のアドレス

RETURN VALUE

true : 文字列が不一致である場合
false : 文字列が一致した場合

EXAMPLE

次のコードは，オブジェクト `my_str` が持つ文字列と文字列 `c_sentence` の比較を行い，その結果を標準出力します.

```
stdstreamio sio;
tstring     my_str      = "User ID : 1234";
const char  *c_sentence = "User name : SUZUKI";

if (my_str != c_sentence) {
    sio.printf("This character string is not corresponding.\n");
}
else {
    sio.printf("This character string is corresponding.\n");
}
```

実行結果

```
This character string is not corresponding.
```

9.5 メンバ関数

全体的な注意事項

size_t 型は符号なし整数として数値を取り扱います．size_t 型を引数に持つメンバ関数に負の値を設定した場合，アボートする可能性が高くなります．負の値を設定しないようにしてください．

9.5.1 length()

NAME

length() — 文字列の長さ

SYNOPSIS

`size_t length() const;`

DESCRIPTION

自身の文字列の長さ ('\0' は含まない) を返します．

RETURN VALUE

文字列長

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列の長さを標準出力します．

```
stdstreamio sio;
tstring     my_str = "User'sFile.txt";

sio.printf("%zu\n", my_str.length());
```

実行結果

```
14
```

9.5.2 max_length()

NAME

max_length() — オブジェクトが扱える文字列の長さの最大値

SYNOPSIS

```
size_t max_length() const;
```

DESCRIPTION

固定長バッファモードでの最大の文字列長を返します．固定長バッファモードではない場合は，0 を返します．

9.5.3 cstr(), c_str()

NAME

cstr(), c_str() — 文字列の先頭アドレス (読み取り専用)

SYNOPSIS

```
const char *cstr() const;
const char *c_str() const;
```

DESCRIPTION

自身が持つ文字列の先頭アドレスを返します．

オブジェクト内の文字列が改変されると，改変後の文字列アドレスを取得します (EXAMPLE-2 参照)．

RETURN VALUE

文字列の先頭アドレス

EXAMPLE-1

次のコードは，オブジェクト my_str を NULL 無しモードで作成し，作成直後の内部文字列格納バッファが NULL ではない事を確認するため，標準出力します．そして，my_str に文字列 *This is a pen.* を代入し，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str(true);

if (my_str.c_str() != NULL) {
    sio.printf("The address of the character string is not NULL.\n");
}
else {
    sio.printf("The address of the character string is NULL.\n");
}

my_str = "This is a pen.";
sio.printf("%s\n", my_str.cstr());
```

実行結果

```
The address of the character string is not NULL.
This is a pen.
```

EXAMPLE-2

次のコードは，オブジェクト my_str が持つ文字列が改変されると，新しい文字列アドレスが取得される事を確認するため，その結果を標準出力します.

```
stdstreamio sio;
tstring     my_str = "JAXA";

sio.printf("%s\n", my_str.cstr());

my_str = "ISAS";

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
JAXA
ISAS
```

9.5.4 str_ptr(), str_ptr_cs()

NAME

str_ptr(), str_ptr_cs() — 文字列の先頭アドレス

SYNOPSIS

`char *str_ptr();` 1
`const char *str_ptr() const;` 2
`const char *str_ptr_cs() const;` 3

DESCRIPTION

自身が持つ文字列の先頭アドレスを返します.

メンバ関数 1 は，オブジェクト内部の文字列バッファに直接書き込みたい場合に利用します. 利用する場合は，書き込みたい文字列の長さにあわせて，resize() メンバ関数 (§9.5.22) などで内部のバッファサイズを調整してください.

メンバ関数 2,3 は，cstr() メンバ関数 (§9.5.3) と同じ動作をします.

str_ptr() メンバ関数の場合では，メンバ関数 1・メンバ関数 2 のどちらが使われるかは，オブジェクトの「const」属性の有無で自動的に決まります.「const」属性が無い場合にはメンバ関数 1 が，有る場合にはメンバ関数 2 が自動的に選択されます.

RETURN VALUE

文字列の先頭アドレス

WARNING

やむをえない場合以外は，このメンバ関数を使わないでください.

9.5.5 cchr()

NAME

cchr() — 指定した位置の文字の読み取り

SYNOPSIS

```
int cchr( size_t pos ) const;
```

DESCRIPTION

自身が持つ文字列の位置 pos にある文字を返します．なお，文字列の先頭位置は 0 です．

PARAMETER

[I] pos 文字列の位置

([I]：入力，[O]：出力)

RETURN VALUE

指定した位置にある文字	:	正常終了
負の値 (エラー)	:	pos に自身の文字列長以上の値を指定した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列に文字列 *User'sFile3.txt* を代入し，my_str の 10 番目の文字を標準出力します．

```
stdstreamio sio;
tstring     my_str = "User'sFile3.txt";
int         i_ret;

if ((i_ret = my_str.cchr(10)) < 0){
    エラー処理
}
sio.printf("%c\n", i_ret);
```

実行結果

```
3
```

9.5.6 at(), at_cs()

NAME

at(), at_cs() — 指定した位置の文字の参照

SYNOPSIS

```
unsigned char &at( size_t pos ); ............................................ 1
const unsigned char &at( size_t pos ) const; ............................... 2
const unsigned char &at_cs( size_t pos ) const; ............................ 3
```

DESCRIPTION

自身が持つ文字列の位置 pos にある文字の参照を返します．なお，文字列の先頭位置は 0 です．

メンバ関数 1 は，文字の読み書き両方に利用でき，メンバ関数 2,3 は，読み取り専用です．

at() メンバ関数の場合では，メンバ関数 1・メンバ関数 2 のどちらが使われるかは，オブジェクトの「const」属性の有無で自動的に決まります.「const」属性が無い場合にはメンバ関数 1 が，有る場合にはメンバ関数 2 が自動的に選択されます.

メンバ関数 1 はオブジェクトの動作モードが通常モードまたは NULL なしモードの場合，文字列長以上の pos を指定しても，文字列長を自動的に pos+1 に調整し，読み書きを行ないます. 固定長バッファモードの場合も同様ですが，pos にオブジェクト作成時の最大文字列長以上の値を指定すると例外が発生します.

メンバ関数 2,3 で文字列長以上の pos を指定した場合は，全ての動作モードで例外が発生します.

PARAMETER

[I] pos 文字列の位置
([I]：入力，[O]：出力)

RETURN VALUE

指定した位置にある文字の参照

EXCEPTION

固定長バッファモードで pos に最大文字列長以上の値を指定した場合 (メンバ関数 1)
pos に文字列長以上の値を指定した場合 (メンバ関数 2,3)

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列の 6 番目の文字を読み取り，その結果を標準出力します.
次に my_str の 9 文字目の位置に文字 X を書き込み，その結果を標準出力します.

```
stdstreamio   sio;
tstring       my_str = "abcdefgh";
unsigned char c_read;

c_read = my_str.at(6);
sio.printf("%c\n", c_read);

my_str.at(9) = 'X';
sio.printf("%s\n", my_str.cstr());
```

実行結果

g
abcdefgh␣X (␣ は空白文字を意味しています.)

9.5.7 update_length()

NAME

update_length() — 内部管理情報の更新 (固定長バッファモード時のみ)

SYNOPSIS

```
tstring &update_length();
```

DESCRIPTION

固定長バッファモード時に，内部文字列バッファにおいて終端文字 ('\0') を探し，文字列長に関するオブジェクト内部情報を更新します.

固定長バッファモードではオブジェクト内部でバッファ長と文字列長とを別々に管理しているため，プログラマが内部バッファに直接キャラクタを書き込んだ場合は，update_length() を使って文字列長に関する内部情報を更新してください.

9.5.8 dprint()

NAME

dprint() — オブジェクト情報を標準エラー出力へ出力 (ユーザのデバッグ用)

SYNOPSIS

```
void dprint() const;
```

DESCRIPTION

自身のオブジェクト情報を標準エラー出力へ出力します.

ユーザ・プログラムのデバッグを目的としたメンバ関数です.

EXAMPLE

次のコードは，オブジェクト my_str の情報を標準エラー出力に出力します．[] にオブジェクトのアドレスが表示されていますが，これは実行環境により異なります.

```
tstring my_str = "X68000 PRO";
my_str.dprint();
```

実行結果

```
sli::tstring[obj=0x7fbffff640] = "X68000 PRO"
```

9.5.9 getstr()

NAME

getstr() — (部分) 文字列を外部バッファにコピー

SYNOPSIS

```
ssize_t getstr( char *dest_str, size_t buf_size ) const;  .................. 1
ssize_t getstr( size_t pos, char *dest_str, size_t buf_size ) const;  ..... 2
```

DESCRIPTION

自身の文字列を，外部バッファdest_str にコピーします.

buf_size には，dest_str のサイズを指定します．返り値には，dest_str のバッファが十分であった場合に書き込まれる文字数を返します．従って，返り値が dest_str のサイズ以上である場合は，バッファが不足している事を表します.

メンバ関数 1 は，自身の文字列を先頭から dest_str へコピーします．

メンバ関数 2 は，自身の文字列を指定された位置 pos から dest_str にコピーします．なお，文字列の先頭位置は 0 です．

オブジェクト内の文字列に対してバッファのサイズが足りない場合でも，dest_str が指す文字列は'\0' で終端します．この場合，メンバ関数 1 であれば文字列の先頭から buf_size-1 文字コピーされ，メンバ関数 2 であれば文字列の位置 pos から buf_size-1 文字コピーされます．

PARAMETER

[O]	dest_str	コピー先の外部バッファアドレス
[I]	buf_size	外部バッファのサイズ
[I]	pos	コピーの開始位置

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	バッファ長が十分な場合にコピーできる文字数 ('\0' は含まない)
負の値 (エラー)	:	dest_str に NULL を設定し，buf_size に 0 以外の値を設定した場合
	:	pos に自身の文字列長を越える値を指定した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列の先頭から外部バッファc_sentence へのコピーを行い，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "JAXA/ISAS";
char        c_sentence[10] ;
int         i_ret  = 0;

if ((i_ret = my_str.getstr(0, c_sentence, sizeof(c_sentence))) < 0 ) {
    エラー処理
}
else if ( sizeof(c_sentence) < i_ret ) {
    sio.printf("The length of buffer is insufficient. \n");
}
else {
    sio.printf("%s\n", c_sentence);
}
```

実行結果

```
JAXA/ISAS
```

9.5.10 copy()

NAME

copy() — (部分) 文字列を外部オブジェクトにコピー

SYNOPSIS

```
ssize_t copy( tstring *dest ) const; ......................................... 1
ssize_t copy( size_t pos, tstring *dest ) const; ............................ 2
ssize_t copy( size_t pos, size_t n, tstring *dest ) const; ................ 3
```

DESCRIPTION

自身の文字列のすべてまたは一部を，指定されたオブジェクト dest にコピーします.

dest の動作モードが固定長バッファモードの場合，dest の最大文字列長以内でコピーを行ないます．なお，この際の返り値は，dest のバッファが十分であった場合に書き込まれる文字数を返します．従って，返り値が dest のバッファサイズ以上である場合は，バッファが不足している事を表します.

メンバ関数 1 は，自身の文字列すべてを dest へコピーします.

メンバ関数 2 およびメンバ関数 3 は，文字列の位置 pos から文字列をコピーします．なお，文字列の先頭位置は 0 です．またメンバ関数 3 は，コピーする文字列の長さ n を指定できます.

Perl や PHP の substr() 関数に相当するメンバ関数です.

PARAMETER

[I]	pos	コピー元の文字列の開始位置
[I]	n	コピーする文字数
[O]	dest	コピー先の外部 tstring クラスのオブジェクト

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	バッファ長が十分な場合にコピーできる文字数
負の値 (エラー)	:	pos にオブジェクト内の文字列長を越える値を指定した場合
	:	dest が NULL の場合

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列の先頭からオブジェクト my_id へコピーを行い，その結果を標準出力します.

```
stdstreamio sio;
tstring     my_id(4);
tstring     my_str   = "User ID : 1234";
int         i_ret    = 0;

if ((i_ret = my_str.copy(10, 4, &my_id)) < 0 ) {
    エラー処理
}
else if ( 4 < i_ret ) {
    sio.printf("The length of buffer is insufficient. \n");
}
else {
```

```
        sio.printf("%s\n", my_id.cstr());
    }
```

実行結果

```
1234
```

9.5.11 swap()

NAME

swap() — オブジェクトのスワップ

SYNOPSIS

```
tstring &swap( tstring &sobj );
```

DESCRIPTION

オブジェクト sobj の内容と自身の内容とを入れ替えます．この時，動作モード (§9.1参照) は入れ替わりません (EXAMPLE 参照).

PARAMETER

[I/O] sobj 自身と内容を入れ替えるオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，オブジェクト my_str1，my_str2 を通常モードで，オブジェクト my_str3 を最大文字列長 4 を指定した固定長バッファモードで作成しています．そして，これらオブジェクトの持つ文字列が swap() によって入れ替わる事を確認するため，標準出力します.

```
    stdstreamio sio;
    tstring     my_str1;
    tstring     my_str2;
    tstring     my_str3(4);

    my_str1 = "ISS/Kibo";
    my_str2 = "JAXA/ISAS";
    my_str3 = "NASA";

    my_str1.swap(my_str2);
    sio.printf("%s\n", my_str1.c_str());
    sio.printf("%s\n", my_str2.c_str());
```

```
my_str1.swap(my_str3);
sio.printf("%s\n", my_str1.c_str());
sio.printf("%s\n", my_str3.c_str());
```

実行結果

```
JAXA/ISAS
ISS/Kibo
NASA
JAXA
```

9.5.12 init()

NAME

init() — オブジェクトの完全初期化

SYNOPSIS

`tstring &init();` 1
`tstring &init( const tstring &src );` 2

DESCRIPTION

自身の初期化を行います.

メンバ関数 1 は，オブジェクトを完全に初期化します．オブジェクトの動作モードが NULL なしモードまたは固定長バッファモードの場合，メモリ領域は保持したまま，文字列バッファの初期化を行います．通常モードの場合，オブジェクト内の文字列バッファに割り当てられたメモリ領域は完全に開放されるため，`cstr()` メンバ関数 (§9.5.3) を実行すると `NULL` が返ります.

メンバ関数 2 は `src` の内容で自身を初期化します.

PARAMETER

[I] `src` コピー元となる文字列を持つオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合 (メンバ関数 2)

EXAMPLE-1

次のコードは，オブジェクト `my_str` が持つ文字列が，`init()` によって変化する事を確認するため，その結果を標準出力します.

```
stdstreamio   sio;
tstring       my_str      = "User'sFile01.txt";
const tstring my_sentence = "JaxaData.txt";
```

```
sio.printf("%s\n", my_str.cstr());

my_str.init(my_sentence);

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
User'sFile01.txt
JaxaData.txt
```

EXAMPLE-2

次のコードは，オブジェクト my_str が持つ文字列が，init() によって完全に初期化する事を確認するため，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "User'sFile01.txt";

sio.printf("%s\n",my_str.cstr());

my_str.init();

sio.printf("%s\n",my_str.cstr());
```

実行結果

```
User'sFile01.txt
(null)
```

9.5.13 printf(), vprintf(), assign(), assignf(), vassignf()

NAME

printf(), vprintf(), assign(), assignf(), vassignf() — オブジェクトの初期化

SYNOPSIS

```
tstring &printf( const char *format, ... );  .................................. 1
tstring &vprintf( const char *format, va_list ap );  ......................... 2
tstring &assign( int ch, size_t n );  ......................................... 3
tstring &assign( const char *str );  .......................................... 4
tstring &assign( const char *str, size_t n );  ............................... 5
tstring &assignf( const char *format, ... );  ................................ 6
tstring &vassignf( const char *format, va_list ap );  ....................... 7
tstring &assign( const tstring &src, size_t pos2 = 0 );  ................... 8
tstring &assign( const tstring &src, size_t pos2, size_t n2 );  ........... 9
```

DESCRIPTION

自身の文字列を，引数で指定された文字列で初期化します．

メンバ関数 1，2，6 およびメンバ関数 7 は，format に従って作成した文字列を用いて初期化します．メンバ関数 1 およびメンバ関数 6 では，可変長引数の各要素データを，メンバ関数 2 およびメンバ関数 7 では，可変長引数のリスト ap を format の変換指定に応じて変換します．format については §8.1.14の解説を参照してください．

メンバ関数 3 は，n 文字の文字 ch からなる文字列で，自身の文字列バッファを初期化します．

メンバ関数 4 およびメンバ関数 5 は，文字列 str で自身の文字列バッファを初期化します．さらにメンバ関数 5 は，初期化に用いる文字列の長さ n を指定できます．n が str の文字列長以上の場合は，str の全文字列を書き込み対象とします．

メンバ関数 8 およびメンバ関数 9 は，オブジェクト src が持つ文字列の位置 pos2 からの文字列を用いて自身を初期化します．なお，文字列の先頭位置は 0 です．メンバ関数 8 は，pos2 は指定しなくても使用できます．その場合は 0 を指定したものとして処理を行います．メンバ関数 9 は，初期化に用いる文字列の長さ n2 を指定できます．

PARAMETER

[I]	format	源泉となる文字列のためのフォーマット指定
[I]	...	format に対応した可変長引数の各要素データ
[I]	ap	format に対応した可変長引数のリスト
[I]	ch	源泉となる文字
[I]	str	源泉となる文字列
[I]	n	ch の個数，または str の長さ
[I]	src	源泉となる文字列を持つ tstring クラスのオブジェクト
[I]	pos2	src が持つ文字列の開始位置 (src の部分文字列を代入する場合)
[I]	n2	書き込まれる文字列の長さ (src の部分文字列の代入をする場合)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

可変長引数の各要素データが指定された変換フォーマットで変換できない値の場合 (メンバ関数 1，2，6，7)

EXAMPLE-1

次のコードは，オブジェクト my_str が持つ文字列が printf() により初期化する事を確認するため，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str;

sio.printf("%s\n",my_str.cstr());

my_str.printf("%s", "TEST_OK");

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
(null)
TEST_OK
```

EXAMPLE-2

このコードは，オブジェクト my_str が持つ文字列に my_sentence の文字列の 11 番目の文字 *e* から 5 文字を用いて my_str を初期化し，その結果を標準出力します.

```
stdstreamio   sio;
tstring       my_str;
const tstring my_sentence = "This is an eraser.";

sio.printf("%s\n", my_str.cstr());

my_str.assign(my_sentence, 11, 5);

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
(null)
erase
```

9.5.14 implode()

NAME

implode() — 文字列配列を区切り文字で結合した文字列をセット

SYNOPSIS

```
tstring &implode( const char *const *arr, const char *delim );
```

DESCRIPTION

引数 arr で指定された文字列配列 (NULL 終端のポインタ配列) を区切り文字列 delim で結合して 1 つの文字列とし，それを自身に格納します.

arr が NULL の場合は何もしません.

PARAMETER

[I]	arr	文字列配列 (NULL 終端のポインタ配列)
[I]	delim	区切り文字列

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

オブジェクト内のバッファの確保に失敗した場合.

EXAMPLE

次のコードは，文字列配列 arr の全要素を区切り文字「,」で結合し，オブジェクト my_str に格納します．結果を確認するため，my_str の内容を標準出力します．

```
stdstreamio sio;
const char *arr[] = {"X1", "MZ2861", "X68k", NULL};
tstring my_str;

my_str.implode(arr, ",");

sio.printf("%s\n", my_str.cstr());
```

実行結果

X1,MZ2861,X68k

9.5.15 import_binary()

NAME

import_binary() — バイナリデータの取り込み

SYNOPSIS

```
tstring &import_binary( const char *buf, size_t bufsize, int altchr = '\0' );
```

DESCRIPTION

内部で this->resize(bufsize) を呼び出した後，buf で指定したバッファから bufsize バイトを読み込み，自身に格納します．altchr に'\0' 以外が指定された場合は，バッファに文字'\0' が含まれていた場合は，altchr に置き換えてから格納します．

altchr を省略すると，バッファ中の文字'\0' をそのまま格納する事ができますが，文字列を検索したりパターンマッチを行うメンバ関数は動作が保証されません．

PARAMETER

[I]	buf	ユーザバッファのアドレス
[I]	bufsize	ユーザバッファのサイズ
[I]	altchr	ユーザバッファに文字'\0' が存在した場合に置き換える文字

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

オブジェクト内のバッファの確保に失敗した場合．

9.5.16 put(), putf(), vputf()

NAME

put(), putf(), vputf() — 任意の位置へ文字または文字列をセット

SYNOPSIS

```
tstring &put( size_t pos1, int ch, size_t n );  ............................ 1
tstring &put( size_t pos1, const char *str );  ............................. 2
tstring &put( size_t pos1, const char *str, size_t n );  ................... 3
tstring &putf( size_t pos1, const char *format, ... );  .................... 4
tstring &vputf( size_t pos1, const char *format, va_list ap );  ........... 5
tstring &put( size_t pos1, const tstring &src, size_t pos2 = 0 );  ....... 6
tstring &put( size_t pos1, const tstring &src, size_t pos2, size_t n2 );   7
```

DESCRIPTION

自身の文字列を位置 pos1 から，引数で指定された文字列で上書きします．なお，オブジェクト内の文字列および指定された文字列の先頭位置は 0 です．

pos1 は任意の値を取る事ができます．引数の指定に対して自身の文字列バッファが不足している場合は，自動的にバッファのサイズを拡張します．この時，追加したバッファを空白文字' ' でパディングし，pos1 の位置に引数で指定された文字列が書き込まれます．ただし，動作モードが固定長バッファモードの場合は，オブジェクト作成時の最大文字列長以上の拡張は行ないません．例えば，最大文字列長以上の pos1 を指定した場合，文字列の終端からバッファサイズ一杯まで空白文字をパディングし，引数で指定した文字，文字列の書き込みは行いません．

メンバ関数 1 は，n 文字の文字 ch からなる文字列を，自身が持つ文字列の位置 pos1 に書き込みます．

メンバ関数 2 およびメンバ関数 3 は，文字列 str を自身が持つ文字列の位置 pos1 に書き込みます．さらにメンバ関数 3 は，書き込む文字列の長さ n を指定できます．n が str の文字列長以上の場合は，str の全文字列を書き込み対象とします．

メンバ関数 4 およびメンバ関数 5 は，format に従って作成した文字列を書き込みます．メンバ関数 4 では，可変長引数の各要素データを，メンバ関数 5 では，可変長引数のリスト ap を format の変換指定に応じて変換します．format については§8.1.14の解説を参照してください．

メンバ関数 6 およびメンバ関数 7 は，オブジェクト src が持つ文字列の位置 pos2 からの文字列を，自身の文字列に書き込みます．メンバ関数 6 は pos2 を指定しなくても使用できます．その場合は 0 を指定したものとして処理を行います．メンバ関数 7 は，書き込まれる文字列の長さ n2 を指定できます．

PARAMETER

[I]	pos1	オブジェクト内の文字列の開始位置
[I]	ch	源泉となる文字
[I]	n	ch の個数，または str の長さ
[I]	str	源泉となる文字列
[I]	format	源泉となる文字列のためのフォーマット指定
[I]	...	format に対応した可変長引数の各要素データ
[I]	ap	format に対応した可変長引数のリスト
[I]	src	源泉となる文字列を持つ tstring クラスのオブジェクト
[I]	pos2	src が持つ文字列の開始位置 (src の部分文字列を代入する場合)
[I]	n2	書き込まれる文字列の長さ (src の部分文字列の代入をする場合)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

可変長引数の各要素データが指定された変換フォーマットで変換できない値の場合 (メンバ関数 4，5)

EXAMPLE

次のコードは，my_sentence の先頭から 4 文字をオブジェクト my_str が持つ文字列の 6 番目の位置へ書き込み，その結果を標準出力します.

```
stdstreamio   sio;
tstring       my_str      = "User'sFile01.txt";
const tstring my_sentence = "Data";

my_str.put(6, my_sentence, 0, 4);

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
User'sData01.txt
```

9.5.17 strcat(), strncat(), append(), appendf(), vappendf()

NAME

strcat(), strncat(), append(), appendf(), vappendf() — 文字または文字列の追加

SYNOPSIS

```
tstring &strcat( const char *str ); ........................................ 1
tstring &strncat( const char *str, size_t n ); ............................. 2
tstring &strcat( const tstring &src, size_t pos2 = 0 ); .................... 3
tstring &strncat( const tstring &src, size_t pos2, size_t n2 ); ........... 4
tstring &append( int ch, size_t n ); ....................................... 5
tstring &append( const char *str ); ........................................ 6
tstring &append( const char *str, size_t n); ............................... 7
tstring &appendf( const char *format, ... ); ............................... 8
tstring &vappendf( const char *format, va_list ap ); ....................... 9
tstring &append( const tstring &src, size_t pos2 = 0 ); ................... 10
tstring &append( const tstring &src, size_t pos2, size_t n2 ); ............ 11
```

DESCRIPTION

自身の文字列に，引数で指定された文字列を追加します.

動作モードが固定長バッファモードの場合は，オブジェクト作成時の最大文字列長以上の拡張を行いません．従って，すでに最大文字列長に達している場合，文字列の追加中に最大文字列長に達した場合は，それ以降の処理は行ないません.

メンバ関数 1，2，6 およびメンバ関数 7 は，文字列 `str` を自身が持つ文字列の終端に追加します．さらにメンバ関数 2 およびメンバ関数 7 は，追加する文字列の長さ `n` を指定できます．`n` が `str` の文字列長以上の場合は，`str` の全文字列を書き込み対象とします．

メンバ関数 3，4，10 およびメンバ関数 11 は，オブジェクト `src` が持つ文字列の位置 `pos2` からの文字列を，自身の文字列の終端に追加します．なお，文字列の先頭位置は 0 です．メンバ関数 3 およびメンバ関数 10 は，`pos2` を指定しなくても使用できます．その場合は 0 を指定したものとして処理を行います．メンバ関数 4 およびメンバ関数 11 は，追加する文字列の長さ `n2` を指定できます．

メンバ関数 5 は，`n` 文字の文字 `ch` からなる文字列を，自身が持つ文字列の終端に追加します．

メンバ関数 8 およびメンバ関数 9 は，`format` に従って作成した文字列を追加します．メンバ関数 8 では，可変長引数の各要素データを，メンバ関数 9 では，可変長引数のリスト `ap` を `format` の変換指定に応じて変換します．`format` については §8.1.14の解説を参照してください．

PARAMETER

[I]	`str`	源泉となる文字列
[I]	`n`	`ch` の個数，または `str` の長さ
[I]	`src`	源泉となる文字列を持つ tstring クラスのオブジェクト
[I]	`pos2`	`src` が持つ文字列の開始位置 (`src` の部分文字列を追加する場合)
[I]	`n2`	追加する文字列の長さ (`src` の部分文字列の追加をする場合)
[I]	`ch`	源泉となる文字
[I]	`format`	源泉となる文字列のためのフォーマット指定
[I]	`...`	`format` に対応した可変長引数の各要素データ
[I]	`ap`	`format` に対応した可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

可変長引数の各要素データが指定された変換フォーマットで変換できない値の場合 (メンバ関数 8，9)

EXAMPLE

次のコードは，オブジェクト `my_str` が持つ文字列に `my_suffix` の内容を追加し，その結果を標準出力します．

```
stdstreamio   sio;
tstring       my_str    = "200X1231";
const tstring my_suffix = ".txt";

my_str.append(my_suffix);
sio.printf("%s\n", my_str.cstr());
```

実行結果

```
200X1231.txt
```

9.5.18 insert(), insertf(), vinsertf()

NAME

insert(), insertf(), vinsertf() — 文字または文字列の挿入

SYNOPSIS

```
tstring &insert( size_t pos1, int ch, size_t n );  ......................... 1
tstring &insert( size_t pos1, const char *str );  .......................... 2
tstring &insert( size_t pos1, const char *str, size_t n );  ................ 3
tstring &insertf( size_t pos1, const char *format, ... );  ................. 4
tstring &vinsertf( size_t pos1, const char *format, va_list ap );  ....... 5
tstring &insert( size_t pos1, const tstring &src, size_t pos2 = 0 );  ..... 6
tstring &insert( size_t pos1, const tstring &src,size_t pos2, size_t n2 );  7
```

DESCRIPTION

自身の文字列の位置 pos1 に，引数で指定された文字列を挿入します．なお，オブジェクト内の文字列および指定された文字列の先頭位置は 0 です．

メンバ関数 1 は，n 文字の文字 ch からなる文字列を，自身が持つ文字列の位置 pos1 に挿入します．

メンバ関数 2 およびメンバ関数 3 は，文字列 str を自身が持つ文字列の位置 pos1 に挿入します．さらにメンバ関数 3 は，挿入する文字列の長さ n を指定できます．

メンバ関数 4 およびメンバ関数 5 は，format に従って作成した文字列を挿入します．メンバ関数 4 では，可変長引数の各要素データを，メンバ関数 5 では，可変長引数のリスト ap を format の変換指定に応じて変換します．format については §8.1.14の解説を参照してください．

メンバ関数 6 およびメンバ関数 7 は，オブジェクト src が持つ文字列の位置 pos2 からの文字列を，自身が持つ文字列の位置 pos1 に挿入します．メンバ関数 6 の pos2 は指定しなくても使用できます．その場合は 0 を指定したものとして処理を行います．メンバ関数 7 は，挿入する文字列の長さ n2 を指定できます．

PARAMETER

[I]	pos1	オブジェクト内の文字列の開始位置
[I]	ch	源泉となる文字
[I]	n	ch の個数または str の長さ
[I]	str	源泉となる文字列
[I]	format	源泉となる文字列のためのフォーマット指定
[I]	...	format に対応した可変長引数の各要素データ
[I]	ap	format に対応した可変長引数のリスト
[I]	src	源泉となる文字列を持つ tstring クラスのオブジェクト
[I]	pos2	src が持つ文字列の開始位置 (src の部分文字列を代入する場合)
[I]	n2	挿入される文字列の長さ (src の部分文字列の代入をする場合)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

可変長引数の各要素データが指定された変換フォーマットで変換できない値の場合 (メンバ関数 4，5)

EXAMPLE

次のコードは，フォーマットに従いオブジェクト my_str が持つ文字列に文字を挿入し，その結果を標準出力します.

```
stdstreamio sio;
tstring     my_str = "123";

sio.printf("%s\n", my_str.cstr());

my_str.insertf(1,"%c",'+');

sio.printf("%s\n", my_str.cstr());

my_str.insertf(3,"%c",'=');

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
123
1+23
1+2=3
```

9.5.19 replace(), replacef(), vreplacef()

NAME

replace(), replacef(), vreplacef() — 文字列の置換

SYNOPSIS

```
tstring &replace( size_t pos1, size_t n1, int ch, size_t n2 );  ........... 1
tstring &replace( size_t pos1, size_t n1, const char *str );  ............. 2
tstring &replace( size_t pos1, size_t n1, const char *str, size_t n2 );  . 3
tstring &replacef( size_t pos1, size_t n1, const char *format, ... );  .... 4
tstring &vreplacef( size_t pos1, size_t n1,
                    const char *format, va_list ap );  ...................... 5
tstring &replace( size_t pos1, size_t n1,
                  const tstring &src, size_t pos2 = 0 );  .................. 6
tstring &replace( size_t pos1, size_t n1, const tstring &src,
                  size_t pos2, size_t n2 );  ................................ 7
```

DESCRIPTION

自身の文字列の位置 pos1 から n1 文字を，指定された文字列で置き換えます．なお，オブジェクト内の文字列および指定された文字列の先頭位置は 0 です.

pos1 に自身の文字列長以上の値を指定した場合，stacat() メンバ関数 (§9.5.17) と同様の処理を行います．pos1 と n1 の和が自身の文字列長よりも大きい場合，また n1，n2 の大小関係により文字列の拡張，収縮が必要な場合は文字列を自動的に調整します．ただし，動作モードが固定長バッファモードの場合は，オブジェクト作成時の最大文字列長以上の拡張は行ないません．

メンバ関数 1 は，自身が持つ文字列の pos1 の位置から n1 文字を，n2 文字の文字 ch からなる文字列で置換します．

メンバ関数 2 およびメンバ関数 3 は，自身が持つ文字列の位置 pos1 から n1 文字を，文字列 str に置換します．さらにメンバ関数 3 では，文字列 str の最初の n2 文字を使う指定ができます．

メンバ関数 4 およびメンバ関数 5 は，format に従って作成した文字列を用いて置換します．メンバ関数 4 では，可変長引数の各要素データを，メンバ関数 5 では，可変長引数のリスト ap を format の変換指定に応じて変換します．format については §8.1.14の解説を参照してください．

メンバ関数 6 およびメンバ関数 7 は，自身が持つ文字列の位置 pos1 から n1 文字を，オブジェクト src が持つ文字列の位置 pos2 からの文字列に置換します．メンバ関数 6 は pos2 を指定しなくても使用できます．その場合は 0 を指定したものとして処理します．メンバ関数 7 では，オブジェクト src が持つ文字列 str の最初の n2 文字を使う指定ができます．

PARAMETER

[I]	pos1	オブジェクト内の文字列の開始位置
[I]	n1	置換する文字数
[I]	ch	源泉となる文字
[I]	n2	ch の個数または str，src が持つ文字列の長さ
[I]	str	源泉となる文字列
[I]	format	源泉となる文字列のためのフォーマット指定
[I]	...	format に対応した可変長引数の各要素データ
[I]	ap	format に対応した可変長引数のリスト
[I]	pos2	src が持つ文字列の開始位置 (src の部分文字列を代入する場合)
[I]	src	源泉となる文字列を持つ tstring クラスのオブジェクト

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
可変長引数の各要素データが指定された変換フォーマットで変換できない値の場合 (メンバ関数 4，5)

EXAMPLE-1

次のコードは，オブジェクト my_str が持つ文字列の 8 番目の文字 Y から 8 文字を指定したフォーマットに従って置換し，その結果を標準出力します．

```
stdstreamio sio;
```

```
tstring     my_str = "UserNameYYYYMMDD.txt";
time_t      jikoku;
struct      tm  *lt;

time(&jikoku);
lt = localtime(&jikoku);

my_str.replacef(8, 8, "%d%d%d", 1900+lt->tm_year, lt->tm_mon, lt->tm_mday);

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
UserName2009219.txt
```

EXAMPLE-2

次のコードは，オブジェクト my_str が持つ文字列の 11 番目の文字 *2* から 2 文字を 4 つの文字 *X* で置換し，その結果を標準出力します.

```
stdstreamio sio;
tstring     my_str = "User ID : 1234";
int         i_ch   = 'X';

my_str.replace(11, 2, i_ch, 4);

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
User ID : 1XXXX4
```

EXAMPLE-3

次のコードは，オブジェクト my_str が持つ文字列の 11 番目の文字 *S* から 5 文字を文字列 *Akar* で置換し，その結果を標準出力します.

```
stdstreamio sio;
tstring     my_str = "My name is Suzuki.";

my_str.replace(11, 5, "Akar", 4);

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
My name is Akari.
```

9.5.20 erase()

NAME

erase() — 文字列の消去

SYNOPSIS

`tstring &erase();` 1

`tstring &erase( size_t pos, size_t n = 1 );` 2

DESCRIPTION

自身が持つ文字列の文字を消去します.

メンバ関数 1 は，すべての文字を消去します (文字列長はゼロになります).

メンバ関数 2 は，位置 `pos` から `n` 文字を消去します．なお，文字列の先頭位置は 0 です．`n` が指定されない場合は，1 文字を消去します.

PARAMETER

[I]	`pos`	消去の開始位置
[I]	`n`	消去する文字数

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト `my_str` が持つ文字列の先頭 1 文字を消去し，その結果を標準出力します.

```
stdstreamio sio;
tstring     my_str(7);

my_str.init("sibuki");

sio.printf("%s\n", my_str.cstr());

my_str.erase(0);

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
sibuki
ibuki
```

9.5.21 clean()

NAME

clean() — 既存の文字列全体を任意の文字でパディング

SYNOPSIS

`tstring &clean( int ch = ' ' );`

DESCRIPTION

自身が持つ文字列全体を，文字 `ch` でパディングします．この `ch` は指定しなくても使用できます．その場合は，空白文字' 'を指定したものとして処理を行います．`clean()` を実行しても文字列長は変化しません．

PARAMETER

[I] `ch` 文字列をパディングする文字
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト `my_str` が持つ文字列を文字 * でパディングし，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "Akari20060222.txt";

my_str.clean('*');
sio.printf("%s\n", my_str.cstr());
```

実行結果

```
*****************
```

9.5.22 resize()

NAME

resize() — 文字列の長さを変更

SYNOPSIS

`tstring &resize( size_t len );`

DESCRIPTION

自身が持つ文字列の長さを `len` に変更します．

文字列長を拡張する場合，空白文字' 'からなる文字列が追加されます．

文字列長を収縮する場合，`len` 以降の文字列は削除されます．

PARAMETER

[I] `len` 変更後の文字列長
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト `my_str` が持つ文字列に 8 文字の文字列を代入した後，文字列長の長さを 3 に変更し，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str;

my_str = "USR TEST";
sio.printf("%s\n", my_str.cstr());

my_str.resize(3);

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
USR TEST
USR
```

9.5.23 resizeby()

NAME

resizeby() — 文字列の長さを相対的に変更

SYNOPSIS

```
tstring &resizeby( ssize_t len );
```

DESCRIPTION

自身が持つ文字列の長さを `len` の長さ分だけ変更します．

文字列長を拡張する場合，空白文字' 'からなる文字列が追加されます．

文字列長を収縮する場合，最後の abs(`len`) 個の文字列は削除されます．

PARAMETER

[I] `len` 文字列長の増分・減分
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

9.5.24 crop()

NAME

crop() — 文字列の切り抜き

SYNOPSIS

`tstring &crop( size_t pos );` 1
`tstring &crop( size_t pos, size_t n );` 2

DESCRIPTION

自身の文字列を，位置 pos から n 個の文字列だけにします．なお，文字列の先頭位置は 0 です．

メンバ関数 1 は，自身が持つ文字列の pos から終端まで切り抜きます．

メンバ関数 2 は，自身が持つ文字列の pos から n 文字を切り抜きます．pos と n の和が文字列長以上の場合，文字の切り抜きは pos 以降の文字列を対象とします．

文字列長以上の値を pos に指定した場合，文字列長は 0 になります．

PARAMETER

[I] pos 切り抜きの開始位置
[I] n 切り抜く文字数
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列の 5 番目の文字 *2* から 8 文字を切り抜き，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "Akari20060222.txt";

my_str.crop(5,8);
sio.printf("%s\n", my_str.cstr());
```

実行結果

20060222

9.5.25 chomp()

NAME

chomp() — 改行文字の除去

SYNOPSIS

```
tstring &chomp( const char *rs = "\n" );
tstring &chomp( const tstring &rs );
```

DESCRIPTION

自身が持つ文字列の右端の改行文字を除去します．改行文字は `rs` で指定します．

例えば，DOS 形式のテキストファイルを扱う場合には,「`str.chomp("\r\n");`」，UNIX 形式，Mac形式，DOS形式のすべての場合に対応させたい場合は,「`str.chomp("\n").chomp("\r");`」とします．

PARAMETER

[I] `rs` 改行文字列

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

9.5.26 trim()

NAME

trim() — 文字列の両端任意文字の除去

SYNOPSIS

```
tstring &trim( int side_space ); ............................................. 1
tstring &trim( const char *side_spaces = " \t\n\r\f\v" ); ................. 2
tstring &trim( const tstring &side_spaces ); .............................. 3
```

DESCRIPTION

自身が持つ文字列の両端にある任意文字を除去します．この任意文字は文字 `side_space` または文字列 `side_spaces` で指定します．

メンバ関数 1 は，自身が持つ文字列の両端にある `side_space` を除去します．

メンバ関数 2 の `side_spaces` は指定しなくても使用できます．その場合は，`"␣\t\n\r\f\v"` を指定したものとして，空白文字，水平タブ文字，改行文字，復帰文字，ファイル区切り文字，垂直タブ文字を除去します (␣ は空白文字を意味しています)．

メンバ関数 2 およびメンバ関数 3 の `side_spaces` は，`" \t"`のような単純な文字リストに加え，正規表現で用いられる `"[A-Z]"`あるいは`"[^A-Z]"`のような指定が可能です．さらに，`"[…]"` の中では，表 19に示す文字クラスが指定できます．

文字クラス	対応する文字	対応する libc の関数
[:alnum:]	英字または数字	isalnum()
[:alpha:]	英字	isalpha()
[:cntrl:]	制御文字	iscntrl()
[:digit:]	10 進数文字	isdigit()
[:graph:]	印字用文字 (空白文字除く)	isgraph()
[:lower:]	英小文字	islower()
[:print:]	印字用文字 (空白文字含む)	isprint()
[:punct:]	句読点文字	ispunct()
[:space:]	空白文字	isspace()
[:upper:]	英大文字	isupper()
[:xdigit:]	16 進数文字	isxdigit()

表 19: [···] で使用できる文字クラス一覧.

例えば，"[[:digit:]]"は"[0-9]"と等価です．その他については，ロケールに依存するものもありますので，対応する libc の関数マニュアルを参照してください．

なお，"[0-9]abcdef"のような表現はできません．"[0-9a-f]"あるいは"0123456789abcdef"のようにします．すなわち，side_spaces は'['で始まる場合は，']'で終わらなければなりません．

PARAMETER

[I] side_space 任意文字
[I] side_spaces 任意文字列
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列から文字列両端にある任意文字を除去し，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "\tThis is a pen. \n";

my_str.trim();
sio.printf("%s\n", my_str.cstr());
```

実行結果

```
This is a pen.
```

9.5.27 ltrim()

NAME

ltrim() — 文字列の左端スペースの除去

SYNOPSIS

```
tstring &ltrim( int side_space );
tstring &ltrim( const char *side_spaces = " \t\n\r\f\v" );
tstring &ltrim( const tstring &side_spaces );
```

DESCRIPTION

自身が持つ文字列の左端の空白文字を除去します．空白文字は side_space または side_spaces で指定します．

side_spaces は，" \t"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]"あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，次に示す文字クラスが指定できます:

[:alnum:], [:alpha:], [:cntrl:], [:digit:], [:graph:], [:lower:], [:print:], [:punct:], [:space:], [:upper:], [:xdigit:].

例えば，"[[:digit:]]"は"[0-9]"と等価です．その他については，ロケールに依存するものもありますので，libc の isalpha() 関数のマニュアルを参照してください．

なお，"[0-9]abcdef"のような表現はできません．"[0-9a-f]"あるいは"0123456789abcdef"のようにします．すなわち，side_spaces が '[' で始まる場合は，']' で終わらなければなりません．

RETURN VALUE

自身の参照

9.5.28 rtrim()

NAME

rtrim() — 文字列の右端スペースの除去

SYNOPSIS

```
tstring &rtrim( int side_space );
tstring &rtrim( const char *side_spaces = " \t\n\r\f\v" );
tstring &rtrim( const tstring &side_spaces );
```

DESCRIPTION

自身が持つ文字列の右端の空白文字を除去します．空白文字は side_space または side_spaces で指定します．

side_spaces は，" \t"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]"あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，次に示す文字クラスが指定できます:

[:alnum:], [:alpha:], [:cntrl:], [:digit:], [:graph:], [:lower:], [:print:], [:punct:], [:space:], [:upper:], [:xdigit:].

例えば，"[[:digit:]]"は"[0-9]"と等価です．その他については，ロケールに依存するものもありますので，libc の isalpha() 関数のマニュアルを参照してください．

なお，"[0-9]abcdef"のような表現はできません．"[0-9a-f]"あるいは"0123456789abcdef"のようにします．すなわち，side_spaces が '[' で始まる場合は，']' で終わらなければなりません．

RETURN VALUE

自身の参照

9.5.29 strreplace()

NAME

strreplace() — 文字列の検索と置換

SYNOPSIS

```
ssize_t strreplace( const char *org_str, const char *new_str, bool all = false );
ssize_t strreplace( size_t pos, const char *org_str, const char *new_str,
                    bool all = false );
ssize_t strreplace( const tstring &org_str, const char *new_str,
                    bool all = false );
ssize_t strreplace( size_t pos, const tstring &org_str, const char *new_str,
                    bool all = false );
ssize_t strreplace( const char *org_str, const tstring &new_str,
                    bool all = false );
ssize_t strreplace( size_t pos, const char *org_str, const tstring &new_str,
                    bool all = false );
ssize_t strreplace( const tstring &org_str, const tstring &new_str,
                    bool all = false );
ssize_t strreplace( size_t pos, const tstring &org_str, const tstring &new_str,
                    bool all = false );
```

DESCRIPTION

自身が持つ文字列の左側から文字列 org_str を検索し，見つかった場合は文字列 new_str で置き換えます．

これらのメンバ関数は，置換が行なわれた場合は置換された文字列の次の位置を返します．この返り値を pos に与えれば，次に org_str が見つかった部分を置換できます．

最後の引数 all が true の場合は，マッチするすべての部分を new_str で置き換えます．

より高度な検索処理を必要とする場合，regreplace() メンバ関数 (§9.5.30) で拡張正規表現を使った置換が行えます．

PARAMETER

[I]	org_str	検出する文字列
[I]	new_str	置換の源泉となる文字列
[I]	pos	文字列検索の開始位置
[I]	all	全置換のフラグ

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定した文字列が見つかった場合，置換された文字列の次の位置
負の値 (エラー)	:	指定した文字または文字列が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	org_str または new_str が NULL の場合
	:	pos にオブジェクト内の文字列長を越える値を指定した場合

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_url が持つ URL 文字列のホスト名 darts.isas.jaxa.jp の部分を置換するコードです.

```
stdstreamio sio;
tstring my_url = "http://darts.isas.jaxa.jp/foo/";

my_url.strreplace("darts.isas.jaxa.jp", "darts.jaxa.jp");
sio.printf("my_url = %s\n", my_url.cstr());
```

実行結果

```
http://darts.jaxa.jp/foo/
```

9.5.30 regreplace()

NAME

regreplace() — 拡張正規表現でマッチした部分の置換を行なう

SYNOPSIS

```
ssize_t regreplace( const char *pat,
                    const char *new_str, bool all = false );  ............... 1
ssize_t regreplace( size_t pos, const char *pat,
                    const char *new_str, bool all = false );  ............... 2
ssize_t regreplace( const tstring &pat,
                    const char *new_str, bool all = false );  ............... 3
ssize_t regreplace( size_t pos, const tstring &pat,
                    const char *new_str, bool all = false );  ............... 4
ssize_t regreplace( const tregex &pat,
                    const char *new_str, bool all = false );  ............... 5
ssize_t regreplace( size_t pos, const tregex &pat,
                    const char *new_str, bool all = false );  ............... 6
ssize_t regreplace( const char *pat,
                    const tstring &new_str, bool all = false );  ........... 7
ssize_t regreplace( size_t pos, const char *pat,
                    const tstring &new_str, bool all = false );  ........... 8
ssize_t regreplace( const tstring &pat,
```

```
                       const tstring &new_str, bool all = false );  .......... 9
ssize_t regreplace( size_t pos, const tstring &pat,
                       const tstring &new_str, bool all = false );  .......... 10
ssize_t regreplace( const tregex &pat,
                       const tstring &new_str, bool all = false );  .......... 11
ssize_t regreplace( size_t pos, const tregex &pat,
                       const tstring &new_str, bool all = false );  .......... 12
```

DESCRIPTION

自身の文字列に対し，pat で指定された POSIX 拡張正規表現 (以下，正規表現) でマッチした部分を文字列 new_str で置き換えます．new_str では，後方参照"\\0"〜"\\9" が利用できます ("\\0"はマッチした部分全体を示します)．バックスラッシュ自身を与えたい場合は，"\\\\" を指定します．

regreplace() メンバ関数は，正規表現マッチ結果が格納されるオブジェクト内部バッファを更新します．引数 all が false の場合，保存された正規表現マッチの結果は，メンバ関数 reg_elem_length()，reg_pos()，reg_length()，reg_cstr()，reg_cstrarray() を使って取得する事ができます．それぞれ，結果の要素数，マッチした部分の位置，マッチした部分の文字列長，マッチした部分の文字列，マッチした部分の文字列に対するポインタ配列を返します．これらのメンバ関数のプロトタイプは次の通りです．

```
size_t reg_elem_length() const;
size_t reg_pos( size_t idx ) const;
size_t reg_length( size_t idx ) const;
const char *reg_cstr( size_t idx ) const;
const char *const *reg_cstrarray() const;
```

引数 idx には，0 から始まる要素番号を指定します．0 番目には，マッチした文字列全体の情報が格納され，1 番目からは，正規表現「(...)」それぞれにマッチした部分文字列の情報が格納されます (つまり，後方参照のための情報です)．reg_cstrarray() メンバ関数の返り値を，tarray_tstring クラス (§10) のオブジェクトに=演算子で代入する事もできます．

引数 all が true の場合は，正規表現マッチ結果がリセットされるので結果情報を取得できません．

メンバ関数 1〜4，7〜10 の場合，正規表現 pat をコンパイルし，その結果を自身が持つ内部バッファに保存し，マッチを行ないます (pat が前回と同じ場合は，再コンパイルしません)．

メンバ関数 5,6,11,12 の場合，正規表現のコンパイル結果を保持している tregex クラスのオブジェクトを指定します．したがって，regmatch() メンバ関数を使う前に，あらかじめ tregex クラスの compile() メンバ関数で正規表現をコンパイルする必要があります (§9.5.59の EXAMPLE-2 を参照)．

いずれの場合も，正規表現のコンパイルに失敗すると，標準エラー出力にその内容を出力します．

自身が持つ文字列の位置 pos から右方向に文字列マッチを試行します．文字列マッチが試行される対象範囲は，文字列終端の'\0' が現れるまでです (改行文字'\n' が現れても処理は終了しません)．pos が指定されない場合は，自身の文字列の左端から探します．なお，文字列の先頭位置は 0 です．

これらのメンバ関数は，置換が行なわれた場合は置換された文字列の次の位置を返します．この返り値を pos に与えれば，次にマッチする部分を置換できます.

最後の引数 all が true の場合は，マッチするすべての部分を new_str で置き換えます.

正規表現の詳細は，§9.5.59を参照してください.

より単純な検索による置換を行なう場合は，処理速度で有利な strreplace() メンバ関数 (§9.5.29) を推奨します.

PARAMETER

[I]	pat	文字パターン (正規表現) または tregex クラスのコンパイル済オブジェクト
[I]	new_str	置換後の文字列
[I]	pos	文字列マッチの開始位置
[I]	all	全置換のフラグ

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	置換された文字列の次の位置
負の値 (エラー)	:	pat にマッチしなかった場合
	:	オブジェクト内に文字列がない場合
	:	pos にオブジェクト内の文字列長を越える値を指定した場合
	:	pat または new_str が NULL である場合
	:	インターバルオペレータ{}が閉じていない場合
	:	リストオペレータ [] が閉じていない場合
	:	未知のキャラクタクラスを設定した場合 ⟦例えば [: up :] の場合が該当します.⟧
	:	正規表現がバックスラッシュで終わっている場合
	:	グループオペレータ () が閉じていない場合
	:	無効な範囲のオペレータの使用の場合 ⟦例えば [9 – 0] の場合が該当します.⟧
	:	サブエクスプレッション\(...\) への無効な後方参照をした場合
	:	無効な後方参照オペレータを使用した場合
	:	グループやリストなどの，パターンオペレータの無効な使用を行った場合 ⟦例えば [0 – 9) の場合が該当します.⟧
	:	'*' が最初の文字としてくるような，無効な繰り返しオペレータを使用した場合 ⟦例えば $pat = " * .txt"$ の場合が該当します.⟧

EXCEPTION

regex ルーチンがメモリを使い果たしている場合
内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，オブジェクト my_url が持つ URL 文字列のホスト名の部分だけを置換するコードです.

```
stdstreamio sio;
tstring my_url = "http://darts.isas.jaxa.jp/foo/";
```

```
    if ( my_url.regreplace("(http://)([^/]+)", "\\1darts.jaxa.jp") < 0 ) {
        エラー処理
    }
    else {
        sio.printf("my_url = %s\n", my_url.cstr());
    }
```

実行結果

```
http://darts.jaxa.jp/foo/
```

9.5.31 tolower()

NAME

tolower() — 大文字を小文字に変換

SYNOPSIS

```
tstring &tolower( size_t pos = 0 );  ........................................ 1
tstring &tolower( size_t pos, size_t n );  .................................. 2
```

DESCRIPTION

自身の文字列が持つアルファベットの大文字を小文字に変換します．なお，文字列の先頭位置は 0 です．

メンバ関数 1 は，自身の文字列の位置 pos から右方向に向かって処理を行います．この pos は指定しなくても使用できます．その場合は 0 を指定したものとして処理を行います．

メンバ関数 2 は，自身の文字列の位置 pos から n 文字の大文字を小文字に変換します．

PARAMETER

[I]	pos	変換の開始位置
[I]	n	変換する文字数

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列を小文字に変換し，その結果を標準出力します．

```
    stdstreamio sio;
    tstring     my_str = "JAXA/ISAS";

    my_str.tolower();

    sio.printf("%s\n", my_str.cstr());
```

実行結果

```
jaxa/isas
```

9.5.32 toupper()

NAME

toupper() — 小文字を大文字に変換

SYNOPSIS

tstring &toupper(size_t pos = 0); 1
tstring &toupper(size_t pos, size_t n); 2

DESCRIPTION

自身の文字列が持つアルファベットの小文字を大文字に変換します．なお，文字列の先頭位置は 0 です．

メンバ関数 1 は，自身の文字列の位置 pos から右方向に向かって処理を行います．この pos は指定しなくても使用できます．その場合は 0 を指定したものとして処理を行います．

メンバ関数 2 は，自身の文字列の位置 pos から n 文字の小文字を大文字に変換します．

PARAMETER

[I] pos 変換の開始位置
[I] n 変換する文字数
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列の 5 番目の文字 i から 4 文字を大文字に変換し，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "jaxa/isas";

my_str.toupper(5,4);

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
jaxa/ISAS
```

9.5.33 expand_tabs()

NAME

expand_tabs() — 水平タブ文字を空白文字に置換

SYNOPSIS

tstring &expand_tabs(size_t tab_width = 8);

DESCRIPTION

自身が持つ文字列の水平タブ文字'\t'を，tab_width の値に桁揃えをして空白文字に置換します．置換の対象範囲は，文字列終端の'\0'が現れるまでです (改行文字'\n'が現れた場合，桁揃えに利用する内部の桁番号がリセットされ，引き続き処理が実行されます)．桁揃えする必要がない場合は，strreplace() メンバ関数 (§9.5.29) や regreplace() メンバ関数 (§9.5.30) を用いてタブ文字を置換することもできます．

tab_width を指定しなかった場合，また，tab_width に 0 を指定した場合は，tab_width に 8 を設定したものとして処理を行います．

tab_width の変更でオブジェクト内の文字列長が大きくなる場合，自動的にバッファのサイズを拡張します．ただし，動作モードが固定長バッファモードの場合は，オブジェクト作成時の最大文字列長以上の拡張は行ないません (EXAMPLE 参照)．

例えば，文字列"a\tbc\tdef がある場合，tab_width=3 で expand_tabs() を実行すると "a␣␣bc␣def" に変換されます (␣ は空白文字を意味しています)．この例では，1 つ目の水平タブ文字を，2 つの空白文字に置換しています．これは水平タブ文字前までの文字列の文字数 1("a") と空白文字の和を，tab_width に桁揃えしているためです．同様に 2 つ目の水平タブ文字前までの文字列の文字数 5 ("a␣␣bc") と空白文字の和が tab_width に桁揃えするために，空白文字 1 つが必要です．従って，2 つ目の水平タブ文字は 1 つの空白文字に置換します．

PARAMETER

[I] tab_width TAB 幅

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_str1 を通常モードで，オブジェクト my_str2 は最大文字列長 11 を指定した固定長バッファモードで作成しています．そして，これらオブジェクトが持つ文字列内の水平タブ文字が，expand_tabs() によって空白文字に置換される際 TAB 幅で調整される事を確認するため，標準出力します．

```
stdstreamio sio;
tstring     my_str1;
tstring     my_str2(11);

my_str1 = "Akari\tIbuki";
my_str2 = "Akari\tIbuki";

sio.printf("%s, %zu\n", my_str1.cstr(), my_str1.length());
sio.printf("%s, %zu\n", my_str2.cstr(), my_str2.length());

my_str1.expand_tabs();
```

```
    my_str2.expand_tabs();

    sio.printf("%s, %zu\n", my_str1.cstr(), my_str1.length());
    sio.printf("%s, %zu\n", my_str2.cstr(), my_str2.length());
```

実行結果

```
Akari Ibuki,11
Akari Ibuki,11
Akari   Ibuki,13(Akari␣␣␣Ibuki)
Akari   Ibu,11(Akari␣␣␣Ibu)
```

(␣ は空白文字を意味しています．)

9.5.34 contract_spaces()

NAME

contract_spaces() — 空白文字を TAB 文字に置換

SYNOPSIS

```
tstring &contract_spaces( size_t tab_width = 8 );
```

DESCRIPTION

自身が持つ文字列の 2 文字以上連続した空白文字' ' すべてを対象にし，指定した TAB 幅 `tab_width` で桁揃えして'\t' で置換します．ただし，置換の対象範囲は，文字列終端の'\0' が現れるまでです (改行文字'\n' が現れた場合，桁揃えに利用する内部の桁番号がリセットされ，引き続き処理が実行されます)．本メンバ関数は §9.5.33の反転動作を行ないます．桁揃えする必要がない場合は，`strreplace()` メンバ関数 (§9.5.29) や `regreplace()` メンバ関数 (§9.5.30) を用いて空白文字を置換することもできます．

`tab_width` を設定しなくても使用できます．その場合と，`tab_width` に 0 を設定した場合は，`tab_width` に 8 が設定されたものとして処理を行います．

例えば，文字列`"abc␣␣␣␣␣␣␣␣de"`(␣ は空白文字を意味しています) がある場合，`tab_width=4` で `contract_spaces()` を実行すると`"abc␣\t␣␣␣de"`に変換されます．この例では，空白文字の前までの文字数 3(`"abc"`) と空白文字の和を，`tab_width` に桁揃えするためには，空白文字 1 つが必要です．従って，1 つ目の空白文字は置換せずにそのまま残ります．続いて置換前の 4 個の空白文字を水平タブ文字に置換します．その後は，`tab_width` 個の空白文字がありませんので，水平タブ文字には置換していません．置換前に 8 個あった空白文字の変換後の内訳は，空白文字 1 つ + 水平タブ文字 1 つ (置換前の空白文字 4 つ分) + 空白文字 3 つです (EXAMPLE 参照)．

PARAMETER

[I] `tab_width` TAB 幅

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト `my_str` が持つ文字列の連続した空白文字が，`contract_spaces()` によって TAB 幅 4 で調整される事を確認するため，標準出力します．

```
stdstreamio sio;
tstring     my_str = "abc        de";

sio.printf("%s\n", my_str.cstr());

my_str.contract_spaces(4);

sio.printf("%s\n", my_str.cstr());
```

実行結果

```
abc        de(abc␣␣␣␣␣␣␣␣de)
abc    de(abc␣\t␣␣␣de)
```

WARNING

`tab_width=1` を設定した時の動作は未定義です．

9.5.35 atoi(), atol(), atoll()

NAME

atoi(), atol(), atoll() — 整数値に変換

SYNOPSIS

```
int atoi( size_t pos = 0 ) const; ........................................... 1
int atoi( size_t pos, size_t n ) const; ..................................... 2
long atol( size_t pos = 0 ) const; .......................................... 3
long atol( size_t pos, size_t n ) const; .................................... 4
long long atoll( size_t pos = 0 ) const; .................................... 5
long long atoll( size_t pos, size_t n ) const; .............................. 6
```

DESCRIPTION

自身が持つ文字列の位置 `pos` 以降の文字を，10 進の整数値に変換し，返します．`atoi()` は文字列を `int` 型整数に変換します．また同様に，`atol()` と `atoll()` は文字列をそれぞれ `long` 型整数，`long long` 型整数に変換します．自身の文字列に `[0-9]` 以外の文字がある場合 (ただし，先頭の符号は除く)，それ以降の文字は処理しません．なお，文字列の先頭位置は 0 です．

メンバ関数 1，3 およびメンバ関数 5 は，位置 `pos` を指定しなくても使用できます．その場合は 0 を指定したものとして処理を行います．

メンバ関数 2，4 およびメンバ関数 6 は，自身の文字列の位置 `pos` から `n` 文字を，整数値に変換します．

PARAMETER

[I] pos 自身が持つ文字列の変換開始位置
[I] n 整数値に変換する文字数
([I] : 入力, [O] : 出力)

RETURN VALUE

整数 : 変換された整数値

EXCEPTION

内部バッファの確保に失敗した場合 (メンバ関数 2, 4, 6)

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列の 2 番目以降の文字を int 型の整数に変換し，その結果を標準出力しています．

```
stdstreamio sio;
tstring     my_str = "1234abc567";

sio.printf("%d\n", my_str.atoi(2));
```

実行結果

34

WARNING

[0-9] 以外の文字が現れた時点で変換は終了します．全ての文字が変換できたかを確認する場合は，endpos 引数のある strtol() メンバ関数 (§9.5.37) を使用してください．

9.5.36 atof()

NAME

atof() — 実数値に変換

SYNOPSIS

```
double atof( size_t pos = 0 ) const; ........................................ 1
double atof( size_t pos, size_t n ) const; ................................ 2
```

DESCRIPTION

自身が持つ文字列の位置 pos 以降の文字を実数値に変換し，返します．オブジェクト内の文字列に実数値として取り扱う事が出来ない文字がある場合，それ以降の文字は処理しません．なお，文字列の先頭位置は 0 です．

実数値に変換する文字列は 10 進数，16 進数，無限，NAN(計算できない数) のいずれかがあります．10 進数は 1 文字以上の 10 進文字列からなっており，少数点を含む事ができます．10 進の指数部は'E' または'e' とその後に置かれる正負記号 (省略可)，およびその後に続く 1 文字以上の 10 進の数字列からなり，10 の何乗であるかを表します．また，FORTRAN 形式の倍精度指数表示 (例えば，1.2345D-10) にも対応しています (EXAMPLE-2 参照)．

16 進数は"0x"または"0X"とその後に続く 1 文字以上の 16 進の数字列からなり，小数点を含む事ができます．この後に 2 進の指数部を指定する事ができます．2 進の指数部は'P' または'p'

と，その後におかれる正負記号 (省略可)，およびその後に続く 1 文字以上の 10 進の数字の列から構成され，2 の何乗であるかを表します (EXAMPLE-3 参照)．小数点と 2 進の指数部は，どちらか一方しか存在できません．

無限は"INF"または"INFINITY"で表され，大文字小文字は区別されません．

NAN は"NAN"(大文字小文字は区別されない) で表され，その後に'(' 文字列')' が続く場合もあります．

メンバ関数 1 は，位置 pos を指定しなくても使用できます．その場合は 0 を指定したものとして処理を行います．

メンバ関数 2 は，自身の文字列の位置 pos から n 文字を実数値に変換します．

PARAMETER

[I]	pos	自身が持つ文字列の変換開始位置
[I]	n	実数値に変換する文字数

([I]：入力，[O]：出力)

RETURN VALUE

実数 ： 変換された double の値

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE-1

次のコードは，オブジェクト my_str が持つ文字列の 2 番目以降の文字を double 型の実数に変換し，その結果を標準出力しています．

```
stdstreamio sio;
tstring     my_str = "1234abc567";

sio.printf("%f\n",  my_str.atof(2));
```

実行結果

34.000000

EXAMPLE-2

次のコードは，オブジェクト my_str が持つ文字列の 1 番目の文字 *2* から 5 文字を double 型の実数に変換し，その結果を標準出力しています．

```
stdstreamio sio;
tstring     my_str = "123D-456";

sio.printf("%f\n",  my_str.atof(1,5));
```

実行結果

0.002300

EXAMPLE-3

次のコードは，オブジェクト my_str が持つ文字列を double 型の実数に変換し，その結果を標準出力しています．

```
stdstreamio sio;
tstring     my_str = "0xabp2";

sio.printf("%f\n", my_str.atof());
```

実行結果

```
684.000000
```

WARNING

基数に対して有効でない数字が現れた時点で変換は終了します．全ての文字が変換できたかを確認する場合は，endpos 引数のある strtod() メンバ関数 (§9.5.39) を使用してください．

9.5.37 strtol(), strtoll()

NAME

strtol(), strtoll() — 整数値に変換

SYNOPSIS

```
long strtol( int base, size_t *endpos ) const; .............................. 1
long strtol( size_t pos, int base, size_t *endpos ) const; ................ 2
long strtol( size_t pos, size_t n, int base, size_t *endpos ) const; ..... 3
long long strtoll( int base, size_t *endpos ) const; ....................... 4
long long strtoll( size_t pos, int base, size_t *endpos ) const; ......... 5
long long strtoll( size_t pos, size_t n, int base, size_t *endpos ) const; 6
```

DESCRIPTION

自身の文字列を，基数 base で整数値に変換し，返します．strtol() は文字列を long 型整数に変換します．また同様に，strtoll() は文字列を long long 型整数に変換します．この base は 2 から 36 までの値，あるいは 0 を指定できます．0 または 16 を指定した場合には，文字列の先頭に'0x' を置く事ができ，文字列は 16 進数として扱われます．これ以外の文字列で base が 0 の場合には，文字列が'0' で始まる時は 8 進数として，それ以外の時は 10 進数として扱われます (EXAMPLE-2 参照)．また，変換されなかった文字の位置を endpos に返します．

メンバ関数 1 およびメンバ関数 4 は，自身の文字列を整数に変換します．

メンバ関数 2，3，5 およびメンバ関数 6 は，自身の文字列を位置 pos から整数に変換します．なお，文字列の先頭位置は 0 です．またメンバ関数 3 およびメンバ関数 6 は，変換に使う部分の文字列長を n で指定できます．

PARAMETER

[I]	pos	自身が持つ文字列の変換開始位置
[I]	n	整数値に変換する文字数
[I]	base	基数
[O]	endpos	自身の文字列の変換されなかった文字の位置

([I]：入力，[O]：出力)

RETURN VALUE

整数 ： 変換された整数値

EXCEPTION

内部バッファの確保に失敗した場合 (メンバ関数 3，6)

EXAMPLE-1

次のコードは，オブジェクト my_str が持つ文字列を long 型の整数に変換し，その結果を標準出力しています.

```
stdstreamio sio;
tstring     my_str = "01234F57";
long        l_ret  = -1;
size_t      endpos = 0;

l_ret = my_str.strtol(0, 0, &endpos);
if (endpos == 0) {
    エラー処理
}
else {
    sio.printf("%ld,%zu\n", l_ret, endpos);
}
```

実行結果

668,5

EXAMPLE-2

次のコードは，オブジェクト my_str が持つ文字列の 2 番目以降の文字を long 型の整数に変換し，その結果を標準出力しています.

```
stdstreamio sio;
tstring     my_str = "1234F57";
long        l_ret  = -1;
size_t      endpos = 0;

l_ret = my_str.strtol(2, 10, &endpos);
if ( endpos == 0 ) {
    エラー処理
}
else {
    sio.printf("%ld, %zu\n", l_ret, endpos);
}
```

実行結果

34,4

9.5.38 strtoul(), strtoull()

NAME

strtoul(), strtoull() — 符号なし整数値に変換

SYNOPSIS

```
unsigned long strtoul( int base, size_t *endpos ) const;  .................. 1
unsigned long strtoul( size_t pos, int base, size_t *endpos ) const;  ..... 2
unsigned long strtoul( size_t pos, size_t n,
                       int base, size_t *endpos ) const;  .................. 3
unsigned long long strtoull( int base, size_t *endpos ) const;  ........... 4
unsigned long long strtoull( size_t pos, int base, size_t *endpos ) const;  5
unsigned long long strtoull( size_t pos, size_t n,
                             int base, size_t *endpos ) const;  ........... 6
```

DESCRIPTION

自身の文字列を，基数 base で符号無し整数値に変換し，返します．strtoul() は文字列を unsigned long 型整数に変換し，strtoll() は文字列を unsigned long long 型整数に変換します．この base は 2 から 36 までの値，あるいは 0 を指定できます．0 または 16 を指定した場合には，文字列の先頭に'0x' を置く事ができ，文字列は 16 進数として扱われます．これ以外の文字列で base が 0 の場合には，文字列が'0' で始まる時は 8 進数として，それ以外の時は 10 進数として扱われます．また，変換されなかった文字の位置を endpos に返します．

メンバ関数 1 およびメンバ関数 4 は，自身の文字列を符号なし整数に変換します．

メンバ関数 2，3，5 およびメンバ関数 6 は，自身の文字列を位置 pos から符号なし整数に変換します．なお，文字列の先頭位置は 0 です．またメンバ関数 3 およびメンバ関数 6 は，変換に使う部分の文字列長を n で指定できます．

PARAMETER

[I]	pos	自身が持つ文字列の変換開始位置
[I]	n	整数値に変換する文字数
[I]	base	基数
[O]	endpos	自身の文字列の変換されなかった文字の位置

([I]：入力，[O]：出力)

RETURN VALUE

整数 ： 変換された整数値

EXCEPTION

内部バッファの確保に失敗した場合 (メンバ関数 3，6)

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列を 10 進数で unsigned long 型の整数に変換し，その結果を標準出力しています．

```
stdstreamio   sio;
tstring       my_str = "-1abc";
unsigned long ul_ret = 0;
size_t        endpos = 0;

ul_ret = my_str.strtoul(10, &endpos);
if ( endpos == 0 ) {
```

```
        エラー処理
    }
    else {
        sio.printf("%lu,%zu\n", ul_ret, endpos);
    }
```

実行結果

```
4294967295,2
```

9.5.39 strtod()

NAME

strtod() — 実数値に変換

SYNOPSIS

```
double strtod( size_t *endpos ) const; ..................................... 1
double strtod( size_t pos, size_t *endpos ) const; ......................... 2
double strtod( size_t pos, size_t n, size_t *endpos ) const; ............. 3
```

DESCRIPTION

自身の文字列を，実数値に変換し，返します．また，変換されなかった文字列の位置を endpos に返します．

FORTRAN 形式の倍精度指数表示 (例えば，1.2345D-10) にも対応しています．その他，変換できる文字列は §9.5.36の解説を参照してください．

メンバ関数 1 は，自身の文字列を実数に変換します．

メンバ関数 2 およびメンバ関数 3 は，自身の文字列を位置 pos から実数に変換します．なお，文字列の先頭位置は 0 です．またメンバ関数 3 は，変換に使う部分の文字列長を n で指定できます．

PARAMETER

[I]	pos	自身が持つ文字列の変換開始位置
[I]	n	実数値に変換する文字数
[O]	endpos	自身の文字列の変換されなかった文字の位置

([I]：入力，[O]：出力)

RETURN VALUE

実数 ： 変換された double の値

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE-1

次のコードは，オブジェクト my_str が持つ文字列-12.3D-4X56 を double 型の実数に変換し，その結果を標準出力しています．

```
    stdstreamio sio;
```

```
tstring     my_str = "-12.3D-4X56";
double      d_ret  = 0;
size_t      endpos = 0;

d_ret = my_str.strtod(&endpos);
if ( endpos == 0 ) {
    エラー処理
}
else {
    sio.printf("%f, %zu\n", d_ret, endpos);
}
```

実行結果

```
-0.001230,8
```

EXAMPLE-2

次のコードは，オブジェクト my_str が持つ文字列 infinity を double 型の実数に変換し，その結果を標準出力しています.

```
stdstreamio sio;
tstring     my_str = "infinity";
double      d_ret  = 0;
size_t      endpos = 0;

d_ret = my_str.strtod(&endpos);
if ( endpos == 0 ) {
    エラー処理
}
else {
    sio.printf("%f, %zu\n", d_ret, endpos);
}
```

実行結果

```
inf,8
```

9.5.40 scanf(), vscanf()

NAME

scanf(), vscanf() — 書式付き入力変換

SYNOPSIS

```
int scanf( const char *format, ... ) const;
int vscanf( const char *format, va_list ap ) const;
```

DESCRIPTION

自身の文字列を format の指定に従って変換し，format 以降の引数に格納します.

format の変換指定に応じて変換した結果を，scanf() では可変長引数の各要素データに，vscanf() では可変長引数のリスト ap に読み込みます．format については §8.1.11の解説を参照してください．

PARAMETER

[I]	format	読み込みフォーマット指定
[O]	...	書き込み先となる可変長引数の各要素データ
[O]	ap	書き込み先となる可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	読み込みと変換が成功した入力要素の個数
EOF(エラー)	:	引数が十分でない，または format が NULL である場合
	:	オブジェクト内に文字列がない場合
	:	format で指定した整数型に変換した際，対応する整数型に格納できるサイズを超えている場合

EXAMPLE

次のコードは，フォーマット *NAME ID : %9s %9s* に従い，オブジェクト my_str が持つ文字列から文字列バッファc_name，c_id への変換を行います．そして，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str     = "NAME ID : SATO 1234";
char        c_name[10];
char        c_id[10];
int         i_ret      = 0;

if ((i_ret = my_str.scanf("NAME ID : %9s %9s", c_name,c_id)) == EOF) {
    エラー処理
}
else {
    sio.printf("%s, %s, %d\n", c_name, c_id, i_ret);
}
```

実行結果

```
SATO, 1234, 2
```

WARNING

format に"%s"を指定する場合，格納バッファの大きさ以上の文字列が入力されるとバッファオーバーランが発生します．この問題を回避する手段については，§8.1.11の WARNING を参照してください．

9.5.41 strcmp(), compare()

NAME

strcmp(), compare() — 文字列の比較

SYNOPSIS

```
int strcmp( const char *str ) const; .......................................... 1
int strcmp( size_t pos1, const char *str ) const; .......................... 2
int strcmp( const tstring &str, size_t pos2 = 0 ) const; .................. 3
int strcmp( size_t pos1, const tstring &str, size_t pos2 = 0 ) const; .... 4
int compare( const char *str ) const; ........................................ 5
int compare( size_t pos1, const char *str ) const; ......................... 6
int compare( const tstring &str, size_t pos2 = 0 ) const; ................. 7
int compare( size_t pos1, const tstring &str, size_t pos2 = 0 ) const; .. 8
```

DESCRIPTION

strcmp() と compare() は名前が異なりますが，同じ動作を行なうメンバ関数です．

strcmp() メンバ関数と compare() メンバ関数は，自身の文字列と指定された文字列 str とを，辞書的に比較し，その結果を返します．比較は文字列内のそれぞれの文字の文字コードに基づいて行います．strncmp() メンバ関数 (§9.5.42) とは異なり，開始位置から全ての文字を比較します．

pos1 には自身の文字列の比較開始位置を，pos2 には指定されたオブジェクト str が持つ文字列の比較開始位置を指定します．なお，オブジェクト内の文字列の先頭位置は 0 です．

メンバ関数 3,4,7,8 は，pos2 を指定しなくても使用できます．その場合は 0 を指定したものとして処理を行います．

PARAMETER

[I]	str	比較に用いる文字列
[I]	pos1	自身が持つ文字列の開始位置
[I]	pos2	オブジェクト str が持つ文字列の開始位置 (str の部分文字列と比較する場合)

([I]：入力，[O]：出力)

RETURN VALUE

0	:	自身の文字列が str と等しい場合
正の値	:	自身の文字列が str に較べて辞書的に大きい場合
負の値	:	自身の文字列が str に較べて辞書的に小さい場合
256(エラー)	:	自身が文字列バッファを持ち，str に NULL を指定した場合
-256(エラー)	:	自身が文字列バッファを持たず，str を指定した場合
	:	pos1 に自身の文字列長を越える値を指定した場合
	:	pos2 にオブジェクト str の文字列長を越える値を指定した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列と外部文字列 *Akari20090303.txt* との比較を行い，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "Akari20060222.txt";

if (my_str.compare("Akari20090303.txt") == 0) {
    sio.printf("The same file\n");
}
```

```
    else {
        sio.printf("Different file\n");
    }
```

実行結果

```
Different file
```

9.5.42 strncmp(), compare()

NAME

strncmp(), compare() — 部分的に文字列を比較

SYNOPSIS

```
int strncmp( const char *str, size_t n ) const; ............................ 1
int strncmp( size_t pos1, const char *str, size_t n ) const; ............. 2
int strncmp( const tstring &str, size_t pos2, size_t n ) const; .......... 3
int strncmp( size_t pos1, const tstring &str,
             size_t pos2, size_t n ) const; ................................ 4
int compare( const char *str, size_t n ) const; ............................ 5
int compare( size_t pos1, const char *str, size_t n ) const; ............. 6
int compare( const tstring &str, size_t pos2, size_t n ) const; .......... 7
int compare( size_t pos1, const tstring &str,
             size_t pos2, size_t n ) const; ................................ 8
```

DESCRIPTION

`strncmp()` と `compare()` は名前が異なりますが，同じ動作を行なうメンバ関数です．

strncmp() メンバ関数と compare() メンバ関数は，自身の文字列と指定された文字列 `str` とを，辞書的に比較し，その結果を返します．比較は文字列内にあるそれぞれの文字の文字コードに基づいて行います．

`pos1` には自身の文字列の比較開始位置を，`pos2` には指定されたオブジェクト `str` が持つ文字列の比較開始位置を指定します．なお，オブジェクト内の文字列の先頭位置は 0 です．

strcmp() メンバ関数 (§9.5.41) とは異なり，開始位置から最初の `n` 文字を比較します．

PARAMETER

[I]	`str`	比較に用いる文字列
[I]	`pos1`	自身の文字列の開始位置
[I]	`pos2`	`str` が持つ文字列の開始位置 (`str` の部分文字列と比較する場合)
[I]	`n`	比較する文字数

([I]：入力，[O]：出力)

RETURN VALUE

0	:	自身の文字列が str と等しい場合
正の値	:	自身の文字列が str に較べて辞書的に大きい場合
負の値	:	自身の文字列が str に較べて辞書的に小さい場合
256(エラー)	:	自身が文字列バッファを持ち，str に NULL を指定した場合
-256(エラー)	:	自身が文字列バッファを持たず，str を指定した場合
	:	自身が文字列バッファを持たず，n を指定した場合
	:	pos1 に自身の文字列長を越える値を指定した場合
	:	pos2 にオブジェクト str の文字列長を越える値を指定した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列の 5 番目の文字 *2* から 8 文字と，外部文字列 *Akari20090303.txt* との比較を行い，その結果を標準出力します.

```
stdstreamio sio;
tstring     my_str = "Akari20060222.txt";

if (my_str.strncmp(5, "20060222", 8) == 0) {
    sio.printf("The same date\n");
}
else {
    sio.printf("Different date\n");
}
```

実行結果

```
The same date
```

9.5.43 strcasecmp(), strncasecmp()

NAME

strcasecmp(), strncasecmp() — 文字列の比較 (大文字/小文字区別なし)

SYNOPSIS

```
int strcasecmp( const char *str ) const; .................................... 1
int strcasecmp( size_t pos1, const char *str ) const; ...................... 2
int strcasecmp( const tstring &str, size_t pos2 = 0 ) const; ............. 3
int strcasecmp( size_t pos1, const tstring &str, size_t pos2 = 0 ) const; 4
int strncasecmp( const char *str, size_t n ) const; ........................ 5
int strncasecmp( size_t pos1, const char *str, size_t n ) const; ......... 6
int strncasecmp( const tstring &str, size_t pos2, size_t n ) const; ...... 7
int strncasecmp( size_t pos1, const tstring &str,
                 size_t pos2, size_t n ) const; ............................ 8
```

DESCRIPTION

自身の文字列と指定された文字列 str とを，アルファベットの大文字と小文字を区別せず辞書的に比較し，その結果を返します．処理の内部では，自身の文字列と文字列 str の双方を小文

字に変換して比較します．小文字に変換後の比較は，文字列内の文字をそれぞれの文字コードに基づいて行います.

pos1 には自身の文字列の比較開始位置を，pos2 には指定されたオブジェクト str が持つ文字列の比較開始位置を指定します．なお，オブジェクト内の文字列の先頭位置は 0 です.

strcasecmp() の場合，開始位置から全ての文字を比較するのに対し，strncasecmp() の場合は，開始位置から最初の n 文字の一致を調べます.

メンバ関数 3 およびメンバ関数 4 は，pos2 を指定しなくても使用できます．その場合は 0 を指定したものとして処理を行います.

PARAMETER

[I]	str	比較に用いる文字列
[I]	pos1	自身の文字列の開始位置
[I]	pos2	オブジェクト str が持つ文字列の開始位置 (str の部分文字列と比較する場合)
[I]	n	比較する文字数

([I]：入力，[O]：出力)

RETURN VALUE

0	:	自身の文字列が str と等しい場合
正の値	:	自身の文字列が str に較べて辞書的に大きい場合
負の値	:	自身の文字列が str に較べて辞書的に小さい場合
256(エラー)	:	自身が文字列バッファを持ち，str に NULL を指定した場合
-256(エラー)	:	自身が文字列バッファを持たず，str を指定した場合
	:	自身が文字列バッファを持たず，n を指定した場合 (メンバ関数 5〜8)
	:	pos1 に自身の文字列長を越える値を指定した場合 (メンバ関数 2，4，6，8)
	:	pos2 に str の文字列長を越える値を指定した場合 (メンバ関数 3，4，7，8)

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列と，外部文字列 *suzuki* との比較を strcmp()，strcasecmp() で行います．strcasecmp() はアルファベットの大文字と小文字を区別しない事を確認するため，その結果を標準出力します.

```
stdstreamio sio;
tstring     my_str = "SUZUKI";

if (my_str.strcmp("suzuki") == 0 ) {
    sio.printf("The same name\n");
} else {
    sio.printf("Different name\n");
}
if (my_str.strcasecmp("suzuki") == 0 ) {
    sio.printf("The same name\n");
} else {
    sio.printf("Different name\n");
}
```

実行結果

```
Different name
The same name
```

9.5.44 isalpha(), isalnum(), isdigit(), islower(), isupper(), 他

NAME

isalpha(), isalnum(), isdigit(), islower(), isupper(), 他 — 文字の分類

SYNOPSIS

```
bool isalnum( size_t pos ) const;
bool isalpha( size_t pos ) const;
bool iscntrl( size_t pos ) const;
bool isdigit( size_t pos ) const;
bool isgraph( size_t pos ) const;
bool islower( size_t pos ) const;
bool isprint( size_t pos ) const;
bool ispunct( size_t pos ) const;
bool isspace( size_t pos ) const;
bool isupper( size_t pos ) const;
bool isxdigit( size_t pos ) const;
```

DESCRIPTION

自身が持つ文字列の位置 pos にある文字を，現在のロケールにより分類し，その結果を返します．なお，文字列の先頭位置は 0 です．全てのメンバ関数は libc の関数と対応しています．これらのメンバ関数と文字との対応は表 18を参照してください．

PARAMETER

[I] pos 分類する文字の位置
([I]：入力，[O]：出力)

RETURN VALUE

true	:	pos にある文字が，そのメンバ関数が対応する文字に一致する場合
false	:	pos にある文字が，そのメンバ関数が対応する文字に不一致である場合
	:	pos に自身の文字列長を越える値を指定した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列の 5 番目に位置する文字が，アルファベットまたは数字である事を確認するため，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "JAXA/ISAS";

if ((my_str.isalnum(5)) == true ) {
    sio.printf("It is an alphabet or figure.\n");
}
else {
```

```
        sio.printf("It is neither an alphabet nor figure.\n");
    }
```

実行結果

```
It is an alphabet or figure.
```

9.5.45 strchr(), find()

NAME

strchr(), find() — 左側からの文字を検索

SYNOPSIS

```
ssize_t strchr( int ch ) const;                              1
ssize_t strchr( size_t pos, int ch ) const;                  2
ssize_t strchr( size_t pos, int ch, size_t *nextpos ) const; 3
ssize_t find( int ch ) const;                                4
ssize_t find( size_t pos, int ch ) const;                    5
ssize_t find( size_t pos, int ch, size_t *nextpos ) const;   6
```

DESCRIPTION

strchr()，find() は名前は違いますが，同じ動作をします．

自身が持つ文字列の左側から右方向に文字 ch を検索し，最初に出現する位置を返します．

pos を指定した場合は，自身が持つ文字列の位置 pos から検索を開始します．なお，文字列の先頭位置は 0 です．

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．nextpos の指す変数には，文字が見つかった場合は見つかった位置の 1 つ右の位置が，見つからなかった場合には自身の文字列長が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	ch	検出する文字
[I]	pos	自身の文字列の開始位置
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定した文字が見つかった場合，その先頭の位置
負の値 (エラー)	:	指定した文字が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	pos に自身の文字列長以上の値を指定した場合 (メンバ関数 2，3，5，6)

EXAMPLE

§9.5.46の EXAMPLE を参照してください．

9.5.46 strstr(), find()

NAME

strstr(), find() — 左側からの文字列を検索

SYNOPSIS

```
ssize_t strstr( const char *str ) const; .................................... 1
ssize_t strstr( size_t pos, const char *str ) const; ....................... 2
ssize_t strstr( size_t pos, const char *str, size_t *nextpos ) const; .... 3
ssize_t strstr( const tstring &str ) const; ................................ 4
ssize_t strstr( size_t pos, const tstring &str ) const; ................... 5
ssize_t strstr( size_t pos, const tstring &str, size_t *nextpos ) const;   6
ssize_t find( const char *str ) const; ...................................... 7
ssize_t find( const char *str, size_t n ) const; ........................... 8
ssize_t find( size_t pos, const char *str ) const; ......................... 9
ssize_t find( size_t pos, const char *str, size_t n ) const; ............ 10
ssize_t find( size_t pos, const char *str, size_t *nextpos ) const; ..... 11
ssize_t find( size_t pos, const char *str, size_t n,
              size_t *nextpos ) const; .................................... 12
ssize_t find( const tstring &str ) const; ................................. 13
ssize_t find( size_t pos, const tstring &str ) const; .................... 14
ssize_t find( size_t pos, const tstring &str, size_t *nextpos ) const;  . 15
```

DESCRIPTION

strstr()，find() は名前は違いますが，同じ動作をします．

自身が持つ文字列の左側から文字列 str を検索し，最初に出現する位置を返します．

pos を指定した場合は，自身の文字列の位置 pos から検索を開始します．なお，文字列の先頭位置は 0 です．

n を指定した場合は，str の先頭から n 文字の文字列と一致する位置を調べます．

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．1 文字以上の長さの文字列が見つかった場合は，見つかった位置からさらに str の長さ分だけ右方向に移動させた位置が nextpos の指す変数に返ります．長さ 0 の文字列が見つかった場合には，見つかった位置から 1 文字分だけ右方向に移動させた位置が自身の文字列長以下であれば，その値が nextpos の指す変数に返ります．それ以外の場合には，自身の文字列長+1 が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	pos	自身の文字列の開始位置
[I]	str	検出する文字列
[I]	n	検出する文字数
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定した文字列が見つかった場合，その先頭の位置
負の値 (エラー)	:	指定した文字列が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	pos に自身の文字列長を越える値を指定した場合
	:	str が NULL の場合 (メンバ関数 1, 2, 3, 7, 8, 9, 10, 11, 12)

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列中から文字列 *ISAS* の位置を左端から検索し，その結果を標準出力します.

```
stdstreamio sio;
tstring     my_str = "JAXA/ISAS/AKARI";
ssize_t     t_ret  = 0;

if ((t_ret = my_str.strstr("ISAS")) < 0 ) {
    エラー処理
}
else {
    sio.printf("%zd\n", t_ret);
}
```

実行結果

5

9.5.47 strrchr(), rfind()

NAME

strrchr(), rfind() — 右側からの文字を検索

SYNOPSIS

```
ssize_t strrchr( int ch ) const; ........................................ 1
ssize_t strrchr( size_t pos, int ch ) const; ............................ 2
ssize_t strrchr( size_t pos, int ch, size_t *nextpos ) const; ........... 3
ssize_t rfind( int ch ) const; .......................................... 4
ssize_t rfind( size_t pos, int ch ) const; .............................. 5
ssize_t rfind( size_t pos, int ch, size_t *nextpos ) const; ............. 6
```

DESCRIPTION

strrchr()，rfind() は名前は違いますが，同じ動作をします.

自身が持つ文字列の右側から左方向に文字 ch を検索し，最初に出現する位置を返します．この位置は文字列の左端からの位置です．なお，文字列の先頭位置は 0 です.

pos を指定した場合は，自身の文字列の位置 pos から左方向に検索を開始します.

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．nextpos の指す変数には，pos が 1 以上で文字が見つかった場合は見つかった位置の 1 つ左の位置が，そうでない場合は自身の文字列長が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます.

PARAMETER

[I]	ch	検出する文字
[I]	pos	自身の文字列の開始位置
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定した文字が見つかった場合，その先頭の位置
負の値 (エラー)	:	指定した文字が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	pos に自身の文字列長以上の値を指定した場合 (メンバ関数 2, 3, 5, 6)

EXAMPLE

§9.5.48の EXAMPLE を参照してください．

9.5.48 strrstr(), rfind()

NAME

strrstr(), rfind() — 右側からの文字列を検索

SYNOPSIS

```
ssize_t strrstr( const char *str ) const;  ................................ 1
ssize_t strrstr( size_t pos, const char *str ) const;  .................... 2
ssize_t strrstr( size_t pos, const char *str, size_t *nextpos ) const;  .. 3
ssize_t strrstr( const tstring &str ) const;  ............................. 4
ssize_t strrstr( size_t pos, const tstring &str ) const;  ................. 5
ssize_t strrstr( size_t pos, const tstring &str, size_t *nextpos ) const;   6
ssize_t rfind( const char *str ) const;  .................................. 7
ssize_t rfind( const char *str, size_t n ) const;  ........................ 8
ssize_t rfind( size_t pos, const char *str ) const;  ...................... 9
ssize_t rfind( size_t pos, const char *str, size_t n ) const;  ........... 10
ssize_t rfind( size_t pos, const char *str, size_t *nextpos ) const;  .... 11
ssize_t rfind( size_t pos, const char *str, size_t n,
               size_t *nextpos ) const;  ................................. 12
ssize_t rfind( const tstring &str ) const;  .............................. 13
ssize_t rfind( size_t pos, const tstring &str ) const;  .................. 14
ssize_t rfind( size_t pos, const tstring &str, size_t *nextpos ) const;    15
```

DESCRIPTION

strrstr()，rfind() は名前は違いますが，同じ動作をします．

自身が持つ文字列の右側から左方向に文字列 str を検索し，最初に出現する位置を返します．この位置は文字列の左端からの位置です．なお，文字列の先頭位置は 0 です．

pos を指定した場合は，自身の文字列の位置 pos から左方向に検索を開始します．

n を指定した場合は，str の先頭から n 文字の文字列と一致する位置を調べます．

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．1 文字以上の長さの文字列が見つかった場合は，見つかった位置か

らさらに str の長さ分だけ左方向に移動させた位置が nextpos の指す変数に返ります．長さ 0 の文字列が見つかった場合には，見つかった位置から 1 文字分だけ左方向に移動させた位置が負数でなければ，その値が nextpos の指す変数に返ります．それ以外の場合には，自身の文字列長+1 が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	pos	オブジェクト内の文字列の開始位置
[I]	str	検出する文字列
[I]	n	検出する文字数
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定した文字列の見つかった場合，その先頭の位置
負の値 (エラー)	:	指定した文字列が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	pos に自身の文字列長を越える値を指定した場合
	:	str が NULL の場合 (メンバ関数 1, 2, 3, 7, 8, 9, 10, 11, 12)

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列中から文字列 *AS* の位置を右端から検索し，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "JAXA/ISAS/NASA";
ssize_t     t_ret  = 0;

if ((t_ret = my_str.rfind("AS")) < 0 ) {
    エラー処理
}
else {
    sio.printf("%zd\n", t_ret);
}
```

実行結果

```
11
```

9.5.49 find_first_of()

NAME

find_first_of() — 左側から文字セット中の文字を検出

SYNOPSIS

```
ssize_t find_first_of( const char *str ) const; ............................ 1
ssize_t find_first_of( const char *str, size_t n ) const; ................. 2
ssize_t find_first_of( size_t pos, const char *str ) const; .............. 3
ssize_t find_first_of( size_t pos, const char *str, size_t n ) const; .... 4
```

```
ssize_t find_first_of( size_t pos, const char *str,
                       size_t *nextpos ) const;
ssize_t find_first_of( size_t pos, const char *str, size_t n,
                       size_t *nextpos ) const;
ssize_t find_first_of( const tstring &str ) const;
ssize_t find_first_of( size_t pos, const tstring &str ) const;
ssize_t find_first_of( size_t pos, const tstring &str,
                       size_t *nextpos ) const;
```
.... 5
.... 6
.... 7
.... 8
.... 9

DESCRIPTION

find_first_of() メンバ関数は，自身が持つ文字列の左側から右方向に，文字セット str に含まれる文字を検索し，最初に出現する位置を返します．なお，文字列の先頭位置は 0 です．

pos を指定した場合は，自身の文字列の位置 pos から検索を開始します．

n を指定した場合は，str の先頭から n 文字をセットとします．

文字セットとは，文字列とは異なり文字の順番に意味をもたず，文字の集合を一つの文字列で表現したものです．例えば文字セットを"ABC"とした場合，'A'，'B'，'C' のいずれかに合致する文字を探します．

tstring.h には，表 20にあるマクロが定義されています．str にセットすると便利でしょう．

マクロ定義	対応する文字セット
CSET_ALNUM	"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
CSET_ALPHA	"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
CSET_LOWER	"abcdefghijklmnopqrstuvwxyz"
CSET_UPPER	"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
CSET_DIGIT	"0123456789"
CSET_XDIGIT	"0123456789abcdefABCDEF"

表 20: 文字セットのマクロ定数定義.

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．nextpos の指す変数には，文字が見つかった場合は見つかった位置の 1 つ右の位置が，見つからなかった場合には自身の文字列長が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

文字セットの指定方法を除いて，strpbrk() メンバ関数 (§9.5.53) と動作は同じです．strpbrk() では文字セットに "[A-Z]" のような表現も使えますので，そちらも検討してください．

PARAMETER

[I]	str	検出対象の文字セット
[I]	n	文字セット str の文字数
[I]	pos	検出の開始位置
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定した文字または文字セットに含まれる文字の位置
負の値 (エラー)	:	指定した文字または文字セットに含まれる文字が見つからなかった場合
	:	pos に自身の文字列長以上の値を指定した場合
	:	オブジェクト内に文字列がない場合
	:	str が NULL である場合 (メンバ関数 1〜6)

EXAMPLE-1

次のコードは，オブジェクト my_str が持つ文字列中から文字セット CSET_DIGIT に含まれる文字が現れる位置を左端から検索し，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "Akari20090306.txt";
ssize_t     t_ret  = 0;

if ((t_ret = my_str.find_first_of(CSET_DIGIT)) < 0 ) {
    エラー処理
}
else {
    sio.printf("%zd\n", t_ret);
}
```

実行結果

5

EXAMPLE-2

次のコードは，オブジェクト my_str が持つ文字列中から文字 I, S, A, S のいずれかが現れる位置を左端から検索し，その結果を標準出力します．(この結果から，§9.5.45の EXAMPLE との違いを確認できます)．

```
stdstreamio sio;
tstring my_str = "JAXA/ISAS/AKARI";
ssize_t t_ret  = 0;

if ((t_ret = my_str.find_first_of("ISAS")) < 0) {
    エラー処理
}
else {
    sio.printf("%zd\n", t_ret);
}
```

実行結果

1

9.5.50 find_last_of()

NAME

find_last_of() — 右側から文字セット中の文字を検出

SYNOPSIS

```
ssize_t find_last_of( const char *str ) const;  ............................ 1
ssize_t find_last_of( const char *str, size_t n ) const;  .................. 2
ssize_t find_last_of( size_t pos, const char *str ) const;  ................ 3
ssize_t find_last_of( size_t pos, const char *str, size_t n ) const;  ..... 4
ssize_t find_last_of( size_t pos, const char *str,
                      size_t *nextpos ) const;  ............................ 5
ssize_t find_last_of( size_t pos, const char *str, size_t n,
                      size_t *nextpos ) const;  ............................ 6
ssize_t find_last_of( const tstring &str ) const;  ......................... 7
ssize_t find_last_of( size_t pos, const tstring &str ) const;  ............ 8
ssize_t find_last_of( size_t pos, const tstring &str,
                      size_t *nextpos ) const;  ............................ 9
```

DESCRIPTION

find_last_of() メンバ関数は，自身が持つ文字列の右側から左方向に，文字 ch または文字セット str に含まれる文字を検索し，最初に出現する位置を返します．この位置は文字列の左端からの位置です．なお，文字列の先頭位置は 0 です．

pos を指定した場合は，自身の文字列の位置 pos から左方向に検索を行います．

n を指定した場合は，str の先頭から n 文字をセットとします．

tstring.h には，文字セットに使用できる CSET_ALNUM，CSET_ALPHA，CSET_LOWER，CSET_UPPER，CSET_DIGIT，CSET_XDIGIT が定義されています．これらの詳細と，文字セットについての説明は §9.5.49の解説を参照してください．

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．nextpos の指す変数には，pos が 1 以上で文字が見つかった場合は見つかった位置の 1 つ左の位置が，そうでない場合は自身の文字列長が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

文字セットの指定方法を除いて，strrpbrk() メンバ関数 (§9.5.54) と動作は同じです．strrpbrk() では文字セットに "[A-Z]" のような表現が使えますので，そちらも検討してください．

PARAMETER

[I]	str	検出対象の文字セット
[I]	n	文字セット str の文字数
[I]	pos	検出の開始位置
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定した文字または文字セットに含まれる文字の位置
負の値 (エラー)	:	指定した文字または文字セットに含まれる文字が見つからなかった場合
	:	pos に自身の文字列長以上の値を指定した場合
	:	オブジェクト内に文字列がない場合
	:	str が NULL である場合 (メンバ関数 1〜6)

EXAMPLE-1

次のコードは，オブジェクト my_str が持つ文字列中から最後に文字 A,S のいずれかが現れる

位置を右端から検索し，その結果を標準出力します．(この結果から，§9.5.47の EXAMPLE との違いを確認できます).

```
stdstreamio sio;
tstring     my_str = "JAXA/ISAS/NASA";
ssize_t     t_ret  = 0;

if ((t_ret = my_str.find_last_of("AS")) < 0) {
    エラー処理
}
else {
    sio.printf("%zd\n", t_ret);
}
```

実行結果

```
13
```

EXAMPLE-2

次のコードは，オブジェクト my_str が持つ文字列中から文字セット CSET_DIGIT に含まれる文字が現れる右端から位置を検索し，その結果を標準出力します．(この結果から，§9.5.49の EXAMPLE-2 との違いを確認できます).

```
stdstreamio sio;
tstring my_str = "Akari20090306.txt";
ssize_t t_ret  = 0;

if ((t_ret = my_str.find_last_of(CSET_DIGIT)) < 0) {
    エラー処理
}
else {
    sio.printf("%zd\n", t_ret);
}
```

実行結果

```
12
```

9.5.51 find_first_not_of()

NAME

find_first_not_of() — 左側から文字セットに含まれない文字の位置を検出

SYNOPSIS

```
ssize_t find_first_not_of( const char *str ) const;  ........................ 1
ssize_t find_first_not_of( const char *str, size_t n ) const;  ............. 2
ssize_t find_first_not_of( size_t pos, const char *str ) const;  ........... 3
ssize_t find_first_not_of( size_t pos, const char *str, size_t n ) const;   4
```

```
ssize_t find_first_not_of( size_t pos, const char *str,
                           size_t *nextpos ) const;  ........................ 5
ssize_t find_first_not_of( size_t pos, const char *str, size_t n,
                           size_t *nextpos ) const;  ........................ 6
ssize_t find_first_not_of( const tstring &str ) const;  .................... 7
ssize_t find_first_not_of( size_t pos, const tstring &str ) const;  ....... 8
ssize_t find_first_not_of( size_t pos, const tstring &str,
                           size_t *nextpos ) const;  ........................ 9
ssize_t find_first_not_of( int ch ) const;  ................................ 10
ssize_t find_first_not_of( size_t pos, int ch ) const;  ................... 11
ssize_t find_first_not_of( size_t pos, int ch, size_t *nextpos ) const;   12
```


DESCRIPTION

find_first_not_of() メンバ関数は，自身が持つ文字列の左側から右方向に，文字 ch または文字セット str に含まれない文字を検索し，最初に出現する位置を返します．なお，文字列の先頭位置は 0 です．

pos を指定した場合は，自身の文字列の位置 pos から検索を開始します．

n を指定した場合は，str の先頭から n 文字をセットとします．

tstring.h には，文字セットに使用できる CSET_ALNUM，CSET_ALPHA，CSET_LOWER，CSET_UPPER，CSET_DIGIT，CSET_XDIGIT が定義されています．これらの詳細と，文字セットについての説明は §9.5.49の解説を参照してください．

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．nextpos の指す変数には，文字が見つかった場合は見つかった位置の 1 つ右の位置が，見つからなかった場合には自身の文字列長が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

strpbrk() メンバ関数 (§9.5.53) で，文字セットを "[^A-Z]" のように指定する方法もあります．そちらも検討してください．

PARAMETER

[I]	ch	検出を除外する文字
[I]	str	文字セット
[I]	n	文字セット str の文字数
[I]	pos	検出の開始位置
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定した文字または文字セットに含まれない文字の位置
負の値 (エラー)	:	指定した文字または文字セットに含まれない文字が見つからなかった場合
	:	pos に自身の文字列長以上の値を指定した場合
	:	オブジェクト内に文字列がない場合
	:	str が NULL である場合 (メンバ関数 1〜6)

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列に，文字 A，J，X のいずれでもない文字が現れる位置を左端から検索し，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "JAXA/ISAS";
ssize_t     t_ret  = 0;

if ((t_ret = my_str.find_first_not_of("AJX")) < 0){
    エラー処理
}
else {
    sio.printf("%zd\n", t_ret);
}
```

実行結果

```
4
```

9.5.52 find_last_not_of()

NAME

find_last_not_of() — 右側から文字セットに含まれない文字の位置を検出

SYNOPSIS

```
ssize_t find_last_not_of( const char *str ) const; ........................ 1
ssize_t find_last_not_of( const char *str, size_t n ) const; ............. 2
ssize_t find_last_not_of( size_t pos, const char *str ) const; ........... 3
ssize_t find_last_not_of( size_t pos, const char *str, size_t n ) const;   4
ssize_t find_last_not_of( size_t pos, const char *str,
                          size_t *nextpos ) const; ........................ 5
ssize_t find_last_not_of( size_t pos, const char *str, size_t n,
                          size_t *nextpos ) const; ........................ 6
ssize_t find_last_not_of( const tstring &str ) const; ..................... 7
ssize_t find_last_not_of( size_t pos, const tstring &str ) const; ....... 8
ssize_t find_last_not_of( size_t pos, const tstring &str,
                          size_t *nextpos ) const; ........................ 9
ssize_t find_last_not_of( int ch ) const; ................................ 10
ssize_t find_last_not_of( size_t pos, int ch ) const; .................... 11
ssize_t find_last_not_of( size_t pos, int ch, size_t *nextpos ) const;  . 12
```

DESCRIPTION

find_last_not_of() メンバ関数は，自身が持つ文字列の右側から左方向に，文字 ch または文字セット str に含まれない文字を検索し，最初に出現する位置を返します．この位置は文字列の左端からの位置です．なお，文字列の先頭位置は 0 です．

pos を指定した場合は，自身の文字列の位置 pos から左方向に検索を開始します．

n を指定した場合は，str の先頭から n 文字をセットとします．

tstring.hには，文字セットに使用できるCSET_ALNUM，CSET_ALPHA，CSET_LOWER，CSET_UPPER，CSET_DIGIT，CSET_XDIGIT が定義されています．これらの詳細と，文字セットについての説明は §9.5.49の解説を参照してください．

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．nextpos の指す変数には，pos が 1 以上で文字が見つかった場合は見つかった位置の 1 つ左の位置が，そうでない場合は自身の文字列長が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

strrpbrk() メンバ関数 (§9.5.54) で，文字セットを "[^A-Z]" のように指定する方法もあります．そちらも検討してください．

PARAMETER

[I]	ch	検出を除外する文字
[I]	str	文字列に含まれていない文字セット
[I]	n	文字セット str の文字数
[I]	pos	検出の開始位置
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定した文字または文字セットに含まれない文字の位置
負の値 (エラー)	:	指定した文字または文字セットに含まれない文字が見つからなかった場合
	:	pos に自身の文字列長以上の値を指定した場合
	:	オブジェクト内に文字列がない場合
	:	str が NULL である場合 (メンバ関数 1〜6)

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列に，文字 N，A，S のいずれでもない文字が現れる位置を右端から検索し，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "JAXA/ISAS";
ssize_t     t_ret  = 0;

if ((t_ret = my_str.find_last_not_of("NASA",3)) < 0) {
    エラー処理
}
else {
    sio.printf("%zd\n", t_ret);
}
```

実行結果

5

9.5.53 strpbrk()

NAME

strpbrk() — 左側から文字セット中の文字を検出

SYNOPSIS

```
ssize_t strpbrk( const char *accept ) const; ................................ 1
ssize_t strpbrk( size_t pos, const char *accept ) const; .................. 2
ssize_t strpbrk( size_t pos, const char *accept, size_t *nextpos ) const;   3
ssize_t strpbrk( const tstring &accept ) const; ............................ 4
ssize_t strpbrk( size_t pos, const tstring &accept ) const; ............... 5
ssize_t strpbrk( size_t pos, const tstring &accept,
                 size_t *nextpos ) const; ................................... 6
```

DESCRIPTION

自身が持つ文字列の左側から右方向に，文字セット accept に含まれる文字を検索し，最初に出現する位置を返します．なお，文字列の先頭位置は 0 です．

pos を指定した場合は，自身の文字列の位置 pos から検索を開始します．

これらのメンバ関数は，find_first_of() メンバ関数 (§9.5.49) の機能に加え，accept には，"xyz"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]"あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説を参照してください．

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．nextpos の指す変数には，文字が見つかった場合は見つかった位置の 1 つ右の位置が，見つからなかった場合には自身の文字列長が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	accept	検出対象の文字セット
[I]	pos	検出の開始位置
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定した文字または文字セットに含まれる文字の位置
負の値 (エラー)	:	指定した文字または文字セットに含まれる文字が見つからなかった場合
	:	pos に自身の文字列長以上の値を指定した場合
	:	オブジェクト内に文字列がない場合
	:	accept が NULL である場合

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列に，文字セット *[:digit:]* に含まれる文字が現れる位置を左端から検索し，その結果を標準出力します．(この結果から §9.5.49の EXAMPLE-1 と同じ結果が得られる事がわかります)

```
stdstreamio sio;
tstring     my_str = "Akari20090306.txt";
ssize_t     t_ret  = 0;

if ((t_ret = my_str.strpbrk("[[:digit:]]")) < 0) {
    エラー処理
}
else {
    sio.printf("%zd\n", t_ret);
}
```

実行結果

```
5
```

9.5.54 strrpbrk()

NAME

strrpbrk() — 右側から文字セット中の文字を検出

SYNOPSIS

```
ssize_t strrpbrk( const char *accept ) const; ............................... 1
ssize_t strrpbrk( size_t pos, const char *accept ) const; ................. 2
ssize_t strrpbrk( size_t pos, const char *accept, size_t *nextpos ) const; 3
ssize_t strrpbrk( const tstring &accept ) const; ........................... 4
ssize_t strrpbrk( size_t pos, const tstring &accept ) const; ............. 5
ssize_t strrpbrk( size_t pos, const tstring &accept,
                  size_t *nextpos ) const; .................................. 6
```

DESCRIPTION

自身が持つ文字列の右側から左方向に，文字セット accept に含まれる文字を検索し，最初に出現する位置を返します．この位置は文字列の左端からの位置です．なお，文字列の先頭位置は 0 です．

pos を指定した場合は，自身の文字列の位置 pos から検索を開始します．

これらのメンバ関数は，find_last_of() メンバ関数 (§9.5.50) の機能に加え，accept には，"xyz"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]"あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説を参照してください．

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．nextpos の指す変数には，pos が 1 以上で文字が見つかった場合は見つかった位置の 1 つ左の位置が，そうでない場合は自身の文字列長が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	accept	検出対象の文字セット
[I]	pos	検出の開始位置
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定した文字または文字セットに含まれる文字の位置
負の値 (エラー)	:	指定した文字または文字セットに含まれる文字が見つからなかった場合
	:	pos に自身の文字列長以上の値を指定した場合
	:	オブジェクト内に文字列がない場合
	:	accept が NULL である場合

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，文字列 my_str を右端から数字がある部分を検索し，見つかった部分を列挙します.

```
stdstreamio sio;
               /* 012345678 */
tstring my_str = "Z80A 4MHz";
size_t pos = my_str.length() - 1;
ssize_t fpos;
while ( 0 <= (fpos=my_str.strrpbrk(pos, "[0-9]", &pos)) ) {
    sio.printf("fpos = %zd  nextpos = %zu\n", fpos, pos);
}
```

結果

```
fpos = 5  nextpos = 4
fpos = 2  nextpos = 1
fpos = 1  nextpos = 0
```

§9.5.53の EXAMPLE も参照してください.

9.5.55 strspn()

NAME

strspn() — 左側から文字セット中の文字が続く長さを調べる

SYNOPSIS

```
size_t strspn( const char *accept ) const; ........................ 1
size_t strspn( size_t pos, const char *accept ) const; ............ 2
size_t strspn( size_t pos, const char *accept, size_t *nextpos ) const; . 3
size_t strspn( const tstring &accept ) const; ..................... 4
size_t strspn( size_t pos, const tstring &accept ) const; ......... 5
size_t strspn( size_t pos, const tstring &accept, size_t *nextpos ) const; 6
size_t strspn( int accept ) const; ................................ 7
size_t strspn( size_t pos, int accept ) const; .................... 8
size_t strspn( size_t pos, int accept, size_t *nextpos ) const; .......... 9
```

DESCRIPTION

自身が持つ文字列の左側から右方向に，文字セット accept が連続する長さを検索し，その長さを返します．

pos を指定した場合は，自身の文字列の位置 pos から検索を開始します．なお，文字列の先頭位置は 0 です．

メンバ関数 1〜6 の accept は，"xyz"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]"あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説を参照してください．

メンバ関数 1〜6 において，文字セット accept が NULL である場合，全ての文字が対象となります (EXAMPLE-3 参照)．

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．このメンバ関数の返り値が 1 以上の場合は，pos からさらに返り値の分だけ右方向に移動させた位置が nextpos の指す変数に返ります．このメンバ関数の返り値が 0 の場合には，pos から 1 文字分だけ右方向に移動させた位置が自身の文字列長未満であれば，その値が nextpos の指す変数に返ります．それ以外の場合には，自身の文字列長が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	accept	検出対象の文字セット
[I]	pos	検出の開始位置
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

正の値	:	文字セット中の文字が連続する長さ
0	:	文字セットがカウントを開始する位置から連続していなかった場合
	:	pos に自身の文字列長以上の値を指定した場合
	:	オブジェクト内に文字列がない場合

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE-1

次のコードは，オブジェクト my_str が持つ文字列に，文字 A，J，X のいずれかが何文字連続しているかを左端から検索し，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "JAXA/ISAS";
size_t      t_ret  = 0;

t_ret = my_str.strspn("AJX");
if (t_ret == 0) {
    エラー処理
}
else {
    sio.printf("%zu\n", t_ret);
```

```
    }
```

実行結果

```
4
```

EXAMPLE-2

次のコードは，オブジェクト my_str が持つ文字列に，文字セット *[:upper:]* に含まれる文字が何文字連続しているかを左端から検索し，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "JAXA/ISAS";
size_t      t_ret  = 0;

t_ret = my_str.strspn("[[:upper:]]");
if (t_ret == 0) {
    エラー処理
}
else {
    sio.printf("%zu\n", t_ret);
}
```

実行結果

```
4
```

EXAMPLE-3

次のコードは，文字セットが NULL の場合の検索を行います．オブジェクト my_str が持つ文字列の 2 番目から，文字セットに含まれる文字が何文字連続しているかを標準出力します．

```
stdstreamio sio;
tstring     my_str = "JAXA/ISAS";
const char  *c_p   = NULL;
size_t      t_ret  = 0;

t_ret = my_str.strspn(2, c_p);
if (t_ret == 0) {
    エラー処理
}
else {
    sio.printf("%zu\n", t_ret);
}
```

実行結果

```
7
```

9.5.56 strrspn()

NAME

strrspn() — 右側から文字セット中の文字が続く長さを調べる

SYNOPSIS

```
size_t strrspn( const char *accept ) const; ................................ 1
size_t strrspn( size_t pos, const char *accept ) const; .................. 2
size_t strrspn( size_t pos, const char *accept, size_t *nextpos ) const;   3
size_t strrspn( const tstring &accept ) const; ............................ 4
size_t strrspn( size_t pos, const tstring &accept ) const; ................ 5
size_t strrspn( size_t pos, const tstring &accept, size_t *nextpos ) const; 6
size_t strrspn( int accept ) const; ........................................ 7
size_t strrspn( size_t pos, int accept ) const; ........................... 8
size_t strrspn( size_t pos, int accept, size_t *nextpos ) const; ......... 9
```

DESCRIPTION

自身が持つ文字列の右側から左方向に，文字セット accept が連続する長さを検索し，その長さを返します.

pos を指定した場合は，自身の文字列の位置 pos から検索を開始します．なお，文字列の先頭位置は 0 です.

メンバ関数 1〜6 の accept は，"xyz"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]"あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説を参照してください.

メンバ関数 1〜6 において，文字セット accept が NULL である場合，全ての文字が対象となります (§9.5.55の EXAMPLE-3 参照).

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．このメンバ関数の返り値が 1 以上の場合は，pos からさらに返り値の分だけ左方向に移動させた位置が nextpos の指す変数に返ります．このメンバ関数の返り値が 0 の場合には，pos から 1 文字分だけ左方向に移動させた位置が負数でなければ，その値が nextpos の指す変数に返ります．それ以外の場合には，自身の文字列長が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます.

PARAMETER

[I]	accept	検出対象の文字セット
[I]	pos	検出の開始位置
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

正の値	:	文字セット中の文字が連続する長さ
0	:	文字セットがカウントを開始する位置から連続していなかった場合
	:	pos に自身の文字列長以上の値を指定した場合
	:	オブジェクト内に文字列がない場合

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列に，数字が連続している部分を右端から検索し，その結果を標準出力します.

```
stdstreamio sio;
size_t pos, len;
               /* 012345678 */
tstring my_str = "Z80A 4MHz";

pos = my_str.length() - 1;
do {
    sio.printf("curpos = %zu  ", pos);
    len = my_str.strrspn(pos, "[0-9]", &pos);
    sio.printf("ret_len = %zu  nextpos = %zu\n", len, pos);
} while ( pos < my_str.length() );
```

実行結果

```
curpos = 8  ret_len = 0  nextpos = 7
curpos = 7  ret_len = 0  nextpos = 6
curpos = 6  ret_len = 0  nextpos = 5
curpos = 5  ret_len = 1  nextpos = 4
curpos = 4  ret_len = 0  nextpos = 3
curpos = 3  ret_len = 0  nextpos = 2
curpos = 2  ret_len = 2  nextpos = 0
curpos = 0  ret_len = 0  nextpos = 9
```

9.5.57 strcspn()

NAME

strcspn() — 左側から文字セットに含まれない文字が続く長さを調べる

SYNOPSIS

```
size_t strcspn( const char *reject ) const; .................................. 1
size_t strcspn( size_t pos, const char *reject ) const; .................... 2
size_t strcspn( size_t pos, const char *reject, size_t *nextpos ) const;   3
size_t strcspn( const tstring &reject ) const; ............................. 4
size_t strcspn( size_t pos, const tstring &reject ) const; ................ 5
size_t strcspn( size_t pos, const tstring &reject, size_t *nextpos ) const; 6
size_t strcspn( int reject ) const; .......................................... 7
size_t strcspn( size_t pos, int reject ) const; ............................ 8
size_t strcspn( size_t pos, int reject, size_t *nextpos ) const; ......... 9
```

DESCRIPTION

自身が持つ文字列の左側から右方向に，文字セット reject が最初に出現するまでの文字列の連続する長さを検索し，その長さを返します.

pos を指定した場合は，自身の文字列の位置 pos から検索を開始します．なお，文字列の先頭位置は 0 です．

メンバ関数 1〜6 の reject は，"ijk"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]"あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説を参照してください．

メンバ関数 1〜6 において，文字セット reject が NULL である場合，対象となる文字セットは全ての文字です．

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．このメンバ関数の返り値が 1 以上の場合は，pos からさらに返り値の分だけ右方向に移動させた位置が nextpos の指す変数に返ります．このメンバ関数の返り値が 0 の場合には，pos から 1 文字分だけ右方向に移動させた位置が自身の文字列長未満であれば，その値が nextpos の指す変数に返ります．それ以外の場合には，自身の文字列長が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

strspn() メンバ関数 (§9.5.55) で，文字セットを "[^A-Z]" のように指定する方法もあります．そちらも検討してください．

PARAMETER

[I]	reject	検出対象外の文字セット
[I]	pos	検出の開始位置
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

正の値	:	文字セットに含まれない文字が連続する長さ
0	:	文字セットに含まれない文字が，カウントを開始する位置から連続していなかった場合
	:	pos に自身の文字列長以上の値を指定した場合
	:	オブジェクト内に文字列がない場合

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列に文字 *0* が現れるまでに連続している文字数を左端から検索し，その結果を標準出力します．

```
stdstreamio sio;
tstring     my_str = "Akari20090309.txt";
int         i_ch   = '0';
size_t      t_ret  = 0;

if ((t_ret = my_str.strcspn(i_ch)) == 0) {
    エラー処理
}
else {
    sio.printf("%zu\n", t_ret);
```

```
}
```

実行結果

```
6
```

9.5.58 strmatch(), fnmatch(), pnmatch()

NAME

strmatch(), fnmatch(), pnmatch() — シェルライクな文字列マッチを試行

SYNOPSIS

```
int strmatch( const char *pat ) const;  ...................................... 1
int strmatch( size_t pos, const char *pat ) const;  .......................... 2
int strmatch( const tstring &pat ) const;  ................................... 3
int strmatch( size_t pos, const tstring &pat ) const;  ....................... 4
int fnmatch( const char *pat ) const;  ....................................... 5
int fnmatch( size_t pos, const char *pat ) const;  ........................... 6
int fnmatch( const tstring &pat ) const;  .................................... 7
int fnmatch( size_t pos, const tstring &pat ) const;  ........................ 8
int pnmatch( const char *pat ) const;  ....................................... 9
int pnmatch( size_t pos, const char *pat ) const;  ........................... 10
int pnmatch( const tstring &pat ) const;  .................................... 11
int pnmatch( size_t pos, const tstring &pat ) const;  ........................ 12
```

DESCRIPTION

自身の文字列に対し，シェルのワイルドカードパターンを用いた文字列マッチを試行し，その結果を返します．

strmatch() は，自身の文字列の位置 pos から右方向に，文字パターン pat によるマッチングを試行します．文字列マッチが試行される対象範囲は，文字列終端の'\0' が現れるまでです (改行文字'\n' が現れても処理は終了しません)．pos が指定されない場合は，文字列の左端から探します．なお，文字列の先頭位置は 0 です．

pat で利用可能なワイルドカードは，'*' と'?' および，正規表現で用いられる"[A-Z]"あるいは"[^A-Z]"のような指定が可能です[11]．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説を参照してください．

fnmatch() は strmatch() に制限を与え，文字列先頭のピリオド'.' を特別に取り扱います．このメンバ関数では，ワイルドカード'*' や'?' が文字列先頭ピリオド'.' にマッチしなくなります．このメンバ関数では，ファイル名検索をする事を想定しています．さらに，pnmatch() はスラッシュ'/' およびスラッシュ直後のピリオドを特別に取り扱います．このメンバ関数はパス名検索をする事を想定しています．

PARAMETER

[I] pat 文字パターン
[I] pos 文字列マッチの開始位置
([I]：入力，[O]：出力)

[11] インターバル「{}」,. 後方参照「\\n」，繰り返し表現「+」，文字集合「\\w」は使えません．

RETURN VALUE

0	:	自身の文字列が pat にマッチした場合
負の値 (エラー)	:	pat にマッチしなかった場合
	:	pos に自身の文字列長を越える値を指定した場合 (メンバ関数 2，4，6，8，10，12)
	:	オブジェクト内に文字列がない場合
	:	pat が NULL である場合

EXAMPLE

次のコードは，オブジェクト my_str が持つ文字列が文字パターン **.txt* にマッチするかを調べ，その結果を標準出力に出力します．

```
stdstreamio sio;
tstring     my_str = "./Akari20090309.txt";

if (my_str.strmatch("*.txt") != 0) {
    sio.printf("The character string is not shown by \"*.txt\".\n");
}
else {
    sio.printf("The character string is shown by \"*.txt\".\n");
}
```

実行結果

```
The character string is shown by "*.txt".
```

9.5.59 regmatch()

NAME

regmatch() — 拡張正規表現による文字列マッチを試行

SYNOPSIS

```
ssize_t regmatch( const char *pat, size_t *ret_span ) const; ............... 1
ssize_t regmatch( size_t pos, const char *pat, size_t *ret_span ) const; . 2
ssize_t regmatch( size_t pos, const char *pat, size_t *ret_span,
                  size_t *nextpos ) const; .................................. 3
ssize_t regmatch( const tstring &pat, size_t *ret_span ) const; ........... 4
ssize_t regmatch( size_t pos, const tstring &pat, size_t *ret_span ) const; 5
ssize_t regmatch( size_t pos, const tstring &pat, size_t *ret_span,
                  size_t *nextpos ) const; .................................. 6
ssize_t regmatch( const tregex &pat, size_t *ret_span ) const; ............ 7
ssize_t regmatch( size_t pos, const tregex &pat, size_t *ret_span ) const; 8
ssize_t regmatch( size_t pos, const tregex &pat, size_t *ret_span,
                  size_t *nextpos ) const; .................................. 9
```

DESCRIPTION

自身の文字列に対し，POSIX 拡張正規表現 (以下，正規表現) による文字列マッチを試行し，その結果を返します．

これらのメンバ関数では，後方参照の情報を得る事ができません．後方参照の情報を得たい場合は，tarray_tstring クラスの regassign() メンバ関数 (§10.4.13) の利用をお勧めします．

メンバ関数 1〜6 の場合，正規表現 pat をコンパイルし，その結果を自身が持つ内部バッファに保存し，マッチを行ないます (pat が前回と同じ場合は，再コンパイルしません)．

メンバ関数 7〜9 の場合，正規表現のコンパイル結果を保持している tregex クラスのオブジェクトを指定します．したがって，regmatch() メンバ関数を使う前に，あらかじめ tregex クラスの compile() メンバ関数で正規表現をコンパイルする必要があります (EXAMPLE-2 を参照)．

いずれの場合も，正規表現のコンパイルに失敗すると，標準エラー出力にその内容を出力します．

自身の文字列の位置 pos から右方向に文字列マッチを試行します．文字列マッチが試行される対象範囲は，文字列終端の'\0' が現れるまでです (改行文字'\n' が現れても処理は終了しません)．pos が指定されない場合は，文字列の左端から探します．なお，文字列の先頭位置は 0 です．

マッチした文字列の長さは*ret_span に返ります．マッチした文字列の長さの情報が不要な場合は，ret_span に NULL を指定する事もできます．

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．pat がマッチし，かつマッチした部分の長さ l が 1 以上の場合は，マッチした位置からさらに l だけ右に移動させた位置が nextpos の指す変数に返ります．マッチした部分の長さが 0 の場合は，マッチした位置から 1 文字分だけ右方向に移動させた位置が自身の文字列長以下であれば，その値が nextpos の指す変数に返ります．それ以外の場合には，自身の文字列長+1 が返ります．nextpos による値の取得が不要な場合は，NULL を与える事もできます．

正規表現の基本単位は，1 文字にマッチする正規表現です．文字パターン pat で利用可能な正規表現の文字列表現方法は表 21の通りです．

リストの中では，文字クラスも指定できます．指定できる文字クラスは §9.5.26の解説を参照してください．文字"]"をマッチ対象にする場合は，リストの先頭におく必要があります．その他，文字"^"はリスト先頭以外，文字"-"はリストの最後にそれぞれおいてください．

後方参照は"\\"の後ろに 0 でない 10 進数値文字 n が続くと，括弧でくくられた部分正規表現の n 番目にマッチした文字並びと同じものにマッチします．部分正規表現の番号付けは，開き括弧"(" の位置が左のものから右のものに向かってなされます．例えば，文字列"abc:def::abc:def" は文字パターン"(\\w+):(\\w+)::\\1:\\2"にマッチします．

PARAMETER

[I]	pos	文字列マッチの開始位置
[I]	pat	文字パターン (正規表現) または tregex クラスのコンパイル済オブジェクト
[O]	ret_span	マッチした文字列の長さ
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

正規表現の文字列表現	意味
`"."`	改行以外の任意の一文字
`"[…]"`	リスト (”[…]”のこと) に含まれるいずれかの一文字
`"[^…]"`	リストに含まれない，いずれかの一文字
`"a|b"`	”a”または”b”のいずれかに一致
`"(ab)"`	グループ化した”ab”と一致
`"\\w"`	英数字 (文字クラス `[[:alnum:]]` と同等)
`"\\W"`	英数字以外 (`[^[:alnum:]]` と同等)
`"^"`	パターンの行頭
`"$"`	パターンの行末
`"\\<"`	単語の先頭の空文字列
`"\\>"`	単語の末尾の空文字列
`"\\b"`	単語の端の空文字列
`"\\B"`	単語の端以外の空文字列
`"?"`	直前の文字の 0 回または 1 回以上の繰り返し
`"*"`	直前の文字の 0 回以上の繰り返し
`"+"`	直前の文字の 1 回以上の繰り返し
`"{n}"`	直前の文字の n 回の繰り返し
`"{n,}"`	直前の文字の n 回以上の繰り返し
`"{n,m}"`	直前の文字の n 回以上，m 回以下の繰り返し
`"\\n"`	後方参照

表 21: 正規表現の文字列表現方法一覧.

非負の値	:	`pat` にマッチした自身の文字列の位置
負の値 (エラー)	:	`pat` にマッチしなかった場合
	:	オブジェクト内に文字列がない場合
	:	`pos` に自身の文字列長を越える値を指定した場合
	:	`pat` が `NULL` である場合
	:	インターバルオペレータ`{}`が閉じていない場合
	:	リストオペレータ [] が閉じていない場合
	:	未知のキャラクタクラスを設定した場合 〚例えば [: up :] の場合が該当します.〛
	:	正規表現がバックスラッシュで終わっている場合
	:	グループオペレータ () が閉じていない場合
	:	無効な範囲のオペレータの使用の場合 〚例えば [9 – 0] の場合が該当します.〛
	:	サブエクスプレッション`\(...\)` への無効な後方参照をした場合
	:	無効な後方参照オペレータを使用した場合
	:	グループやリストなどの，パターンオペレータの無効な使用を行った場合 〚例えば [0 – 9) の場合が該当します.〛
	:	`'*'` が最初の文字としてくるような，無効な繰り返しオペレータを使用した場合 〚例えば $pat = " * .txt"$ の場合が該当します.〛

EXCEPTION

regex ルーチンがメモリを使い果たしている場合
内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE-1

次のコードは，オブジェクト my_str が持つ文字列に数字が 4 つ続いている文字列の位置を求めるため，文字パターン *[[:digit:]]{4}* を設定し，文字列マッチを行います．そして，その結果を標準出力に出力します．

```
stdstreamio sio;
tstring     my_str = "User ID : 1234";
size_t      t_span = 0;
ssize_t     t_ret = 0;

if ((t_ret = my_str.regmatch("[[:digit:]]{4}", &t_span)) < 0) {
    エラー処理
}
else {
    sio.printf("%zd, %zu\n", t_ret, t_span);
}
```

実行結果

10,4

EXAMPLE-2

EXAMPLE-1 と同じ処理を行なうコードですが，regmatch() を使う前にあらかじめ正規表現をコンパイルしている点が異なります．正規表現のコンパイルは，compile() メンバ関数を使って my_pat.complie("[[:digit:]]{4}") のように行ないます．コンパイル処理のエラーチェックには cregex() を利用します．

```
stdstreamio sio;
tregex      my_pat;
tstring     my_str = "User ID : 1234";
size_t      t_span = 0;
ssize_t     t_ret = 0;

my_pat.compile("[[:digit:]]{4}");        /* ←ここでコンパイルしている */
if ( my_pat.cregex() == NULL ) {
    /* エラー処理: 正規表現のコンパイルに失敗 */
}
if ((t_ret = my_str.regmatch(my_pat, &t_span)) < 0) {
    エラー処理
}
else {
    sio.printf("%zd, %zu\n", t_ret, t_span);
}
```

```
my_pat.init();                              /* コンパイル結果を破棄 */
```

10 TARRAY_TSTRING クラス

tarray_tstring クラスを使えば，文字列配列を簡単に扱う事ができます．配列の 1 つ 1 つの要素は tstring クラス (§9) のオブジェクトであり，tstring クラスの API と融合する事で使いやすい文字列配列 API を実現しています．

次のような特徴を持ちます．

- メモリは自動確保されるため，オブジェクトを作った後にすぐに代入が可能．
- いつでも文字列へポインタ配列 (NULL 終端) を取得でき，libc の `execvp()` のような関数との連携が容易．
- `printf()` の記法が多くのメンバ関数で利用可能．
- `[]` または at() メンバ関数を通して，tstring クラス (§9) の豊富なメンバ関数が利用可能．
- 空白区切，TAB 区切り，CSV 形式の文字列を簡単に分割し配列化．
- 全配列要素に対して一気に文字列の編集を行なう事ができるメンバ関数が利用可能 (例: `chomp()`, `trim()` 等)．使い方は，tstring クラス (§9) の場合と同じ．
- 配列に対して，POSIX 拡張正規表現を含む検索処理のための API を用意．

tarray_tstring クラスを使う場合は,「`#include <sli/tarray_tstring.h>`」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は,「`using namespace sli;`」もコードに書いてください．

簡単な使用例を次に示します．

```
#include <sli/stdstreamio.h>
#include <sli/asarray_tstring.h>
using namespace sli;

int main()
{
    stdstreamio sio;
    tarray_tstring my_arr;
    size_t i=0;
    my_arr[i++] = "MacOSX";                          /* 要素 0 に "MacOSX" を代入 */
    my_arr[i++].printf("Linux");                     /* 要素 1 に "Linux" を代入 */
    my_arr.at(i).printf("Solaris");                  /* 要素 2 に "Solaris" を代入 */
    for ( i=0 ; i < my_arr.length() ; i++ ) {        /* すべての要素を表示 */
        sio.printf("%zu ... [%s]\n", i, my_arr.cstr(i));
    }
}
```

実行結果

```
0 ... [MacOSX]
1 ... [Linux]
2 ... [Solaris]
```

10.1 オブジェクトの作り方

3 通りの方法で，オブジェクトに初期値を与える事ができます[12)].

1 つ目は，何も引数を指定しない方法です.

```
tarray_tstring my_arr0;
```

この状態では，文字列用のバッファもポインタ配列用のバッファも確保されていません.

2 つ目は，可変長引数で与える方法です.

```
tarray_tstring my_arr0("tokyo", "osaka", "nagoya", NULL);
```

この場合は，与えられた文字列で配列を初期化します．引数の最後は必ず NULL を与えます.

3 つ目は，char *[] 型の NULL 終端のポインタ配列を与える方法です．例えば，main() 関数の argv をそのまま与える事ができます.

```
tarray_tstring my_arr0(argv);
```

この場合，与えられた文字列配列 argv でオブジェクト内の配列を初期化します.

10.2 メンバ関数一覧

表 22 はメンバ関数一覧です.

	メンバ関数名	機能
§10.3.1	[]	指定要素の文字列オブジェクトの参照
§10.3.2	=	文字列配列を代入
§10.3.3	+=	文字列配列の追加
§10.3.4	+=	文字列要素の追加
§10.4.1	length()	文字列配列の長さ (個数)，または値文字列の長さ
§10.4.2	cstrarray()	値文字列のポインタ配列 (NULL 終端) のアドレス
§10.4.3	cstr(), c_str()	指定要素の値文字列のアドレス
§10.4.4	at(), at_cs()	指定要素の文字列オブジェクトの参照
§10.4.5	dprint()	オブジェクト情報を標準エラー出力へ出力 (ユーザプログラムのデバッグ用)
§10.4.6	copy()	配列 (の一部) を外部オブジェクトへコピー
§10.4.7	swap()	オブジェクトの入替え
§10.4.8	init()	オブジェクトの完全初期化
§10.4.9	assign(), assignf()	オブジェクトの初期化と代入 (1 つの文字列を指定)
§10.4.10	assign(), vassign()	オブジェクトの初期化と代入 (複数の文字列または文字列配列を指定)
§10.4.11	explode()	引数の文字列を分割し，配列に代入 (単純かつ高速)
§10.4.12	split()	引数の文字列を分割し，配列に代入 (多機能版)
§10.4.13	regassign()	引数の文字列に正規表現マッチを行ない，その結果を配列に代入

表 22: tarray_tstring クラスで利用可能なメンバ関数一覧 (次ページに続く).

12) tstring クラス (§9) のような動作モードはありません.

	メンバ関数名	機能
§10.4.14	put(), putf()	任意の要素位置へ文字列を n 個セット
§10.4.15	put(), vput()	任意の要素位置へ複数の文字列または文字列配列をセット
§10.4.16	append(), appendf()	要素の追加 (1 つの文字列を指定)
§10.4.17	append(), vappend()	要素の追加 (複数の文字列または文字列配列を指定)
§10.4.18	insert(), insertf()	要素の挿入 (1 つの文字列を指定)
§10.4.19	insert(), vinsert()	要素の挿入 (複数の文字列または文字列配列を指定)
§10.4.20	replace(), replacef()	要素の置換 (1 つの文字列を指定)
§10.4.21	replace(), vreplace()	要素の置換 (複数の文字列または文字列配列を指定)
§10.4.22	erase()	要素の削除
§10.4.23	clean()	既存の配列の要素値すべてを任意の文字列でパディング
§10.4.24	resize()	配列の長さを変更
§10.4.25	resizeby()	配列の長さを相対的に変更
§10.4.26	crop()	配列の切り抜き
§10.4.27	chomp()	全要素の改行文字の除去
§10.4.28	trim()	全要素の両端スペースの除去
§10.4.29	ltrim()	全要素の左端スペースの除去
§10.4.30	rtrim()	全要素の右端スペースの除去
§10.4.31	strreplace()	全要素の文字列検索と置換
§10.4.32	regreplace()	全要素の正規表現による文字列検索と置換
§10.4.33	tolower()	全要素の大文字を小文字に置換
§10.4.34	toupper()	全要素の小文字を大文字に置換
§10.4.35	expand_tabs()	全要素の TAB 文字を空白文字に置換
§10.4.36	contract_spaces()	全要素の空白文字を TAB 文字に置換
§10.4.37	find_elem()	左側 (先頭) から配列要素を検索
§10.4.38	rfind_elem()	右側 (最後尾) から配列要素を検索
§10.4.39	find()	配列の左側 (先頭) から文字列を検索
§10.4.40	rfind()	配列の右側 (最後尾) から文字列を検索
§10.4.41	find_matched_str()	パターンにマッチする要素 (文字列) を検索
§10.4.42	find_matched_fn()	パターンにマッチする要素 (ファイル名) を検索
§10.4.43	find_matched_pn()	パターンにマッチする要素 (パス名) を検索
§10.4.44	regmatch() [Normal 版]	拡張正規表現で文字列を検索
§10.4.45	regmatch() [Advanced 版]	拡張正規表現で文字列を検索

表 22: tarray_tstring クラスで利用可能なメンバ関数一覧 (続き).

10.3 演算子

10.3.1 []

NAME

[] — 指定要素の文字列オブジェクトの参照

SYNOPSIS

```
tstring &operator[]( size_t index ); ........................................ 1
const tstring &operator[]( size_t index ) const; ........................... 2
```

DESCRIPTION

添え字で指定された配列要素 (tstring クラス (§9) のオブジェクト) の参照を返します.「[]」の直後に「.」で接続し，tstring クラス (§9) のメンバ関数を使う事ができます (EXAMPLE では tstring クラスの「=」演算子と cstr() メンバ関数を使っています).

メンバ関数 1 は読み書き両用で at() と同じ動作をします．メンバ関数 2 は読み取り専用で at_cs() と同じ動作をします.

index に配列の長さ以上の値が指定された場合，メンバ関数 1 では配列の長さが自動拡張されますが，メンバ関数 2 では例外が発生します．なお，配列の先頭の要素番号は 0 です.

メンバ関数 1・メンバ関数 2 のどちらが使われるかは，オブジェクトの「const」属性の有無で自動的に決まります.「const」属性が無い場合にはメンバ関数 1 が，有る場合にはメンバ関数 2 が自動的に選択されます.

at()，at_cs() についての詳細は §10.4.4を参照してください.

PARAMETER

[I] index 0 から始まる要素番号

RETURN VALUE

添え字で指定された配列要素の参照

EXCEPTION

内部バッファの確保に失敗した場合 (メンバ関数 1)
index に配列長以上の値が指定された場合 (メンバ関数 2)

EXAMPLE

次のコードは，オブジェクト my_arr が持つ文字列配列に，オペレータ「[]」を使って文字列 "camellia"を追加し，その結果を標準出力します．length() に関しては §10.4.1の解説を参照してください.

```
stdstreamio sio;

tarray_tstring my_arr("hawthorn", "oak", NULL);
my_arr[2] = "camellia";

/* 表示 */
for ( size_t i=0 ; i < my_arr.length() ; i++ ) {
    sio.printf("[%s]\n", my_arr[i].cstr());
}
```

実行結果

```
[hawthorn]
[oak]
[camellia]
```

10.3.2 =

NAME

= — 文字列配列を代入

SYNOPSIS

```
tarray_tstring &operator=(const tarray_tstring &obj); ...................... 1
const char *const *operator=(const char *const *elements); ................ 2
```

DESCRIPTION

演算子の右側 (引数) で指定されたオブジェクトの内容または文字列配列を自身に代入します.

PARAMETER

[I]	`obj`	tarray_tstring クラスのオブジェクト
[I]	`elements`	文字列のポインタ配列のアドレス (NULL 終端)

RETURN VALUE

自身の参照 (メンバ関数 1)
内部文字列バッファへのポインタ配列 (メンバ関数 2)

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合 (メンバ関数 1)

EXAMPLE

次のコードは，オブジェクト my_arr に，オペレータ「=」を使って文字列のポインタ配列 menu の内容を文字列配列として代入し，その結果を標準出力します．cstr()，length() に関しては §10.4.3，§10.4.1の解説を参照してください.

```
stdstreamio sio;

const char *menu[] = {"rice ball", "sushi", "tofu", NULL};
tarray_tstring my_arr;
my_arr = menu;          /* ’=’ は tarray_tstring クラスの演算子 */

/* 表示 */
for ( size_t i=0 ; i < my_arr.length() ; i++ ) {
    sio.printf("[%s]\n", my_arr.cstr(i));
}
```

実行結果

```
[rice ball]
```

```
[sushi]
[tofu]
```

10.3.3 +=

NAME

+= — 文字列配列の追加

SYNOPSIS

```
tarray_tstring &operator+=(const tarray_tstring &obj);   .................... 1
const char *const *operator+=(const char *const *elements);   ............... 2
```

DESCRIPTION

自身の配列に，演算子の右側 (引数) で指定された文字列配列の追加を行います.

PARAMETER

[I]	`obj`	tarray_tstring クラスのオブジェクト
[I]	`elements`	文字列のポインタ配列のアドレス (NULL 終端)

RETURN VALUE

自身の参照 (メンバ関数 1)
内部文字列バッファへのポインタ配列 (メンバ関数 2)

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合 (メンバ関数 1)

EXAMPLE

次のコードは，オブジェクト my_arr の文字列配列に，オペレータ「+=」を使って文字列配列 addTree を追加し，その結果を標準出力します. cstr()，length() に関しては §10.4.3, §10.4.1の解説を参照してください.

```
stdstreamio sio;

tarray_tstring my_arr("ginkgo", "Japanese apricot", "maple", NULL);

const char *addTree[] = {"oak", "cherry tree", NULL};
my_arr += addTree;

/* 表示 */
for ( size_t i=0 ; i < my_arr.length() ; i++ ) {
    sio.printf("[%s]\n", my_arr.cstr(i));
}
```

実行結果

```
[ginkgo]
[Japanese apricot]
```

```
[maple]
[oak]
[cherry tree]
```

10.3.4 +=

NAME

+= — 文字列要素の追加

SYNOPSIS

```
tarray_tstring &operator+=(const char *str);   ........ 1
tarray_tstring &operator+=(const tstring &str);  ...... 2
```

DESCRIPTION

自身の配列に，演算子の右側 (引数) で指定された文字列を追加します.

PARAMETER

[I] str 文字列のアドレス (メンバ関数 1)
tstring クラスのオブジェクト (メンバ関数 2)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_arr が持つ文字列配列に，オペレータ「+=」を使って文字列 "wisteria"を追加し，その結果を標準出力します．cstr()，length() に関しては §10.4.3, §10.4.1の解説を参照してください.

```
stdstreamio sio;

tarray_tstring my_arr("nandina", "elm", NULL);
my_arr += "wisteria";

/* 表示 */
for ( size_t i=0 ; i < my_arr.length() ; i++ ) {
    sio.printf("[%s]\n", my_arr.cstr(i));
}
```

実行結果

```
[nandina]
[elm]
[wisteria]
```

10.4 メンバ関数

全体的な注意事項

size_t 型は符号なし整数として数値を取り扱います．size_t 型を引数に持つメンバ関数に負の値を設定した場合，アボートする可能性が高くなります．負の値を設定しないようにしてください．

10.4.1 length()

NAME

length() — 文字列配列の長さ (個数)，または値文字列の長さ

SYNOPSIS

`size_t length() const;` 1

`size_t length( size_t index ) const;` 2

DESCRIPTION

自身の配列の長さ (個数) を返します (メンバ関数 1).

引数 `index` が指定された場合は，引数の要素番号に該当する要素の文字列の長さを返します (メンバ関数 2)．なお，配列の先頭の要素番号は `0` です.

RETURN VALUE

文字列配列数 (メンバ関数 1)，または指定要素の文字列長 (メンバ関数 2)

EXAMPLE

次のコードは，オブジェクト `my_arr` が持つ文字列配列の配列数と，各要素の文字列長を標準出力します．`at()` に関しては §10.4.4の解説を参照してください.

```
stdstreamio sio;

tarray_tstring my_arr;
my_arr.at(0).printf("Hello");
my_arr.at(1).printf("Hoge");

sio.printf("my_arr length = %zu\n",my_arr.length());
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("my_arr index%zu length = %zu\n",i, my_arr.length(i));
}
```

実行結果

```
my_arr length = 2
my_arr index0 length = 5
my_arr index1 length = 4
```

10.4.2 cstrarray()

NAME

cstrarray() — 値文字列のポインタ配列 (NULL 終端) のアドレス

SYNOPSIS

```
const char *const *cstrarray() const;
```

DESCRIPTION

自身が持つ文字列配列のそれぞれの文字列要素についてのポインタ配列のアドレスを返します．ポインタ配列は，必ず NULL で終端しています．

RETURN VALUE

文字列バッファへのポインタ配列 (NULL 終端)

EXAMPLE

次のコードは，オブジェクト my_arr に tmp の内容を文字列配列として代入後，文字列のポインタ配列のアドレスを取得し，各要素の内容を標準出力します．

```
stdstreamio sio;

/* オブジェクトの初期化 */
const char *tmp[] = {"linux", "windows", "mac", NULL};
tarray_tstring my_arr(tmp);

/* 文字列のポインタ配列のアドレスを取得する */
const char *const *ptr = my_arr.cstrarray();
if ( ptr != NULL ) {
    int i;
    for ( i=0 ; ptr[i] != NULL ; i++ ) {
        sio.printf("%d ... [%s]\n", i, ptr[i]);
    }
}
```

実行結果

```
0 ... [linux]
1 ... [windows]
2 ... [mac]
```

§3.4.3に cstrarray() メンバ関数を用いた execvp 関数の使用例があります．

10.4.3 cstr(), c_str()

NAME

cstr(), c_str() — 指定要素の値文字列のアドレス

SYNOPSIS

```
const char *cstr( size_t index ) const;   ........ 1
const char *c_str( size_t index ) const;  ........ 2
```

DESCRIPTION

index で指定された要素の値文字列のアドレスを返します．index に配列の長さ以上の値が指定された場合，NULL が返ります．なお，配列の先頭の要素番号は 0 です．

cstr()，c_str() は名前は違いますが，同じ動作をします．

RETURN VALUE

値文字列のアドレス

EXAMPLE

次のコードは，オブジェクト my_arr に tmp の内容を文字列配列として代入後，my_arr の各要素の先頭アドレスを取得し，内容を標準出力します．

```
stdstreamio sio;

/* オブジェクトの初期化 */
const char *tmp[] = {"linux", "windows", "mac", NULL};
tarray_tstring my_arr(tmp);

/* 表示 */
for ( size_t i=0 ; i < my_arr.length() ; i++ ) {
    sio.printf("[%s]\n", my_arr.cstr(i));
}
```

実行結果

```
[linux]
[windows]
[mac]
```

10.4.4 at(), at_cs()

NAME

at(), at_cs() — 指定要素の文字列オブジェクトの参照

SYNOPSIS

```
tstring &at( size_t index ); ........................ 1
const tstring &at( size_t index ); const ............ 2
const tstring &at_cs( size_t index ) const; ......... 3
```

DESCRIPTION

index で指定された配列要素 (tstring クラス (§9) のオブジェクト) の参照を返します．これらメンバ関数の直後に「.」で接続し，tstring クラス (§9) のメンバ関数を使う事ができます (EXAMPLE では tstring クラスの cstr() メンバ関数を使っています)．なお，配列の先頭の要素番号は 0 です．

メンバ関数 1 は，文字列の読み書き両方に利用でき，メンバ関数 2,3 は，読み取り専用です．

at() メンバ関数の場合では，メンバ関数 1・メンバ関数 2 のどちらが使われるかは，オブジェクトの「const」属性の有無で自動的に決まります.「const」属性が無い場合にはメンバ関数 1 が，有る場合にはメンバ関数 2 が自動的に選択されます.

メンバ関数 1 で配列長以上の index が指定された場合は，新しい配列要素が作られ，""(長さ 0 の文字列) が代入されます．バッファの確保に失敗した場合を除き，例外は発生しません.

メンバ関数 2,3 で配列長以上の index が指定された場合は，例外が発生します.

PARAMETER

[I] index 要素番号

([I]：入力，[O]：出力)

RETURN VALUE

指定された要素番号に該当する文字列 (tstring) オブジェクトの参照

EXCEPTION

内部バッファの確保に失敗した場合 (メンバ関数 1)

配列長以上の index が指定された場合 (メンバ関数 2,3)

EXAMPLE

次のコードは，文字列配列 my_arr の要素番号 0 の要素に文字列を代入し，my_arr の要素番号 0 と 1 の内容を標準出力します.

```
stdstreamio sio;

tarray_tstring my_arr;
my_arr.at(0) = "Hello";
sio.printf("my_arr[0] = %s\n",my_arr.at(0).cstr());
sio.printf("my_arr[1] = %s\n",my_arr.at(1).cstr());
```

実行結果

```
my_arr[0] = Hello
my_arr[1] =
```

10.4.5 dprint()

NAME

dprint() — オブジェクト情報を標準エラー出力へ出力 (ユーザのデバッグ用)

SYNOPSIS

```
void dprint() const;
```

DESCRIPTION

自身のオブジェクト情報を標準エラー出力へ出力します.

ユーザ・プログラムのデバッグを目的としたメンバ関数です.

EXAMPLE

次のコードは，オブジェクト my_array の情報を標準エラー出力に出力します．実行結果では，[] にオブジェクトのアドレスが表示されていますが，これは実行環境により異なります．

```
tarray_tstring my_array("MZ-80B", "MZ-2861", "X1C", "X1 turboZ", NULL);
my_array.dprint();
```

実行結果

```
sli::tarray_tstring[obj=0x7fbffff3e0] = {"MZ-80B", "MZ-2861", "X1C", "X1 turboZ"}
```

10.4.6 copy()

NAME

copy() — 文字列配列 (の一部) を別の文字列配列にコピー

SYNOPSIS

```
ssize_t copy( tarray_tstring *dest ) const;  ................................ 1
ssize_t copy( size_t index, tarray_tstring *dest ) const;  ................. 2
ssize_t copy( size_t index, size_t n, tarray_tstring *dest ) const;  ...... 3
```

DESCRIPTION

自身の文字列配列のすべてまたは一部を，dest で指定されたオブジェクトにコピーします．返り値は，dest に書き込まれる要素数です．

メンバ関数 1 は，文字列配列すべてを dest へコピーします．

メンバ関数 2 および 3 は，文字列配列の要素番号 index からの要素をコピーします．なお，配列の先頭の要素番号は 0 です．またメンバ関数 3 は，コピーする要素の個数 n を指定できます．

index + n の値がコピー元の要素数を超える場合，index 位置から最後の要素迄コピーされます．index の値がコピー元の要素数を超える場合，dest の内容は消去され，返り値が-1 となります．

PARAMETER

[O]	dest	コピー先の tarray_tstring クラスのオブジェクト
[I]	index	コピー元 (自身) の配列要素の位置
[I]	n	コピーする要素数

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	コピーした要素数
負の値 (エラー)	:	index に文字列配列長以上の値が指定された場合

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，文字列配列 my_menu の要素番号 2 から 2 個の要素を，文字列配列 dest_arr へコピーし，コピー先 dest_arr の内容を標準出力します．

```
stdstreamio sio;

tarray_tstring my_menu("pickles", "natto", "tempura", "sukiyaki", NULL);
tarray_tstring dest_arr;

my_menu.copy(2, 2, &dest_arr);
for ( size_t i = 0 ; i < dest_arr.length() ; i++ ) {
    sio.printf("dest_arr[%zu] = %s\n", i, dest_arr.cstr(i));
}
```

実行結果

```
dest_arr[0] = tempura
dest_arr[1] = sukiyaki
```

10.4.7 swap()

NAME

swap() — 文字列配列オブジェクトの入替え

SYNOPSIS

`tarray_tstring &swap( tarray_tstring &sobj );`

DESCRIPTION

オブジェクト `sobj` の内容と自身の内容とを入れ替えます.

PARAMETER

[I/O] `sobj` 内容を入れ替える tarray_tstring クラスのオブジェクト
([I]：入力, [O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは, 文字列配列 `myMenu_arr` と文字列配列 `myTree_arr` を入替え, その結果を標準出力します.

```
stdstreamio sio;

tarray_tstring myMenu_arr("rice ball", "sushi", "tofu", NULL);
tarray_tstring myTree_arr("Pine", "Ginkgo", "Magnolia", NULL);

myMenu_arr.swap(myTree_arr);
for ( size_t i = 0 ; i < myMenu_arr.length() ; i++ ) {
    sio.printf("myMenu_arr[%zu] = %s\n", i, myMenu_arr.cstr(i));
}
```

実行結果

```
myMenu_arr[0] = Pine
myMenu_arr[1] = Ginkgo
myMenu_arr[2] = Magnolia
```

10.4.8 init()

NAME

init() — オブジェクトの完全初期化

SYNOPSIS

`tarray_tstring &init();` 1
`tarray_tstring &init(const tarray_tstring &obj);` 2

DESCRIPTION

自身の文字列配列の初期化を行います.

メンバ関数 1 は，文字列配列を完全に初期化します．文字列配列の配列と文字列バッファに割り当てられたメモリ領域は完全に開放されるため，`cstrarray()` メンバ関数 (§10.4.2) を実行すると `NULL` が返ります.

メンバ関数 2 は `obj` の内容で初期化します (`obj` の内容すべてを自身にコピーします).

PARAMETER

[I] `obj` tarray_tstring クラスのオブジェクト (コピー元)
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合 (メンバ関数 2)

EXAMPLE

次のコードは，オブジェクト `my_arr` を `my_treeArr` の内容で初期化し，その内容を標準出力します．更に `init()` で完全に初期化し，配列長を標準出力します.

```
stdstreamio sio;

tarray_tstring my_treeArr("pine", "willow", NULL);
tarray_tstring my_arr;

my_arr.init(my_treeArr);
/* 表示 */
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("[%s]\n", my_arr.cstr(i));
}
```

```
/* 完全初期化 */
my_arr.init();
/* 表示 */
sio.printf("my_arr.length = [%zu]\n", my_arr.length());
```

実行結果

```
[pine]
[willow]
my_arr.length = [0]
```

10.4.9 assign(), assignf(), vassignf()

NAME

assign(), assignf(), vassignf() — オブジェクトの初期化と代入 (1 つの文字列を指定)

SYNOPSIS

```
tarray_tstring &assign( const char *str, size_t n ); ........................ 1
tarray_tstring &assign( const tstring &str, size_t n ); .................... 2
tarray_tstring &assignf( size_t n, const char *format, ... ); ............. 3
tarray_tstring &vassignf( size_t n, const char *format, va_list ap ); .... 4
```

DESCRIPTION

自身の配列要素を n 個とし，すべての要素に指定された文字列を代入します.

メンバ関数 1，2 は，n 個の配列要素に文字列 str を代入します.

メンバ関数 3，4 は，n 個の配列要素に，format で指定された書式に従って作成した文字列を代入します．メンバ関数 3 では，可変長引数の各要素データを format の変換指定に応じて変換します．メンバ関数 4 では，可変長引数のリスト ap を format の変換指定に応じて変換します．format については §8.1.14の解説を参照してください.

PARAMETER

[I]	n	配列の要素数
[I]	str	源泉となる文字列
[I]	format	源泉となる文字列のためのフォーマット指定
[I]	...	format に対応した可変長引数の各要素データ
[I]	ap	format に対応した可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

可変長引数の各要素データが指定された変換フォーマットで変換できない値の場合 (メンバ関数 3，4)

EXAMPLE

次のコードは，書式指定を用いて，文字列「***」+文字列「Japanese quince」で初期化し，3 つの要素を持つ文字列配列 my_arr を作成します．確認のため，その結果を標準出力します．

```
stdstreamio sio;

tarray_tstring my_arr;
my_arr.assignf(3, "***%s", "Japanese quince");
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("my_arr[%zu] = %s\n", i, my_arr.cstr(i));
}
```

実行結果

```
my_arr[0] = ***Japanese quince
my_arr[1] = ***Japanese quince
my_arr[2] = ***Japanese quince
```

10.4.10 assign(), vassign()

NAME

assign(), vassign() — オブジェクトの初期化と代入 (複数の文字列または文字列配列を指定)

SYNOPSIS

```
tarray_tstring &assign( const char *el0, const char *el1, ... );  ........ 1
tarray_tstring &vassign( const char *el0, const char *el1, va_list ap );   2
tarray_tstring &assign( const char *const *elements );  .................... 3
tarray_tstring &assign( const char *const *elements, size_t n );  ........ 4
tarray_tstring &assign( const tarray_tstring &src, size_t idx2 = 0 );  .... 5
tarray_tstring &assign( const tarray_tstring &src, size_t idx2,
                        size_t n2 );  ........................................ 6
```

DESCRIPTION

el0，el1，... で指定された複数の文字列，または elements，src で指定された文字列配列で，自身の文字列配列を初期化します．

メンバ関数 1,2 では，el0，el1 と可変長引数または可変長引数のリスト ap を指定します．可変長引数は NULL で終端していなければなりません．

メンバ関数 3,4 では，引数 elements に文字列のポインタ配列を指定します．メンバ関数 3 の場合，ポインタ配列は NULL で終端していなければなりません．メンバ関数 4 の場合は，n で要素数を指定できます．elements の個数 (NULL に達するまで) を超えた n が指定された場合は，n は無視されます．

メンバ関数 5 およびメンバ関数 6 は，源泉となる文字列配列 src の開始要素位置を idx2 で，要素数を n2 で指定できます．メンバ関数 5 は，idx2 を指定しなくても使用できます．その場合は 0 が指定されたものとして処理を行います．メンバ関数 6 は，源泉となる要素の個数 n2 を指定できます．なお，配列の先頭の要素番号は 0 です．

PARAMETER

[I]	`el0`	要素に入る文字列 (0 番目)
[I]	`el1`	要素に入る文字列 (1 番目)
[I]	`...`	要素に入る文字列 (2 番目以降; 要 NULL 終端)
[I]	`ap`	要素に入る文字列の可変長引数のリスト (2 番目以降; NULL 終端)
[I]	`elements`	要素に入る文字列のポインタ配列 (メンバ関数 3 の場合，NULL で終端)
[I]	`n`	配列 elements の要素数
[I]	`src`	源泉となる文字列配列を持つ tarray_tstring クラスのオブジェクト
[I]	`idx2`	src 中の要素の開始位置 (src の部分配列を代入する場合)
[I]	`n2`	src 中の要素の個数 (src の部分配列を代入する場合)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，文字列配列 myTree の要素番号 1 から 2 個の要素を用いて，文字列配列 my_arr を初期化します．確認のため，その結果を標準出力します．

```
stdstreamio sio;

const tarray_tstring myTree("fir", "magnolia", "dogwood", NULL);
tarray_tstring my_arr;

/* 配列 myTree の要素番号 1 から 2 個 */
my_arr.assign(myTree,1,2);
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("*** my_arr[%zu] = %s\n", i, my_arr.cstr(i));
}
```

実行結果

```
*** my_arr[0] = magnolia
*** my_arr[1] = dogwood
```

10.4.11 explode()

NAME

explode() — 文字列を分割し文字列配列へ代入 (単純かつ高速)

SYNOPSIS

```
tarray_tstring &explode( const char *src_str, const char *delim );   ...... 1
tarray_tstring &explode( const tstring &src_str, const char *delim )  ..... 2
```

DESCRIPTION

explode() メンバ関数は，文字列 src_str を区切り文字列 delim で分割し，その結果を文字列配列として自身へ代入します．区切り文字列は常に要素と要素との間に存在しているものとして扱うので，分割後の要素は文字列長ゼロもあり得ます．

split() メンバ関数 (§10.4.12) とは異なり動作が非常に単純なので，相対的に高速に動作します．

PARAMETER

[I]	`src_str`	分割対象の文字列
[I]	`delim`	区切り文字列

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

split() メンバ関数 (§10.4.12) の EXAMPLE をご覧ください．

10.4.12 split()

NAME

split() — 文字列を分割し文字列配列へ代入 (多機能版)

SYNOPSIS

```
tarray_tstring &split( const char *src_str, const char *delims,
                       bool zero_str, const char *quotations,
                       int escape, bool rm_escape );  ....................... 1
tarray_tstring &split( const tstring &src_str, const char *delims,
                       bool zero_str, const char *quotations,
                       int escape, bool rm_escape );  ....................... 2
tarray_tstring &split( const char *src_str, const char *delims,
                       bool zero_str = false );  ............................ 3
tarray_tstring &split( const tstring &src_str, const char *delims,
                       bool zero_str = false );  ............................ 4
```

DESCRIPTION

文字列 `src_str` を区切り文字で分割し，その結果を文字列配列として自身へ代入します．区切り文字は、文字リストとして `delims` で与え，`" \t"`のような単純な文字リストに加え，正規表現で用いられる `"[A-Z]"`あるいは`"[^A-Z]"`のような指定が可能です．さらに，`"[…]"`の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説と表 19 を参照してください．

分割後の要素として，ゼロの文字列長を許すかどうかを，`zero_str` で指定できます．`zero_str` が `false` の場合，文字列長 0 の要素の作成を許可しません．`zero_str` が `true` の場合，文字列長 0 の要素の作成を許可します (csv 形式の場合などに使います)．`zero_str` の指定がない場合は，`false` の扱いとなります．

クォーテーション等,「特定の文字」で囲まれた文字列は分割しない場合，`quotations` に「特定の文字」を指定します．たとえば，シングルクォーテーションで囲む文字列を分割対象外とする場合は，`"'"`と指定します．

エスケープ文字は escape で指定します．分割後のエスケープ文字を削除する場合は，rm_escape を true にセットします．ただし，quotations で指定された文字で囲まれた部分のエスケープ文字は削除しません．

PARAMETER

[I]	src_str	分割対象の文字列
[I]	delims	区切り文字を含む文字列
[I]	zero_str	分割後に長さ 0 の文字列要素を許すか否か (true/false)
[I]	quotations	クォーテーション文字を含む文字列
[I]	escape	エスケープ文字
[I]	rm_escape	エスケープ文字を削除するかどうかのフラグ (true/false)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，文字列 line を文字列" "で分割し，その結果をオブジェクト my_arr に代入し，標準出力します．

```
stdstreamio sio;

const char *line = "Fragrant olive is 'KINMOKUSEI'. It is good smell.";
tarray_tstring my_arr;
my_arr.split(line, " ", false);
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("my_arr[%zu] ===> %s\n", i, my_arr.cstr(i));
}
```

実行結果

```
my_arr[0] ===> Fragrant
my_arr[1] ===> olive
my_arr[2] ===> is
my_arr[3] ===> 'KINMOKUSEI'.
my_arr[4] ===> It
my_arr[5] ===> is
my_arr[6] ===> good
my_arr[7] ===> smell.
```

§3.4.7にメンバ関数 1 を使った例があります．

CSV 形式の文字列を分割する例は §10.4.28 の EXAMPLE で紹介しています．

10.4.13 regassign()

NAME

regassign() — 引数の文字列に正規表現マッチを行ない，その結果を配列に代入

SYNOPSIS

```
tarray_tstring &regassign( const char *src_str, const char *pat );  ....... 1
tarray_tstring &regassign( const char *src_str, size_t pos,
                           const char *pat );  ........................... 2
tarray_tstring &regassign( const char *src_str, size_t pos,
                           const char *pat, size_t *nextpos );  ........... 3
tarray_tstring &regassign( const tstring &src_str, const char *pat );  .... 4
tarray_tstring &regassign( const tstring &src_str, size_t pos,
                           const char *pat );  ........................... 5
tarray_tstring &regassign( const tstring &src_str, size_t pos,
                           const char *pat, size_t *nextpos );  ........... 6
tarray_tstring &regassign( const char *src_str, const tstring &pat );  .... 7
tarray_tstring &regassign( const char *src_str, size_t pos,
                           const tstring &pat );  ........................ 8
tarray_tstring &regassign( const char *src_str, size_t pos,
                           const tstring &pat, size_t *nextpos );  ....... 9
tarray_tstring &regassign( const tstring &src_str, const tstring &pat);   10
tarray_tstring &regassign( const tstring &src_str, size_t pos,
                           const tstring &pat );  ...................... 11
tarray_tstring &regassign( const tstring &src_str, size_t pos,
                           const tstring &pat, size_t *nextpos );  ...... 12
tarray_tstring &regassign( const char *src_str, const tregex &pat );  .... 13
tarray_tstring &regassign( const char *src_str, size_t pos,
                           const tregex &pat );  ....................... 14
tarray_tstring &regassign( const char *src_str, size_t pos,
                           const tregex &pat, size_t *nextpos );  ....... 15
tarray_tstring &regassign( const tstring &src_str, const tregex &pat );   16
tarray_tstring &regassign( const tstring &src_str, size_t pos,
                           const tregex &pat );  ....................... 17
tarray_tstring &regassign( const tstring &src_str, size_t pos,
                           const tregex &pat, size_t *nextpos );  ....... 18
```

DESCRIPTION

文字列 src_str に対し，pat で指定された POSIX 拡張正規表現 (以下，正規表現) による文字列マッチを試行し，マッチした場合は後方参照されるべき部分文字列を，文字列配列として自身に格納します (配列の長さは 1 以上になります)．マッチしなかった場合や正規表現に誤りがある等 (詳細は §9.5.59の RETURN VALUE の項をご覧ください) の原因で処理にエラーが発生した場合は，自身の配列が初期化されて何も代入されません (配列の長さが 0 になります)．

マッチした部分の位置は，reg_pos() メンバ関数で得る事ができます．プロトタイプは次のとおりです．

```
size_t reg_pos( size_t idx ) const;
```

引数 idx には，0 から始まる要素番号を指定します．要素番号 0 の要素には，マッチした文字列全体の情報が格納され，要素番号 1 からは，正規表現「(...)」それぞれにマッチした部分文字列の情報が格納されます (つまり，後方参照のための情報です).

メンバ関数 1〜12 の場合，正規表現 pat をコンパイルし，その結果を自身が持つ内部バッファに保存し，マッチを行ないます (pat が前回と同じ場合は，再コンパイルしません).

メンバ関数 13〜18 の場合，正規表現のコンパイル結果を保持している tregex クラスのオブジェクトを指定します．したがって，regassign() メンバ関数を使う前に，あらかじめ tregex クラスの compile() メンバ関数で正規表現をコンパイルする必要があります (EXAMPLE を参照).

いずれの場合も，正規表現のコンパイルに失敗すると，標準エラー出力にその内容を出力します.

pos が指定されない場合は，文字列 src_str の左端からマッチを試行しますが，指定された場合は，文字列 src_str の位置 pos からマッチを試行します．文字列マッチが試行される対象範囲は，文字列終端の'\0' が現れるまでです (改行文字'\n' が現れても処理は終了しません). なお，文字列の先頭位置は 0 です.

連続して文字や文字列を検索する場合に，nextpos を使って，次の呼び出しで pos に与えるべき値を得る事ができます．nextpos の指す変数には，マッチした場合はその位置からさらにマッチした部分の長さ分 (マッチした部分の長さが 0 の場合は 1 文字分) だけ右方向に移動させた位置が，マッチしなかった場合には文字列 src_str の長さ+1 が返ります．nextpos が不要な場合は，NULL を与える事もできます.

正規表現の詳細は，§9.5.59を参照してください.

PARAMETER

[I]	src_str	マッチング対象の文字列
[I]	pos	文字列マッチの開始位置
[I]	pat	文字パターン (正規表現) または tregex クラスのコンパイル済オブジェクト
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

regex ルーチンがメモリを使い果たしている場合
内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，文字列 "OS = linux" について，キーワードと値とを取り出しています．my_elms.cstr(0) には，my_pat 全体についてマッチした部分の文字列が入り，my_elms.cstr(1) 以降に，後方参照の部分文字列が入ります．正規表現のコンパイルは，compile() メンバ関数を使って my_pat.complie("([^]+)([ ]*=[ ]*)([^]+)") のように行ないます．コンパイル処理のエラーチェックには cregex() を利用します.

```
stdstreamio sio;
tarray_tstring my_elms;
tstring my_str = "OS = linux";
tregex my_pat;

my_pat.compile("([^ ]+)([ ]*=[ ]*)([^ ]+)");
if ( my_pat.cregex() == NULL ) {
    /* エラー処理: 正規表現のコンパイルに失敗 */
}
my_elms.regassign(my_str, my_pat);
if ( my_elms.length() == 4 ) {
    sio.printf("keyword=[%s] value=[%s]\n",
               my_elms.cstr(1), my_elms.cstr(3));
}
```

実行結果

```
keyword=[OS] value=[linux]
```

§3.4.8にメンバ関数 4 を使った例があります.

10.4.14 put(), putf(), vputf()

NAME

put(), putf(), vputf() — 任意の要素位置へ文字列を n 個セット

SYNOPSIS

```
tarray_tstring &put( size_t index, const char *str, size_t n );  ..........  1
tarray_tstring &put( size_t index, const tstring &str, size_t n );  ......  2
tarray_tstring &putf( size_t index, size_t n, const char *format, ... );    3
tarray_tstring &vputf( size_t index, size_t n, const char *format,
                       va_list ap );  ....................................  4
```

DESCRIPTION

自身の配列の要素番号 `index` の位置から，指定された文字列を持つ要素 n 個で上書きします．なお，配列の先頭の要素番号は 0 です．

`index` は任意の値を取る事ができます．引数の指定に対して配列内の要素数が不足している場合は，自動的にサイズを拡張します．要素が空の状態において，例えば，`my_arr.put(0,"",6)` と `my_arr.put(2,"",4)` の結果は同じです．要素数 4 個の配列に対して `my_arr.put(2,"",4)` とすると，要素数は 6 個となり，要素番号 2 以降の要素は`""`となります．

メンバ関数 1 およびメンバ関数 2 は，文字列 `str` を配列の要素番号 `index` から n 個の要素へ書き込みます．書き込み前の配列の長さが `index + n` に満たない場合，書き込み後の配列の長さは `index + n` に拡張されます．

メンバ関数 3 では，可変長引数の各要素データを，メンバ関数 4 では，可変長引数のリスト `ap` を `format` の変換指定に応じて変換してできた文字列を，要素番号 `index` から n 個の要素へ書き込みます．`format` については §8.1.14の解説を参照してください．

PARAMETER

[I]	`index`	自身の配列の書き込み位置
[I]	`n`	要素の個数
[I]	`str`	源泉となる文字列
[I]	`format`	源泉となる文字列のためのフォーマット指定
[I]	`...`	`format` に対応した可変長引数の各要素データ
[I]	`ap`	`format` に対応した可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

可変長引数の各要素データが，指定された変換フォーマットで変換できない値の場合 (メンバ関数 3，4)

EXAMPLE

次のコードは，文字列配列 `my_arr` の要素番号 1 から 2 個の要素に文字列`"elm"`を書き込み，結果を標準出力します．

```
stdstreamio sio;

tarray_tstring my_arr;

my_arr.put(1, "elm", 2);
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("my_arr[%zu] = %s\n", i, my_arr.cstr(i));
}
```

実行結果

```
my_arr[0] =
my_arr[1] = elm
my_arr[2] = elm
```

10.4.15 put(), vput()

NAME

put(), vput() — 任意の要素位置へ複数の文字列または文字列配列をセット

SYNOPSIS

```
tarray_tstring &put( size_t index, const char *el0,
                     const char *el1, ... );  ........................ 1
tarray_tstring &vput( size_t index, const char *el0, const char *el1,
                      va_list ap );  ................................. 2
tarray_tstring &put( size_t index, const char *const *elements );  ....... 3
```

```
tarray_tstring &put( size_t index, const char *const *elements,
                     size_t n ); ............................................ 4
tarray_tstring &put( size_t index, const tarray_tstring &src,
                     size_t idx2 = 0 ); ..................................... 5
tarray_tstring &put( size_t index, const tarray_tstring &src,
                     size_t idx2, size_t n2 ); .............................. 6
```

DESCRIPTION

el0, el1, ... で指定された複数の文字列，または elements, src で指定された文字列配列を，自身の配列の要素位置 index から書き込みます (上書きします).

index は任意の値を取る事ができます．引数の指定に対して配列内の要素数が不足している場合は，自動的にサイズを拡張します.

メンバ関数 1,2 では，el0, el1 と可変長引数または可変長引数のリスト ap を指定します．可変長引数は NULL で終端していなければなりません.

メンバ関数 3,4 では，引数 elements に文字列のポインタ配列を指定します．メンバ関数 3 の場合，ポインタ配列は NULL で終端していなければなりません．メンバ関数 4 の場合は，n で要素数を指定できます．elements の個数 (NULL に達するまで) を超えた n が指定された場合は，n は無視されます.

メンバ関数 5 およびメンバ関数 6 は，源泉となる文字列配列 src の開始要素位置を idx2 で，要素数を n2 で指定できます．メンバ関数 5 は，idx2 を指定しなくても使用できます．その場合は 0 が指定されたものとして処理を行います．メンバ関数 6 は，源泉となる要素の個数 n2 を指定できます．なお，配列の先頭の要素番号は 0 です.

PARAMETER

[I]	index	自身の配列の書き込み位置
[I]	el0	源泉となる文字列 (0 番目)
[I]	el1	源泉となる文字列 (1 番目)
[I]	...	源泉となる文字列 (2 番目以降; 要 NULL 終端)
[I]	ap	源泉となる文字列の可変長引数のリスト (2 番目以降; 要 NULL 終端)
[I]	elements	源泉となる文字列のポインタ配列 (メンバ関数 3 の場合，NULL で終端)
[I]	n	配列 elements の要素数
[I]	src	源泉となる文字列配列を持つ tarray_tstring クラスのオブジェクト
[I]	idx2	src 中の要素の開始位置 (src の部分配列を代入する場合)
[I]	n2	src 中の要素の個数 (src の部分配列を代入する場合)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，文字列配列 tree_arr の要素番号 1 から 2 個の要素を文字列配列 my_arr の要素番号 2 以降の要素に書き込み，結果を標準出力します.

```
stdstreamio sio;

tarray_tstring my_arr;

const char *mytree[] = {"maple", "larch", "camphor", NULL};
const tarray_tstring tree_arr(mytree);

my_arr.put(2, tree_arr, 1, 2);
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("my_arr[%zu] = %s\n", i, my_arr.cstr(i));
}
```

実行結果

```
my_arr[0] =
my_arr[1] =
my_arr[2] = larch
my_arr[3] = camphor
```

10.4.16 append(), appendf(), vappendf()

NAME

append(), appendf(), vappendf() — 要素の追加 (1 つの文字列を指定)

SYNOPSIS

```
tarray_tstring &append( const char *str, size_t n );  ........................ 1
tarray_tstring &append( const tstring &str, size_t n );  .................... 2
tarray_tstring &appendf( size_t n, const char *format, ... );  ............. 3
tarray_tstring &vappendf( size_t n, const char *format, va_list ap );  .... 4
```

DESCRIPTION

自身の配列の最後に，指定された文字列を持つ要素 n 個を追加します．なお，配列の先頭の要素番号は 0 です．

メンバ関数 1 とメンバ関数 2 は，文字列配列に文字列 str を n 個追加します．

メンバ関数 3 とメンバ関数 4 は，文字列配列に format で指定された書式に従って作成した文字列を n 個追加します．メンバ関数 3 では，可変長引数の各要素データを，メンバ関数 4 では，可変長引数のリスト ap を書式指定に応じて変換します．format については §8.1.14の解説を参照してください．

PARAMETER

[I]	n	追加要素の個数
[I]	str	源泉となる文字列
[I]	format	源泉となる文字列のためのフォーマット指定
[I]	...	format に対応した可変長引数の各要素データ
[I]	ap	format に対応した可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

可変長引数の各要素データが指定された変換フォーマットで変換できない値の場合 (メンバ関数 3，4)

EXAMPLE

次のコードは，文字列配列 `my_arr` に文字列を追加し，その結果を標準出力します.

```
stdstreamio sio;

tarray_tstring my_arr("maple", "larch", NULL);

const tstring mytrr = "gardenia";
my_arr.append(mytrr,2);
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("my_arr[%zu] = %s\n", i, my_arr.cstr(i));
}
```

実行結果

```
my_arr[0] = maple
my_arr[1] = larch
my_arr[2] = gardenia
my_arr[3] = gardenia
```

10.4.17 append(), vappend()

NAME

append(), vappend() — 要素の追加 (複数の文字列または文字列配列を指定)

SYNOPSIS

```
tarray_tstring &append( const char *el0, const char *el1, ... );  ......... 1
tarray_tstring &vappend( const char *el0, const char *el1,
                         va_list ap );  ..................................... 2
tarray_tstring &append( const char *const *elements );  .................... 3
tarray_tstring &append( const char *const *elements, size_t n );  ......... 4
tarray_tstring &append( const tarray_tstring &src, size_t idx2 = 0 );  .... 5
tarray_tstring &append( const tarray_tstring &src, size_t idx2,
                        size_t n2 );  ....................................... 6
```

DESCRIPTION

`el0`，`el1`，`...` で指定された複数の文字列，または `elements`，`src` で指定された文字列配列を，自身の配列の最後尾以降に追加します.

メンバ関数 1,2 では，el0，el1 と可変長引数または可変長引数のリスト ap を指定します．可変長引数は NULL で終端していなければなりません．

メンバ関数 3,4 では，引数 elements に文字列のポインタ配列を指定します．メンバ関数 3 の場合，ポインタ配列は NULL で終端していなければなりません．メンバ関数 4 の場合は，n で要素数を指定できます．elements の個数 (NULL に達するまで) を超えた n が指定された場合は，n は無視されます．

メンバ関数 5 およびメンバ関数 6 は，源泉となる文字列配列 src の開始要素位置を idx2 で，要素数を n2 で指定できます．メンバ関数 5 は，idx2 を指定しなくても使用できます．その場合は 0 が指定されたものとして処理を行います．メンバ関数 6 は，源泉となる要素の個数 n2 を指定できます．なお，配列の先頭の要素番号は 0 です．

PARAMETER

[I]	el0	源泉となる文字列 (0 番目)
[I]	el1	源泉となる文字列 (1 番目)
[I]	...	源泉となる文字列 (2 番目以降; 要 NULL 終端)
[I]	ap	源泉となる文字列の可変長引数のリスト (2 番目以降; 要 NULL 終端)
[I]	elements	源泉となる文字列のポインタ配列 (メンバ関数 3 の場合，NULL で終端)
[I]	n	配列 elements の要素数
[I]	src	源泉となる文字列配列を持つ tarray_tstring クラスのオブジェクト
[I]	idx2	src 中の要素の開始位置 (src の部分配列を追加する場合)
[I]	n2	src 中の要素の個数 (src の部分配列を追加する場合)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，文字列配列 my_arr に文字列配列 tree_arr の要素番号 2 から 2 個の要素を文字列を追加し，その結果を標準出力します．

```
stdstreamio sio;

tarray_tstring tree_arr("chestnut", "zelkova", "crape myrtle", "daphne", NULL);
tarray_tstring my_arr;
my_arr.at(0) = "chestnut";

my_arr.append(tree_arr,2,2);
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("my_arr[%zu] = %s\n", i, my_arr.cstr(i));
}
```

実行結果

```
my_arr[0] = chestnut
```

```
my_arr[1] = crape myrtle
my_arr[2] = daphne
```

10.4.18 insert(), insertf(), vinsertf()

NAME

insert(), insertf(), vinsertf() — 要素の挿入 (1 つの文字列を指定)

SYNOPSIS

```
tarray_tstring &insert( size_t index, const char *str, size_t n );  ....... 1
tarray_tstring &insert( size_t index, const tstring &str, size_t n );  .... 2
tarray_tstring &insertf( size_t index, size_t n, const char *format, ... );  3
tarray_tstring &vinsertf( size_t index, size_t n,
                          const char *format, va_list ap );  ............... 4
```

DESCRIPTION

自身の文字列配列の要素番号 index の位置に，指定された文字列を持つ要素 n 個を挿入します．なお，配列の先頭の要素番号は 0 です．

index に配列の長さ以上の値を指定する場合，index に自身の配列の長さを与えたものとみなします．

メンバ関数 1 とメンバ関数 2 は，文字列配列に文字列 str を n 個挿入します．

メンバ関数 3 とメンバ関数 4 は，文字列配列に format で指定された書式に従って作成した文字列を n 個挿入します．メンバ関数 3 では，可変長引数の各要素データを，メンバ関数 4 では，可変長引数のリスト ap を書式指定に応じて変換します．format については §8.1.14の解説を参照してください．

PARAMETER

[I]	index	自身の配列の挿入位置
[I]	n	追加要素の個数
[I]	str	源泉となる文字列
[I]	format	源泉となる文字列のためのフォーマット指定
[I]	...	format に対応した可変長引数の各要素データ
[I]	ap	format に対応した可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，文字列配列 my_arr の要素番号 1 の要素位置に，文字列"palm"，"fir"を挿入し，その結果を標準出力します．

```
stdstreamio sio;

tarray_tstring my_arr("cycad", "dogwood", NULL);
```

```
    my_arr.insert(1, "palm", "fir", NULL);
    for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
        sio.printf("my_arr[%zu] = %s\n", i, my_arr.cstr(i));
    }
```

実行結果

```
my_arr[0] = cycad
my_arr[1] = palm
my_arr[2] = fir
my_arr[3] = dogwood
```

10.4.19 insert(), vinsert()

NAME

insert(), vinsert() — 要素の挿入 (複数の文字列または文字列配列を指定)

SYNOPSIS

```
tarray_tstring &insert( size_t index, const char *el0,
                        const char *el1, ... );  ............................ 1
tarray_tstring &vinsert( size_t index, const char *el0,
                         const char *el1, va_list ap );  ................... 2
tarray_tstring &insert( size_t index, const char *const *elements );  ..... 3
tarray_tstring &insert( size_t index, const char *const *elements,
                        size_t n );  ...................................... 4
tarray_tstring &insert( size_t index, const tarray_tstring &src,
                        size_t idx2 = 0 );  ................................ 5
tarray_tstring &insert( size_t index, const tarray_tstring &src,
                        size_t idx2, size_t n2 );  ......................... 6
```

DESCRIPTION

el0，el1，... で指定された複数の文字列，または elements，src で指定された文字列配列を，自身の文字列配列の指定位置 index に挿入します．

index に自身の配列の長さ以上の値を指定する場合，index に自身の配列の長さを与えたものとみなします．

メンバ関数 1,2 では，el0，el1 と可変長引数または可変長引数のリスト ap を指定します．可変長引数は NULL で終端していなければなりません．

メンバ関数 3,4 では，引数 elements に文字列のポインタ配列を指定します．メンバ関数 3 の場合，ポインタ配列は NULL で終端していなければなりません．メンバ関数 4 の場合は，n で要素数を指定できます．elements の個数 (NULL に達するまで) を超えた n が指定された場合は，n は無視されます．

メンバ関数 5 およびメンバ関数 6 は，源泉となる文字列配列 src の開始要素位置を idx2 で，要素数を n2 で指定できます．メンバ関数 5 は，idx2 を指定しなくても使用できます．その場合は 0 が指定されたものとして処理を行います．メンバ関数 6 は，源泉となる要素の個数 n2 を指定できます．なお，配列の先頭の要素番号は 0 です．

PARAMETER

[I]	`index`	自身の配列の挿入位置
[I]	`el0`	源泉となる文字列 (0 番目)
[I]	`el1`	源泉となる文字列 (1 番目)
[I]	`...`	源泉となる文字列 (2 番目以降; 要 `NULL` 終端)
[I]	`ap`	源泉となる文字列の可変長引数のリスト (2 番目以降; 要 `NULL` 終端)
[I]	`elements`	源泉となる文字列のポインタ配列 (メンバ関数 3 の場合，`NULL` で終端)
[I]	`n`	配列 `elements` の要素数
[I]	`src`	源泉となる文字列配列を持つ tarray_tstring クラスのオブジェクト
[I]	`idx2`	`src` 中の要素の開始位置 (`src` の部分配列を挿入する場合)
[I]	`n2`	`src` 中の要素の個数 (`src` の部分配列を挿入する場合)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，文字列配列 `my_arr` の要素番号 1 の要素位置に，`addTree` の要素番号 0 から 2 個の要素を挿入し，その結果を標準出力します.

```
stdstreamio sio;

const char *addTree[] = {"cycad", "dogwood", NULL};
tarray_tstring my_arr("hawthorn", "oak", NULL);

my_arr.insert(1, addTree, 2);
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("my_arr[%zu] = %s\n", i, my_arr.cstr(i));
}
```

実行結果

```
my_arr[0] = hawthorn
my_arr[1] = cycad
my_arr[2] = dogwood
my_arr[3] = oak
```

10.4.20 replace(), replacef(), vreplacef()

NAME

replace(), replacef(), vreplacef() — 要素の置換 (1 つの文字列を指定)

SYNOPSIS

```
tarray_tstring &replace( size_t idx1, size_t n1,
```

```
                              const char *str, size_t n2 );  ..................... 1
tarray_tstring &replace( size_t idx1, size_t n1,
                              const tstring &str, size_t n2 );  .................. 2
tarray_tstring &replacef( size_t idx1, size_t n1,
                               size_t n2, const char *format, ... );  ........... 3
tarray_tstring &vreplacef( size_t idx1, size_t n1,
                                size_t n2, const char *format, va_list ap );  . 4
```

DESCRIPTION

自身の配列の要素の位置 idx1 から n1 個の要素を，指定された文字列を持つ要素 n2 個で置換します．なお，配列の先頭の要素番号は 0 です．

idx1 に配列要素数以上の値が指定された場合，append() メンバ関数 (§10.4.16) と同様の処理を行います．idx1 と n1 の和が配列の要素数よりも大きい場合や，また n1，n2 の大小関係により配列の拡張，収縮が必要な場合は要素数を自動的に調整します．

メンバ関数 1 およびメンバ関数 2 は，文字列配列内の要素番号 idx1 から n1 個の要素を，n2 個の文字列 str で置換します．

メンバ関数 3 とメンバ関数 4 は，文字列配列の要素番号 idx1 から n1 個の要素を，format で指定された書式に従って作成した文字列 n2 個で置換します．メンバ関数 3 では，可変長引数の各要素データを，メンバ関数 4 では，可変長引数のリスト ap を書式指定に応じて変換します．format については §8.1.14の解説を参照してください．

PARAMETER

[I]	idx1	自身の配列の開始位置
[I]	n1	置換される要素数
[I]	n2	指定された文字列が代入される要素の個数
[I]	format	源泉となる文字列のためのフォーマット指定
[I]	...	format に対応した可変長引数の各要素データ
[I]	ap	format に対応した可変長引数のリスト
[I]	str	源泉となる文字列

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
可変長引数の各要素データが，指定された変換フォーマットで変換できない値の場合 (メンバ関数 3，4)

EXAMPLE

次のコードは，文字列配列 my_arr の要素番号 1 から 1 個の要素を，文字列"linden"に置換し，その結果を標準出力します．

```
stdstreamio sio;

const char *tree[] = {"willow", "pine", "fir", NULL};
```

```
    tarray_tstring my_arr = tree;

    my_arr.replace(1, 1, "linden", 1);
    for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
        sio.printf("my_arr[%zu] = %s\n", i, my_arr.cstr(i));
    }
```

実行結果

```
my_arr[0] = willow
my_arr[1] = linden
my_arr[2] = fir
```

10.4.21 replace(), vreplace()

NAME

replace(), vreplace() — 要素の置換 (複数の文字列または文字列配列を指定)

SYNOPSIS

```
tarray_tstring &replace( size_t idx1, size_t n1,
                         const char *el0, const char *el1, ... );  ....... 1
tarray_tstring &vreplace( size_t idx1, size_t n1,
                          const char *el0, const char *el1, va_list ap );  2
tarray_tstring &replace( size_t idx1, size_t n1,
                         const char *const *elements );  .................. 3
tarray_tstring &replace( size_t idx1, size_t n1,
                         const char *const *elements, size_t n2 );  ...... 4
tarray_tstring &replace( size_t idx1, size_t n1,
                         const tarray_tstring &src, size_t idx2 = 0 );  .. 5
tarray_tstring &replace( size_t idx1, size_t n1, const tarray_tstring &src,
                         size_t idx2, size_t n2 );  ....................... 6
```

DESCRIPTION

自身の文字列配列の要素番号 idx1 から n1 個の要素を，el0，el1，... で指定された複数の文字列，または elements，src で指定された文字列配列で置換します．

idx1 に配列要素数以上の値が指定された場合，append() メンバ関数 (§10.4.16) と同様の処理を行います．idx1 と n1 の和が配列の要素数よりも大きい場合，また n1，n2 の大小関係により配列の拡張，収縮が必要な場合は要素数を自動的に調整します．

メンバ関数 1,2 では，el0，el1 と可変長引数または可変長引数のリスト ap を指定します．可変長引数は NULL で終端していなければなりません．

メンバ関数 3,4 では，引数 elements に文字列のポインタ配列を指定します．メンバ関数 3 の場合，ポインタ配列は NULL で終端していなければなりません．メンバ関数 4 の場合は，n2 で要素数を指定できます．elements の個数 (NULL に達するまで) を超えた n2 が指定された場合は，n2 は無視されます．

メンバ関数 5 およびメンバ関数 6 は，源泉となる文字列配列 `src` の開始要素位置を `idx2` で，要素数を `n2` で指定できます．メンバ関数 5 は，`idx2` を指定しなくても使用できます．その場合は `0` が指定されたものとして処理を行います．メンバ関数 6 は，源泉となる要素の個数 `n2` を指定できます．なお，配列の先頭の要素番号は `0` です．

PARAMETER

[I]	`idx1`	自身の配列の開始位置
[I]	`n1`	置換される要素数
[I]	`el0`	源泉となる文字列 (0 番目)
[I]	`el1`	源泉となる文字列 (1 番目)
[I]	`...`	源泉となる文字列 (2 番目以降; 要 `NULL` 終端)
[I]	`ap`	源泉となる文字列の可変長引数のリスト (2 番目以降; 要 `NULL` 終端)
[I]	`elements`	源泉となる文字列のポインタ配列 (メンバ関数 3 の場合，`NULL` で終端)
[I]	`n2`	配列 `elements` の要素数，または `src` 中の要素の個数 (`src` の部分配列を代入する場合)
[I]	`src`	源泉となる文字列配列を持つ tarray_tstring クラスのオブジェクト
[I]	`idx2`	`src` 中の要素の開始位置 (`src` の部分配列を代入する場合)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，文字列配列 `my_tree` の要素番号 1 から 1 個の要素を，文字列配列 `my_addTree` の要素番号 1 から 2 個の要素で置換し，その結果を標準出力します．

```
stdstreamio sio;

tarray_tstring my_tree("willow", "pine", "fir", NULL);
tarray_tstring my_addTree("linden", "beech", "holly", NULL);

my_tree.replace(1, 1, my_addTree, 1, 2);
for ( size_t i = 0 ; i < my_tree.length() ; i++ ) {
    sio.printf("my_tree[%zu] = %s\n", i, my_tree.cstr(i));
}
```

実行結果

```
my_tree[0] = willow
my_tree[1] = beech
my_tree[2] = holly
my_tree[3] = fir
```

10.4.22 erase()

NAME

erase() — 要素の削除

SYNOPSIS

```
tarray_tstring &erase(); ...................................................... 1
tarray_tstring &erase( size_t index, size_t num_elements = 1 ); .......... 2
```

DESCRIPTION

自身が持つ文字列配列の要素を削除します.

メンバ関数 1 は，すべての配列要素を削除します (配列長はゼロになります).

メンバ関数 2 は，要素番号 `index` の要素から `num_elements` 個の要素を削除します．なお，配列の先頭の要素番号は 0 です．`num_elements` が指定されない場合は，1 つの要素を削除します.

削除した分だけ，配列長は短くなります.

`index` に配列の長さ以上の値が指定された場合，単に無視されます.

PARAMETER

[I]	`index`	要素番号
[I]	`num_elements`	要素の個数

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，文字列配列 `my_menu` の要素番号 1 から 2 個の要素を削除し，その結果を標準出力します.

```
stdstreamio sio;

tarray_tstring my_menu("rice ball", "sushi", "tofu", NULL);
my_menu.erase(1,2);
for ( size_t i = 0 ; i < my_menu.length() ; i++ ) {
    sio.printf("my_menu[%zu] = %s\n", i, my_menu.cstr(i));
}
```

実行結果

```
my_menu[0] = rice ball
```

10.4.23 clean()

NAME

clean() — 既存の配列の要素値すべてを任意の文字列でパディング

SYNOPSIS

`tarray_tstring &clean(const char *str = "");` 1
`tarray_tstring &clean(const tstring &str);` 2

DESCRIPTION

自身が持つ配列の全要素を文字列 `str` でパディングします．引数 `str` は指定しなくても使用できます．その場合は，文字列`""`が指定されたものとして処理を行います．`clean()` を実行しても文字列配列長は変化しません．

PARAMETER

[I] `str` 文字列配列をパディングするための文字列
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト `my_arr` に `tree` の値を文字列配列として設定後，全要素に `paulownia` を設定します．確認の為，要素値を標準出力します．

```
stdstreamio sio;

const char *tree[] = {"katsura", "torreya", NULL};
tarray_tstring my_arr = tree;

my_arr.clean("paulownia");
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("my_arr[%zu] = %s\n",i, my_arr.cstr(i));
}
```

実行結果

```
my_arr[0] = paulownia
my_arr[1] = paulownia
```

10.4.24 resize()

NAME

resize() — 文字列配列の長さを変更

SYNOPSIS

```
tarray_tstring &resize( size_t new_num_elements );
```

DESCRIPTION

自身の文字列配列の長さを new_num_elements に変更します．

文字列配列長を拡張する場合，空白文字""からなる要素が追加されます．

文字列配列長を収縮する場合，new_num_elements 以降の文字列要素は削除されます．

PARAMETER

[I] new_num_elements 変更後の文字列配列長

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_arr に tree の値を文字列配列として設定後，文字列配列長を 2 に変更します．確認の為，要素値の内容を標準出力します．

```
stdstreamio sio;

const char *tree[] = {"andromeda", "yew", "Japanese pagoda tree", NULL};
tarray_tstring my_arr = tree;
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("my_arr[%zu] = %s\n",i, my_arr.cstr(i));
}

my_arr.resize(2);
for ( size_t i = 0 ; i < my_arr.length() ; i++ ) {
    sio.printf("my_arr[%zu] = %s\n",i, my_arr.cstr(i));
}
```

実行結果

```
my_arr[0] = andromeda
my_arr[1] = yew
my_arr[2] = Japanese pagoda tree

my_arr[0] = andromeda
my_arr[1] = yew
```

10.4.25 resizeby()

NAME

resizeby() — 文字列配列の長さを相対的に変更

SYNOPSIS

```
tarray_tstring &resizeby( ssize_t len );
```

DESCRIPTION

自身の文字列配列の長さを len の長さ分だけ変更します.

文字列配列長を拡張する場合，空白文字""からなる要素が追加されます.

文字列配列長を収縮する場合，最後の abs(len) 個の文字列要素は削除されます.

PARAMETER

[I] len 配列長の増分・減分
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

10.4.26 crop()

NAME

crop() — 文字列配列の切り抜き

SYNOPSIS

```
tarray_tstring &crop( size_t idx, size_t len );
tarray_tstring &crop( size_t idx );
```

DESCRIPTION

自身の配列を，要素番号 idx から len 個の要素だけにします．len を省略した時は，idx 以降の配列だけにします.

PARAMETER

[I] idx 切り出し要素の開始位置
[I] len 要素の個数
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

10.4.27 chomp()

NAME

chomp() — 全要素の改行文字の除去

SYNOPSIS

```
tarray_tstring &chomp( const char *rs = "\n" );
tarray_tstring &chomp( const tstring &rs );
```

DESCRIPTION

自身の文字列配列の全要素の右端の改行文字を除去します.

このメンバ関数は，配列の全要素で tstring クラスの chomp() メンバ関数 (§9.5.25) を実行します．詳細は §9.5.25をご覧ください.

PARAMETER

[I] `rs` 改行文字列

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

10.4.28 trim()

NAME

trim() — 全要素の両端スペースの除去

SYNOPSIS

```
tarray_tstring &trim( const char *side_spaces = " \t\n\r\f\v" );
tarray_tstring &trim( const tstring &side_spaces );
tarray_tstring &trim( int side_space );
```

DESCRIPTION

自身の文字列配列の全要素について，文字列両端にある任意文字を除去します.

`side_spaces` には，`" \t"`のような単純な文字リストに加え，正規表現で用いられる `"[A-Z]"` あるいは`"[^A-Z]"`のような指定が可能です．さらに，`"[…]"`の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説と表 19 を参照してください.

このメンバ関数は，配列の全要素で tstring クラスの trim() メンバ関数を実行します．詳細は §9.5.26をご覧ください.

PARAMETER

[I] `side_space` 任意文字

[I] `side_spaces` 任意文字列

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，CSV 形式の文字列を split() メンバ関数 (§10.4.12) で各要素に分割して配列として自身に代入し，trim() を使って各要素の左右の余分な空白文字を除去しています．

```
tarray_tstring my_arr;
my_arr.split(" MZ-2500, PC-8801MR2  ,FV77AV  ", ",", true);
my_arr.dprint();
my_arr.trim();
my_arr.dprint();
```

実行結果

```
sli::tarray_tstring[obj=0x7fbffff470] = {" MZ-2500", " PC-8801MR2  ", "FV77AV  "}
sli::tarray_tstring[obj=0x7fbffff470] = {"MZ-2500", "PC-8801MR2", "FV77AV"}
```

10.4.29 ltrim()

NAME

ltrim() — 全要素の左端スペースの除去

SYNOPSIS

```
tarray_tstring &ltrim( const char *side_spaces = " \t\n\r\f\v" );
tarray_tstring &ltrim( const tstring &side_spaces );
tarray_tstring &ltrim( int side_space );
```

DESCRIPTION

自身の文字列配列の全要素について，文字列左端にある任意文字を除去します．

side_spaces には，" \t"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]" あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説と表 19 を参照してください．

このメンバ関数は，配列の全要素で tstring クラスの ltrim() メンバ関数を実行します．詳細は §9.5.27をご覧ください．

PARAMETER

[I] side_space 任意文字
[I] side_spaces 任意文字列
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

10.4.30 rtrim()

NAME

rtrim() — 全要素の右端スペースの除去

SYNOPSIS

```
tarray_tstring &rtrim( const char *side_spaces = " \t\n\r\f\v" );
tarray_tstring &rtrim( const tstring &side_spaces );
tarray_tstring &rtrim( int side_space );
```

DESCRIPTION

自身の文字列配列の全要素について，文字列右端にある任意文字を除去します．

side_spaces には，" \t"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]" あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説と表 19 を参照してください．

このメンバ関数は，配列の全要素で tstring クラスの rtrim() メンバ関数を実行します．詳細は §9.5.28をご覧ください．

PARAMETER

[I]	side_space	任意文字
[I]	side_spaces	任意文字列

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

10.4.31 strreplace()

NAME

strreplace() — 全要素の文字列検索と置換

SYNOPSIS

```
tarray_tstring &strreplace( const char *org_str, const char *new_str,
                            bool all = false );
tarray_tstring &strreplace( const tstring &org_str, const char *new_str,
                            bool all = false );
tarray_tstring &strreplace( const char *org_str, const tstring &new_str,
                            bool all = false );
tarray_tstring &strreplace( const tstring &org_str, const tstring &new_str,
                            bool all = false );
```

DESCRIPTION

自身の文字列配列の全要素について，文字列の左側から文字列 org_str を検索し，見つかった場合は文字列 new_str で置き換えます．

このメンバ関数は，配列の全要素で tstring クラスの strreplace() メンバ関数を実行します (pos は 0 が設定されます)．詳細は §9.5.29をご覧ください．

PARAMETER

[I]	org_str	検出する文字列
[I]	new_str	置換の源泉となる文字列
[I]	all	全置換のフラグ

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，全要素について大文字の N を小文字に置き換えます．

```
tarray_tstring my_arr("NEC", "NVIDIA", "HYNIX", NULL);
my_arr.strreplace("N", "n", true);
my_arr.dprint();
```

実行結果

```
sli::tarray_tstring[obj=0x7fbffff470] = {"nEC", "nVIDIA", "HYnIX"}
```

10.4.32 regreplace()

NAME

regreplace() — 全要素の正規表現による文字列検索と置換

SYNOPSIS

```
tarray_tstring &regreplace( const char *pat,
                            const char *new_str, bool all = false );
tarray_tstring &regreplace( const tstring &pat,
                            const char *new_str, bool all = false );
tarray_tstring &regreplace( const tregex &pat,
                            const char *new_str, bool all = false );
tarray_tstring &regreplace( const char *pat,
                            const tstring &new_str, bool all = false );
tarray_tstring &regreplace( const tstring &pat,
                            const tstring &new_str, bool all = false );
tarray_tstring &regreplace( const tregex &pat,
                            const tstring &new_str, bool all = false );
```

DESCRIPTION

自身の文字列配列の全要素について，pat で指定された POSIX 拡張正規表現 (以下，正規表現) でマッチした部分を文字列 new_str で置き換えます．new_str では，後方参照"\\0"～"\\9"

が利用できます ("\\0"はマッチした部分全体を示します)．バックスラッシュ自身を与えたい場合は，"\\\\"を指定します.

このメンバ関数は，配列の全要素で tstring クラスの regreplace() メンバ関数を実行します (pos は 0 が設定されます)．詳細は §9.5.30をご覧ください.

正規表現を必要としない場合は，strreplace() メンバ関数 (§10.4.31) の方が動作が高速です.

PARAMETER

[I]	pat	文字パターン (正規表現) または tregex クラスのコンパイル済オブジェクト
[I]	new_str	置換後の文字列
[I]	all	全置換のフラグ

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，全要素について "[-]" にマッチする部分をアンダーバーで置き換えます.

```
tarray_tstring my_arr("MZ-2000", "PC-88VA", "X1 turboZ III", NULL);
my_arr.regreplace("[- ]", "_", true);
my_arr.dprint();
```

実行結果

```
sli::tarray_tstring[obj=0x7fbffff470] = {"MZ_2000", "PC_88VA", "X1_turboZ_III"}
```

10.4.33 tolower()

NAME

tolower() — 全要素の大文字を小文字に置換

SYNOPSIS

```
tarray_tstring &tolower();
```

DESCRIPTION

自身の文字列配列の全要素のアルファベットの大文字を小文字に変換します.

このメンバ関数は，配列の全要素で tstring クラスの tolower() メンバ関数を実行します．詳細は §9.5.31をご覧ください.

RETURN VALUE

自身の参照

10.4.34 toupper()

NAME

toupper() — 全要素の小文字を大文字に置換

SYNOPSIS

```
tarray_tstring &toupper();
```

DESCRIPTION

自身の文字列配列の全要素のアルファベットの小文字を大文字に変換します.

このメンバ関数は，配列の全要素で tstring クラスの toupper() メンバ関数を実行します．詳細は §9.5.32をご覧ください.

RETURN VALUE

自身の参照

10.4.35 expand_tabs()

NAME

expand_tabs() — 全要素の TAB 文字を空白文字に置換

SYNOPSIS

```
tarray_tstring &expand_tabs( size_t tab_width = 8 );
```

DESCRIPTION

自身の文字列配列の全要素について，水平タブ文字'\t'を，tab_width の値に桁揃えをして空白文字に置換します.

このメンバ関数は，配列の全要素で tstring クラスの expand_tabs() メンバ関数を実行します. 詳細は §9.5.33をご覧ください.

PARAMETER

[I] tab_width TAB 幅
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

10.4.36 contract_spaces()

NAME

contract_spaces() — 全要素の空白文字を TAB 文字に置換

SYNOPSIS

```
tarray_tstring &contract_spaces( size_t tab_width = 8 );
```

DESCRIPTION

自身の文字列配列の全要素について，2 文字以上連続した空白文字' 'すべてを対象にし，指定された TAB 幅 `tab_width` で桁揃えして'\t' で置換します．

このメンバ関数は，配列の全要素で tstring クラスの contract_spaces() メンバ関数を実行します．詳細は §9.5.34をご覧ください．

PARAMETER

[I] `tab_width` TAB 幅

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

10.4.37 find_elem()

NAME

find_elem() — 左側 (先頭) から配列要素を検索

SYNOPSIS

```
ssize_t find_elem( const char *str ) const;
ssize_t find_elem( size_t idx, const char *str ) const;
ssize_t find_elem( size_t idx, const char *str, size_t *nextidx ) const;
ssize_t find_elem( const tstring &str ) const;
ssize_t find_elem( size_t idx, const tstring &str ) const;
ssize_t find_elem( size_t idx, const tstring &str, size_t *nextidx ) const;
```

DESCRIPTION

自身の配列要素の左側から文字列 `str` に完全に一致する要素を検索し，見つかった場合はその要素番号を返し，見つからなければ負数を返します．

特定の要素から検索を開始する場合には，引数 `idx` で開始位置を指定してください．なお，配列の先頭の要素番号は `0` です．

連続して要素を検索する場合に，`nextidx` を使って，次の呼び出しで `idx` に与えるべき値を得る事ができます．`nextidx` の指す変数には，要素が見つかった場合は見つかった位置の 1 つ右の位置が，見つからなかった場合には自身が持つ配列の長さが返ります．`nextidx` による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	`idx`	配列要素の検索開始位置
[I]	`str`	検出する要素値に一致する文字列
[O]	`nextidx`	次回の `idx`(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	str に一致する要素が見つかった場合，その要素番号
負の値 (エラー)	:	str に一致する要素が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	idx に配列長以上の値が指定された場合
	:	str が NULL の場合

EXAMPLE

次のコードは，文字列配列 my_arr から "INTEL" に一致する要素を前から順に列挙します．idx が検出を開始する要素番号で，これらのアドレスを find_elem() メンバ関数の最後に与える事で自動的に適切な値が入るようになっています．

```
stdstreamio sio;
tarray_tstring my_arr;
size_t idx = 0;
ssize_t fidx;
my_arr.assign("ZILOG", "INTEL", "INTEL", "MOTOROLA", "MOS", NULL);
while ( 0 <= (fidx=my_arr.find_elem(idx, "INTEL", &idx)) ) {
    sio.printf("in : fidx=%zd nextidx=%zu\n", fidx, idx);
}
sio.printf("out: fidx=%zd nextidx=%zu\n", fidx, idx);
```

実行結果

```
in : fidx=1 nextidx=2
in : fidx=2 nextidx=3
out: fidx=-1 nextidx=5
```

10.4.38 rfind_elem()

NAME

rfind_elem() — 右側 (最後尾) から配列要素を検索

SYNOPSIS

```
ssize_t rfind_elem( const char *str ) const;
ssize_t rfind_elem( size_t idx, const char *str ) const;
ssize_t rfind_elem( size_t idx, const char *str, size_t *nextidx ) const;
ssize_t rfind_elem( const tstring &str ) const;
ssize_t rfind_elem( size_t idx, const tstring &str ) const;
ssize_t rfind_elem( size_t idx, const tstring &str, size_t *nextidx ) const;
```

DESCRIPTION

自身の配列要素の右側から文字列 str に完全に一致する要素を検索し，見つかった場合はその要素番号を返し，見つからなければ負数を返します．

特定の要素から検索を開始する場合には，引数 idx で開始位置を指定してください．なお，配列の先頭の要素番号は 0 です．

連続して要素を検索する場合に，nextidx を使って，次の呼び出しで idx に与えるべき値を得る事ができます．nextidx の指す変数には，idx が 1 以上で要素が見つかった場合は見つかった位置の 1 つ左の位置が，そうでない場合には自身が持つ配列の長さが返ります．nextidx による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	idx	配列要素の検索開始位置
[I]	str	検出する要素値に一致する文字列
[O]	nextidx	次回の idx(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	str に一致する要素が見つかった場合，その要素番号
負の値 (エラー)	:	str に一致する要素が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	idx に配列長以上の値が指定された場合
	:	str が NULL の場合

EXAMPLE

次のコードは，文字列配列 my_arr から "INTEL" に一致する要素を配列の最後尾から順に列挙します．idx が検出を開始する要素番号で，これらのアドレスを rfind_elem() メンバ関数の最後に与える事で自動的に適切な値が入るようになっています．

```
stdstreamio sio;
tarray_tstring my_arr;
size_t idx;
ssize_t fidx;
my_arr.assign("ZILOG", "INTEL", "INTEL", "MOTOROLA", "MOS", NULL);
idx = my_arr.length() - 1;
while ( 0 <= (fidx=my_arr.rfind_elem(idx, "INTEL", &idx)) ) {
    sio.printf("in : fidx=%zd nextidx=%zu\n", fidx, idx);
}
sio.printf("out: fidx=%zd nextidx=%zu\n", fidx, idx);
```

実行結果

```
in : fidx=2 nextidx=1
in : fidx=1 nextidx=0
out: fidx=-1 nextidx=5
```

10.4.39 find()

NAME

find() — 配列の左側 (先頭) から文字列を検索

SYNOPSIS

```
ssize_t find( const char *str, ssize_t *pos_r ) const;
ssize_t find( size_t idx, size_t pos, const char *str,
```

```
                ssize_t *pos_r ) const;
ssize_t find( size_t idx, size_t pos, const char *str,
                ssize_t *pos_r,  size_t *nextidx, size_t *nextpos ) const;
ssize_t find( const tstring &str, ssize_t *pos_r ) const;
ssize_t find( size_t idx, size_t pos, const tstring &str,
                ssize_t *pos_r ) const;
ssize_t find( size_t idx, size_t pos, const tstring &str,
                ssize_t *pos_r, size_t *nextidx, size_t *nextpos ) const;
```

DESCRIPTION

自身の配列要素の左側から文字列 str を含む要素を検索し，メンバ関数の返り値として，見つかった場合はその要素番号を返し，見つからなければ負数を返します．同時に，見つかった場合は pos_r が指す変数に，その要素における文字列の位置も返します．

特定の要素の特定の文字列の位置から検索を開始する場合には，引数 idx，pos で開始位置を指定してください．なお，配列も文字列も先頭の要素番号は 0 です．引数 idx，pos を省略した場合，要素番号 0，文字列の先頭から検索します．

連続して要素を検索する場合に，nextidx，nextpos を使って，次の呼び出しで idx，pos に与えるべき値を得る事ができます．これらの値については，文章よりもコードの例を見ていただく方がわかりやすいと思います．この後の EXAMPLE をご覧ください．

pos_r，nextidx，nextpos による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	idx	配列要素の検索開始位置
[I]	pos	文字列の検索開始位置
[I]	str	検出する文字列
[O]	pos_r	見つかった場合，その要素における文字列の位置
[O]	nextidx	次回の idx(連続検索の時に利用)
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定された文字列が見つかった場合，その要素番号
負の値 (エラー)	:	指定された文字列が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	idx に配列長以上の値が指定された場合
	:	pos に文字列長を越える値が指定された場合
	:	str が NULL の場合

EXAMPLE

次のコードは，文字列配列 my_arr から "80" を含む部分を前から順に列挙します．idx と pos が検出を開始する要素番号と文字列の位置で，これらのアドレスを find() メンバ関数の最後の 2 つに与える事で自動的に適切な値が入るようになっています．

```
stdstreamio sio;
tarray_tstring my_arr;
size_t idx = 0, pos = 0;
```

```
    ssize_t fidx, fpos;
    my_arr.assign("Z80", "8080", "8086", "6800", "6502", NULL);
    while ( 0 <= (fidx=my_arr.find(idx, pos, "80", &fpos, &idx, &pos)) ) {
        sio.printf("in : fidx=%zd fpos=%zd nextidx=%zu nextpos=%zu\n",
                   fidx, fpos, idx, pos);
    }
    sio.printf("out: fidx=%zd fpos=%zd nextidx=%zu nextpos=%zu\n",
               fidx, fpos, idx, pos);
```

実行結果

```
in : fidx=0 fpos=1 nextidx=0 nextpos=3
in : fidx=1 fpos=0 nextidx=1 nextpos=2
in : fidx=1 fpos=2 nextidx=1 nextpos=4
in : fidx=2 fpos=0 nextidx=2 nextpos=2
in : fidx=3 fpos=1 nextidx=3 nextpos=3
out: fidx=-1 fpos=-1 nextidx=5 nextpos=5
```

10.4.40 rfind()

NAME

rfind() — 配列の右側 (最後尾) から文字列を検索

SYNOPSIS

```
ssize_t rfind( const char *str, ssize_t *pos_r ) const;
ssize_t rfind( size_t idx, size_t pos, const char *str,
               ssize_t *pos_r ) const;
ssize_t rfind( size_t idx, size_t pos, const char *str,
               ssize_t *pos_r,  size_t *nextidx, size_t *nextpos ) const;
ssize_t rfind( const tstring &str, ssize_t *pos_r ) const;
ssize_t rfind( size_t idx, size_t pos, const tstring &str,
               ssize_t *pos_r ) const;
ssize_t rfind( size_t idx, size_t pos, const tstring &str,
               ssize_t *pos_r, size_t *nextidx, size_t *nextpos ) const;
```

DESCRIPTION

自身の配列要素の右側から文字列 `str` を含む要素を検索し，メンバ関数の返り値として，見つかった場合はその要素番号を返し，見つからなければ負数を返します．同時に，見つかった場合は `pos_r` が指す変数に，その要素における文字列の位置も返します．

特定の要素の特定の文字列の位置から検索を開始する場合には，引数 `idx`，`pos` で開始位置を指定してください．なお，配列も文字列も先頭の要素番号 (左端) は `0` です．引数 `idx`，`pos` を省略した場合，最後の要素 (要素の個数-1)，文字列の最後尾 (文字列長) から検索します．

連続して要素を検索する場合に，`nextidx`，`nextpos` を使って，次の呼び出しで `idx`，`pos` に与えるべき値を得る事ができます．これらの値については，文章よりもコードの例を見ていただく方がわかりやすいと思います．この後の EXAMPLE をご覧ください．

`pos_r`，`nextidx`，`nextpos` による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	`idx`	配列要素の検索開始位置
[I]	`pos`	文字列の検索開始位置
[I]	`str`	検出する文字列
[O]	`pos_r`	見つかった場合，その要素における文字列の位置
[O]	`nextidx`	次回の `idx`(連続検索の時に利用)
[O]	`nextpos`	次回の `pos`(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定された文字列が見つかった場合，その要素番号
負の値 (エラー)	:	指定された文字列が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	`idx` に配列長以上の値が指定された場合
	:	`pos` に文字列長を越える値が指定された場合
	:	`str` が `NULL` の場合

EXAMPLE

次のコードは，文字列配列 `my_arr` から `"80"` を含む部分を，最後尾から順に列挙します．`idx` と `pos` が検出を開始する要素番号と文字列の位置で，これらのアドレスを rfind() メンバ関数の最後の 2 つに与える事で自動的に適切な値が入るようになっています．

```
stdstreamio sio;
tarray_tstring my_arr;
size_t idx, pos;
ssize_t fidx, fpos;
my_arr.assign("Z80", "8080", "8086", "6800", "6502", NULL);
idx = my_arr.length() - 1;
pos = my_arr.length(idx);
while ( 0 <= (fidx=my_arr.rfind(idx, pos, "80", &fpos, &idx, &pos)) ) {
    sio.printf("in : fidx=%zd fpos=%zd nextidx=%zu nextpos=%zu\n",
               fidx, fpos, idx, pos);
}
sio.printf("out: fidx=%zd fpos=%zd nextidx=%zu nextpos=%zu\n",
           fidx, fpos, idx, pos);
```

実行結果

```
in : fidx=3 fpos=1 nextidx=2 nextpos=4
in : fidx=2 fpos=0 nextidx=1 nextpos=4
in : fidx=1 fpos=2 nextidx=1 nextpos=0
in : fidx=1 fpos=0 nextidx=0 nextpos=3
in : fidx=0 fpos=1 nextidx=5 nextpos=4
out: fidx=-1 fpos=-1 nextidx=5 nextpos=4
```

10.4.41 find_matched_str()

NAME

find_matched_str() — パターンにマッチする要素 (文字列) を検索

SYNOPSIS

```
ssize_t find_matched_str( const char *pat ) const;
ssize_t find_matched_str( size_t idx, const char *pat ) const;
ssize_t find_matched_str( size_t idx, const char *pat, size_t *nextidx ) const;
ssize_t find_matched_str( const tstring &pat ) const;
ssize_t find_matched_str( size_t idx, const tstring &pat ) const;
ssize_t find_matched_str( size_t idx, const tstring &pat, size_t *nextidx ) const;
```

DESCRIPTION

自身の配列要素の左側から順に，シェルのワイルドカードパターンを用いた文字列マッチを試行し，マッチした場合は要素番号を返します.

文字列先頭のピリオド'.'やスラッシュ'/'を特別扱いする必要がある場合は，find_matched_fn() メンバ関数 (§10.4.42) または find_matched_pn() メンバ関数 (§10.4.43) を利用してください.

このメンバ関数は，配列の全要素に対して順に tstring クラスの strmatch() メンバ関数を実行します (pos は 0 が設定されます). 詳細は §9.5.58をご覧ください.

特定の要素から検索を開始する場合には，引数 idx で開始位置を指定してください. なお，配列の先頭の要素番号は 0 です.

連続して要素を検索する場合に，nextidx を使って，次の呼び出しで idx に与えるべき値を得る事ができます. nextidx の指す変数には，マッチした要素が見つかった場合は見つかった位置の 1 つ右の位置が，見つからなかった場合には自身が持つ配列の長さが返ります. nextidx による値の取得が不要な場合は，NULL を与える事もできます.

PARAMETER

[I]	idx	配列要素の検索開始位置
[I]	pat	検出する要素値に一致する文字列
[O]	nextidx	次回の idx(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	マッチする要素が見つかった場合，その要素番号
負の値 (エラー)	:	マッチする要素が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	idx に配列長以上の値が指定された場合
	:	pat が NULL の場合

EXAMPLE

次のコードは，文字列配列 my_arr から "MO*" にマッチする要素を前から順に列挙します. idx が検出を開始する要素番号で，これらのアドレスを find_matched_str() メンバ関数の最後に与える事で自動的に適切な値が入るようになっています.

```
stdstreamio sio;
tarray_tstring my_arr;
```

```
    size_t idx = 0;
    ssize_t fidx;
    my_arr.assign("ZILOG", "INTEL", "MOTOROLA", "MOS", "AMD", NULL);
    while ( 0 <= (fidx=my_arr.find_matched_str(idx, "MO*", &idx)) ) {
        sio.printf("in : fidx=%zd nextidx=%zu\n", fidx, idx);
    }
    sio.printf("out: fidx=%zd nextidx=%zu\n", fidx, idx);
```

実行結果

```
in : fidx=2 nextidx=3
in : fidx=3 nextidx=4
out: fidx=-1 nextidx=5
```

10.4.42 find_matched_fn()

NAME

find_matched_fn() — パターンにマッチする要素 (ファイル名) を検索

SYNOPSIS

```
ssize_t find_matched_fn( const char *pat ) const;
ssize_t find_matched_fn( size_t idx, const char *pat ) const;
ssize_t find_matched_fn( size_t idx, const char *pat, size_t *nextidx ) const;
ssize_t find_matched_fn( const tstring &pat ) const;
ssize_t find_matched_fn( size_t idx, const tstring &pat ) const;
ssize_t find_matched_fn( size_t idx, const tstring &pat, size_t *nextidx ) const;
```

DESCRIPTION

自身の配列要素の左側から順に，シェルのワイルドカードパターンを用いた文字列マッチを試行し，マッチした場合は要素番号を返します．

find_matched_fn() メンバ関数では，ファイル名検索をする事を想定しており，文字列先頭のピリオド'.' を特別に取り扱います．つまり，ワイルドカード'*' や'?' は先頭のピリオドにマッチしないようになっています．

スラッシュ'/' を特別扱いする必要がある場合は，find_matched_pn() メンバ関数 (§10.4.43) を利用してください．

このメンバ関数は，配列の全要素に対して順に tstring クラスの fnmatch() メンバ関数を実行します (pos は 0 が設定されます)．詳細は §9.5.58をご覧ください．

特定の要素から検索を開始する場合には，引数 idx で開始位置を指定してください．なお，配列の先頭の要素番号は 0 です．

連続して要素を検索する場合に，nextidx を使って，次の呼び出しで idx に与えるべき値を得る事ができます．nextidx の指す変数には，マッチした要素が見つかった場合は見つかった位置の 1 つ右の位置が，見つからなかった場合には自身が持つ配列の長さが返ります．nextidx による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	`idx`	配列要素の検索開始位置
[I]	`pat`	検出する要素値に一致する文字列
[O]	`nextidx`	次回の `idx`(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	マッチする要素が見つかった場合，その要素番号
負の値 (エラー)	:	マッチする要素が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	`idx` に配列長以上の値が指定された場合
	:	`pat` が `NULL` の場合

EXAMPLE

§10.4.41の EXAMPLE をご覧ください．

10.4.43 find_matched_pn()

NAME

find_matched_pn() — パターンにマッチする要素 (パス名) を検索

SYNOPSIS

```
ssize_t find_matched_pn( const char *pat ) const;
ssize_t find_matched_pn( size_t idx, const char *pat ) const;
ssize_t find_matched_pn( size_t idx, const char *pat, size_t *nextidx ) const;
ssize_t find_matched_pn( const tstring &pat ) const;
ssize_t find_matched_pn( size_t idx, const tstring &pat ) const;
ssize_t find_matched_pn( size_t idx, const tstring &pat, size_t *nextidx ) const;
```

DESCRIPTION

自身の配列要素の左側から順に，シェルのワイルドカードパターンを用いた文字列マッチを試行し，マッチした場合は要素番号を返します．

find_matched_pn() メンバ関数では，パス名検索をする事を想定しており，文字列先頭のピリオド'.'，スラッシュ'/' およびスラッシュ直後のピリオドを特別に取り扱います．ワイルドカード'*' や'?' はそれらの文字にマッチしないようになっています．

このメンバ関数は，配列の全要素に対して順に tstring クラスの pnmatch() メンバ関数を実行します (`pos` は `0` が設定されます)．詳細は §9.5.58をご覧ください．

特定の要素から検索を開始する場合には，引数 `idx` で開始位置を指定してください．なお，配列の先頭の要素番号は `0` です．

連続して要素を検索する場合に，`nextidx` を使って，次の呼び出しで `idx` に与えるべき値を得る事ができます．`nextidx` の指す変数には，マッチした要素が見つかった場合は見つかった位置の 1 つ右の位置が，見つからなかった場合には自身が持つ配列の長さが返ります．`nextidx` による値の取得が不要な場合は，NULL を与える事もできます．

PARAMETER

[I]	`idx`	配列要素の検索開始位置
[I]	`pat`	検出する要素値に一致する文字列
[O]	`nextidx`	次回の `idx`(連続検索の時に利用)

([I] : 入力, [O] : 出力)

RETURN VALUE

非負の値	:	マッチする要素が見つかった場合，その要素番号
負の値 (エラー)	:	マッチする要素が見つからなかった場合
	:	オブジェクト内に文字列がない場合
	:	`idx` に配列長以上の値が指定された場合
	:	`pat` が `NULL` の場合

EXAMPLE

§10.4.41の EXAMPLE をご覧ください．

10.4.44 regmatch() [Normal 版]

NAME

regmatch() — 拡張正規表現で文字列を検索

SYNOPSIS

```
ssize_t regmatch( const char *pat, ssize_t *pos_r,
                  size_t *span_r ) const;  ...................................  1
ssize_t regmatch( size_t idx, size_t pos, const char *pat,
                  ssize_t *pos_r, size_t *span_r ) const;  .................  2
ssize_t regmatch( size_t idx, size_t pos, const char *pat,
                  ssize_t *pos_r, size_t *span_r,
                  size_t *nextidx, size_t *nextpos ) const;  ..............  3
ssize_t regmatch( const tstring &pat, ssize_t *pos_r,
                  size_t *span_r ) const;  ...................................  4
ssize_t regmatch( size_t idx, size_t pos, const tstring &pat,
                  ssize_t *pos_r, size_t *span_r ) const;  .................  5
ssize_t regmatch( size_t idx, size_t pos, const tstring &pat,
                  ssize_t *pos_r, size_t *span_r,
                  size_t *nextidx, size_t *nextpos ) const;  ..............  6
ssize_t regmatch( const tregex &pat, ssize_t *pos_r,
                  size_t *span_r ) const;  ...................................  7
ssize_t regmatch( size_t idx, size_t pos, const tregex &pat,
                  ssize_t *pos_r, size_t *span_r ) const;  .................  8
ssize_t regmatch( size_t idx, size_t pos, const tregex &pat,
                  ssize_t *pos_r, size_t *span_r,
                  size_t *nextidx, size_t *nextpos ) const;  ..............  9
```

DESCRIPTION

自身の配列要素の左側から，`pat` で指定された POSIX 拡張正規表現 (以下，正規表現) にマッ

チする部分を含む要素を検索し，メンバ関数の返り値として，マッチした場合はその要素番号を返し，マッチしなければ負数を返します．同時に，マッチした場合は pos_r，span_r が指す変数に，その要素におけるマッチした部分の文字位置と長さも返します．

特定の要素の特定の文字列の位置から検索を開始する場合には，引数 idx，pos で開始位置を指定してください．なお，配列も文字列も先頭の要素番号は 0 です．引数 idx，pos を省略した場合，要素番号 0，文字列の先頭から検索します．

メンバ関数 1〜6 の場合，正規表現 pat をコンパイルし，その結果を自身が持つ内部バッファに保存し，マッチを行ないます (pat が前回と同じ場合は，再コンパイルしません)．

メンバ関数 7〜9 の場合，正規表現のコンパイル結果を保持している tregex クラスのオブジェクトを指定します．したがって，regmatch() メンバ関数を使う前に，あらかじめ tregex クラスの compile() メンバ関数で正規表現をコンパイルする必要があります (EXAMPLE を参照)．

いずれの場合も，正規表現のコンパイルに失敗すると，標準エラー出力にその内容を出力します．

連続して要素を検索する場合に，nextidx，nextpos を使って，次の呼び出しで idx，pos に与えるべき値を得る事ができます．これらの値については，文章よりもコードの例を見ていただく方がわかりやすいと思います．この後の EXAMPLE をご覧ください．

pos_r，nextidx，nextpos による値の取得が不要な場合は，NULL を与える事もできます．

正規表現の詳細は，§9.5.59を参照してください．

PARAMETER

[I]	idx	配列要素の検索開始位置
[I]	pos	文字列の検索開始位置
[I]	pat	検索に使う正規表現
[O]	pos_r	マッチした場合，その要素におけるマッチした部分の文字位置
[O]	span_r	マッチした場合，その要素におけるマッチした部分の文字列長
[O]	nextidx	次回の idx(連続検索の時に利用)
[O]	nextpos	次回の pos(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定された正規表現がマッチした場合，その要素番号
負の値 (エラー)	:	指定された正規表現がマッチしなかった場合
	:	オブジェクト内に文字列がない場合
	:	idx に配列長以上の値が指定された場合
	:	pos に文字列長を越える値が指定された場合
	:	pat で指定された正規表現が不正な場合 (詳細は §9.5.59を参照)

EXAMPLE

次のコードは，文字列配列 my_arr において正規表現 "http://[^/]+\\.jp/" を含む部分を前から順に列挙します．idx と pos が検出を開始する要素番号と文字列の位置で，これらのアドレスを regmatch() メンバ関数の最後の 2 つに与える事で自動的に適切な値が入るようになっています．

```
stdstreamio sio;
```

```
tarray_tstring my_arr("http://www.jaxa.jp/", "http://www.noao.edu/", NULL);
size_t fspan, idx = 0, pos = 0;
ssize_t fidx, fpos;
tregex pat;
pat.compile("http://[^/]+\\.jp/");
while ( 0 <= (fidx=my_arr.regmatch(idx,pos,pat, &fpos,&fspan,&idx,&pos)) ) {
    sio.printf("in : fidx=%zd fpos=%zd fspan=%zd nextidx=%zu nextpos=%zu\n",
               fidx, fpos, fspan, idx, pos);
}
sio.printf("out: fidx=%zd fpos=%zd fspan=%zd nextidx=%zu nextpos=%zu\n",
           fidx, fpos, fspan, idx, pos);
pat.init();
```

実行結果

```
in : fidx=0 fpos=0 fspan=19 nextidx=0 nextpos=19
out: fidx=-1 fpos=-1 fspan=0 nextidx=2 nextpos=21
```

10.4.45 regmatch() [Advanced 版]

NAME

regmatch() — 拡張正規表現で文字列を検索

SYNOPSIS

```
ssize_t regmatch( const char *pat, tarray_tstring *result );  ............. 1
ssize_t regmatch( size_t idx, size_t pos, const char *pat,
                  tarray_tstring *result );  ................................. 2
ssize_t regmatch( size_t idx, size_t pos, const char *pat,
                  tarray_tstring *result,
                  size_t *nextidx, size_t *nextpos );  ...................... 3
ssize_t regmatch( const tstring &pat, tarray_tstring *result );  .......... 4
ssize_t regmatch( size_t idx, size_t pos, const tstring &pat,
                  tarray_tstring *result );  ................................. 5
ssize_t regmatch( size_t idx, size_t pos, const tstring &pat,
                  tarray_tstring *result,
                  size_t *nextidx, size_t *nextpos );  ...................... 6
ssize_t regmatch( const tregex &pat, tarray_tstring *result ) const;  ..... 7
ssize_t regmatch( size_t idx, size_t pos, const tregex &pat,
                  tarray_tstring *result ) const;  ......................... 8
ssize_t regmatch( size_t idx, size_t pos, const tregex &pat,
                  tarray_tstring *result,
                  size_t *nextidx, size_t *nextpos ) const;  .............. 9
```

DESCRIPTION

自身の配列要素の左側から，pat で指定された POSIX 拡張正規表現 (以下，正規表現) にマッチする部分を含む要素を検索し，メンバ関数の返り値として，マッチした場合はその要素番号

を返し，マッチしなければ負数を返します．同時に，マッチした場合は文字列配列オブジェクト `result` に，その要素におけるマッチした部分の後方参照を含む情報も返します．

これらのメンバ関数は，引数オブジェクト `result` の regassign() メンバ関数を利用して正規表現マッチを行なっています．したがって，マッチした部分の結果情報を得る方法は，regassign() メンバ関数を利用した場合と同様です．それについての詳細は §10.4.13 をご覧ください．

特定の要素の特定の文字列の位置から検索を開始する場合には，引数 `idx`，`pos` で開始位置を指定してください．なお，配列も文字列も先頭の要素番号は `0` です．引数 `idx`，`pos` を省略した場合，要素番号 0，文字列の先頭から検索します．

メンバ関数 1〜6 の場合，正規表現 `pat` をコンパイルし，その結果を自身が持つ内部バッファに保存し，マッチを行ないます (`pat` が前回と同じ場合は，再コンパイルしません)．

メンバ関数 7〜9 の場合，正規表現のコンパイル結果を保持している tregex クラスのオブジェクトを指定します．したがって，regmatch() メンバ関数を使う前に，あらかじめ tregex クラスの compile() メンバ関数で正規表現をコンパイルする必要があります (EXAMPLE を参照)．

いずれの場合も，正規表現のコンパイルに失敗すると，標準エラー出力にその内容を出力します．

連続して要素を検索する場合に，`nextidx`，`nextpos` を使って，次の呼び出しで `idx`，`pos` に与えるべき値を得る事ができます．これらの値については，文章よりもコードの例を見ていただく方がわかりやすいと思います．§10.4.44の EXAMPLE をご覧ください．

`nextidx`，`nextpos` による値の取得が不要な場合は，NULL を与える事もできます．

正規表現の詳細は，§9.5.59を参照してください．

PARAMETER

[I]	`idx`	配列要素の検索開始位置
[I]	`pos`	文字列の検索開始位置
[I]	`pat`	検索に使う正規表現
[O]	`result`	マッチした場合，その要素におけるマッチした部分の結果情報
[O]	`nextidx`	次回の `idx`(連続検索の時に利用)
[O]	`nextpos`	次回の `pos`(連続検索の時に利用)

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定された正規表現がマッチした場合，その要素番号
負の値 (エラー)	:	指定された正規表現がマッチしなかった場合
	:	オブジェクト内に文字列がない場合
	:	`idx` に配列長以上の値が指定された場合
	:	`pos` に文字列長を越える値が指定された場合
	:	`pat` で指定された正規表現が不正な場合 (詳細は §9.5.59を参照)
	:	`result` が NULL である場合
	:	`result` に自身が指定された場合

EXAMPLE-1

次のコードは，文字列配列`my_arr`において正規表現 `"^([ ]*)([^= ]+)([ ]*=[ ]*)([^= ]*)"` にマッチする要素あった場合に，後方参照要素 2 と 4 をそれぞれ key，value として表示します．

```
stdstreamio sio;
tarray_tstring my_arr("HEADER",
                      " ARCHITECTURE = x86_64 / CPU = AMD", "OS = Linux  ",
                      NULL);
tarray_tstring my_result;
size_t pos = 0, idx = 0;
tregex pat;
pat.compile("^([ ]*)([^= ]+)([ ]*=[ ]*)([^= ]*)");
while ( 0 <= my_arr.regmatch(idx, pos, pat, &my_result, &idx, &pos) ) {
    if ( my_result.length() == 5 ) {
        sio.printf("key=[%s] value=[%s]\n",
                   my_result.cstr(2), my_result.cstr(4));
    }
}
pat.init();
```

実行結果

```
key=[ARCHITECTURE] value=[x86_64]
key=[OS] value=[Linux]
```

EXAMPLE-2

次のコードも EXAMPLE-1 と同じ結果となりますが，こちらの例では正規表現の「^」を使わず，各要素の「最初にマッチした部分」を取り出す方法を用いています．安全性を考えると，EXAMPLE-1 の方が良いコードです．

```
stdstreamio sio;
tarray_tstring my_arr("HEADER",
                      " ARCHITECTURE = x86_64 / CPU = AMD", "OS = Linux  ",
                      NULL);
tarray_tstring my_result;
ssize_t i;
tregex pat;
pat.compile("([^= ]+)([ ]*=[ ]*)([^= ]*)");
for ( i=0 ; 0 <= (i=my_arr.regmatch(i, 0, pat, &my_result)) ; i++ ) {
    if ( my_result.length() == 4 ) {
        sio.printf("key=[%s] value=[%s]\n",
                   my_result.cstr(1), my_result.cstr(3));
    }
}
pat.init();
```

11 ASARRAY_TSTRING クラス

asarray_tstring クラスを使えば，文字列の連想配列を簡単に扱う事ができます．通常の文字列配列にインデックスを付加するという実装となっているので，読み取り時の高速アクセスと通常の配列の持つメリットとをあわせもった連想配列です．

オブジェクト内部では，キーと値の管理に tarray_tstring クラス (§10) を利用しており，tarray_tstring クラスと tstring クラス (§9) の API と融合する事で使いやすい文字列配列 API を実現しています．

次のような特徴を持ちます．

- メモリは自動確保されるため，オブジェクトを作った後にすぐに代入が可能．
- `printf()` の記法が多くのメンバ関数で利用可能．
- `[]`，at() メンバ関数, atf() メンバ関数を通して，tstring クラスの豊富なメンバ関数が利用可能．
- 空白区切，TAB 区切り，CSV 形式の文字列を簡単に分割可能．
- キーは辞書で管理されるため，値の取り出しが高速．
- 全配列要素に対して一気に文字列の編集を行なう事ができるメンバ関数が利用可能 (例: `chomp()`, `trim()` 等)．使い方は，tstring クラス (§9) の場合と同じ．
- keys() メンバ関数 (§11.4.8)，values() メンバ関数 (§11.4.9) を通じて，内部キー配列・値配列に対して，tarray_tstring クラスによる検索処理 API(正規表現など) を利用可能．

asarray_tstring クラスを使う場合は,「`#include <sli/asarray_tstring.h>`」とコードに書いてください．また，namespace 宣言 (§4.1) が必要な場合は,「`using namespace sli;`」もコードに書いてください．

簡単な使用例を次に示します．

```
#include <sli/stdstreamio.h>
#include <sli/asarray_tstring.h>
using namespace sli;

int main()
{
    stdstreamio sio;
    asarray_tstring my_aarr;
    /* キーを "VENDOR" として "RedHat" を代入 */
    my_aarr["VENDOR"] = "RedHat";
    /* キーを "OS" として "Linux" を代入 */
    my_aarr["OS"] = "Linux";
    /* キーを "VERSION_0" "VERSION_1" "VERSION_2" として "2" "4" "30" を代入 */
    size_t i=0;
    my_aarr.atf("VERSION_%d",i++).printf("2");
    my_aarr.atf("VERSION_%d",i++).printf("4");
    my_aarr.atf("VERSION_%d",i++).printf("30");
    /* すべての要素を表示 */
    for ( i=0 ; i < my_aarr.length() ; i++ ) {
        const char *key = my_aarr.key(i);
        sio.printf("%s ... [%s]\n", key, my_aarr.cstr(key));
    }

    return 0;
}
```

実行結果

```
VENDOR ... [RedHat]
OS ... [Linux]
VERSION_0 ... [2]
VERSION_1 ... [4]
VERSION_2 ... [30]
```

11.1 オブジェクトの作り方

3 通りの方法で，オブジェクトに初期値を与える事ができます[13)].

1 つ目は，何も引数を指定しない方法です.

```
asarray_tstring my_arr0;
```

この状態では，文字列用のバッファもポインタ配列用のバッファも確保されていません.

2 つ目は，可変引数で与える方法です.

```
asarray_tstring my_arr0("OS","Solaris", "VENDOR","Sun", NULL);
```

この場合は，与えられたキー文字列と値文字列で連想配列を初期化します．キー文字列 0，値文字列 0，キー文字列 1，値文字列 1，…の順に引数を与えます．引数の最後は必ず NULL を与えてください.

3 つ目は，構造体 asarrdef_tstring 型の配列を与える方法です．例を示します.

```
asarrdef_tstring my_def0[] = { {"OS","Solaris"}, {"VENDOR","Sun"},
                               {NULL,NULL} };
asarray_tstring my_arr0(my_def0);
```

構造体の配列の最後の要素には必ず {NULL,NULL} を与えてください.

11.2 メンバ関数一覧

表 23 はメンバ関数一覧です.

	メンバ関数名	機能
§11.3.1	[]	指定されたキーに該当する要素値オブジェクト (tstring クラス) の参照
§11.3.2	=	オブジェクトのコピー
§11.4.1	length()	連想配列の長さ (個数)，値文字列の長さ
§11.4.2	cstrarray()	値文字列のポインタ配列 (NULL 終端)
§11.4.3	cstr(), cstrf()	指定されたキー，または要素番号に該当する値文字列
§11.4.4	at(), atf()	指定されたキー，または要素番号に該当する要素値 (tstring クラスのオブジェクト) の参照
§11.4.5	at_cs(), atf_cs()	指定されたキー，または要素番号に該当する要素値 (tstring クラスのオブジェクト) の参照 (読み取り専用)
§11.4.6	index()	キー文字列に該当する要素番号を取得
§11.4.7	key()	要素番号に該当するキー文字列を取得

表 23: asarray_tstring クラスで利用可能なメンバ関数一覧 (次ページに続く).

13) tstring クラス (§9) のような動作モードはありません.

	メンバ関数名	機能
§11.4.8	keys()	キー文字列の配列オブジェクトを参照 (読み取りのみ)
§11.4.9	values()	値文字列の配列オブジェクトを参照 (読み取りのみ)
§11.4.10	dprint()	オブジェクト情報を標準エラー出力へ出力 (ユーザプログラムのデバッグ用)
§11.4.11	swap()	オブジェクトの入替え
§11.4.12	init()	オブジェクトの完全初期化
§11.4.13	assign(), assignf()	オブジェクトの初期化と要素の代入 (1 つのキー・値のセットを指定)
§11.4.14	assign(), vassign()	オブジェクトの初期化と要素の代入 (複数のキー・値のセットを指定)
§11.4.15	assign_keys()	複数の文字列または文字列配列をキーに設定
§11.4.16	assign_values()	複数の文字列または文字列配列を値に設定
§11.4.17	split_keys()	文字列を分割し，キーに設定
§11.4.18	split_values()	文字列を分割し，値に設定
§11.4.19	append(), appendf()	要素を追加 (1 つのキー・値のセットを指定)
§11.4.20	append(), vappend()	要素を追加 (複数のキー・値のセットを指定)
§11.4.21	insert(), insertf()	要素を挿入 (1 つのキー・値のセットを指定)
§11.4.22	insert(), vinsert()	要素を挿入 (複数のキー・値のセットを指定)
§11.4.23	erase()	要素を削除
§11.4.24	clean()	既存の連想配列の要素値すべてを任意の文字列でパディング
§11.4.25	rename_a_key()	キー文字列の変更
§11.4.26	chomp()	全要素の改行文字の除去
§11.4.27	trim()	全要素の両端スペースの除去
§11.4.28	ltrim()	全要素の左端スペースの除去
§11.4.29	rtrim()	全要素の右端スペースの除去
§11.4.30	strreplace()	全要素の文字列検索と置換
§11.4.31	regreplace()	全要素の正規表現による文字列検索と置換
§11.4.32	tolower()	全要素の大文字を小文字に置換
§11.4.33	toupper()	全要素の小文字を大文字に置換
§11.4.34	expand_tabs()	全要素の TAB 文字を空白文字に置換
§11.4.35	contract_spaces()	全要素の空白文字を TAB 文字に置換

表 23: asarray_tstring クラスで利用可能なメンバ関数一覧 (続き).

11.3 演算子

11.3.1 []

NAME

[] — 指定されたキーに該当する要素値 (tstring クラスのオブジェクト) の参照

SYNOPSIS

```
tstring &operator[]( const char *key );  ..................................... 1
const tstring &operator[]( const char *key ) const;  ........................ 2
```

DESCRIPTION

キーに対応する連想配列の要素値 (tstring クラス; §9) の参照を返します.「[]」の直後に「.」で接続し，tstring クラス (§9) のメンバ関数を使う事ができます (EXAMPLE では tstring クラスの「=」演算子と assign() メンバ関数を使っています).

メンバ関数 1 は読み書き両用で，at() メンバ関数と同じ動作をします．メンバ関数 2 は読み取り専用で，at_cs() メンバ関数と同じ動作をします.

存在しないキー文字列が指定された場合，メンバ関数 1 では指定されたキー文字列と値""のセットを連想配列に追加しますが，メンバ関数 2 では例外が発生します.

メンバ関数 1・メンバ関数 2 のどちらが使われるかは，オブジェクトの「const」属性の有無で自動的に決まります.「const」属性が無い場合にはメンバ関数 1 が，有る場合にはメンバ関数 2 が自動的に選択されます.

at()，at_cs() については §11.4.4を参照してください.

PARAMETER

[I] key 連想配列のキー文字列

RETURN VALUE

キーに対応する連想配列の要素値 (tstring クラスのオブジェクト) の参照

EXCEPTION

指定されたキー文字列が NULL の場合
内部バッファの確保に失敗した場合 (メンバ関数 1)
存在しないキー文字列が指定された場合 (メンバ関数 2)

EXAMPLE

次のコードは，オブジェクト my_asarr に連想配列のキーと値を設定します．演算子「=」も assign() も動作は同じです.

```
asarray_tstring my_asarr;
my_asarr["google"] = "Larry Page";
my_asarr["YouTube"].assign("Steve Chen");
```

11.3.2 =

NAME

= — asarray_tstring クラスのオブジェクトのコピー

SYNOPSIS

```
asarray_tstring &operator=(const asarray_tstring &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定された `asarray_tstring` クラスのオブジェクトを自身に代入します.

PARAMETER

[I] `obj` asarray_tstring クラスのオブジェクト (コピー元)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，オブジェクト `my_asarrObj` に，オブジェクト `my_asarr` が持つ連想配列を代入し，その結果を標準出力します．`cstr()` に関しては §11.4.3の解説を参照してください.

```
stdstreamio sio;

asarray_tstring my_asarr;
my_asarr["linux"]   = "Linus Torvalds";
my_asarr["windows"] = "Bill Gates";
my_asarr["mac"]     = "Steve Jobs";

asarray_tstring my_asarrObj;
my_asarrObj = my_asarr;
/* すべての要素を表示 */
for ( size_t i=0 ; i < my_asarrObj.length() ; i++ ) {
    const char *key = my_asarrObj.key(i);
    sio.printf("%s ... [%s]\n", key, my_asarrObj.cstr(key));
}
```

実行結果

```
linux ...  [Linus Torvalds]
windows ...  [Bill Gates]
mac ...  [Steve Jobs]
```

11.4 メンバ関数

11.4.1 length()

NAME

length() — 連想配列の長さ (個数)，値文字列の長さ

SYNOPSIS

```
size_t length() const; ........................................................ 1
size_t length( const char *key ) const; ...................................... 2
```

DESCRIPTION

メンバ関数 1 は，自身の連想配列の長さ (個数) を返します．

メンバ関数 2 は，引数で指定されたキー文字列に該当する値の文字列長を返します．

PARAMETER

[I] key 連想配列のキー文字列

([I]：入力，[O]：出力)

RETURN VALUE

連想配列の要素数または，指定されたキーに該当する値の文字列長

EXAMPLE

次のコードは，連想配列 my_asarr の配列の長さと，キー"google"に対応する値"Larry Page"の文字列長を標準出力します．

```
stdstreamio sio;

asarray_tstring my_asarr;
my_asarr["linux"]   = "Linus Torvalds";
my_asarr["google"]  = "Larry Page";
my_asarr["mac"]     = "Steve Jobs";

sio.printf("my_asarr total length   ... [%zu]\n", my_asarr.length());
sio.printf("my_asarr key='google' length ... [%zu]\n",
            my_asarr.length("google"));
```

実行結果

```
my_asarr total length ...  [3]
my_asarr key='google' length ...  [10]
```

11.4.2 cstrarray()

NAME

cstrarray() — 連想配列の値文字列のポインタ配列 (NULL 終端)

SYNOPSIS

```
const char *const *cstrarray() const;
```

DESCRIPTION

自身が持つ連想配列の値文字列のポインタ配列を返します．ポインタ配列は，必ず NULL で終端しています．

RETURN VALUE

連想配列の値文字列へのポインタ配列 (NULL 終端)

EXAMPLE

次のコードは，連想配列 my_asarr の値文字列のポインタ配列を取得し，その値を標準出力します．

```
stdstreamio sio;

asarray_tstring my_asarr;
my_asarr["MIT"] = "cambridge";
my_asarr["Princeton"] = "New Jersey";
my_asarr["Berkeley"] = "California";

const char *const *my_ptr;
my_ptr = my_asarr.cstrarray();
if ( my_ptr != NULL ) {
    for ( int i = 0 ; my_ptr[i] != NULL ; i++ ) {
        sio.printf("%d ... [%s]\n", i, my_ptr[i]);
    }
}
```

実行結果

```
0 ...  [cambridge]
1 ...  [New Jersey]
2 ...  [California]
```

11.4.3 cstr(), c_str(), cstrf(), vcstrf()

NAME

cstr(), c_str(), cstrf(), vcstrf() — 指定されたキー，または要素番号に該当する値文字列

SYNOPSIS

```
const char *cstr( const char *key ) const;  ................ 1
const char *c_str( const char *key ) const;  ............... 2
const char *cstrf( const char *fmt, ... ) const;  .......... 3
const char *vcstrf( const char *fmt, va_list ap ) const;  .. 4
const char *cstr( size_t index ) const;  ................... 5
```

DESCRIPTION

メンバ関数 1 と 2 は，指定されたキーに該当する連想配列の値文字列を返します．

メンバ関数 3 と 4 は，指定したいキー文字列を printf() 関数と同様のフォーマットと可変引数でセットできます．メンバ関数 3 では，可変長引数の各要素データを fmt の変換指定に応じて変換します．メンバ関数 4 では，可変長引数のリスト ap を fmt の変換指定に応じて変換します．fmt については §8.1.14の解説を参照してください．

メンバ関数 5 は，指定された要素番号に該当する連想配列の値文字列を返します．なお，配列の先頭の要素番号は 0 です．

キーが不正な場合や，要素番号に配列の長さ以上の値が指定された場合，エラーとして NULL が返ります.

PARAMETER

[I]	key	連想配列のキー文字列
[I]	fmt	キー文字列のためのフォーマット指定
[I]	...	fmt に対応した可変長引数の各要素データ
[I]	ap	fmt に対応した可変長引数のリスト
[I]	index	要素番号

([I]：入力，[O]：出力)

RETURN VALUE

指定されたキー，または要素番号に該当する文字列のアドレス (正常終了)
NULL(エラー) ： キー，または要素番号が不正な場合

EXAMPLE

次のコードは，連想配列 my_asarr のキー"Riyuu"に対応する文字列を標準出力します.

```
stdstreamio sio;

asarray_tstring my_asarr;
my_asarr["Itamu Hito"]    = "Arata Tendou";
my_asarr["Kucyu Buranko"] = "Hideo Okuda";
my_asarr["Riyuu"]         = "Miyuki Miyabe";

sio.printf("my_asarr c_str ... [%s]\n", my_asarr.c_str("Riyuu"));
```

実行結果

```
my_asarr c_str ...  [Miyuki Miyabe]
```

11.4.4 at(), atf()

NAME

at(), atf() — 指定されたキー，または要素番号に該当する要素値 (tstring クラスのオブジェクト) の参照

SYNOPSIS

```
tstring &at( const char *key ); ........................................ 1
tstring &atf( const char *fmt, ... ); .................................. 2
tstring &vatf( const char *fmt, va_list ap ); .......................... 3
tstring &at( size_t index ); ........................................... 4
const tstring &at( const char *key ) const; ............................ 5
const tstring &atf( const char *fmt, ... ) const; ...................... 6
const tstring &vatf( const char *fmt, va_list ap ) const; .............. 7
const tstring &at( size_t index ) const; ............................... 8
```

DESCRIPTION

引数のキー文字列 (メンバ関数 1〜3, 5〜7)，または要素番号 (メンバ関数 4, 8) に対応する要素値 (tstring クラス; §9) の参照を返します．これらメンバ関数の直後に「.」で接続し，tstring クラス (§9) のメンバ関数を使う事ができます (EXAMPLE では tstring クラスの「=」演算子と cstr() メンバ関数を使っています)．メンバ関数 1〜4 は要素の読み書き両方に利用できますが，メンバ関数 5〜8 は読み取り専用です．

メンバ関数 2,3,6,7 は，指定したいキー文字列を printf() 関数と同様のフォーマットと可変引数でセットできます．メンバ関数 2,6 では，可変長引数の各要素データを `fmt` の変換指定に応じて変換します．メンバ関数 3,7 では，可変長引数のリスト `ap` を `fmt` の変換指定に応じて変換します．`fmt` については §8.1.14の解説を参照してください．

存在しないキー文字列が指定された場合，メンバ関数 1〜3 では指定されたキー文字列と値"" のセットを連想配列に追加しますが，メンバ関数 5〜7 では例外が発生します．

メンバ関数 4,8 の場合，`index` に配列の長さ以上の値を指定すると，例外が発生します．なお，配列の先頭の要素番号は `0` です．

メンバ関数 1〜4・メンバ関数 5〜8 のどちらが使われるかは，オブジェクトの「const」属性の有無で自動的に決まります.「const」属性が無い場合にはメンバ関数 1〜4 が，有る場合にはメンバ関数 5〜8 が自動的に選択されます．

PARAMETER

[I]	`key`	連想配列のキー文字列
[I]	`fmt`	キー文字列のためのフォーマット指定
[I]	`...`	`fmt` に対応した可変長引数の各要素データ
[I]	`ap`	`fmt` に対応した可変長引数のリスト
[I]	`index`	要素番号

([I]：入力，[O]：出力)

RETURN VALUE

指定されたキー，または要素番号に該当する要素値 (tstring クラスのオブジェクト) の参照

EXCEPTION

指定されたキー文字列が NULL の場合
存在しないキー文字列が指定された場合 (メンバ関数 5〜7 の場合)
指定された要素番号が不正な場合 (メンバ関数 4,8 の場合)
内部バッファの確保に失敗した場合 (メンバ関数 1〜3,6,7)

EXAMPLE

次のコードは，連想配列 `my_asarr` にキー`"Yasushi Inoue"`，値に`"Tougyu"`の組を追加し，確認の為，標準出力します．

```
stdstreamio sio;

asarray_tstring my_asarr;
my_asarr["Takeshi kaikou"]  = "Hadaka no Oosama";
my_asarr["Koubou Abe"]      = "Kabe";
my_asarr["Kenzaburou Ooe"]  = "Shiiku";
```

```
    my_asarr.at("Yasushi Inoue") = "Tougyu";
    sio.printf("my_asarr c_str ... [%s]\n", my_asarr.at("Yasushi Inoue").cstr());
```

実行結果

```
my_asarr c_str ...  [Tougyu]
```

11.4.5 at_cs(), atf_cs()

NAME

at_cs(), atf_cs() — 指定されたキー，または要素番号に該当する要素値 (tstring クラスのオブジェクト) の参照 (読み取り専用)

SYNOPSIS

```
const tstring &at_cs( const char *key ) const;  ............................. 1
const tstring &atf_cs( const char *fmt, ... ) const;  ....................... 2
const tstring &vatf_cs( const char *fmt, va_list ap ) const;  ............. 3
const tstring &at_cs( size_t index ) const;  ............................... 4
```

DESCRIPTION

引数のキー文字列 (メンバ関数 1〜3)，または要素番号 (メンバ関数 4) に対応する要素値 (tstring クラス; §9) の参照を返します．これらのメンバ関数は，読み取り専用です．

メンバ関数 2,3 は，指定したいキー文字列を printf() 関数と同様のフォーマットと可変引数でセットできます．メンバ関数 2 では，可変長引数の各要素データを fmt の変換指定に応じて変換します．メンバ関数 3 では，可変長引数のリスト ap を fmt の変換指定に応じて変換します．fmt については §8.1.14の解説を参照してください．

なお，配列の先頭の要素番号は 0 です．

存在しないキー文字列や要素番号を指定すると，例外が発生します．

PARAMETER

[I]	key	連想配列のキー文字列
[I]	fmt	キー文字列のためのフォーマット指定
[I]	...	fmt に対応した可変長引数の各要素データ
[I]	ap	fmt に対応した可変長引数のリスト
[I]	index	要素番号

([I]：入力，[O]：出力)

RETURN VALUE

指定されたキー，または要素番号に該当する要素値 (tstring クラスのオブジェクト) の参照

EXCEPTION

指定されたキー文字列が NULL の場合
存在しないキー文字列が指定された場合 (メンバ関数 1〜3 の場合)
指定された要素番号が不正な場合
内部バッファの確保に失敗した場合 (メンバ関数 2,3)

11.4.6 index(), indexf(), vindexf()

NAME

index(), indexf(), vindexf() — キー文字列に該当する要素番号を取得

SYNOPSIS

`ssize_t index( const char *key ) const;` 1

`ssize_t indexf( const char *fmt, ... ) const;` 2

`size_t vindexf( const char *fmt, va_list ap ) const;` 3

DESCRIPTION

キー文字列に該当する要素番号を取得します．なお，配列の先頭の要素番号は 0 です．

メンバ関数 2,3 は，指定したいキー文字列を printf() 関数と同様のフォーマットと可変引数でセットできます．メンバ関数 2 では，可変長引数の各要素データを fmt の変換指定に応じて変換します．メンバ関数 3 では，可変長引数のリスト ap を fmt の変換指定に応じて変換します．fmt については §8.1.14の解説を参照してください．

PARAMETER

[I]	key	キー文字列
[I]	fmt	キー文字列のためのフォーマット指定
[I]	...	fmt に対応した可変長引数の各要素データ
[I]	ap	fmt に対応した可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

非負の値	:	指定されたキー文字列が見つかった場合，該当する要素番号
負の値 (エラー)	:	指定されたキー文字列が見つからなかった場合

EXCEPTION

内部バッファの確保に失敗した場合 (メンバ関数 2,3)

EXAMPLE

次のコードは，オブジェクト my_country が持つ連想配列のなかで，キーが"Muritaniya"である要素番号を取得し，標準出力します．

```
stdstreamio sio;

asarray_tstring my_country;
my_country["Saudi Arabia"] = "Abdullah bin Abdulaziz al-Saud";
my_country["Muritaniya"] = "Mohamed Ould Abdel Aziz";
my_country["Cyprus"] = "Demetris Christofias";

sio.printf("my_country.index(\"Muritaniya\") ... [%zd]\n",
           my_country.index("Muritaniya"));
```

実行結果

```
my_country.index("Muritaniya") ...  [1]
```

11.4.7 key()

NAME

key() — 要素番号に該当するキー文字列を取得

SYNOPSIS

```
const char *key( size_t index ) const;
```

DESCRIPTION

index で指定された要素番号に該当するキー文字列を取得します．なお，配列の先頭の要素番号は 0 です．

index に配列の長さ以上の値が指定された場合，NULL を返します．

PARAMETER

[I] index 要素番号
([I]：入力，[O]：出力)

RETURN VALUE

キー文字列の内部バッファのアドレス (正常終了)
NULL(エラー) ： 配列の長さ以上の要素番号が指定された場合

EXAMPLE

次のコードは，オブジェクト my_city が持つ連想配列のキーを 0 から順に取得し，取得したキーと値を標準出力します．

```
stdstreamio sio;

asarray_tstring my_city;
my_city["Yokohama"] = "Fumiko Hayashi";
my_city["Osaka"]    = "Kunio Hiramatsu";
my_city["Fukuoka"]  = "Hiroshi Yoshida";

for ( size_t i=0 ; i < my_city.length() ; i++ ) {
    const char *key = my_city.key(i);
    sio.printf("#%zx -> %s ... [%s]\n", i, key, my_city.cstr(key));
}
```

実行結果

```
#0 -> Yokohama ...  [Fumiko Hayashi]
#1 -> Osaka ...  [Kunio Hiramatsu]
#2 -> Fukuoka ...  [Hiroshi Yoshida]
```

11.4.8 keys()

NAME

keys() — キー文字列の配列オブジェクトを参照 (読み取りのみ)

SYNOPSIS

```
const tarray_tstring &keys() const;
```

DESCRIPTION

オブジェクト内で管理しているキー文字列の配列オブジェクト (tarray_tstring クラス; §10) の参照を返します．このメンバ関数の直後に「.」で接続し，tarray_tstring クラスのメンバ関数を使う事ができます．利用可能な tarray_tstring クラスのメンバ関数は，キー文字列を改変しないもの，すなわち const 属性を有するものに限ります．

RETURN VALUE

キー文字列の配列オブジェクト (tarray_tstring クラス) の参照

11.4.9 values()

NAME

values() — 値文字列の配列オブジェクトを参照 (読み取りのみ)

SYNOPSIS

```
const tarray_tstring &values() const;
```

DESCRIPTION

オブジェクト内で管理している値文字列の配列オブジェクト (tarray_tstring クラス; §10) の参照を返します．このメンバ関数の直後に「.」で接続し，tarray_tstring クラスのメンバ関数を使う事ができます．利用可能な tarray_tstring クラスのメンバ関数は，値文字列を改変しないもの，すなわち const 属性を有するものに限ります．

RETURN VALUE

値文字列の配列オブジェクト (tarray_tstring クラス) の参照

11.4.10 dprint()

NAME

dprint() — オブジェクト情報を標準エラー出力へ出力 (ユーザのデバッグ用)

SYNOPSIS

```
void dprint() const;
```

DESCRIPTION

自身のオブジェクト情報を標準エラー出力へ出力します．

ユーザ・プログラムのデバッグを目的としたメンバ関数です．

EXAMPLE

次のコードは，オブジェクト `my_array` の情報を標準エラー出力に出力します．実行結果では，`[]` にオブジェクトのアドレスが表示されていますが，これは実行環境により異なります．

```
asarray_tstring my_array("CPU","Sparc", "OS","Solaris", NULL);
my_array.dprint();
```

実行結果

```
sli::asarray_tstring[obj=0xbffff3d0] = { {"CPU", "Sparc"}, {"OS", "Solaris"} }
```

11.4.11 swap()

NAME

swap() — オブジェクトの入替え

SYNOPSIS

```
asarray_tstring &swap( asarray_tstring &sobj );
```

DESCRIPTION

オブジェクト sobj の内容と自身の内容とを入れ替えます.

PARAMETER

[I/O] sobj 内容を入れ替える asarray_tstring クラスのオブジェクト
([I] : 入力, [O] : 出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは, オブジェクト my_africa と my_america が持つ連想配列を入れ替え, 確認のため, 各内容を標準出力します.

```
stdstreamio sio;

asarray_tstring my_africa;
my_africa["Rwanda"] = "Kigali";
my_africa["Cameroon"] = "Yaounde";

asarray_tstring my_america;
my_america["Honduras"] = "Tegucigalpa";
my_america["Jamaica"] = "Kingston";

my_africa.swap(my_america);
for ( size_t i=0 ; i < my_africa.length() ; i++ ) {
    const char *africa_key = my_africa.key(i);
    const char *africa__key = my_america.key(i);
    sio.printf("[%s]:%s <===> [%s]:%s\n", africa_key,
               my_africa.cstr(africa_key),
               africa__key, my_africa.cstr(africa_key));
}
```

実行結果

```
[Honduras]:Tegucigalpa <===> [Rwanda]:Tegucigalpa
[Jamaica]:Kingston <===> [Cameroon]:Kingston
```

11.4.12 init()

NAME

init() — オブジェクトの完全初期化

SYNOPSIS

```
asarray_tstring &init(); ........................................ 1
asarray_tstring &init(const asarray_tstring &obj); .............. 2
```

DESCRIPTION

自身の連想配列の初期化を行います.

メンバ関数 1 は，オブジェクトを完全に初期化します．連想配列の配列バッファや文字列バッファ等に割り当てられたメモリ領域は完全に開放されます．`init()` を実行後，`cstrarray()` メンバ関数 (§11.4.2) を実行すると `NULL` が返ります.

メンバ関数 2 は `obj` の内容で初期化を行います (`obj` の内容すべてを自身にコピーします).

PARAMETER

[I] `obj` asarray_tstring クラスのオブジェクト (コピー元)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

メモリ破壊を起こした場合 (メンバ関数 2)

EXAMPLE

次のコードは，連想配列 `my_asarr` を `IgNobel_asarr` で初期化し，確認の為，標準出力します.

```
stdstreamio sio;

asarray_tstring IgNobel_asarr;
asarray_tstring my_asarr;
IgNobel_asarr["2008"] = "Toshiyuki Nakagaki";
IgNobel_asarr["2007"] = "Mayu Yamamoto";
IgNobel_asarr["2006"] = "Dr.Nakamatsu";

my_asarr.init(IgNobel_asarr);
for ( size_t i=0 ; i < my_asarr.length() ; i++ ) {
    const char *key = my_asarr.key(i);
    sio.printf("%s ... [%s]\n", key, my_asarr.cstr(key));
}
```

実行結果

```
2008 ... [Toshiyuki Nakagaki]
2007 ... [Mayu Yamamoto]
2006 ... [Dr.Nakamatsu]
```

11.4.13 assign(), assignf(), vassignf()

NAME

assign(), assignf(), vassignf() — オブジェクトの初期化と要素の代入 (1 つのキー・値のセットを指定)

SYNOPSIS

```
asarray_tstring &assign( const char *key, const char *val );  .............. 1
asarray_tstring &assign( const char *key, const tstring &val );  ........... 2
asarray_tstring &assignf( const char *key, const char *fmt, ... );  ....... 3
asarray_tstring &vassignf( const char *key, const char *fmt, va_list ap );  4
```

DESCRIPTION

自身の連想配列を，指定された 1 つの要素 (キー・値の組合せ) で初期化します.

メンバ関数 1，2 は，キー key と値 val で初期化します.

メンバ関数 3，4 は，指定したい値文字列を printf() 関数と同様のフォーマットと可変引数でセットできます. メンバ関数 3 では，可変長引数の各要素データを fmt の変換指定に応じて変換します. メンバ関数 4 では，可変長引数のリスト ap を fmt の変換指定に応じて変換します. fmt については §8.1.14の解説を参照してください.

PARAMETER

[I]	key	連想配列に設定するキー文字列
[I]	val	連想配列に設定する値文字列
[I]	fmt	設定する値文字列のためのフォーマット指定
[I]	...	fmt に対応した可変長引数の各要素データ
[I]	ap	fmt に対応した可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，連想配列 my_asarr に 1 つのキー・値のセットを設定し，その結果を標準出力します.

```
stdstreamio sio;

asarray_tstring my_asarr;

const char *key0 = "Everest";
const char *val0 = "Nepal";
my_asarr.assign(key0,val0);
for ( size_t i=0 ; i < my_asarr.length() ; i++ ) {
    const char *key = my_asarr.key(i);
```

```
        sio.printf("%s ... [%s]\n", key, my_asarr.cstr(key));
    }
```

実行結果

```
Everest ...  [Nepal]
```

11.4.14 assign(), vassign()

NAME

assign(), vassign() — オブジェクトの初期化と要素の代入 (複数のキー・値のセットを指定)

SYNOPSIS

```
asarray_tstring &assign( const asarray_tstring &src );  .................... 1
asarray_tstring &assign( const asarrdef_tstring elements[] );  ............ 2
asarray_tstring &assign( const asarrdef_tstring elements[], size_t n );  . 3
asarray_tstring &assign( const char *key0, const char *val0,
                         const char *key1, ... );  ......................... 4
asarray_tstring &vassign( const char *key0, const char *val0,
                          const char *key1, va_list ap );  ................. 5
```

DESCRIPTION

自身の連想配列を，指定された複数の要素 (キー・値の組合せ) で初期化します.

メンバ関数 1 は，asarray_tstring クラスのオブジェクト src の内容を自身に代入します.

メンバ関数 2，3 は，asarrdef_tstring 型 (構造体) の配列の内容を設定します (メンバ関数 2 の場合，elements は {NULL,NULL} で終端している必要があります)．メンバ関数 3 は，elements の先頭から n 個分を設定します．elements の個数 ({NULL,NULL} に達するまで) を超えた n が指定された場合は，n は無視されます.

メンバ関数 4 およびメンバ関数 5 は，key0，val0，key1 とそれらに続く可変長引数にセットされた複数の const char *型のキーと値の組合せで，連想配列を作成します (NULL で終端している必要があります).

PARAMETER

[I]	src	源泉となる要素を持つ asarray_tstring クラスのオブジェクト
[I]	elements	源泉となる要素を持つ asarrdef_tstring 型 (構造体) の配列 (メンバ関数 2 の場合，{NULL,NULL} で終端)
[I]	n	配列 elements の個数
[I]	key0, key1	連想配列に設定するキー文字列
[I]	val0	連想配列に設定する値文字列
[I]	...	キー・値となる文字列の可変長引数の各要素データ (要 NULL 終端)
[I]	ap	キー・値となる文字列の可変長引数のリスト (要 NULL 終端)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，連想配列 `my_asarr` に構造体 asarrdef_tstring 型の配列の内容を設定し，その結果を標準出力します．配列の最後には，キーと値の両方に `NULL` を設定します．

```
stdstreamio sio;

asarray_tstring my_asarr;
const asarrdef_tstring mount_elem[] = { {"K2","China"},
        {"Kangchenjunga","Nepal"}, {"Mount Kenya","Kenya"}, {NULL,NULL} };

my_asarr.assign(mount_elem);
for ( size_t i=0 ; i < my_asarr.length() ; i++ ) {
    const char *key = my_asarr.key(i);
    sio.printf("%s ... [%s]\n", key, my_asarr.cstr(key));
}
```

実行結果

```
K2 ...  [China]
Kangchenjunga ...  [Nepal]
Mount Kenya ...  [Kenya]
```

11.4.15 assign_keys()

NAME

assign_keys() — 複数の文字列または文字列配列をキーに設定

SYNOPSIS

```
asarray_tstring &assign_keys( const char *key0, ... );  ..................... 1
asarray_tstring &vassign_keys( const char *key0, va_list ap );  ........... 2
asarray_tstring &assign_keys( const char *const *keys );  .................. 3
asarray_tstring &assign_keys( const tarray_tstring &keys );  .............. 4
```

DESCRIPTION

指定された複数の文字列 `key0, ...,` または文字列配列 `keys` を，自身の連想配列のキーに設定します．

引数で指定されたキーの個数が連想配列の個数に設定されます (引数で指定されたキーの個数を越える連想配列要素は削除されます)．

メンバ関数 1,2 の可変引数，メンバ関数 3 のポインタ配列 `keys` は `NULL` で終端していなければなりません．

PARAMETER

[I]	`key0`	キー文字列
[I]	`...`	キー文字列の可変長引数の各要素データ (`NULL` 終端)
[I]	`ap`	キー文字列の可変長引数のリスト (`NULL` 終端)
[I]	`keys`	キー文字列に設定する文字列配列 (メンバ関数 3 の場合，`NULL` で終端)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

§11.4.16の EXAMPLE を参照してください．

11.4.16 assign_values()

NAME

assign_values() — 複数の文字列または文字列配列を値に設定

SYNOPSIS

```
asarray_tstring &assign_values( const char *val0, ... );  .................. 1
asarray_tstring &vassign_values( const char *val0, va_list ap );  ......... 2
asarray_tstring &assign_values( const char *const *values );  ............. 3
asarray_tstring &assign_values( const tarray_tstring &values );  .......... 4
```

DESCRIPTION

指定された複数の文字列 `val0`，`...`，または文字列配列 `values` を，自身の連想配列の値に設定します．

引数で指定された値の個数が自身の連想配列の個数を越えた場合，越えた部分の値は捨てられます．

メンバ関数 1,2 の可変引数，メンバ関数 3 のポインタ配列 `values` は `NULL` で終端していなければなりません．

PARAMETER

[I]	`val0`	値文字列
[I]	`...`	値文字列の可変長引数の各要素データ (`NULL` 終端)
[I]	`ap`	値文字列の可変長引数のリスト (`NULL` 終端)
[I]	`values`	値文字列に設定する文字列配列 (メンバ関数 3 の場合，`NULL` で終端)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

assign_keys() と assign_values() とを使って，連想配列を初期化しています．

```
asarray_tstring my_array;
my_array.assign_keys("CPU", "OS", NULL);
my_array.assign_values("PentiumPro", "Linux", NULL);
my_array.dprint();
```

実行結果

```
sli::asarray_tstring[obj=0xbffff3d0] = { {"CPU", "PentiumPro"}, {"OS", "Linux"} }
```

11.4.17 split_keys()

NAME

split_keys() — 文字列を分割し，キーに設定

SYNOPSIS

```
asarray_tstring &split_keys( const char *src_str, const char *delims,
                             bool zero_str, const char *quotations,
                             int escape, bool rm_escape );  ................ 1
asarray_tstring &split_keys( const char *src_str, const char *delims,
                             bool zero_str = false );  ...................... 2
asarray_tstring &split_keys( const tstring &src_str, const char *delims,
                             bool zero_str, const char *quotations,
                             int escape, bool rm_escape );  ................ 3
asarray_tstring &split_keys( const tstring &src_str, const char *delims,
                             bool zero_str = false );  ...................... 4
```

DESCRIPTION

文字列 src_str を区切り文字で分割し，その結果を自身の連想配列のキーに入れます．区切り文字は、文字リストとして delims で与え，" \t"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]"あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説と表 19 を参照してください．

文字列の分割後に得られたキーの個数が連想配列の個数に設定されます (文字列の分割後に得られたキーの個数を越える連想配列要素は削除されます)．

分割後のキー要素として，ゼロの文字列長を許すかどうかを，zero_str で指定できます．zero_str が false の場合，文字列長 0 のキー要素の作成を許可しません．zero_str が true の場合，文字列長 0 のキー要素の作成を許可します (csv 形式の場合などに使います)．zero_str の指定がない場合は，false の扱いとなります．

クォーテーション等,「特定の文字」で囲まれた文字列は分割しない場合，quotations に「特定の文字」を指定します．たとえば，シングルクォーテーションで囲む文字列を分割対象外とする場合は，"'"と指定します．

エスケープ文字は`escape`で指定します．分割後のエスケープ文字を削除する場合は，`rm_escape`を`true`にセットします．ただし，`quotations`で指定された文字で囲まれた部分のエスケープ文字は削除しません．

このメンバ関数だけでうまくキーが取り出せない場合は，一旦 tarray_tstring クラスのオブジェクトにキー文字列を作成し，assign_keys() メンバ関数 (§11.4.15) を使って自身のキーを設定する方法もあります．

PARAMETER

[I]	`src_str`	分割対象の文字列
[I]	`delims`	区切り文字を含む文字列
[I]	`zero_str`	長さ 0 の区切り結果の文字列を許すか (true/false)
[I]	`quotations`	クォーテーション文字を含む文字列
[I]	`escape`	エスケープ文字
[I]	`rm_escape`	エスケープ文字を削除するかどうかのフラグ (true/false)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

§11.4.18の EXAMPLE を参照して下さい．

§3.5.5にメンバ関数 2 を使った例があります．

11.4.18 split_values()

NAME

split_values() — 文字列を分割し，値に設定

SYNOPSIS

```
asarray_tstring &split_values( const char *src_str, const char *delims,
                               bool zero_str, const char *quotations,
                               int escape, bool rm_escape );  ............. 1
asarray_tstring &split_values( const char *src_str, const char *delims,
                               bool zero_str = false );  ................... 2
asarray_tstring &split_values( const tstring &src_str, const char *delims,
                               bool zero_str, const char *quotations,
                               int escape, bool rm_escape );  ............. 3
asarray_tstring &split_values( const tstring &src_str, const char *delims,
                               bool zero_str = false );  ................... 4
```

DESCRIPTION

文字列`src_str`を区切り文字で分割し，その結果を自身の連想配列の値に入れます．区切り文字は、文字リストとして`delims`で与え，`" \t"`のような単純な文字リストに加え，正規表現で

用いられる "[A-Z]"あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは§9.5.26の解説と表 19 を参照してください．

文字列の分割後に得られた値の個数が，自身の連想配列の個数を越えた場合，越えた部分の値は捨てられます．

分割後の値の要素として，ゼロの文字列長を許すかどうかを，zero_str で指定できます．zero_str が false の場合，文字列長 0 の値の要素の作成を許可しません．zero_str が true の場合，文字列長 0 の値の要素の作成を許可します (csv 形式の場合などに使います)．zero_str の指定がない場合は，false の扱いとなります．

クォーテーション等,「特定の文字」で囲まれた文字列は分割しない場合，quotations に「特定の文字」を指定します．たとえば，シングルクォーテーションで囲む文字列を分割対象外とする場合は，"'"と指定します．

エスケープ文字は escape で指定します．分割後のエスケープ文字を削除する場合は，rm_escape を true にセットします．ただし，quotations で指定された文字で囲まれた部分のエスケープ文字は削除しません．

PARAMETER

[I]	src_str	分割対象の文字列
[I]	delims	区切り文字を含む文字列
[I]	zero_str	長さ 0 の区切り結果の文字列を許すか (true/false)
[I]	quotations	クォーテーション文字を含む文字列
[I]	escape	エスケープ文字
[I]	rm_escape	エスケープ文字を削除するかどうかのフラグ (true/false)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，区切り文字" "とクォーテーション文字を指定して文字列を分割し，分割した文字列を連想配列 my_arr にキーとして設定します．次に，区切り文字","とクォーテーション文字を指定して文字列を分割し，分割した文字列を連想配列 my_arr に値として設定しています．確認の為，設定したキーと値を標準出力します．

```
stdstreamio sio;

const char *line =
      "'Camellia sasanqua' 'Chrysanthemum morifolium' 'Cyclamen persicum'";
asarray_tstring my_arr;
my_arr.split_keys(line, " ", false, "'", 0);
for ( size_t i=0 ; i < my_arr.length() ; i++ ) {
    const char *key = my_arr.key(i);
    sio.printf("%s ... [%s]\n", key, my_arr.cstr(key));
```

```
    }

    const char *val =
              "'Camellia,pink','Chrysanthemum,yellow','Cyclamen,pink'";
    my_arr.split_values(val, ",", false, "'", 0);
    for ( size_t i=0 ; i < my_arr.length() ; i++ ) {
        const char *key = my_arr.key(i);
        sio.printf("%s ... [%s]\n", key, my_arr.cstr(key));
    }
```

実行結果

```
'Camellia sasanqua' ...  ['Camellia,pink']
'Chrysanthemum morifolium' ...  ['Chrysanthemum,yellow']
'Cyclamen persicum' ...  ['Cyclamen,pink']
```

§3.5.5 および §11.4.27 にメンバ関数 2 を使った例があります.

11.4.19 append(), appendf(), vappendf()

NAME

append(), appendf(), vappendf() — 要素を追加 (1 つのキー・値のセットを指定)

SYNOPSIS

```
asarray_tstring &append( const char *key, const char *val );  ............. 1
asarray_tstring &append( const char *key, const tstring &val );  ........... 2
asarray_tstring &appendf( const char *key, const char *fmt, ... );  ....... 3
asarray_tstring &vappendf( const char *key, const char *fmt, va_list ap );  4
```

DESCRIPTION

自身の連想配列に，指定された 1 つの要素 (キー・値の組合せ) を追加します.

メンバ関数 1，2 は，キー `key` と値 `val` を追加します.

メンバ関数 3，4 は，指定したい値文字列を printf() 関数と同様のフォーマットと可変引数でセットできます．メンバ関数 3 では，可変長引数の各要素データを `fmt` の変換指定に応じて変換します．メンバ関数 4 では，可変長引数のリスト `ap` を `fmt` の変換指定に応じて変換します. `fmt` については §8.1.14の解説を参照してください.

キーが重複した場合，実行時に標準エラー出力に警告が出力され，処理が行われません.

PARAMETER

[I]	`key`	連想配列に追加するキー文字列
[I]	`val`	連想配列に追加する値文字列
[I]	`fmt`	追加する値文字列のためのフォーマット指定
[I]	`...`	`fmt` に対応した可変長引数の各要素データ
[I]	`ap`	`fmt` に対応した可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，appendf() メンバ関数を使って，オブジェクト my_asarr が持つ連想配列に 1 つ要素を追加し，その結果を標準出力します.

```
stdstreamio sio;

asarray_tstring my_asarr("rice","China", "coffee","Brazile", NULL);
const char *key_cacao = "cacao";

my_asarr.appendf(key_cacao,"### No.1 is %s ###","Cote d'Ivoire");
for ( size_t i=0 ; i < my_asarr.length() ; i++ ) {
    const char *key = my_asarr.key(i);
    sio.printf("%s ... [%s]\n", key, my_asarr.cstr(key));
}
```

実行結果

```
rice ...  [China]
coffee ...  [Brazile]
cacao ...  [### No.1 is Cote d'Ivoire ###]
```

11.4.20 append(), vappend()

NAME

append(), vappend() — 要素を追加 (複数のキー・値のセットを指定)

SYNOPSIS

```
asarray_tstring &append( const asarray_tstring &src );  .................... 1
asarray_tstring &append( const asarrdef_tstring elements[] );  ............ 2
asarray_tstring &append( const asarrdef_tstring elements[], size_t n );  . 3
asarray_tstring &append( const char *key0, const char *val0,
                         const char *key1, ... );  ......................... 4
asarray_tstring &vappend( const char *key0, const char *val0,
                          const char *key1, va_list ap );  ................ 5
```

DESCRIPTION

自身の連想配列に，指定された複数の要素 (キー・値の組合せ) を追加します.

メンバ関数 1 は，asarray_tstring クラスのオブジェクト src の内容を追加します.

メンバ関数 2，3 は，asarrdef_tstring 型 (構造体) の配列の内容を追加します (メンバ関数 2 の場合，elements は {NULL,NULL} で終端している必要があります)．メンバ関数 3 は，elements の先頭から n 個分を追加します．elements の個数 ({NULL,NULL} に達するまで) を超えた n が指定された場合は，n は無視されます.

メンバ関数 4 およびメンバ関数 5 は，key0，val0，key1 とそれらに続く可変長引数にセットされた const char *型のキーと値の組合せを，連想配列の要素として追加します (NULL で終端している必要があります).

キーが重複した場合，実行時に標準エラー出力に警告が出力され，処理が行われません.

PARAMETER

[I]	src	源泉となる要素を持つ asarray_tstring クラスのオブジェクト
[I]	elements	源泉となる要素を持つ asarrdef_tstring 型 (構造体) の配列 (メンバ関数 2 の場合，{NULL,NULL} で終端)
[I]	n	配列 elements の個数
[I]	key0, key1	連想配列に追加するキー文字列
[I]	val0	連想配列に追加する値文字列
[I]	...	キー・値となる文字列の可変長引数の各要素データ (要 NULL 終端)
[I]	ap	キー・値となる文字列の可変長引数のリスト (要 NULL 終端)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_asarr が持つ連想配列に，キーと値の組で定義した配列 foods を追加し，その結果を標準出力します．配列 foods の最後には，キーと値の両方に NULL を設定します.

```
stdstreamio sio;

asarray_tstring my_asarr("rice","China", "coffee","Brazile", NULL);
const asarrdef_tstring foods[] = { {"banana","India"},
                                   {"wheat","China"}, {NULL,NULL} };

my_asarr.append(foods);
for ( size_t i=0 ; i < my_asarr.length() ; i++ ) {
    const char *key = my_asarr.key(i);
    sio.printf("%s ... [%s]\n", key, my_asarr.cstr(key));
}
```

実行結果

```
rice ...  [China]
coffee ...  [Brazile]
banana ...  [India]
wheat ...  [China]
```

11.4.21 insert(), insertf(), vinsertf()

NAME

insert(), insertf(), vinsertf() — 要素を挿入 (1 つのキー・値のセットを指定)

SYNOPSIS

```
asarray_tstring &insert( const char *key,
                         const char *newkey, const char *newval ); ....... 1
asarray_tstring &insert( const char *key,
                         const char *newkey, const tstring &newval ); .... 2
asarray_tstring &insertf( const char *key,
                          const char *newkey, const char *fmt, ... ); .... 3
asarray_tstring &vinsertf( const char *key,
                           const char *newkey, const char *fmt, va_list ap ); 4
```

DESCRIPTION

自身の連想配列のキー key の要素位置の前に，指定された 1 つの要素 (キー・値の組合せ) を挿入します.

メンバ関数 1，2 は，キー newkey と値 newval の組合せを要素として挿入します.

メンバ関数 3，4 は，指定したい値文字列を printf() 関数と同様のフォーマットと可変引数でセットできます. メンバ関数 3 では，可変長引数の各要素データを fmt の変換指定に応じて変換します. メンバ関数 4 では，可変長引数のリスト ap を fmt の変換指定に応じて変換します. fmt については §8.1.14の解説を参照してください.

キーが重複した場合，実行時に警告が出力され，処理が行われません.

PARAMETER

[I]	key	挿入位置にある自身の連想配列のキーの文字列 (キーの前に挿入)
[I]	newkey	連想配列に挿入するキー文字列
[I]	newval	連想配列に挿入する値文字列
[I]	fmt	挿入する値文字列のためのフォーマット指定
[I]	...	fmt に対応した可変長引数の各要素データ
[I]	ap	fmt に対応した可変長引数のリスト

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_asarr が持つ連想配列に，1 つの要素を挿入し，その結果を標準出力します.

```
stdstreamio sio;

asarray_tstring my_asarr("Nile River","Africa", "the Amazon","South America",
```

```
                                  NULL);
    my_asarr.insert("the Amazon", "Chang River", "China");
    for ( size_t i=0 ; i < my_asarr.length() ; i++ ) {
        const char *key = my_asarr.key(i);
        sio.printf("%s ... [%s]\n", key, my_asarr.cstr(key));
    }
```

実行結果

```
Nile River ...  [Africa]
Chang River ...  [China]
the Amazon ...  [South America]
```

11.4.22 insert(), vinsert()

NAME

insert(), vinsert() — 要素を挿入 (複数のキー・値のセットを指定)

SYNOPSIS

```
asarray_tstring &insert( const char *key,
                         const asarray_tstring &src );  .................... 1
asarray_tstring &insert( const char *key,
                         const asarrdef_tstring elements[] );  ............ 2
asarray_tstring &insert( const char *key,
                         const asarrdef_tstring elements[], size_t n );  . 3
asarray_tstring &insert( const char *key,
                         const char *key0, const char *val0,
                         const char *key1, ... );  .......................... 4
asarray_tstring &vinsert( const char *key,
                          const char *key0, const char *val0,
                          const char *key1, va_list ap );  .................. 5
```

DESCRIPTION

自身の連想配列のキー key の要素位置の前に，指定された複数の要素 (キー・値の組合せ) を挿入します．

メンバ関数 1 は，asarray_tstring クラスのオブジェクト src の内容を挿入します．

メンバ関数 2，3 は，asarrdef_tstring 型 (構造体) の配列の内容を挿入します．(メンバ関数 2 の場合，elements は {NULL,NULL} で終端している必要があります)．メンバ関数 3 は，elements の先頭から n 個分を挿入します．elements の個数 ({NULL,NULL} に達するまで) を超えた n が指定された場合は，n は無視されます．

メンバ関数 4 およびメンバ関数 5 は，key0，val0，key1 とそれらに続く可変長引数にセットされた const char *型のキーと値の組合せを，連想配列の要素として挿入します (NULL で終端している必要があります)．

キーが重複した場合，実行時に標準エラー出力に警告が出力され，処理が行われません．

PARAMETER

[I]	key	挿入位置にある連想配列のキー文字列 (キーの前に挿入)
[I]	src	源泉となる要素を持つ asarray_tstring クラスのオブジェクト
[I]	elements	源泉となる要素を持つ asarrdef_tstring 型 (構造体) の配列 (メンバ関数 2 の場合，{NULL,NULL} で終端)
[I]	n	配列 elements の個数
[I]	key0, key1	連想配列に挿入するキー文字列
[I]	val0	連想配列に挿入する値文字列
[I]	...	キー・値となる文字列の可変長引数の各要素データ (要 NULL 終端)
[I]	ap	キー・値となる文字列の可変長引数のリスト (要 NULL 終端)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_asarr が持つ連想配列に，オブジェクト my_lake が持つ連想配列を挿入し，その結果を標準出力します．配列 lakes の最後には，キーと値の両方に NULL を設定します．

```
stdstreamio sio;

const asarrdef_tstring lakes[] = { {"Lake Superior","North America"},
                                   {"Lake Victoria","Tanzania"}, {NULL,NULL} };
asarray_tstring my_lake(lakes);
asarray_tstring my_asarr("Caspian Sea","Eurasia",
                         "Aral Sea","Kazakhstan", NULL);

my_asarr.insert("Aral Sea", my_lake);
for ( size_t i=0 ; i < my_asarr.length() ; i++ ) {
    const char *key = my_asarr.key(i);
    sio.printf("%s ... [%s]\n", key, my_asarr.cstr(key));
}
```

実行結果

```
Caspian Sea ...  [Eurasia]
Lake Superior ...  [North America]
Lake Victoria ...  [Tanzania]
Aral Sea ...  [Kazakhstan]
```

11.4.23 erase()

NAME

erase() — 要素 (キーと値のセット) の削除

SYNOPSIS

```
asarray_tstring &erase(); ...................................................... 1
asarray_tstring &erase( const char *key, size_t num_elements = 1 ); ...... 2
```

DESCRIPTION

自身の連想配列の要素を消去します.

メンバ関数 1 は，すべての要素を消去します (配列長はゼロになります).

メンバ関数 2 は，key で指定されたキーに該当する要素から num_elements 個の要素を消去します．num_elements が指定されない場合は，1 つの要素を消去します.

消去した分だけ，配列長は短くなります.

PARAMETER

[I]	key	キー文字列
[I]	num_elements	削除する要素の個数

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，オブジェクト my_asarr が持つ連想配列のキー"Democratic People's Republic of Korea" とその値を削除し，削除後の my_asarr の内容を標準出力します.

```
stdstreamio sio;

asarray_tstring my_asarr;
my_asarr["Kingdom of Lesotho"] = "Letsie III";
my_asarr["Democratic People's Republic of Korea"] = "Kim Jong-il";
my_asarr["Republic of Cote d'Ivoire"] = "Laurent Gbagbo";

my_asarr.erase("Democratic People's Republic of Korea");
for ( size_t i=0 ; i < my_asarr.length() ; i++ ) {
    const char *key = my_asarr.key(i);
    sio.printf("%s ... [%s]\n", key, my_asarr.cstr(key));
}
```

実行結果

```
Kingdom of Lesotho ... [Letsie III]
Republic of Cote d'Ivoire ... [Laurent Gbagbo]
```

11.4.24 clean()

NAME

clean() — 既存の連想配列の要素値すべてを任意の文字列でパディング

SYNOPSIS

`asarray_tstring &clean(const char *str = "");` 1
`asarray_tstring &clean(const tstring &str);` 2

DESCRIPTION

自身の連想配列の要素値すべてを，文字列 `str` でパディングします．この `str` は指定しなくても使用できます．その場合は，長さ 0 の文字列が指定されたものとして処理を行います．`clean()` を実行してもキーと配列長は変化しません.

PARAMETER

[I] `str` 連想配列の値をパディングする文字列
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，連想配列 `my_asarr` がもつ各文字列を `Grammy Award` でパディングし，その結果を標準出力します.

```
stdstreamio sio;

asarray_tstring my_asarr;
my_asarr["U2"] = "Beautiful Day";
my_asarr["Eric Clapton"] = "Tears In Heaven";
my_asarr["TOTO"] = "Rosanna";

my_asarr.clean("Grammy Award");
for ( size_t i=0 ; i < my_asarr.length() ; i++ ) {
    const char *key = my_asarr.key(i);
    sio.printf("%s ... [%s]\n", key, my_asarr.cstr(key));
}
```

実行結果

```
U2 ... [Grammy Award]
Eric Clapton ... [Grammy Award]
TOTO ... [Grammy Award]
```

11.4.25 rename_a_key()

NAME

rename_a_key() — キー文字列の変更

SYNOPSIS

```
asarray_tstring &rename_a_key( const char *org_key, const char *new_key );
```

DESCRIPTION

自身の連想配列のキー文字列 `org_key` を `new_key` で指定された文字列に変更します.

`org_key` に連想配列に存在しないキー文字列が指定された場合や，`new_key` が重複するキー文字列だった場合は，標準エラー出力にエラーメッセージを出力します.

PARAMETER

[I] `org_key` 元のキー文字列
[I] `new_key` 変更後のキー文字列
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

11.4.26 chomp()

NAME

chomp() — 全要素の値文字列の改行文字の除去

SYNOPSIS

```
asarray_tstring &chomp( const char *rs = "\n" );
asarray_tstring &chomp( const tstring &rs );
```

DESCRIPTION

自身の連想配列の全要素の値文字列の右端の改行文字を除去します.

このメンバ関数は，連想配列の全要素で tstring クラスの chomp() メンバ関数 (§9.5.25) を実行します．詳細は §9.5.25をご覧ください.

PARAMETER

[I] `rs` 改行文字列
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

11.4.27 trim()

NAME

trim() — 全要素の値文字列の両端スペースの除去

SYNOPSIS

```
asarray_tstring &trim( const char *side_spaces = " \t\n\r\f\v" );
asarray_tstring &trim( const tstring &side_spaces );
asarray_tstring &trim( int side_space );
```

DESCRIPTION

自身の連想配列の全要素の値文字列について，文字列両端にある任意文字を除去します．

`side_spaces` には，`" \t"`のような単純な文字リストに加え，正規表現で用いられる `"[A-Z]"` あるいは`"[^A-Z]"`のような指定が可能です．さらに，`"[…]"`の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説と表 19 を参照してください．

このメンバ関数は，連想配列の全要素で tstring クラスの trim() メンバ関数を実行します．詳細は §9.5.26をご覧ください．

PARAMETER

[I] `side_space` 任意文字
[I] `side_spaces` 任意文字列
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，CSV 形式の文字列を split_values() メンバ関数 (§11.4.18) で各要素に分割し，それぞれを連想配列の値としてオブジェクトに代入し，`trim()` を使って各要素の左右の余分な空白文字を除去しています．

```
stdstreamio sio;
asarray_tstring my_arr;
size_t i;
my_arr.assign_keys("CPU", "ChipSet", NULL);
my_arr.split_values(" Pentium4, E7205   ", ",", true);
for ( i=0 ; i < my_arr.length() ; i++ ) {
    const char *key = my_arr.key(i);
    sio.printf("%s ... [%s]\n", key, my_arr.cstr(key));
}
my_arr.trim();
for ( i=0 ; i < my_arr.length() ; i++ ) {
    const char *key = my_arr.key(i);
    sio.printf("%s ... [%s]\n", key, my_arr.cstr(key));
}
```

実行結果

```
CPU ... [ Pentium4]
ChipSet ... [ E7205   ]
CPU ... [Pentium4]
ChipSet ... [E7205]
```

11.4.28 ltrim()

NAME

ltrim() — 全要素の値文字列の左端スペースの除去

SYNOPSIS

```
asarray_tstring &ltrim( const char *side_spaces = " \t\n\r\f\v" );
asarray_tstring &ltrim( const tstring &side_spaces );
asarray_tstring &ltrim( int side_space );
```

DESCRIPTION

自身の連想配列の全要素の値文字列について，文字列左端にある任意文字を除去します．

side_spaces には，" \t"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]" あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説と表 19 を参照してください．

このメンバ関数は，連想配列の全要素で tstring クラスの ltrim() メンバ関数を実行します．詳細は §9.5.27をご覧ください．

PARAMETER

[I]	side_space	任意文字
[I]	side_spaces	任意文字列

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

11.4.29 rtrim()

NAME

rtrim() — 全要素の値文字列の右端スペースの除去

SYNOPSIS

```
asarray_tstring &rtrim( const char *side_spaces = " \t\n\r\f\v" );
asarray_tstring &rtrim( const tstring &side_spaces );
asarray_tstring &rtrim( int side_space );
```

DESCRIPTION

自身の連想配列の全要素の値文字列について，文字列右端にある任意文字を除去します．

side_spaces には，" \t"のような単純な文字リストに加え，正規表現で用いられる "[A-Z]" あるいは"[^A-Z]"のような指定が可能です．さらに，"[…]"の中では，文字クラスが指定できます．指定できる文字クラスは §9.5.26の解説と表 19 を参照してください．

このメンバ関数は，連想配列の全要素で tstring クラスの rtrim() メンバ関数を実行します．詳細は §9.5.28をご覧ください．

PARAMETER

[I]	side_space	任意文字
[I]	side_spaces	任意文字列

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

11.4.30 strreplace()

NAME

strreplace() — 全要素の値文字列の文字列検索と置換

SYNOPSIS

```
asarray_tstring &strreplace( const char *org_str, const char *new_str,
                             bool all = false );
asarray_tstring &strreplace( const tstring &org_str, const char *new_str,
                             bool all = false );
asarray_tstring &strreplace( const char *org_str, const tstring &new_str,
                             bool all = false );
asarray_tstring &strreplace( const tstring &org_str, const tstring &new_str,
                             bool all = false );
```

DESCRIPTION

自身の連想配列の全要素の値文字列について，文字列の左側から文字列 org_str を検索し，見つかった場合は文字列 new_str で置き換えます．

このメンバ関数は，連想配列の全要素で tstring クラスの strreplace() メンバ関数を実行します (pos は 0 が設定されます)．詳細は §9.5.29をご覧ください．

PARAMETER

[I]	org_str	検出する文字列
[I]	new_str	置換の源泉となる文字列
[I]	all	全置換のフラグ

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，全要素について空白文字をアンダーバー「_」に置き換えます.

```
stdstreamio sio;
asarray_tstring my_arr("OS","Solaris 9",
                       "VENDOR","Sun Microsystems, Inc.", NULL);
size_t i;
my_arr.strreplace(" ", "_", true);
for ( i=0 ; i < my_arr.length() ; i++ ) {
    const char *key = my_arr.key(i);
    sio.printf("%s ... [%s]\n", key, my_arr.cstr(key));
}
```

実行結果

```
OS ... [Solaris_9]
VENDOR ... [Sun_Microsystems,_Inc.]
```

11.4.31 regreplace()

NAME

regreplace() — 全要素の値文字列の正規表現による文字列検索と置換

SYNOPSIS

```
asarray_tstring &regreplace( const char *pat,
                             const char *new_str, bool all = false );
asarray_tstring &regreplace( const tstring &pat,
                             const char *new_str, bool all = false );
asarray_tstring &regreplace( const tregex &pat,
                             const char *new_str, bool all = false );
asarray_tstring &regreplace( const char *pat,
                             const tstring &new_str, bool all = false );
asarray_tstring &regreplace( const tstring &pat,
                             const tstring &new_str, bool all = false );
asarray_tstring &regreplace( const tregex &pat,
                             const tstring &new_str, bool all = false );
```

DESCRIPTION

自身の連想配列の全要素の値文字列について，`pat` で指定された POSIX 拡張正規表現 (以下, 正規表現) でマッチした部分を文字列 `new_str` で置き換えます．`new_str` では，後方参照`"\\0"`

〜"\\9" が利用できます ("\\0"はマッチした部分全体を示します)．バックスラッシュ自身を与えたい場合は，"\\\\"を指定します．

このメンバ関数は，連想配列の全要素で tstring クラスの regreplace() メンバ関数を実行します (pos は 0 が設定されます)．詳細は §9.5.30をご覧ください．

正規表現を必要としない場合は，strreplace() メンバ関数 (§11.4.30) の方が動作が高速です．

PARAMETER

[I]	pat	文字パターン (正規表現) または tregex クラスのコンパイル済オブジェクト
[I]	new_str	置換後の文字列
[I]	all	全置換のフラグ

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，エスケープ文字「\」をすべての要素値から消去するものです．全要素の値について正規表現 "([\\])(.)" のマッチを試行し，マッチした場合は後方参照要素 2 で置き換えています．

```
stdstreamio sio;
asarray_tstring my_arr("OS","Solaris",
                       "VENDOR","Sun\\ Microsystems\\,\\ Inc.", NULL);
size_t i;
my_arr.regreplace("([\\])(.)", "\\2", true);
for ( i=0 ; i < my_arr.length() ; i++ ) {
    const char *key = my_arr.key(i);
    sio.printf("%s ... [%s]\n", key, my_arr.cstr(key));
}
```

実行結果

```
OS ... [Solaris]
VENDOR ... [Sun Microsystems, Inc.]
```

11.4.32 tolower()

NAME

tolower() — 全要素の値文字列の大文字を小文字に置換

SYNOPSIS

```
asarray_tstring &tolower();
```

DESCRIPTION

自身の連想配列の全要素の値文字列のアルファベットの大文字を小文字に変換します.

このメンバ関数は，連想配列の全要素で tstring クラスの tolower() メンバ関数を実行します. 詳細は §9.5.31をご覧ください.

RETURN VALUE

自身の参照

11.4.33 toupper()

NAME

toupper() — 全要素の値文字列の小文字を大文字に置換

SYNOPSIS

```
asarray_tstring &toupper();
```

DESCRIPTION

自身の連想配列の全要素の値文字列のアルファベットの小文字を大文字に変換します.

このメンバ関数は，連想配列の全要素で tstring クラスの toupper() メンバ関数を実行します. 詳細は §9.5.32をご覧ください.

RETURN VALUE

自身の参照

11.4.34 expand_tabs()

NAME

expand_tabs() — 全要素の値文字列の TAB 文字を空白文字に置換

SYNOPSIS

```
asarray_tstring &expand_tabs( size_t tab_width = 8 );
```

DESCRIPTION

自身の連想配列の全要素の値文字列について，水平タブ文字'\t'を，`tab_width` の値に桁揃えをして空白文字に置換します.

このメンバ関数は，連想配列の全要素で tstring クラスの expand_tabs() メンバ関数を実行します. 詳細は §9.5.33をご覧ください.

PARAMETER

[I] `tab_width` TAB 幅

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

11.4.35 contract_spaces()

NAME

contract_spaces() — 全要素の値文字列の空白文字を TAB 文字に置換

SYNOPSIS

```
asarray_tstring &contract_spaces( size_t tab_width = 8 );
```

DESCRIPTION

自身の連想配列の全要素の値文字列について，2 文字以上連続した空白文字' 'すべてを対象にし，指定された TAB 幅 `tab_width` で桁揃えして`'\t'`で置換します.

このメンバ関数は，連想配列の全要素で tstring クラスの contract_spaces() メンバ関数を実行します．詳細は §9.5.34をご覧ください.

PARAMETER

[I] `tab_width` TAB 幅

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

---

12 MDARRAY_*クラス

mdarray_*クラスは，C 言語の主な型の多次元配列を簡単に扱えるクラスです．IDL のような感覚で，多次元配列に対する演算が可能です．

次のような特徴を持ちます．

- 配列の要素にアクセスする時に必要に応じてメモリ領域を自動確保する「自動リサイズモード」，画像データの扱いに適した「手動リサイズモード」の 2 つの動作モードを持つ．
- C 言語の主な型 (float，double，int 等) に対応したクラス (mdarray_float，mdarray_double，mdarray_int 等) が用意されており，異なるクラス間で演算・代入が可能．
- スカラー値，配列全体に対して演算子「+」,「-」,「*」,「/」,「+=」,「-=」,「*=」,「/=」,「=」が利用可能．
- 配列全体に対して演算を行なう数学関数 (`sin()`，`log()` 等) を用意 (libc の math.h で定義されている関数のほとんどが利用可能)．

表 24に利用可能なクラス一覧を示します．どのクラスも，親クラスである「mdarray」を継承しています (mdarray クラスについては上級編を参照)．この実装により，一部のクラス (表 24で「演算子による演算」が「N/A」となっているもの) を除いて，演算子「+」,「-」,「*」,「/」,「+=」,「-=」,「*=」,「/=」による配列全体に対する演算が行なえるようになっています．

クラス名	C 言語の型	演算子による演算
mdarray_float	float	OK
mdarray_double	double	OK
mdarray_uchar	unsigned char	OK
mdarray_short	short	OK
mdarray_int	int	OK
mdarray_long	long	OK
mdarray_llong	long long	OK
mdarray_int16	int16_t	OK
mdarray_int32	int32_t	OK
mdarray_int64	int64_t	OK
mdarray_size	size_t	N/A
mdarray_ssize	ssize_t	OK
mdarray_bool	bool	N/A
mdarray_uintptr	uintptr_t	N/A
mdarray_fcomplex	float complex	OK
mdarray_dcomplex	double complex	OK

表 24: 利用可能なクラス一覧．

表 24に示したクラスを使う場合は，ユーザコードの先頭に次のように書きます．

```
#include <sli/mdarray.h>
#include <sli/mdarray_math.h>
```

`mdarray_math.h` はクラスを利用するだけの場合は必要ありませんが，数学関数を配列全体に対して

使う場合に必要です．また，namespace 宣言 (§4.1) が必要な場合は,「using namespace sli;」もコードに書いてください．

簡単な使用例を次に示します．

```
#include <sli/stdstreamio.h>
#include <sli/mdarray.h>
#include <sli/mdarray_math.h>
using namespace sli;

int main()
{
    stdstreamio sio;
    size_t i;
    mdarray_long my_larr;
    mdarray_double my_darr;
    my_larr[0] = 100;
    my_larr[1] = 10;
    my_larr[2] = 1;
    my_darr = log10(my_larr);
    for ( i=0 ; i < my_darr.length() ; i++ ) {  /* my_darr のすべての要素を表示 */
        sio.printf("%zu ... [%g]\n", i, my_darr[i]);
    }
}
```

実行結果

```
0 ... [2]
1 ... [1]
2 ... [0]
```

12.1 オブジェクトの作り方

いくつかの方法で，オブジェクトを初期化する事ができます．この時，動作モードを指定できます．

動作モードは「自動リサイズモード」「手動リサイズモード」の 2 種類があります．

「自動リサイズモード」では，演算子「[]」，メンバ関数 at() 等を使って配列要素にアクセスしたり，演算子「+=」,「-=」等を使って配列の演算を行ったりする時に必要に応じて次元数と配列サイズを自動的に拡張します．一方,「手動リサイズモード」では，明示的にリサイズを指示しない限りバッファサイズを変更しません (画像データの扱いに適しています)．

「動作モード」により動作が異なるメンバ関数については，表 26 に「動作モードサポート」の「◯」で示しています．

12.1.1 何も引数を指定しない方法

この場合,「自動リサイズモード」で初期化されます．

```
    mdarray_double my_darr;
```

この状態ではバッファは確保されていません．この後,

```
my_darr[3] = 1.23;
my_darr(3,2,1) = 4.56;
```

のようにすると自動的にバッファが確保されます．前者の場合，バッファ数が 4 の 1 次元配列とな

り，後者の場合，バッファ数が 4×3×2 の 3 次元配列となります．値を与えない部分の初期値は 0 です．この時使われる初期値は，assign_default() メンバ関数で変更する事ができます．

12.1.2 配列の大きさを与える方法

次のようにすると，オブジェクト作成時に配列の大きさを与える事ができます．

```
    mdarray_double my_darr(false, 1920,1080,3);
```

最初の引数では必ず動作モードを与えます．「自動リサイズモード」で初期化したい場合は `true` を，そうでない場合は `false` を与えます．2 つ目の引数から，1 次元目，2 次元目，3 次元目 (3 次元まで) の要素数を与えます．上記の例では，1 次元目から順に，1920×1080×3 の配列を確保しています．

n 次元の配列を確保したい場合は，

```
    const size_t nelemx = {800, 600, 3, 2}
    mdarray_double my_darr(false, nelemx, 4);
```

のように与えます．この例は 4 次元の場合で，1 次元目から順に 800×600×3×2 の配列を確保しています．

12.1.3 配列の大きさと初期値を与える方法

オブジェクト作成時に初期値を与える方法としては，任意のバッファの先頭アドレスを与える方法と，任意のバッファに対するポインタ配列を与える方法があります．ただし初期値を与える場合は，次元数は 3 までに制限されます．

次のように，コンストラクタの引数に**最初の引数では必ず動作モードを与え**，次いで各次元の要素数，配列のアドレスを与えます．

```
    const double my_data1[] = {0.02, 0.2, 2.0};
    mdarray_double my_darr1(false, 3, my_data1);
```

```
    const double my_data2[][3] = {{0.02, 0.2, 2.0}, {20.0, 200.0, 2000.0}};
    mdarray_double my_darr2(false, 3,2, *my_data2);
```

```
    const double my_data3[][2][3] = {{{0.02, 0.2, 2.0}, {20.0, 200.0, 2000.0}},
                                     {{0.04, 0.4, 4.0}, {40.0, 400.0, 4000.0}}};
    mdarray_double my_darr3(false, 3,2,2, **my_data3);
```

これらはそれぞれ，1 次元，2 次元，3 次元の場合に初期値を与える方法を示しています．

コンストラクタの引数に，配列データに対するポインタ配列を与える事ができます．ユーザが作成した関数の引数を使って，オブジェクト内の配列に初期値を与える例を示します．

```
int my_function( const double *ptr[], int nx, int ny )
{
    mdarray_double my_darr2(false, nx,ny, ptr);
        :
        :
}
```

12.2 数学関数

表 25 に示す数学関数が利用できます.

関数プロトタイプ	機能
`mdarray cbrt( const mdarray &obj );`	立方根
`mdarray sqrt( const mdarray &obj );`	平方根
`mdarray asin( const mdarray &obj );`	逆正弦
`mdarray acos( const mdarray &obj );`	逆余弦
`mdarray atan( const mdarray &obj );`	逆正接
`mdarray acosh( const mdarray &obj );`	逆双曲線余弦
`mdarray asinh( const mdarray &obj );`	逆双曲線正弦
`mdarray atanh( const mdarray &obj );`	逆双曲線正接
`mdarray exp( const mdarray &obj );`	底が e の指数関数
`mdarray exp2( const mdarray &obj );`	底が 2 の指数関数
`mdarray expm1( const mdarray &obj );`	引き数の指数から 1 を引いた値
`mdarray log( const mdarray &obj );`	自然対数
`mdarray log1p( const mdarray &obj );`	引き数に 1 を加えた値の対数
`mdarray log10( const mdarray &obj );`	底が 10 の対数
`mdarray sin( const mdarray &obj );`	正弦
`mdarray cos( const mdarray &obj );`	余弦
`mdarray tan( const mdarray &obj );`	正接
`mdarray sinh( const mdarray &obj );`	双曲線正弦
`mdarray cosh( const mdarray &obj );`	双曲線余弦
`mdarray tanh( const mdarray &obj );`	双曲線正接
`mdarray erf( const mdarray &obj );`	誤差関数
`mdarray erfc( const mdarray &obj );`	相補誤差関数
`mdarray ceil( const mdarray &obj );`	引き数より小さくない最小の整数値
`mdarray floor( const mdarray &obj );`	引き数を越えない最大の整数値
`mdarray round( const mdarray &obj );`	最も近い整数値に丸める
`mdarray trunc( const mdarray &obj );`	0 に近い方の整数値に丸める
`mdarray fabs( const mdarray &obj );`	絶対値
`mdarray hypot( const mdarray &obj, float v );`	ユークリッド距離関数
`mdarray hypot( const mdarray &obj, double v );`	
`mdarray hypot( float v, const mdarray &obj );`	
`mdarray hypot( double v, const mdarray &obj );`	
`mdarray hypot( const mdarray &src0, const mdarray &src1 );`	
`mdarray pow( const mdarray &obj, float v );`	累乗
`mdarray pow( const mdarray &obj, double v );`	
`mdarray pow( float v, const mdarray &obj );`	
`mdarray pow( double v, const mdarray &obj );`	
`mdarray pow( const mdarray &src0, const mdarray &src1 );`	
`mdarray fmod( const mdarray &obj, float v );`	剰余
`mdarray fmod( const mdarray &obj, double v );`	
`mdarray fmod( float v, const mdarray &obj );`	
`mdarray fmod( double v, const mdarray &obj );`	
`mdarray fmod( const mdarray &src0, const mdarray &src1 );`	
`mdarray remainder( const mdarray &obj, float v );`	剰余
`mdarray remainder( const mdarray &obj, double v );`	
`mdarray remainder( float v, const mdarray &obj );`	
`mdarray remainder( double v, const mdarray &obj );`	
`mdarray remainder( const mdarray &src0, const mdarray &src1 );`	

表 25: 利用可能な数学関数一覧.

プロトタイプにある「mdarray クラス」は表 24に示したクラスの親クラスです. したがって, mdarray_double クラスなどのオブジェクトを「mdarray クラスの引数」へ与える事も可能ですし,「mdarray

クラスの返り値」を mdarray_double クラスなどのメンバ関数の引数で受ける事も可能です.
チュートリアルの §3.6.4 に数学関数を配列に対して使っている例があります. そちらもご覧ください.

12.3 メンバ関数一覧

表 26 はメンバ関数一覧です. 親クラスである mdarray で定義しているものも, 継承クラス (mdarray_double 等) で再定義・追加定義しているものも, すべてを記載しています.

	メンバ関数名	機能	動作モードサポート
§12.4.1	`[]`	指定された 1 要素値の参照 (1 次元)	
§12.4.2	`()`	指定された 1 要素値の参照 (1〜3 次元)	
§12.4.3	`=`	配列を代入 (属性もコピー)	
§12.4.4	`=`	スカラー値を代入	
§12.4.5	`+=`	自身へ配列を加算	◯
§12.4.6	`+=`	自身へスカラー値を加算	
§12.4.7	`-=`	自身から配列を減算	◯
§12.4.8	`-=`	自身からスカラー値を減算	
§12.4.9	`*=`	自身へ配列を乗算	◯
§12.4.10	`*=`	自身へスカラー値を乗算	
§12.4.11	`/=`	自身から配列を除算	◯
§12.4.12	`/=`	自身からスカラー値を除算	
§12.4.13	`+`	自身に配列を加算した結果を格納したオブジェクトを返す	
§12.4.14	`+`	自身にスカラー値を加算した結果を格納したオブジェクトを返す	
§12.4.15	`-`	自身から配列を減算した結果を格納したオブジェクトを返す	
§12.4.16	`-`	自身からスカラー値を減算した結果を格納したオブジェクトを返す	
§12.4.17	`*`	自身に配列を乗算した結果を格納したオブジェクトを返す	
§12.4.18	`*`	自身にスカラー値を乗算した結果を格納したオブジェクトを返す	
§12.4.19	`/`	自身から配列を除算した結果を格納したオブジェクトを返す	
§12.4.20	`/`	自身からスカラー値を除算した結果を格納したオブジェクトを返す	
§12.4.21	`==`	比較	
§12.4.22	`!=`	比較 (否定形)	
§12.5.1	`size_type()`	型を表す整数 (型種別)	
§12.5.2	`bytes()`	1 要素のバイト数	
§12.5.3	`dim_length()`	配列の次元数	
§12.5.4	`length()`	(各次元の) 要素の個数	
§12.5.5	`byte_length()`	(各次元の) 配列の総バイト長	
§12.5.6	`col_length()`	配列の列の長さ	
§12.5.7	`row_length()`	配列の行の長さ	
§12.5.8	`layer_length()`	配列のレイヤ数	

表 26: mdarray_*クラスで利用可能なメンバ関数一覧 (次ページに続く).

	メンバ関数名	機能	動作モードサポート
§12.5.9	at(), at_cs()	指定された 1 要素値の参照 (1〜3 次元)	
§12.5.10	dvalue()	double 型に変換した 1 要素の値	
§12.5.11	lvalue(), llvalue()	long 型，long long 型に変換した 1 要素の値	
§12.5.12	default_value() assign_default()	サイズ拡張時の初期値の取得・設定	
§12.5.13	auto_resize(), set_auto_resize()	リサイズモードの取得・設定	
§12.5.14	rounding() set_rounding()	四捨五入の可否の取得・設定	
§12.5.15	dprint()	オブジェクト情報を標準エラー出力へ出力 (ユーザプログラムのデバッグ用)	
§12.5.16	carray(), array_ptr()	指定要素のアドレスの取得	
§12.5.17	get_elements()	自身の配列をユーザ・バッファへコピー	
§12.5.18	put_elements()	ユーザ・バッファの配列を自身へコピー	
§12.5.19	getdata()	自身の配列をユーザ・バッファへコピー	
§12.5.20	putdata()	ユーザ・バッファの配列を自身へコピー	
§12.5.21	reverse_endian()	必要に応じてエンディアンを反転	
§12.5.22	init()	配列の初期化	
§12.5.23	assign()	1 要素へ値を代入	○
§12.5.24	put()	任意の要素位置へ値をセット	○
§12.5.25	swap()	要素間での値の入れ替え	
§12.5.26	move()	要素間での値のコピー	
§12.5.27	cpy()	要素間での値のコピー (自動拡張あり)	
§12.5.28	insert()	要素の挿入	
§12.5.29	crop()	要素の切り出し	
§12.5.30	erase()	要素の削除	
§12.5.31	resize()	配列の長さを変更	
§12.5.32	resizeby()	配列の長さを相対的に変更	
§12.5.33	increase_dim()	次元数の拡張	
§12.5.34	decrease_dim()	次元数の縮小	
§12.5.35	swap()	別オブジェクトとの内容の入れ替え	
§12.5.36	convert()	配列の全要素の値変換	
§12.5.37	ceil()	全要素値に対し小数部を切り上げる	
§12.5.38	floor()	全要素値に対し小数部を切り下げる	
§12.5.39	round()	全要素値に対し四捨五入	
§12.5.40	trunc()	全要素値に対し小数部を切り捨て	
§12.5.41	abs()	全要素値に対し絶対値をとる	
§12.5.42	compare()	配列オブジェクトの比較	

表 26: mdarray_*クラスで利用可能なメンバ関数一覧 (続き).

	メンバ関数名	機能	動作モードサポート
§12.5.43	`copy()`	配列を別オブジェクトへコピー	
§12.5.44	`copy()`	配列の一部を別オブジェクトへコピー (画像データ向き)	
§12.5.45	`cut()`	配列の全値を切り出す	
§12.5.46	`cut()`	配列の一部を切り出す (画像データ向き)	
§12.5.47	`clean()`	要素をデフォルト値でパディング (画像データ向き)	
§12.5.48	`fill()`	要素値の書換え (画像データ向き)	
§12.5.49	`add()`	要素値の加算 (画像データ向き)	
§12.5.50	`multiply()`	要素値の乗算 (画像データ向き)	
§12.5.51	`paste()`	配列オブジェクトの貼り付け (画像データ向き)	
§12.5.52	`add()`	配列オブジェクトの加算 (画像データ向き)	
§12.5.53	`subtract()`	配列オブジェクトの減算 (画像データ向き)	
§12.5.54	`multiply()`	配列オブジェクトの乗算 (画像データ向き)	
§12.5.55	`divide()`	配列オブジェクトの除算 (画像データ向き)	

表 26: mdarray_*クラスで利用可能なメンバ関数一覧 (続き).

12.4 演算子

リファレンス部では，解説対象であるクラス (つまり，mdarray クラスの継承クラス) の名前を「mdarray_type」と記載しています．実際には，それが「mdarray_double」「mdarray_int」等に置き換わります．

12.4.1 []

NAME

`[]` — 指定された 1 要素値の参照 (1 次元)

SYNOPSIS

```
mdarray_type &operator[]( ssize_t idx0 ); ................................. 1
const mdarray_type &operator[]( ssize_t idx0 ) const; ..................... 2
```

DESCRIPTION

`[]` で指定した要素値の参照を返します．`[]` の場合，与える事ができる引数は 1 つだけです．2 次元，3 次元の配列を扱う場合は，`()` を使用します (§12.4.2を参照してください)．

メンバ関数 1 は読み書き両用で `at()` と同じ動作を，メンバ関数 2 は読み取り専用で `at_cs()` と同じ動作をします．

メンバ関数 1 を使用して値を読み書きする際，動作モードが「自動リサイズモード」の場合，配列サイズは指定された要素番号に従って自動的にリサイズされます．動作モードが「手動リサイズモード」の場合，配列サイズを超えた要素へ値を代入しても，無視されるだけでエラーとはなりません．現在の配列サイズを超えた要素へ値の代入を行うには，予め resize() メンバ関数などで配列サイズを拡張する必要があります．resize() メンバ関数については §12.5.31を参照して下さい．

「手動リサイズモード」で配列サイズを超えた要素を読み取ると，浮動小数点値の場合は `NAN` が，整数値の場合は `INDEF_UCHAR`，`INDEF_INT16`，`INDEF_INT32`，`INDEF_INT64` のいずれかが返ります．INDEF の値については，各型における最小の整数値が設定されています．

メンバ関数 1・メンバ関数 2 のどちらが使われるかは，オブジェクトの「const」属性の有無で自動的に決まります.「const」属性が無い場合にはメンバ関数 1 が，有る場合にはメンバ関数 2 が自動的に選択されます．

`at()`，`at_cs()` についての詳細は §12.5.9を参照してください．

PARAMETER

[I] `idx0` 次元番号 0 の次元 (1 次元目) の要素番号

([I]：入力，[O]：出力)

RETURN VALUE

要素の値の参照

EXCEPTION

自動リサイズモードで内部バッファの確保に失敗した場合 (メンバ関数 1)

EXAMPLE

次のコードは，mdarray_llong クラス (配列の型は long long) のオブジェクト `my_mdarr` に値を代入しています．

```
mdarray_llong my_mdarr;
my_mdarr[0] = 17090000;
my_mdarr[1] = 9980000;
my_mdarr[2] = 9620000;
```

12.4.2 ()

NAME

() — 指定された 1 要素値の参照 (1〜3 次元)

SYNOPSIS

```
mdarray_type &operator()( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                          ssize_t idx2 = MDARRAY_INDEF);  .................. 1
const mdarray_type &operator()( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                                ssize_t idx2 = MDARRAY_INDEF) const;  ..... 2
```

DESCRIPTION

() で指定した要素値の参照を返します．引数は 3 つまでとれるので，3 次元までは要素番号をそのまま引数に与えるだけですが，4 次元以上の場合は，配列の次元を 3 次元に縮退させた場合の 3 次元目 (次元番号 2) の要素番号を idx2 に設定する事で，n 次元の配列を扱う事ができます．

メンバ関数 1 は読み書き両用で at() と同じ動作を，メンバ関数 2 は読み取り専用で at_cs() と同じ動作をします．

この関数を使用して値を読み書きする際，動作モードが「自動リサイズモード」の場合，配列サイズは指定された要素番号に従って自動的にリサイズされます．動作モードが「手動リサイズモード」の場合，配列サイズを超えた要素へ値を代入しても，無視されるだけでエラーとはなりません．現在の配列サイズを超えた要素へ値の代入を行うには，予め resize() メンバ関数などで配列サイズを拡張する必要があります．resize() メンバ関数については §12.5.31を参照して下さい．

指定された要素番号が負だった場合や，「手動リサイズモード」で配列サイズを超えた要素を読み取ると，浮動小数点値の場合は NAN が，整数値の場合は INDEF_UCHAR，INDEF_INT16，INDEF_INT32，INDEF_INT64 のいずれかが返ります．INDEF の値については，各型における最小の整数値が設定されています．

メンバ関数 1・メンバ関数 2 のどちらが使われるかは，オブジェクトの「const」属性の有無で自動的に決まります．「const」属性が無い場合にはメンバ関数 1 が，有る場合にはメンバ関数 2 が自動的に選択されます．

引数に，MDARRAY_INDEF を明示的に与えないでください．

at()，at_cs() についての詳細は §12.5.9を参照してください．

PARAMETER

[I] idx0 次元番号 0 の次元 (1 次元目) の要素番号
[I] idx1 次元番号 1 の次元 (2 次元目) の要素番号 (省略可)
[I] idx2 次元番号 2 の次元 (3 次元目) の要素番号 (省略可)
([I]：入力，[O]：出力)

RETURN VALUE

要素の値の参照

EXCEPTION

自動リサイズモードで内部バッファの確保に失敗した場合 (メンバ関数 1)

EXAMPLE

次のコードは，mdarray_double クラス (配列の型は double) のオブジェクト my_mdarr に値を代入しています．この場合，3×2×1 の 3 次元配列ができます．

```
mdarray_double my_mdarr;
my_mdarr(2,1,0) = 170.9;
```

12.4.3 =

NAME

= — 配列を代入 (属性もコピー)

SYNOPSIS

```
mdarray_type &operator=(const mdarray_type &obj);  ........................... 1
mdarray_type &operator=(const mdarray &obj);  ................................ 2
```

DESCRIPTION

obj の内容をすべて (配列長，リサイズ等の属性) を自身にコピーします．

メンバ関数 2 の引数は，親クラスである mdarray クラスです．したがって，自身とは異なる継承クラスのオブジェクトも指定できます．その場合，一旦全値を 0 に初期化し，+=演算子 (§12.4.5) を使って全値を加算し，リサイズ等の属性をコピーします．

PARAMETER

[I] obj mdarray(継承) クラスのオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，mdarray_llong クラスのオブジェクト my_mdarr に mdarray_long クラスのオブジェクト area_mdarr を代入し，その結果を標準出力します．length() に関しては，§12.5.4の解説を参照してください．

```
stdstreamio sio;

mdarray_llong my_mdarr;
```

```
    mdarray_long area_mdarr;
    area_mdarr[0] = 17090000;
    area_mdarr[1] = 9980000;
    area_mdarr[2] = 9620000;

    my_mdarr = area_mdarr;
    for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
        sio.printf("my_mdarr value[%zu]... [%lld]\n", i, my_mdarr[i]);
    }
```

実行結果

```
my_mdarr value[0]...  [17090000]
my_mdarr value[1]...  [9980000]
my_mdarr value[2]...  [9620000]
```

12.4.4 =

NAME

= — スカラー値を代入

SYNOPSIS

```
mdarray_type &operator=(double v);      1
mdarray_type &operator=(long long v);   2
mdarray_type &operator=(long v);        3
mdarray_type &operator=(int v);         4
```

DESCRIPTION

演算子の右側 (引数) で指定した数値 (スカラー値) を代入をします．自動的なサイズ拡張は行いません．予め要素数を設定し，バッファを確保する必要があります．

PARAMETER

[I] v スカラー値

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，1 次元配列を持つオブジェクト `mdarr` にスカラー値 125 を代入し，その結果を標準出力します．`length()` に関しては，§12.5.4の解説を参照してください．

```
    stdstreamio sio;
    mdarray_int my_mdarr(false, 2);

    my_mdarr = 125;
    for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
```

```
        sio.printf("my_mdarr value[%zu]... [%d]\n", i, my_mdarr[i]);
    }
```

実行結果

```
my_mdarr value[0]...  [125]
my_mdarr value[1]...  [125]
```

12.4.5 +=

NAME

+= — 自身へ配列を加算

SYNOPSIS

```
mdarray_type &operator+=(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray(継承) クラスのオブジェクトの配列を自身に加算します．引数は，親クラスである mdarray クラスですので，自身とは異なる継承クラスのオブジェクトも指定できます．その場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

「自動リサイズモード」が設定されている場合，各次元サイズに関して自身よりも `obj` の方が大きい場合，自動的にリサイズします．

PARAMETER

[I] `obj` mdarray(継承) クラスのオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

12.4.6 +=

NAME

+= — 自身へスカラー値を加算

SYNOPSIS

```
mdarray_type &operator+=(double v);
mdarray_type &operator+=(long long v);
mdarray_type &operator+=(long v);
mdarray_type &operator+=(int v);
```

DESCRIPTION

演算子の右側 (引数) で指定したスカラー値を自身の要素すべてに対して加算します．自身とは型が異なる引数の場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

PARAMETER

[I] v スカラー値
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，mdarray_int クラスのオブジェクト my_mdarr に 50 を加算し，その結果を標準出力します．length() に関しては，§12.5.4の解説を参照してください．

```
stdstreamio sio;

mdarray_int my_mdarr(false, 2);
my_mdarr = 25;
my_mdarr += 50;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%d]\n", i, my_mdarr[i]);
}
```

実行結果

```
my_mdarr value[0]...  [75]
my_mdarr value[1]...  [75]
```

12.4.7 -=

NAME

−= — 自身から配列を減算

SYNOPSIS

```
mdarray_type &operator-=(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray(継承) クラスのオブジェクトの配列を自身から減算します．引数は，親クラスである mdarray クラスですので，自身とは異なる継承クラスのオブジェクトも指定できます．その場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

「自動リサイズモード」が設定されている場合，各次元サイズに関して自身よりも obj の方が大きい場合，自動的にリサイズします．

PARAMETER

[I] obj mdarray(継承) クラスのオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，mdarray_long クラスのオブジェクト my_mdarr から mdarray_int クラスのオブジェクト subst_mdarr を減算し，その結果を標準出力します．length() に関しては，§12.5.4の解説を参照してください．

```
stdstreamio sio;

mdarray_long my_mdarr(false, 2);
my_mdarr = 100;

mdarray_int subst_mdarr(false, 2);
subst_mdarr[0] = 10;
subst_mdarr[1] = 20;

my_mdarr -= subst_mdarr;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%ld]\n", i, my_mdarr[i]);
}
```

実行結果

```
my_mdarr value[0]...  [90]
my_mdarr value[1]...  [80]
```

12.4.8 -=

NAME

−= — 自身からスカラー値を減算

SYNOPSIS

```
mdarray_type &operator-=(double v);
mdarray_type &operator-=(long long v);
mdarray_type &operator-=(long v);
mdarray_type &operator-=(int v);
```

DESCRIPTION

自身の要素すべてに対して，演算子の右側 (引数) で指定したスカラーで減算します．自身とは型が異なる引数の場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

PARAMETER

[I]　v　スカラー値
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

12.4.9 *=

NAME

*= — 自身へ配列を乗算

SYNOPSIS

```
mdarray_type &operator*=(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray(継承) クラスのオブジェクトの配列を自身に乗算します．引数は，親クラスである mdarray クラスですので，自身とは異なる継承クラスのオブジェクトも指定できます．その場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

「自動リサイズモード」が設定されている場合，各次元サイズに関して自身よりも `obj` の方が大きい場合，自動的にリサイズします．

PARAMETER

[I] `obj` mdarray(継承) クラスのオブジェクト

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

メモリ破壊を起こした場合

EXAMPLE

次のコードは，mdarray_long クラスのオブジェクト `my_mdarr` に mdarray_int クラス `mdarrPlus` を乗算し，その結果を標準出力します．`length()` に関しては，§12.5.4の解説を参照してください．

```
stdstreamio sio;

mdarray_long my_mdarr(false, 2);
my_mdarr = 50;

mdarray_int multi_mdarr;
multi_mdarr[0] = 10;
multi_mdarr[1] = 20;
multi_mdarr[2] = 30;

my_mdarr *= multi_mdarr;
```

```
    for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
        sio.printf("my_mdarr value[%zu]... [%ld]\n", i, my_mdarr[i]);
    }
```

実行結果

```
my_mdarr value[0]...  [500]
my_mdarr value[1]...  [1000]
```

12.4.10 *=

NAME

*= — 自身へスカラー値を乗算

SYNOPSIS

```
mdarray_type &operator*=(double v);
mdarray_type &operator*=(long long v);
mdarray_type &operator*=(long v);
mdarray_type &operator*=(int v);
```

DESCRIPTION

演算子の右側 (引数) で指定したスカラー値を自身の要素すべてに対して乗算します．自身とは型が異なる引数の場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

PARAMETER

[I] v スカラー値

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

12.4.11 /=

NAME

/= — 自身から配列を除算

SYNOPSIS

```
mdarray_type &operator/=(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray(継承) クラスのオブジェクトの配列を自身から除算します．引数は，親クラスである mdarray クラスですので，自身とは異なる継承クラスのオブジェクトも指定できます．その場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

「自動リサイズモード」が設定されている場合，各次元サイズに関して自身よりも obj の方が大きい場合，自動的にリサイズします．

PARAMETER

[I] `obj` mdarray(継承) クラスのオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，mdarray_long クラスのオブジェクト `my_mdarr` を mdarray_int クラスの `div_mdarr` で除算し，その結果を標準出力します．`length()` に関しては，§12.5.4の解説を参照してください．

```
stdstreamio sio;

mdarray_long my_mdarr(false, 2);
my_mdarr = 50;

mdarray_int div_mdarr;
div_mdarr[0] = 1;
div_mdarr[1] = 2;
div_mdarr[2] = 5;

my_mdarr /= div_mdarr;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%ld]\n", i, my_mdarr[i]);
}
```

実行結果

```
my_mdarr value[0]...  [50]
my_mdarr value[1]...  [25]
```

12.4.12 /=

NAME

/= — 自身からスカラー値を除算

SYNOPSIS

```
mdarray_type &operator/=(double v);
mdarray_type &operator/=(long long v);
mdarray_type &operator/=(long v);
mdarray_type &operator/=(int v);
```

DESCRIPTION

自身の要素すべてに対して，演算子の右側 (引数) で指定したスカラー値で除算します．自身とは型が異なる引数の場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

PARAMETER

[I] `v` スカラー値
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

12.4.13 +

NAME

+ — 自身に配列を加算した結果を格納したオブジェクトを返す

SYNOPSIS

```
mdarray operator+(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray(継承) クラスのオブジェクトの配列と，自身を加算した結果を格納したオブジェクトを作成して返します．引数は，親クラスである mdarray クラスですので，自身とは異なる継承クラスのオブジェクトも指定できます．その場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

返されるオブジェクトの動作モードや rounding 属性は，自身の場合と同じです．

PARAMETER

[I] `obj` mdarray(継承) クラスのオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

演算結果を格納したオブジェクト

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

§3.6.3に配列に対するオペレータの使用例があります．

12.4.14 +

NAME

+ — 自身にスカラー値を加算した結果を格納したオブジェクトを返す

SYNOPSIS

```
mdarray operator+(double v);
```

```
mdarray operator+(float v);
mdarray operator+(long long v);
mdarray operator+(long v);
mdarray operator+(int v);
```

DESCRIPTION

演算子の右側 (引数) で指定したスカラー値を，自身の要素すべてに対して加算した結果を格納したオブジェクトを作成して返します．自身とは型が異なる引数の場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

返されるオブジェクトの動作モードや rounding 属性は，自身の場合と同じです．

PARAMETER

[I] `v` スカラー値
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

12.4.15 -

NAME

– — 自身から配列を減算した結果を格納したオブジェクトを返す

SYNOPSIS

```
mdarray operator-(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray(継承) クラスのオブジェクトの配列を自身から減算した結果を格納したオブジェクトを作成して返します．引数は，親クラスである mdarray クラスですので，自身とは異なる継承クラスのオブジェクトも指定できます．その場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

返されるオブジェクトの動作モードや rounding 属性は，自身の場合と同じです．

PARAMETER

[I] `obj` mdarray(継承) クラスのオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

演算結果を格納したオブジェクト

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

§3.6.3に配列に対するオペレータの使用例があります．

12.4.16 -

NAME

− — 自身からスカラー値を減算した結果を格納したオブジェクトを返す

SYNOPSIS

```
mdarray operator-(double v);
mdarray operator-(float v);
mdarray operator-(long long v);
mdarray operator-(long v);
mdarray operator-(int v);
```

DESCRIPTION

演算子の右側 (引数) で指定したスカラー値を，自身の要素それぞれから減算した結果を格納したオブジェクトを作成して返します．自身とは型が異なる引数の場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

返されるオブジェクトの動作モードや rounding 属性は，自身の場合と同じです．

PARAMETER

[I] `v` スカラー値
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

12.4.17 *

NAME

∗ — 自身に配列を乗算した結果を格納したオブジェクトを返す

SYNOPSIS

```
mdarray operator*(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray(継承) クラスのオブジェクトの配列と，自身を乗算した結果を格納したオブジェクトを作成して返します．引数は，親クラスである mdarray クラスですので，自身とは異なる継承クラスのオブジェクトも指定できます．その場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

返されるオブジェクトの動作モードや rounding 属性は，自身の場合と同じです．

PARAMETER

[I] `obj` mdarray(継承) クラスのオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

演算結果を格納したオブジェクト

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

§3.6.3に配列に対するオペレータの使用例があります.

12.4.18 *

NAME

∗ — 自身にスカラー値を乗算した結果を格納したオブジェクトを返す

SYNOPSIS

```
mdarray operator*(double v);
mdarray operator*(float v);
mdarray operator*(long long v);
mdarray operator*(long v);
mdarray operator*(int v);
```

DESCRIPTION

演算子の右側 (引数) で指定したスカラー値を，自身の要素すべてに対して乗算した結果を格納したオブジェクトを作成して返します．自身とは型が異なる引数の場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます.

返されるオブジェクトの動作モードや rounding 属性は，自身の場合と同じです.

PARAMETER

[I] v スカラー値
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

12.4.19 /

NAME

/ — 自身から配列を除算した結果を格納したオブジェクトを返す

SYNOPSIS

```
mdarray operator/(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray(継承) クラスのオブジェクトの配列を自身から除算した結果を格納したオブジェクトを作成して返します．引数は，親クラスである mdarray クラスですので，自身とは異なる継承クラスのオブジェクトも指定できます．その場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

返されるオブジェクトの動作モードや rounding 属性は，自身の場合と同じです．

PARAMETER

[I] `obj` mdarray(継承) クラスのオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

演算結果を格納したオブジェクト

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

§3.6.3に配列に対するオペレータの使用例があります．

12.4.20 /

NAME

/ — 自身からスカラー値を除算した結果を格納したオブジェクトを返す

SYNOPSIS

```
mdarray operator/(double v);
mdarray operator/(float v);
mdarray operator/(long long v);
mdarray operator/(long v);
mdarray operator/(int v);
```

DESCRIPTION

演算子の右側 (引数) で指定したスカラー値を，自身の要素それぞれから除算した結果を格納したオブジェクトを作成して返します．自身とは型が異なる引数の場合，通常のスカラー演算の場合と同様の型変換処理が行なわれます．

返されるオブジェクトの動作モードや rounding 属性は，自身の場合と同じです．

PARAMETER

[I] `v` スカラー値
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

12.4.21 ==

NAME

== — 比較

SYNOPSIS

`bool operator==(const mdarray &obj) const;`

DESCRIPTION

演算子の右側 (引数) で指定した mdarray(継承) クラスのオブジェクトと自身とを比較します．`obj` と配列サイズ・要素の値が等しければ，`true` を返します．`obj` と配列サイズ・要素の値が等しくなけば，`false` を返します．

このメンバ関数は，内部で compare() メンバ関数 (§12.5.42) を使っています．

PARAMETER

[I] `obj` mdarray(継承) クラスのオブジェクト

([I]：入力，[O]：出力)

RETURN VALUE

true ： 配列サイズ，要素の値が一致した場合

false ： 配列サイズ，要素の値が不一致である場合

EXAMPLE

次のコードは，mdarray_uchar クラスのオブジェクト `my_mdarr` と mdarray_long クラスのオブジェクト `comp_mdarr` を比較し，その結果を標準出力します．

```
stdstreamio sio;

mdarray_uchar my_mdarr(false, 3);
my_mdarr = 20;

mdarray_long comp_mdarr;
comp_mdarr[0] = 20;
if (my_mdarr == comp_mdarr) {
    sio.printf("true\n");
} else {
    sio.printf("false\n");
}
```

実行結果

```
false
```

12.4.22 !=

NAME

!= — 比較 (否定形)

SYNOPSIS

```
bool operator!=(const mdarray &obj) const;
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray(継承) クラスのオブジェクトと自身とを比較 (否定形) します．`obj` と等しくなければ，`true` を返します．`obj` と等しければ，`false` を返します．

このメンバ関数は，内部で compare() メンバ関数 (§12.5.42) を使っています．

PARAMETER

[I] `obj` mdarray(継承) クラスのオブジェクト

([I]：入力，[O]：出力)

RETURN VALUE

true : 配列サイズ，要素の値が不一致である場合

false : 配列サイズ，要素の値が一致した場合

EXAMPLE

次のコードは，mdarray_uchar クラスのオブジェクト `my_mdarr` と mdarray_long クラスの `comp_mdarr` を比較し，その結果を標準出力します．

```
stdstreamio sio;

mdarray_uchar my_mdarr(false, 3);
my_mdarr = 20;

mdarray_long comp_mdarr;
comp_mdarr[0] = 20;
if (my_mdarr != comp_mdarr) {
    sio.printf("true\n");
} else {
    sio.printf("false\n");
}
```

実行結果

```
true
```

12.5 メンバ関数

リファレンス部では，解説対象であるクラス (つまり，mdarray クラスの継承クラス) の名前を「mdarray_type」と記載しています．実際には，それが「mdarray_double」「mdarray_int」等に置き換わります．

12.5.1 size_type()

NAME

size_type() — 型を表す整数 (型種別)

SYNOPSIS

```
ssize_t size_type() const;
```

DESCRIPTION

自身が持つ配列要素の型を表す整数 (型種別) を取得します.

このメンバで返される値は,「sli/size_types.h」の中で，次の表のとおり定義されています.

C 言語の型	定数名	実際の値	説明
float	FLOAT_ZT	-4	単精度浮動小数点型
double	DOUBLE_ZT	-8	倍精度浮動小数点型
fcomplex	FCOMPLEX_ZT	-7	単精度複素数型
dcomplex	DCOMPLEX_ZT	-15	倍精度複素数型
unsigned char	UCHAR_ZT	1	符号なし 1 バイト整数型
short	SHORT_ZT	処理系依存	符号付き整数型
int	INT_ZT	処理系依存	符号付き整数型
long	LONG_ZT	処理系依存	符号付き整数型
long long	LLONG_ZT	処理系依存	符号付き整数型
int16_t	INT16_ZT	2	符号付き 2 バイト整数型
int32_t	INT32_ZT	4	符号付き 4 バイト整数型
int64_t	INT64_ZT	8	符号付き 8 バイト整数型
size_t	SIZE_ZT	処理系依存	符号なし整数型
ssize_t	SSIZE_ZT	処理系依存	符号付き整数型
bool	BOOL_ZT	処理系依存	論理型
uintptr_t	UINTPTR_ZT	処理系依存	アドレス幅と同じサイズの符号なし整数型

利用可能なクラスについては，表 24をご覧ください.

型情報を扱う必要がある場合には，-4 のような実際の値をそのままコードに書くのではなく，表にある定数名を使用してください.

RETURN VALUE

型を表す整数

EXAMPLE

次のコードは，mdarray_int32 クラスのオブジェクト my_mdarr を作り，my_mdarr の size_type を標準出力します.

```
stdstreamio sio;

mdarray_int32 my_mdarr;
sio.printf("*** my_mdarr size_type... [%zd]\n", my_mdarr.size_type());
```

実行結果

```
*** my_mdarr size_type... [4]
```

12.5.2 bytes()

NAME

bytes() — 1 要素のバイト数

SYNOPSIS

```
size_t bytes() const;
```

DESCRIPTION

自身が持つ配列の 1 要素のバイト長を返します.

RETURN VALUE

1 要素のバイト長

EXAMPLE

次のコードは，mdarray_double クラスのオブジェクト my_mdarr を作成し，1 要素のバイト長を標準出力します.

```
stdstreamio sio;

mdarray_double my_mdarr;
sio.printf("*** my_mdarr bytes... [%zu]\n", my_mdarr.bytes());
```

実行結果

```
*** my_mdarr bytes... [8]
```

12.5.3 dim_length()

NAME

dim_length() — 配列の次元数

SYNOPSIS

```
size_t dim_length() const;
```

DESCRIPTION

自身が持つ配列の次元数を返します.

RETURN VALUE

配列の次元数

EXAMPLE

次のコードは，オブジェクト my_mdarr3dim がもつ配列の次元数を標準出力します.

```
stdstreamio sio;

mdarray_float my_mdarr3dim(false, 3, 4, 5);
sio.printf("*** my_mdarr3dim dim... [%zu]\n", my_mdarr3dim.dim_length());
```

実行結果

```
*** my_mdarr3dim dim... [3]
```

12.5.4 length()

NAME

length() — 要素の個数

SYNOPSIS

size_t length() const; 1
size_t length(size_t dim_index) const; 2

DESCRIPTION

自身が持つ配列要素の個数を返します．引数の指定がない場合は，全要素数 (次元 1 の個数 × 次元 2 の個数 × 次元 3 の個数 ×...) を返します．dim_index を引数に指定した場合は，次元番号が dim_index の次元の要素の個数を返します．dim_index の開始番号は 0 です．

PARAMETER

[I] dim_index 次元番号 (開始の値は 0)
([I]：入力，[O]：出力)

RETURN VALUE

要素の個数

EXAMPLE

次のコードは，オブジェクト my_mdarr3dim がもつ全要素数と次元番号 0 の次元の要素数を標準出力します．

```
stdstreamio sio;

mdarray_float my_mdarr3dim(false, 3, 4, 5);
sio.printf("*** my_mdarr3dim length... [%zu]\n", my_mdarr3dim.length());
sio.printf("*** my_mdarr3dim length 1dim... [%zu]\n", my_mdarr3dim.length(0));
```

実行結果

```
*** my_mdarr3dim length... [60]
*** my_mdarr3dim length 1dim... [3]
```

12.5.5 byte_length()

NAME

byte_length() — (各次元の) 配列の総バイト長

SYNOPSIS

size_t byte_length() const; 1
size_t byte_length(size_t dim_index) const; 2

DESCRIPTION

自身が持つ配列の総バイト長を返します．dim_index を引数に指定した場合は，次元番号が dim_index の次元の総バイト長を返します．

PARAMETER

[I] dim_index 次元番号 (開始の値は 0)
([I]：入力，[O]：出力)

RETURN VALUE

配列の総バイト長，または指定次元の総バイト長

EXAMPLE

次のコードは，3 次元配列 my_mdarr3dim がもつ配列の総バイト長と，次元番号が 2 の次元 (3 次元目) の総バイト長を標準出力します.

```
stdstreamio sio;

mdarray_float my_mdarr3dim(false, 3, 4, 5);
sio.printf("*** my_mdarr3dim byte_length... [%zu]\n",
            my_mdarr3dim.byte_length());
sio.printf("*** my_mdarr3dim byte_length 3dim... [%zu]\n",
            my_mdarr3dim.byte_length(2));
```

実行結果

```
*** mdarr3dim byte_length... [240]
*** mdarr3dim byte_length 3dim... [20]
```

12.5.6 col_length()

NAME

length() — 配列の列の長さ

SYNOPSIS

```
size_t col_length() const;
```

DESCRIPTION

自身が持つ配列の列の長さを返します.

RETURN VALUE

配列の列の長さ

EXAMPLE

次のコードは，3 次元配列 my_mdarr3dim がもつ列の長を標準出力します.

```
stdstreamio sio;

mdarray_float my_mdarr3dim(false, 3, 4, 5);
sio.printf("*** my_mdarr3dim col... [%zu]\n", my_mdarr3dim.col_length());
```

実行結果

```
*** my_mdarr3dim col... [3]
```

12.5.7 row_length()

NAME

row_length() — 配列の行の長さ

SYNOPSIS

```
size_t row_length() const;
```

DESCRIPTION

自身が持つ配列の行の長さを返します.

RETURN VALUE

配列の行の長さ

EXAMPLE

次のコードは，3 次元配列 my_mdarr3dim がもつ配列の行の長さを標準出力します.

```
stdstreamio sio;

mdarray_float my_mdarr3dim(false, 3, 4, 5);
sio.printf("*** my_mdarr3dim row... [%zu]\n", my_mdarr3dim.row_length());
```

実行結果

```
*** my_mdarr3dim row... [4]
```

12.5.8 layer_length()

NAME

layer_length() — 配列のレイヤ数

SYNOPSIS

```
size_t layer_length() const;
```

DESCRIPTION

自身が持つ配列のレイヤ数を返します．1 次元または 2 次元配列の場合は，1 を返します．3 次元以上の場合は，配列の次元を 3 次元に縮退させた場合の 3 次元目 (次元番号 2) の長さを返します.

RETURN VALUE

配列の次元数

EXAMPLE

次のコードは，3 次元配列 my_mdarr3dim がもつレイヤ数を標準出力します.

```
stdstreamio sio;

mdarray_float my_mdarr3dim(false, 3, 4, 5);
sio.printf("*** my_mdarr3dim layer... [%zu]\n", my_mdarr3dim.layer_length());
```

実行結果

```
*** my_mdarr3dim layer... [5]
```

12.5.9 at(), at_cs()

NAME

at(), at_cs() — 指定された 1 要素値の参照 (1〜3 次元)

SYNOPSIS

```
type &at( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
          ssize_t idx2 = MDARRAY_INDEF );  .................................. 1
const type &at( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                ssize_t idx2 = MDARRAY_INDEF ) const;  ...................... 2
const type &at_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                   ssize_t idx2 = MDARRAY_INDEF ) const;  .................. 3
```

DESCRIPTION

idx0，idx1，idx2 で指定された配列要素の値を設定・取得します．

メンバ関数 1 は読み書き両用で，メンバ関数 2，3 は読み取り専用です．

メンバ関数 1 を使用して値を読み書きする際，動作モードが「自動リサイズモード」の場合，配列サイズは指定された要素番号に従って自動的にリサイズされます．動作モードが「手動リサイズモード」の場合，配列サイズを超えた要素へ値を代入しても，無視されるだけでエラーとはなりません．現在の配列サイズを超えた要素へ値の代入を行うには，予め resize() メンバ関数などで配列サイズを拡張する必要があります．resize() メンバ関数については§12.5.31を参照して下さい．

「手動リサイズモード」で配列サイズを超えた要素を読み取ると，浮動小数点値の場合は NAN が，整数値の場合は INDEF_UCHAR，INDEF_INT16，INDEF_INT32，INDEF_INT64 のいずれかが返ります．INDEF の値については，各型における最小の整数値が設定されています．

at() メンバ関数の場合，メンバ関数 1・メンバ関数 2 のどちらが使われるかは，オブジェクトの「const」属性の有無で自動的に決まります．「const」属性が無い場合にはメンバ関数 1 が，有る場合にはメンバ関数 2 が自動的に選択されます．

引数に，MDARRAY_INDEF を明示的に与えないでください．

PARAMETER

[I] idx0 次元番号 0 の次元 (1 次元目) の要素番号
[I] idx1 次元番号 1 の次元 (2 次元目) の要素番号 (省略可)
[I] idx2 次元番号 2 の次元 (3 次元目) の要素番号 (省略可)
([I]：入力，[O]：出力)

RETURN VALUE

要素の値の参照

EXCEPTION

内部バッファの確保に失敗した場合 (メンバ関数 1，自動リサイズモード)
オブジェクト内の要素の型が，メンバ関数の戻り値の型より小さい場合

EXAMPLE

次のコードは，mdarray_float クラスのオブジェクト `my_fmdarr` の要素に at() メンバ関数を使用して値を設定した後，at() メンバ関数を使用して各要素の値を標準出力します.

```
stdstreamio sio;

mdarray_float my_fmdarr;

my_fmdarr.at(0) = 1000.1;
my_fmdarr.at(1) = 2000.2;
for ( size_t i = 0 ; i < my_fmdarr.length() ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%5.1f]\n", i, my_fmdarr.at(i));
}
```

実行結果

```
my_fmdarr value[0]...  [1000.1]
my_fmdarr value[1]...  [2000.2]
```

12.5.10 dvalue()

NAME

dvalue() — double 型に変換した 1 要素の値

SYNOPSIS

```
double dvalue( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
               ssize_t idx2 = MDARRAY_INDEF ) const;
```

DESCRIPTION

自身が持つ配列の要素値を double 型に変換して返します．配列サイズを超えた要素を指定した場合，`NAN` を返します.

引数に，`MDARRAY_INDEF` を明示的に与えないでください.

PARAMETER

[I]	`idx0`	次元番号 0 の次元 (1 次元目) の要素番号
[I]	`idx1`	次元番号 1 の次元 (2 次元目) の要素番号 (省略可)
[I]	`idx2`	次元番号 2 の次元 (3 次元目) の要素番号 (省略可)

([I]：入力，[O]：出力)

RETURN VALUE

double 型に変換した要素の値	:	正常終了
`NAN`(エラー)	:	要素の型がサポートされない型の場合 配列サイズを超えた要素を指定した場合

EXAMPLE

次のコードは，mdarray_float クラスのオブジェクト `my_mdarry` に値を設定した後，要素の値を double 型で取得し，標準出力します.

```
stdstreamio sio;

mdarray_float my_mdarry;
my_mdarry[0] = 123.456;
sio.printf("my_mdarry dvalue... [%6.3f]\n", my_mdarry.dvalue(0));
```

実行結果

```
my_mdarry dvalue...  [123.456]
```

12.5.11 lvalue(), llvalue()

NAME

lvalue(), llvalue() — long 型または long long 型に変換した 1 要素の値

SYNOPSIS

```
long lvalue( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
             ssize_t idx2 = MDARRAY_INDEF ) const;  ......................... 1
long long llvalue( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                   ssize_t idx2 = MDARRAY_INDEF ) const;  .................. 2
```

DESCRIPTION

自身が持つ配列の要素値を long 型，または long long 型に変換して返します．

自身の型が浮動小数点の場合，デフォルトでは小数点以下は切り捨てられます．小数点以下を四捨五入したい場合は，予め set_rounding() メンバ関数を使用して，四捨五入を行う設定にします．set_rounding() メンバ関数については §12.5.14を参照下さい．

配列サイズを超えた要素を指定した場合，それぞれ INDEF_LONG，INDEF_LLONG が返ります．INDEF の値については，各型における最小の整数値が設定されています．

引数に，MDARRAY_INDEF を明示的に与えないでください．

PARAMETER

[I] idx0 次元番号 0 の次元 (1 次元目) の要素番号
[I] idx1 次元番号 1 の次元 (2 次元目) の要素番号 (省略可)
[I] idx2 次元番号 2 の次元 (3 次元目) の要素番号 (省略可)
([I]：入力，[O]：出力)

RETURN VALUE

long 型または long long 型に変換した要素の値	:	正常終了
INDEF_LONG または INDEF_LLONG	:	要素の型がサポートされない型の場合 配列サイズを超えた要素を指定した場合

EXAMPLE

次のコードは，mdarray_float クラスのオブジェクト my_mdarry に値を設定した後，要素の値を long 型と long long 型で取得し標準出力します．

```
stdstreamio sio;
```

```
mdarray_float my_mdarry;
my_mdarry[0] = 123.556;
sio.printf("my_mdarry lvalue... [%ld]\n", my_mdarry.lvalue(0));
my_mdarry.set_rounding(true);
sio.printf("my_mdarry llvalue... [%lld]\n", my_mdarry.llvalue(0));
```

実行結果

```
my_mdarry lvalue...  [123]
my_mdarry llvalue...  [124]
```

12.5.12 default_value(), assign_default()

NAME

default_value(), assign_default() — サイズ拡張時の初期値の取得・設定

SYNOPSIS

`type default_value() const;` 1
`mdarray_type &assign_default( type value );` 2

DESCRIPTION

メンバ関数 1 は，自身が持つサイズ拡張時の初期値を取得します.

メンバ関数 2 は，配列サイズ拡張時の初期値を設定します．設定された値は既存の要素には作用せず，サイズ拡張時に有効となります.

RETURN VALUE

メンバ関数 1 は，サイズ拡張時の初期値が返ります.

メンバ関数 2 は，自身の参照を返します.

EXCEPTION

内部バッファの確保に失敗した場合 (メンバ関数 2)

EXAMPLE

次のコードは，mdarray_float クラスのオブジェクト `my_mdarr` のサイズ拡張時の初期値を標準出力します.

```
stdstreamio sio;

mdarray_float my_mdarr;
my_mdarr.assign_default(50);
sio.printf("*** my_mdarr defval... [%f]\n",
           my_mdarr.default_value());
```

実行結果

```
*** my_mdarr defval... [50.000000]
```

12.5.13 auto_resize(), set_auto_resize()

NAME

auto_resize(), set_auto_resize() — リサイズモードの取得・設定

SYNOPSIS

```
bool auto_resize() const; ...................................................... 1
mdarray_type &set_auto_resize( bool tf ); ................................... 2
```

DESCRIPTION

リサイズモードを真 (true)，または偽 (false) で取得 (メンバ関数 1)・設定 (メンバ関数 2) します.

動作モードが「自動リサイズモード」の場合は true(=1)，動作モードが「手動リサイズモード」の場合は false(=0) です.

動作モードが機能するメンバ関数については，表 26の「動作モードサポート」のカラムをご覧ください.

RETURN VALUE

メンバ関数 1 は，動作モードを返します (自動リサイズモードの場合は true).

メンバ関数 2 は，自身の参照を返します.

EXAMPLE

次のコードは，`mdarr0dim` を自動リサイズモード，`mdarr3dim` を手動リサイズモードで定義した後，各配列のリサイズモードを標準出力します.

```
stdstreamio sio;

mdarray_float my_mdarr0dim;
sio.printf("*** my_mdarr0dim auto_resize... [%d]\n",
            (int)(my_mdarr0dim.auto_resize()));
mdarray_float my_mdarr3dim(false, 3, 4, 5);
sio.printf("*** my_mdarr3dim auto_resize... [%d]\n",
            (int)(my_mdarr3dim.auto_resize()));
```

実行結果

```
*** my_mdarr0dim auto\_resize... [1]
*** my_mdarr3dim auto\_resize... [0]
```

12.5.14 rounding(), set_rounding()

NAME

rounding(), set_rounding() — 四捨五入の可否の取得・設定

SYNOPSIS

```
bool rounding() const; ......................................................... 1
mdarray_type &set_rounding( bool tf ); ........................................ 2
```

DESCRIPTION

メンバ関数 2 は，いくつかの高レベルメンバ関数において浮動小数点数を整数に変換する時に，四捨五入を行うか否を設定します．メンバ関数 1 の返り値は，四捨五入するように設定されている場合は真 (true)，四捨五入しないように設定されている場合は偽 (false) となります．

オブジェクト生成時の初期状態では，四捨五入しないように設定されています．

「=」演算子または init() メンバ関数を使用して，オブジェクトのコピーを行なった場合，四捨五入可否の属性も，コピーされます．init() メンバ関数については §12.5.22を参照下さい．

四捨五入の属性が機能するメンバ関数は次のとおりです:

lvalue()，llvalue() メンバ関数 (§12.5.11)，assign_default() メンバ関数 (§12.5.12)，assign() メンバ関数 (§12.5.23)，画像向きメンバ関数全般．

RETURN VALUE

メンバ関数 1 は，四捨五入動作の属性を返します (四捨五入するように設定されている場合は true)．

メンバ関数 2 は，自身の参照を返します．

EXAMPLE

次のコードは，mdarray_llong クラスのオブジェクト `my_mdarr` を作り，四捨五入の設定前後で，同じ実数値を代入しています．確認の為，代入した値を標準出力します．

```
stdstreamio sio;

mdarray_llong my_mdarr;
my_mdarr.assign(1.618, 0);
sio.printf("my_mdarr value[0]... [%lld]\n", my_mdarr[0]);

my_mdarr.set_rounding(true);
my_mdarr.assign(1.618, 1);
sio.printf("my_mdarr value[1]... [%lld]\n", my_mdarr[1]);
```

実行結果

```
my_imdarr value[0]...  [1]
my_imdarr value[1]...  [2]
```

12.5.15 dprint()

NAME

dprint() — オブジェクト情報を標準エラー出力へ出力 (ユーザのデバッグ用)

SYNOPSIS

```
void dprint() const;
```

DESCRIPTION

自身のオブジェクト情報を，標準エラー出力へ出力します．

ユーザ・プログラムのデバッグを目的としたメンバ関数です．

EXAMPLE

次のコードは，オブジェクト my_array の情報を標準エラー出力に出力します．[] にオブジェクトのアドレスが表示されていますが，これは実行環境により異なります．

```
mdarray_int my_array(false, 3,2,1);
my_array(2,0,0) = 100;
my_array(0,1,0) = 200;
my_array.dprint();
```

実行結果

```
sli::mdarray[obj=0x7fbffff630, sz_type=4, dim=(3,2,1)] = {
 { { 0,0,100 },
   { 200,0,0 } }
}
```

12.5.16 carray(), array_ptr()

NAME

carray(), array_ptr() — 指定要素のアドレスの取得

SYNOPSIS

```
const type *carray() const;  .............................................. 1
const type *carray( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                    ssize_t idx2 = MDARRAY_INDEF ) const;  ................. 2
type *array_ptr();  ....................................................... 3
type *array_ptr( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                 ssize_t idx2 = MDARRAY_INDEF );  ......................... 4
const type *array_ptr() const;  ........................................... 5
const type *array_ptr( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                       ssize_t idx2 = MDARRAY_INDEF ) const;  ............. 6
const type *array_ptr_cs() const;  ........................................ 7
const type *array_ptr_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                          ssize_t idx2 = MDARRAY_INDEF ) const;  .......... 8
```

DESCRIPTION

自身が持つ配列の指定要素のアドレスを取得します．メンバ関数 1，2，5〜8 は，読み取り専用のアドレスを取得します．

array_ptr() メンバ関数の場合,「const」属性なしのメンバ関数 3,4・「const」属性ありのメンバ関数 5,6 のどちらが使われるかは，オブジェクトの「const」属性の有無で自動的に決まります．「const」属性が無い場合にはメンバ関数 3,4 が，有る場合にはメンバ関数 5,6 が自動的に選択されます．

引数に，MDARRAY_INDEF を明示的に与えないでください．

PARAMETER

[I] idx0 次元番号 0 の次元 (1 次元目) の要素番号
[I] idx1 次元番号 1 の次元 (2 次元目) の要素番号 (省略可)
[I] idx2 次元番号 2 の次元 (3 次元目) の要素番号 (省略可)
([I]：入力，[O]：出力)

RETURN VALUE

指定要素のアドレス

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_fmdarr の (0,1) での値のアドレスを取得し，値を標準出力します．

```
stdstreamio sio;
mdarray_float my_fmdarr(false, 2,2);
my_fmdarr(0,0) = 1000;
my_fmdarr(1,0) = 2000;
my_fmdarr(0,1) = 3000;
my_fmdarr(1,1) = 4000;

const float *mycarray_ptr = my_fmdarr.carray(0, 1);
sio.printf("*** my_fmdarr carray[0] ---> [%f] *** \n",  mycarray_ptr[0]);
```

実行結果

```
*** my_fmdarr carray[0] ---> [3000.000000] ***
```

12.5.17 get_elements()

NAME

get_elements() — 自身の配列をユーザ・バッファへコピー

SYNOPSIS

```
ssize_t get_elements( type *dest_buf, size_t elem_size,
                      ssize_t idx0 = 0, ssize_t idx1 = MDARRAY_INDEF,
                      ssize_t idx2 = MDARRAY_INDEF ) const;
```

DESCRIPTION

自身の配列の内容を，dest_buf で指定されたユーザ・バッファへコピーします．バッファの大きさ elem_size は，要素の個数で与えます．idx，idx1，idx2 でコピー元を指定します．

引数に，MDARRAY_INDEF を明示的に与えないでください．

PARAMETER

[O] dest_buf ユーザ・バッファのアドレス
[I] elem_size コピーする要素の個数
[I] idx0 次元番号 0 の次元 (1 次元目) の要素番号 (コピー元，省略可)
[I] idx1 次元番号 1 の次元 (2 次元目) の要素番号 (コピー元，省略可)
[I] idx2 次元番号 2 の次元 (3 次元目) の要素番号 (コピー元，省略可)
([I]：入力，[O]：出力)

RETURN VALUE

ユーザのバッファ長が十分な場合にコピーされる要素数

EXCEPTION

メモリ破壊を起こした場合

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_fmdarr の内容をユーザバッファmyfloat へ取り出します．確認の為，myfloat の値を標準出力します．

```
stdstreamio sio;

float my_data[] = {1000, 2000, 3000, 4000};
mdarray_float my_fmdarr(false, 2,2, my_data);

float myfloat[4];
my_fmdarr.get_elements(myfloat, sizeof(myfloat)/sizeof(float));
for ( int i = 0 ; i < sizeof(myfloat)/sizeof(float) ; i++ ) {
    sio.printf("myfloat value[%d]... [%f]\n", i, myfloat[i]);
}
```

実行結果

```
myfloat_ptr value[0]...  [1000.000000]
myfloat_ptr value[1]...  [2000.000000]
myfloat_ptr value[2]...  [3000.000000]
myfloat_ptr value[3]...  [4000.000000]
```

12.5.18 put_elements()

NAME

put_elements() — ユーザ・バッファの配列を自身へコピー

SYNOPSIS

```
ssize_t put_elements( const type *src_buf, size_t elem_size, ssize_t idx0 = 0,
                      ssize_t idx1 = MDARRAY_INDEF, ssize_t idx2 = MDARRAY_INDEF );
```

DESCRIPTION

src_buf で指定されたユーザ・バッファの内容を自身の配列へコピーします．バッファの大きさ elem_size は，要素の個数で与えます．idx0，idx1，idx2 でコピー先を指定します．

引数に，MDARRAY_INDEF を明示的に与えないでください．

PARAMETER

[I]	src_buf	ユーザ・バッファのアドレス
[I]	elem_size	コピーする要素の個数
[I]	idx0	次元番号 0 の次元 (1 次元目) の要素番号 (コピー先，省略可)
[I]	idx1	次元番号 1 の次元 (2 次元目) の要素番号 (コピー先，省略可)
[I]	idx2	次元番号 2 の次元 (3 次元目) の要素番号 (コピー先，省略可)

([I] : 入力，[O] : 出力)

RETURN VALUE

ユーザのバッファ長が十分な場合にコピーされる要素数

EXCEPTION

メモリ破壊を起こした場合

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_fmdarr へユーザ・バッファmy_float の内容を書き込みます．確認の為，my_fmdarr の要素の値を標準出力します．

```
stdstreamio sio;

mdarray_float my_fmdarr(false, 2,2);

float my_float[] = {1000, 2000, 3000, 4000};
my_fmdarr.put_elements(my_float, sizeof(my_float)/sizeof(float));

for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
  for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
               i, j, my_fmdarr(i, j));
  }
}
```

実行結果

```
my_fmdarr value(0,0)...  [1000.000000]
my_fmdarr value(1,0)...  [2000.000000]
my_fmdarr value(0,1)...  [3000.000000]
my_fmdarr value(1,1)...  [4000.000000]
```

12.5.19 getdata()

NAME

getdata() — 自身の配列をユーザ・バッファへコピー

SYNOPSIS

```
ssize_t getdata( void *dest_buf, size_t buf_size, ssize_t idx0 = 0,
                 ssize_t idx1 = MDARRAY_INDEF, ssize_t idx2 = MDARRAY_INDEF ) const;
```

DESCRIPTION

自身の配列の内容を，dest_buf で指定されたユーザ・バッファへコピーします．バッファの大きさ buf_size は，バイト単位で与えます．idx，idx1，idx2 でコピー元を指定します．

引数に，MDARRAY_INDEF を明示的に与えないでください．

PARAMETER

[O]	`dest_buf`	ユーザ・バッファのアドレス
[I]	`buf_size`	バッファサイズ (バイト単位)
[I]	`idx0`	次元番号 0 の次元 (1 次元目) の要素番号 (コピー元，省略可)
[I]	`idx1`	次元番号 1 の次元 (2 次元目) の要素番号 (コピー元，省略可)
[I]	`idx2`	次元番号 2 の次元 (3 次元目) の要素番号 (コピー元，省略可)

([I]：入力，[O]：出力)

RETURN VALUE

ユーザのバッファ長 (buf_size) が十分な場合にコピーされるバイトサイズ

EXCEPTION

メモリ破壊を起こした場合

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト `my_fmdarr` の内容をユーザバッファ`myfloat` へ取り出します．確認の為，`myfloat` の値を標準出力します．

```
stdstreamio sio;

mdarray_float my_fmdarr(false, 2,2);
my_fmdarr(0,0) = 1000;
my_fmdarr(1,0) = 2000;
my_fmdarr(0,1) = 3000;
my_fmdarr(1,1) = 4000;

float myfloat[4];
my_fmdarr.getdata((void *)myfloat, sizeof(myfloat));
for ( int i = 0 ; i < sizeof(myfloat)/sizeof(float) ; i++ ) {
    sio.printf("myfloat value[%d]... [%f]\n", i, myfloat[i]);
}
```

実行結果

```
myfloat value[0]...  [1000.000000]
myfloat value[1]...  [2000.000000]
myfloat value[2]...  [3000.000000]
myfloat value[3]...  [4000.000000]
```

12.5.20 putdata()

NAME

putdata() — ユーザ・バッファの配列を自身へコピー

SYNOPSIS

```
ssize_t putdata( const void *src_buf, size_t buf_size, ssize_t idx0 = 0,
                 ssize_t idx1 = MDARRAY_INDEF, ssize_t idx2 = MDARRAY_INDEF );
```

DESCRIPTION

src_buf で指定されたユーザ・バッファの内容を自身の配列へコピーします．バッファの大きさ buf_size は，バイト単位で与えます．idx0，idx1，idx2 でコピー先を指定します．

引数に，MDARRAY_INDEF を明示的に与えないでください．

PARAMETER

[I]	src_buf	ユーザ・バッファのアドレス
[I]	buf_size	ユーザ・バッファのサイズ (バイト単位)
[I]	idx0	次元番号 0 の次元 (1 次元目) の要素番号 (コピー先，省略可)
[I]	idx1	次元番号 1 の次元 (2 次元目) の要素番号 (コピー先，省略可)
[I]	idx2	次元番号 2 の次元 (3 次元目) の要素番号 (コピー先，省略可)

([I]：入力，[O]：出力)

RETURN VALUE

ユーザのバッファ長 (buf_size) が十分な場合にコピーされるバイトサイズ

EXCEPTION

メモリ破壊を起こした場合

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_fmdarr へユーザ・バッファmy_float の内容を書き込みます．確認の為，my_fmdarr の要素の値を標準出力します．

```
stdstreamio sio;

mdarray_float my_fmdarr(false, 2,2);

float my_float[] = {1000, 2000, 3000, 4000};
my_fmdarr.putdata((const void *)my_float, sizeof(my_float));

for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
  for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
               i, j, my_fmdarr(i, j));
  }
}
```

実行結果

```
my_fmdarr value(0,0)...  [1000.000000]
my_fmdarr value(1,0)...  [2000.000000]
my_fmdarr value(0,1)...  [3000.000000]
my_fmdarr value(1,1)...  [4000.000000]
```

12.5.21 reverse_endian()

NAME

reverse_endian() — 必要に応じてエンディアンを反転

SYNOPSIS

```
mdarray_type &reverse_endian( bool is_little_endian );  ..................... 1
mdarray_type &reverse_endian( bool is_little_endian,
                              size_t begin, size_t length );  ............. 2
```

DESCRIPTION

このメンバ関数は，自身の配列をバイナリデータとしてファイルに保存したい時，あるいはファイルのバイナリデータを自身の配列に取り込みたい時に使います．

ファイルにデータを保存したい時は，このメンバ関数を呼び出してファイル保存に適したエンディアンに変換し，carray() メンバ関数 (§12.5.16) などで取得したアドレスをストリーム書き込み用の関数に与えて内容を書き込んだ後，再度このメンバ関数を呼び出して，エンディアンを元に戻します．

ファイルからデータを読み込みたい時は，array_ptr() メンバ関数 (§12.5.16) などで取得したアドレスをストリーム読み取り用の関数に与えて内容を読み込んだ後，このメンバ関数を呼び出して処理系に適したエンディアンに変換します．

上記のいずれの場合も，ファイルに保存されるべきデータがビッグエンディアンならば，第 1 引数に `false` をセットしまず (リトルエンディアンなら `true` です)．

このメンバ関数は，処理系によって使い分けが必要とならないように作られています．例えば，ファイルにビッグエンディアンのデータを保存したいので，`is_little_endian` に `false` を指定し，このメンバ関数を呼び出したとします．この時，マシンがビッグエンディアンであれば，実際には反転処理は行われません (マシンがリトルエンディアンであれば，反転処理が行われます)．次に，オブジェクト内のバイナリデータをそのままファイルに保存すれば，指定したバイトオーダーのバイナリファイルができます．その後，再度同じ引数でこのメンバ関数を呼び出して，エンディアンが反転されている場合は元に戻す処理を行います．

従って，ファイルへの保存に際しては，このメンバ関数は同じ引数で 2 回呼び出す事が前提です．

なお，引数 `begin` と `length` とを設定する事で，配列の一部分だけのエンディアン変換を行なう事もできます．

PARAMETER

[I]	`is_little_endian`	1 回目の処理後のメモリ上の値が little endian なら true をセット
[I]	`begin`	部分的に変換する場合の配列要素の開始番号 (先頭は 0)
[I]	`length`	部分的に変換する場合の配列要素の長さ

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト `my_mdarr` の内容をビッグエンディアンでバイナリファイル出力します．

```
stdstreamio sio;

mdarray_int my_mdarr(false, 2,2);
```

```
my_mdarr(0,0) = 10;
my_mdarr(1,0) = 20;
my_mdarr(0,1) = 30;
my_mdarr(1,1) = 40;

my_mdarr.reverse_endian(false);
const void *mydata_ptr = my_mdarr.data_ptr();

if ( fio.openf("w", "%s", "binary.dat") < 0 ) {
  //エラー処理
}
if ( fio.write(mydata_ptr, my_mdarr.byte_length()) < 0 ) {
  //エラー処理
}
my_mdarr.reverse_endian(false);

fio.close();
```

実行結果

binary.dat ファイルの内容：

```
"00 00 00 0A"
"00 00 00 14"
"00 00 00 1E"
"00 00 00 28"
```

§3.6.10にエンディアン変換の説明があります.

12.5.22 init()

NAME

init() — 配列の初期化

SYNOPSIS

```
mdarray_type &init(); ........................................................ 1
mdarray_type &init( bool auto_resize ); ...................................... 2
mdarray_type &init( bool auto_resize,
                    const size_t naxisx[], size_t ndim ); .................... 3
mdarray_type &init( bool auto_resize, size_t naxis0 ); ....................... 4
mdarray_type &init( bool auto_resize, size_t naxis0, size_t naxis1 ); ........ 5
mdarray_type &init( bool auto_resize, size_t naxis0, size_t naxis1,
                    size_t naxis2 ); ......................................... 6
mdarray_type &init( bool auto_resize, size_t naxis0,
                    const type vals[] ); ..................................... 7
mdarray_type &init( bool auto_resize, size_t naxis0, size_t naxis1,
```

```
                          const type vals[] );  ................................... 8
    mdarray_type &init( bool auto_resize, size_t naxis0, size_t naxis1,
                          const type *const vals[] );  ............................ 9
    mdarray_type &init( bool auto_resize, size_t naxis0, size_t naxis1,
                          size_t naxis2, const type vals[] );  .................. 10
    mdarray_type &init( bool auto_resize, size_t naxis0, size_t naxis1,
                          size_t naxis2, const type *const *const vals[] );  .. 11
    mdarray_type &init( const mdarray_type &obj );  ............................ 12
```


DESCRIPTION

自身の配列を初期化します.

メンバ関数 1 は，配列サイズ 0 としてオブジェクトを初期化します．動作モードは「自動リサイズモード」になります.

メンバ関数 2〜11 は，引数 auto_resize に動作モードを与え，その次の引数から配列のサイズ，要素の初期化のための配列のアドレスを与えます.

メンバ関数 12 は，obj の配列の内容や属性等すべてを自身にコピーします.

メンバ関数 1〜11 に関しては，オブジェクト作成時の場合 (コンストラクタ) と全く同じように引数を与えます．オブジェクト作成時の引数と動作モードについては，§12.1 をご覧ください.

PARAMETER

[I]	auto_resize	動作モード.「自動リサイズモード」の場合は true
[I]	ndim	配列次元数
[I]	naxisx[]	各次元の要素数
[I]	naxis0	次元番号 0 の次元 (1 次元目) の要素数
[I]	naxis1	次元番号 1 の次元 (2 次元目) の要素数
[I]	naxis2	次元番号 2 の次元 (3 次元目) の要素数
[I]	vals	コピー元となる配列のアドレス，またはポインタ配列のアドレス
[I]	obj	コピー元となるオブジェクト

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，2×3 の配列を持つオブジェクト my_mdarr を作り，各次元の長さを標準出力します.

```
stdstreamio sio;

mdarray_int my_mdarr;
my_mdarr.init(false, 2,3);
sio.printf("*** my_mdarr 0 dim length ====> [%zu] *** \n",
```

```
                my_mdarr.length(0));
    sio.printf("*** my_mdarr 1st dim length ====> [%zu] *** \n",
                my_mdarr.length(1));
```

実行結果

```
*** my_mdarr 0 dim length ====> [2] ***
*** my_mdarr 1st dim length ====> [3] ***
```

12.5.23 assign()

NAME

assign() — 1 要素へ値を代入

SYNOPSIS

```
mdarray_type &assign( double value, ssize_t idx0,
                      ssize_t idx1 = MDARRAY_INDEF, ssize_t idx2 = MDARRAY_INDEF );
```

DESCRIPTION

自身の配列の，idx$n$ で指定された 1 要素に値を設定します．

浮動小数点値を整数型の要素に代入する場合，デフォルトでは小数点以下は切り捨てます．小数点以下を四捨五入したい場合は，予め set_rounding() メンバ関数を使用して，オブジェクトを四捨五入を行う設定にします．set_rounding() メンバ関数については §12.5.14を参照して下さい．

動作モードが自動リサイズモードの場合，指定された要素番号に従って配列サイズが自動的にリサイズされます．

動作モードが手動リサイズモードの場合，配列サイズを超えた要素へ値を代入しても，無視されるだけでエラーとはなりません．配列サイズを超えた要素へ値の代入を行うには，予め resize() メンバ関数でサイズを拡張する必要があります．resize() メンバ関数については §12.5.31を参照下さい．

引数に，MDARRAY_INDEF を明示的に与えないでください．

PARAMETER

[I]	value	double 型の値
[I]	idx0	次元番号 0 の次元 (1 次元目) の要素番号
[I]	idx1	次元番号 1 の次元 (2 次元目) の要素番号 (省略可)
[I]	idx2	次元番号 2 の次元 (3 次元目) の要素番号 (省略可)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合 (自動リサイズモードの場合)

EXAMPLE

次のコードは，mdarray_float クラスのオブジェクト my_mdarr の 1 要素に値を設定し，各要素の値を標準出力します．

```
stdstreamio sio;

mdarray_float my_mdarr;
my_mdarr.assign(200.0, 1);
for ( size_t i = 0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr lvalue[%zu]... [%ld]\n", i, my_mdarr.lvalue(i));
}
```

実行結果

```
my_mdarr lvalue[0]...  [0]
my_mdarr lvalue[1]...  [200]
```

12.5.24 put()

NAME

put() — 任意の要素位置へ値をセット

SYNOPSIS

```
mdarray_type &put( type value, ssize_t idx, size_t len );  .................. 1
mdarray_type &put( type value,
                   size_t dim_index, ssize_t idx, size_t len );  ........... 2
```

DESCRIPTION

自身が持つ配列の要素番号 `idx` に，値 `value` を len 個書き込みます．なお，要素番号および次元番号は 0 から始まる数値です．

`idx` と `len` は任意の値を取る事ができます.「自動リサイズモード」の場合，引数の指定に対してオブジェクト内の配列長が不足している場合は，自動的に配列のリサイズを行ないます．この時，追加した要素のうち値が書き込まれない部分は，デフォルト値でパディングします.「手動リサイズモード」の場合は，`idx`，`len` で指定された部分のうち，配列サイズを越える部分については処理が行なわれません．

メンバ関数 1 は，値 `value` を要素番号 `idx` から `len` 個に書き込みます．

メンバ関数 2 は，値 `value` をオブジェクト内の配列の次元番号 `dim_index` の次元の要素番号 `idx` から `len` 個に書き込みます．`dim_index` が 1 以上の場合，下位次元の全要素に書き込まれます．

PARAMETER

[I]	`value`	セットする値
[I]	`idx`	要素番号
[I]	`len`	要素の個数
[I]	`dim_index`	次元番号

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合 (自動リサイズモードの場合)

EXAMPLE

次のコードは，1 次元の配列を持つオブジェクト my_smdarr の 2 番目の要素から 2 つに値を設定し，要素の値を標準出力します.

```
stdstreamio sio;

mdarray_short my_smdarr(false, 3);
my_smdarr.put(12, 1,2);
for ( size_t i = 0 ; i < my_smdarr.length() ; i++ ) {
    sio.printf("my_smdarr value[%zu]... [%hd]\n", i, my_smdarr[i]);
}
```

実行結果

```
my_smdarr value[0]...  [0]
my_smdarr value[1]...  [12]
my_smdarr value[2]...  [12]
```

12.5.25 swap()

NAME

swap() — 要素間での値の入れ替え

SYNOPSIS

```
mdarray_type &swap( ssize_t idx_src, size_t len, ssize_t idx_dst );  ...... 1
mdarray_type &swap( size_t dim_index,
                    ssize_t idx_src, size_t len, ssize_t idx_dst );  ...... 2
```

DESCRIPTION

自身の配列要素間で値を入れ替えます.

メンバ関数 1 は，要素番号 idx_src から len 個分の要素を，要素番号 idx_dst から len 個分の要素と入れ替えます．idx_dst + len が配列サイズを超える場合は，配列サイズまでの処理が行われます.

メンバ関数 2 は，次元番号 dim_index の要素番号 idx_src から len 個分の要素を，要素番号 idx_dst から len 個分の要素と入れ替えます．idx_dst + len が配列サイズを超える場合は，配列サイズまでの処理が行われます.

入れ替える領域が重なった場合，重なっていない src の領域に対してのみ入れ替え処理が行われます.

PARAMETER

[I]	idx_src	入れ替え元の要素番号
[I]	len	入れ替え元の要素の長さ
[I]	idx_dst	入れ替え先の要素番号
[I]	dim_index	次元番号

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，mdarray_uchar クラスのオブジェクト my_cmdarr の次元番号 1(2 次元目) の要素番号 0 から 1 個の要素と，要素番号 1 の要素を入れ替え，各要素の値を標準出力します．

```
stdstreamio sio;

unsigned char my_char[] = {51, 52, 101, 102};
mdarray_uchar my_cmdarr(false, 2,2, my_char);

my_cmdarr.swap( 1, 0, 1, 1 );
for ( size_t j = 0 ; j < my_cmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_cmdarr.length(0) ; i++ ) {
        sio.printf("my_cmdarr value(%zu,%zu)... [%hhu]\n",
                   i, j, my_cmdarr(i, j));
    }
}
```

実行結果

```
my_cmdarr value(0,0)...  [101]
my_cmdarr value(1,0)...  [102]
my_cmdarr value(0,1)...  [51]
my_cmdarr value(1,1)...  [52]
```

12.5.26 move()

NAME

move() — 要素間での値のコピー

SYNOPSIS

```
mdarray_type &move( ssize_t idx_src, size_t len, ssize_t idx_dst,
                    bool clr );  ........................................... 1
mdarray_type &move( size_t dim_index, ssize_t idx_src, size_t len, ssize_t idx_dst,
                    bool clr );  ........................................... 2
```

DESCRIPTION

自身の配列要素間で値をコピーします．

clr に false を指定する場合，コピー元の値は残ります．clr に true を指定する場合，コピー元の値は残らず，デフォルト値で埋められます．idx_dst に既存の配列長より大きな値を設定しても，配列サイズは変わりません．この点が次の cpy() メンバ関数 (§12.5.27) とは異なります．

メンバ関数 1 では，常に次元番号 0 の次元 (1 次元目) を処理対象とします．メンバ関数 2 では，次元番号 dim_index で処理対象とする次元を指定できます．

PARAMETER

[I]	`idx_src`	コピー元の要素番号
[I]	`len`	コピー元の要素の長さ
[I]	`idx_dst`	コピー先の要素番号
[I]	`clr`	コピー元の値のクリア可否
[I]	`dim_index`	次元番号

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，mdarray_uchar クラスのオブジェクト `my_cmdarr` の要素間でコピーを行い，コピー元の値をクリアします．確認の為，要素の値を標準出力します．

```
stdstreamio sio;

mdarray_uchar my_cmdarr(false, 3);
my_cmdarr[0] = 99;
my_cmdarr[1] = 98;
my_cmdarr[2] = 97;
my_cmdarr.move( 2, 1, 0, true );
for ( size_t i = 0 ; i < my_cmdarr.length() ; i++ ) {
    sio.printf("my_cmdarr value[%zu]... [%hhu]\n",
               i, my_cmdarr[i]);
}
```

実行結果

```
my_cmdarr value[0]...  [97]
my_cmdarr value[1]...  [98]
my_cmdarr value[2]...  [0]
```

12.5.27 cpy()

NAME

cpy() — 要素間での値のコピー (自動拡張あり)

SYNOPSIS

```
mdarray_type &cpy( ssize_t idx_src, size_t len, ssize_t idx_dst,
              bool clr );  ...................................... 1
mdarray_type &cpy( size_t dim_index, ssize_t idx_src, size_t len, ssize_t idx_dst,
              bool clr );  ...................................... 2
```

DESCRIPTION

自身の配列要素間で値をコピーします．

clr に false を指定する場合，コピー元の値は残ります．clr に true を指定する場合，コピー元の値は残らず，デフォルト値で埋められます．idx_dst + len が既存の配列長より大きい場合，配列サイズは自動拡張されます．

メンバ関数 1 では，常に次元番号 0 の次元 (1 次元目) を処理対象とします．メンバ関数 2 では，次元番号 dim_index で処理対象とする次元を指定できます．

PARAMETER

[I]	idx_src	コピー元の要素番号
[I]	len	コピー元の要素の長さ
[I]	idx_dst	コピー先の要素番号
[I]	clr	コピー元の値のクリア可否
[I]	dim_index	次元番号

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，mdarray_llong クラスのオブジェクト my_lmdarr の要素間でコピーを行い，コピー元の値を残します．確認の為，要素の値を標準出力します．

```
stdstreamio sio;

mdarray_llong my_lmdarr;
my_lmdarr[0] = -2147483646;
my_lmdarr[1] = 2147483647;
my_lmdarr.cpy(1, 1, 2, false);
for ( size_t i = 0 ; i < my_lmdarr.length(0) ; i++ ) {
    sio.printf("my_lmdarr value[%zu]... [%lld]\n", i, my_lmdarr[i]);
}
```

実行結果

```
my_lmdarr value[0]...  [-2147483646]
my_lmdarr value[1]...  [2147483647]
my_lmdarr value[2]...  [2147483647]
```

12.5.28 insert()

NAME

insert() — 要素の挿入

SYNOPSIS

```
mdarray_type &insert( ssize_t idx, size_t len );  ........................... 1
mdarray_type &insert( size_t dim_index, ssize_t idx, size_t len );  ...... 2
```

DESCRIPTION

自身の配列の要素位置 idx に，len 個分の要素を挿入します．なお，挿入される要素の値はデフォルト値です．

メンバ関数 1 では，常に次元番号 0 の次元 (1 次元目) を処理対象とします．メンバ関数 2 では，次元番号 dim_index で処理対象とする次元を指定できます．

PARAMETER

[I]	idx	挿入位置の要素番号
[I]	len	要素の個数
[I]	dim_index	次元番号

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，mdarray_long クラスのオブジェクト my_mdarr の 1 番目の要素の前に 2 個，デフォルト値 (0) の要素を挿入します．確認の為，その結果を標準出力します．

```
stdstreamio sio;

mdarray_long my_mdarr(false, 2);
my_mdarr[0] = -2147483646;
my_mdarr[1] = 2147483647;
my_mdarr.insert( 1, 2 );
for ( size_t i = 0 ; i < my_mdarr.length(0) ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%ld]\n", i, my_mdarr[i]);
}
```

実行結果

```
my_mdarr value[0]...  [-2147483646]
my_mdarr value[1]...  [0]
my_mdarr value[2]...  [0]
my_mdarr value[3]...  [2147483647]
```

12.5.29 crop()

NAME

crop() — 要素の切り出し

SYNOPSIS

```
mdarray_type &crop( ssize_t idx, size_t len );  ............................. 1
mdarray_type &crop( size_t dim_index, ssize_t idx, size_t len );  ........ 2
```

DESCRIPTION

自身の配列を，要素位置 idx から len 個の要素だけにします．

メンバ関数 1 では，常に次元番号 0 の次元 (1 次元目) を処理対象とします．メンバ関数 2 では，次元番号 dim_index で処理対象とする次元を指定できます．

PARAMETER

[I]	idx	切り出し開始位置の要素番号
[I]	len	要素の個数
[I]	dim_index	次元番号

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，mdarray_uchar クラスのオブジェクト my_cmdarr の次元番号 0(1 次元目) の要素番号 1 から 1 個分の要素を切り出します．確認の為，要素の値を標準出力します．

```
stdstreamio sio;

mdarray_uchar my_cmdarr(false, 2, 3);
my_cmdarr(0,0) = 124;
my_cmdarr(1,0) = 125;
my_cmdarr(0,1) = 126;
my_cmdarr(1,1) = 127;
my_cmdarr.crop( 0, 1, 1 );
for ( size_t j = 0 ; j < my_cmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_cmdarr.length(0) ; i++ ) {
        sio.printf("my_cmdarr value(%zu, %zu)... [%hhu]\n",
                   i, j, my_cmdarr(i, j));
    }
}
```

実行結果

```
my_cmdarr value(0, 0)...  [125]
my_cmdarr value(0, 1)...  [127]
my_cmdarr value(0, 2)...  [0]
```

12.5.30 erase()

NAME

erase() — 要素の削除

SYNOPSIS

```
mdarray_type &erase( ssize_t idx, size_t len ); ............................ 1
mdarray_type &erase( size_t dim_index, ssize_t idx, size_t len ); ........ 2
```

DESCRIPTION

自身の配列から指定された要素を削除します．削除した分，長さは短くなります．

メンバ関数 1 では，常に次元番号 0 の次元 (1 次元目) を処理対象とします．メンバ関数 2 では，次元番号 dim_index で処理対象とする次元を指定できます．

PARAMETER

[I]	idx	開始位置の要素番号
[I]	len	要素の個数
[I]	dim_index	次元番号

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，1 次元配列を持つオブジェクト my_mdarr の要素の要素番号 1 から 1 個の要素を削除し，各要素の値を標準出力します．

```
stdstreamio sio;

mdarray_llong my_mdarr(false, 3);
my_mdarr[0] = 0;
my_mdarr[1] = 2147483646;
my_mdarr[2] = 2147483647;

my_mdarr.erase( 1, 1 );
for ( size_t i = 0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%lld]\n", i, my_mdarr[i]);
}
```

実行結果

```
my_mdarr value[0]... [0]
my_mdarr value[1]... [2147483647]
```

12.5.31 resize()

NAME

resize() — 配列の長さを変更

SYNOPSIS

```
mdarray_type &resize( size_t len );  ........................................ 1
mdarray_type &resize( size_t dim_index, size_t len );  ...................... 2
mdarray_type &resize( const mdarray &src );  ................................ 3
```

DESCRIPTION

自身が持つ配列の長さを変更します．

配列長を拡張する場合，要素の値はデフォルト値で埋められます．配列長を収縮する場合，len 以降の要素は削除されます．

メンバ関数 1 は，常に次元番号 0 の次元 (1 次元目) を処理対象とします．メンバ関数 2 は，次元番号 dim_index で処理対象とする次元を指定できます．メンバ関数 3 は，自身の次元数と配列長を，オブジェクト src が持つものと同じ大きさにします．

PARAMETER

[I]	len	要素の個数
[I]	dim_index	次元番号

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_cmdarr の次元番号 1(2 次元目) の配列長を 3 に拡張し，その結果を標準出力します．

```
stdstreamio sio;

mdarray_uchar my_cmdarr(false, 2, 2);
my_cmdarr(0,0) = 70;
my_cmdarr(1,0) = 71;
my_cmdarr(0,1) = 36;
my_cmdarr(1,1) = 37;

my_cmdarr.resize( 1, 3 );
for ( size_t j = 0 ; j < my_cmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_cmdarr.length(0) ; i++ ) {
        sio.printf("my_cmdarr value(%zu, %zu)... [%hhu]\n",
                   i, j, my_cmdarr(i, j));
    }
}
```

実行結果

```
my_cmdarr value(0, 0)...  [70]
my_cmdarr value(1, 0)...  [71]
```

```
my_cmdarr value(0, 1)...  [36]
my_cmdarr value(1, 1)...  [37]
my_cmdarr value(0, 2)...  [0]
my_cmdarr value(1, 2)...  [0]
```

3.6.3にサイズ変更の例があります.

12.5.32 resizeby()

NAME

resizeby() — 配列の長さを相対的に変更

SYNOPSIS

`mdarray_type &resizeby( ssize_t len );` 1
`mdarray_type &resizeby( size_t dim_index, ssize_t len );` 2

DESCRIPTION

自身が持つ配列の長さを `len` の指定分，拡張・縮小します.

サイズの縮小は、`len` にマイナス値を指定することによって行います．`resizeby()` 後の配列サイズは，元の配列の長さに `len` を加えたものとなります.

メンバ関数 1 では，常に次元番号 0 の次元 (1 次元目) を処理対象とします．メンバ関数 2 では，次元番号 `dim_index` で処理対象とする次元を指定できます.

PARAMETER

[I]	`len`	要素個数の増分・減分
[I]	`dim_index`	次元番号

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，1 次元配列を持つオブジェクト `my_cmdarr` の配列長を縮小し，確認の為，値を標準出力します.

```
stdstreamio sio;

mdarray_uchar my_cmdarr(false, 3);

my_cmdarr.resizeby( -2 );
for ( size_t i = 0 ; i < my_cmdarr.length(0) ; i++ ) {
    sio.printf("my_cmdarr value[%d]... [%hhu]\n", i, my_cmdarr[i]);
}
```

実行結果

```
my_cmdarr value[0]...  [0]
```

12.5.33 increase_dim()

NAME

increase_dim() — 次元数の拡張

SYNOPSIS

```
mdarray_type &increase_dim();
```

DESCRIPTION

自身が持つ配列の次元数を 1 つ拡張します.

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合

EXAMPLE

次のコードは，3 次元配列を持つオブジェクト my_mdarr の次元数を 1 つ拡張し，次元数を標準出力します.

```
stdstreamio sio;

mdarray_uchar my_mdarr(false, 1, 2, 3);
my_mdarr.increase_dim();
sio.printf("my_mdarr dim... [%zu]\n",  my_mdarr.dim_length());
```

実行結果

```
my_mdarr dim...  [4]
```

12.5.34 decrease_dim()

NAME

decrease_dim() — 次元数の縮小

SYNOPSIS

```
mdarray_type &decrease_dim();
```

DESCRIPTION

自身が持つ配列の次元を 1 つ縮小します.

RETURN VALUE

自身の参照

EXCEPTION

内部バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，3 次元配列を持つオブジェクト my_mdarr の次元数を 1 つ縮小し，次元数を標準出力します.

```
stdstreamio sio;

mdarray_uchar my_mdarr(false, 1, 2, 3);
my_mdarr.decrease_dim();
sio.printf("my_mdarr dim... [%zu]\n", my_mdarr.dim_length());
```

実行結果

```
my_mdarr dim... [2]
```

12.5.35 swap()

NAME

swap() — 別オブジェクトとの内容の入れ替え

SYNOPSIS

```
mdarray_type &swap( mdarray_type &sobj );
```

DESCRIPTION

指定されたオブジェクト sobj の内容と自身の内容を入れ替えます．配列サイズ等すべての状態が入れ替えます.

PARAMETER

[I/O] sobj 自身と同じクラスのオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，1 次元配列を持つオブジェクト my_fmdarr と swap_mdarr を入れ替え，my_fmdarr の要素の値を標準出力します.

```
stdstreamio sio;

mdarray_float my_fmdarr(false, 2);
my_fmdarr[0] = 1000;
my_fmdarr[1] = 2000;

mdarray_float swap_mdarr(false, 2);
```

```
swap_mdarr[0] = 100;
swap_mdarr[1] = 200;
my_fmdarr.swap(swap_mdarr);
for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%g]\n", i, my_fmdarr.dvalue(i));
}
```

実行結果

```
my_fmdarr value[0]...  [100]
my_fmdarr value[1]...  [200]
```

12.5.36 convert()

NAME

convert() — 配列の全要素の値変換

SYNOPSIS

```
mdarray_type &convert( void (*func)(const type [],type [],size_t,bool,void *),
                       void *user_ptr );
```

DESCRIPTION

ユーザ定義関数 func 経由で，自身の全要素の値を書き換えます．

ユーザ定義関数の第 1 引数には配列の元の各要素アドレスが，第 2 引数には変換後の各要素のアドレスが，第 3 引数にはユーザ関数で変換すべき要素の個数が，第 5 引数には user_ptr が与えられます．第 4 引数はユーザ関数中のコードでは無視してください．ユーザ関数のコードでは，処理結果を第 2 引数で示されるアドレスに書き込んでください．

PARAMETER

[I] sz_type 要素の型種別
[I] func ユーザ関数のアドレス
[I] user_ptr ユーザ関数の最後に与えられるユーザのポインタ
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

12.5.37 ceil()

NAME

ceil() — 浮動小数点型の全要素値の切り上げ

SYNOPSIS

```
mdarray_type &ceil();
```

DESCRIPTION

自身の配列 (浮動小数点型) の全要素の小数部の値を切り上げます．

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，1 次元配列を持つオブジェクト my_fmdarr に小数点以下を持つ値を設定後，切り上げ処理を行います．確認の為，要素の値を標準出力します．

```
stdstreamio sio;

mdarray_float my_fmdarr;

my_fmdarr[0] = 1000.1;
my_fmdarr[1] = 2000.6;
my_fmdarr.ceil();

for ( size_t i = 0 ; i < my_fmdarr.length() ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%f]\n", i, my_fmdarr[i]);
}
```

実行結果

```
my_fmdarr value[0]...  [1001.000000]
my_fmdarr value[1]...  [2001.000000]
```

12.5.38 floor()

NAME

floor() — 浮動小数点型の全要素値の小数部の切り下げ

SYNOPSIS

```
mdarray_type &floor();
```

DESCRIPTION

自身の配列 (浮動小数点型) の全要素の小数部を，それぞれの要素の値を越えない最大の整数値に切り下げます．

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，1 次元配列を持つオブジェクト my_fmdarr に小数点以下を持つ値を設定後，切り下げ処理を行います．確認の為，要素の値を標準出力します．

```
stdstreamio sio;

mdarray_float my_fmdarr;

my_fmdarr[0] = 1000.1;
```

```
my_fmdarr[1] = 2000.9;
my_fmdarr.floor();

for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%f]\n", i, my_fmdarr[i]);
}
```

実行結果

```
my_fmdarr value[0]...  [1000.000000]
my_fmdarr value[1]...  [2000.000000]
```

12.5.39 round()

NAME

round() — 浮動小数点型の全要素値の四捨五入

SYNOPSIS

`mdarray_type &round();`

DESCRIPTION

自身の配列 (浮動小数点型) の全要素値の小数部を四捨五入し，整数値にします.

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，1 次元配列を持つオブジェクト mdarrf に小数点以下の値を持つ値を設定後，四捨五入処理を行い，確認の為，各要素の値を標準出力します.

```
stdstreamio sio;

mdarray_float my_fmdarr;
my_fmdarr[0] = 1000.5;
my_fmdarr[1] = -1000.5;
my_fmdarr.round();

for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%f]\n", i, my_fmdarr[i]);
}
```

実行結果

```
my_fmdarr value[0]...  [1001.000000]
my_fmdarr value[1]...  [-1001.000000]
```

12.5.40 trunc()

NAME

trunc() — 浮動小数点型の全要素値の小数部の切り捨て

SYNOPSIS

```
mdarray_type &trunc();
```

DESCRIPTION

自身の配列 (浮動小数点型) の全要素の値の小数部を切り捨て，0 に近い方の整数値にします.

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，1 次元配列を持つオブジェクト my_fmdarr に小数点以下持つ値を設定後，小数部の切り捨て処理を行います．確認の為，各素の値を標準出力します.

```
stdstreamio sio;

mdarray_float my_fmdarr;
my_fmdarr[0] = 1.7;
my_fmdarr[1] = -1.7;
my_fmdarr.trunc();

for ( size_t i = 0 ; i < my_fmdarr.length() ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%f]\n", i, my_fmdarr[i]);
}
```

実行結果

```
my_fmdarr value[0]...  [1.000000]
my_fmdarr value[1]...  [-1.000000]
```

12.5.41 abs()

NAME

abs() — 全要素値に対し絶対値をとる

SYNOPSIS

```
mdarray_type &abs();
```

DESCRIPTION

自身の配列の全要素について絶対値をとります.

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，1 次元配列を持つオブジェクト `my_fmdarr` に負の値を設定後，各要素の絶対値を標準出力します.

```
stdstreamio sio;

mdarray_float my_fmdarr;

my_fmdarr[0] = -1000.1;
my_fmdarr[1] = -2000.6;
my_fmdarr.abs();

for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%5.1f]\n", i, my_fmdarr[i]);
}
```

実行結果

```
my_fmdarr value[0]...  [1000.1]
my_fmdarr value[1]...  [2000.6]
```

12.5.42 compare()

NAME

compare() — 配列オブジェクトの比較

SYNOPSIS

`bool compare(const mdarray &obj) const;`

DESCRIPTION

自身と指定されたオブジェクト `obj` の配列が等しいかどうかを返します.

引数は mdarray クラスなので，自身とは異なる型の配列を持つオブジェクトを指定できます. 配列の型が異なっても配列長と値が等しければ真 true(=1)，異なれば偽 false(=0) を返します.

PARAMETER

[I] `obj` mdarray(継承) クラスのオブジェクト

([I]：入力，[O]：出力)

RETURN VALUE

true ： 配列サイズ，要素の値が一致した場合

false ： 配列サイズ，要素の値が不一致である場合

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト `my_fmdarr` と `my_i64mdarr` を比較し，結果を標準出力します.

```
stdstreamio sio;
```

```
mdarray_float my_fmdarr(false, 2,2);
my_fmdarr(0,0) = 1000;

mdarray_int64 my_i64mdarr(false, 2,2);
my_i64mdarr(0,0) = 1000;

sio.printf("*** my_fmdarr compare [%d] *** \n",
           (int)my_fmdarr.compare(my_i64mdarr));
```

実行結果

```
*** my_fmdarr compare [1] ***
```

12.5.43 copy()

NAME

copy() — 配列を別オブジェクトへコピー

SYNOPSIS

`ssize_t copy( mdarray_type *dest ) const;`

DESCRIPTION

自身の全ての内容を指定されたオブジェクト `dest` へコピーします．

コピー先へは，コピー元の配列長，値等全ての属性をコピーします．自身 (コピー元) の配列に変化はありません．

PARAMETER

[O] `dest` コピー先のオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

コピーした要素数 (列数 × 行数 × レイヤ数)

EXCEPTION

バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト `my_cmdarr` を `my_fmdarr` へコピーします．確認の為，要素の値を標準出力します．

```
stdstreamio sio;

mdarray_float my_fmdarr;
float my_values[] = {99, 101, 98, 102};
mdarray_float my_cmdarr(false, 2, 2, my_values);

ssize_t copy_size = my_cmdarr.copy( &my_fmdarr );
```

```
    for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
        for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
            sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
                        i, j, my_fmdarr(i, j));
        }
    }
```

実行結果

```
my_fmdarr value(0,0)...  [98]
my_fmdarr value(1,0)...  [99]
my_fmdarr value(0,1)...  [101]
my_fmdarr value(1,1)...  [102]
```

12.5.44 copy()

NAME

copy() — 配列の一部を別オブジェクトへコピー (画像データ向き)

SYNOPSIS

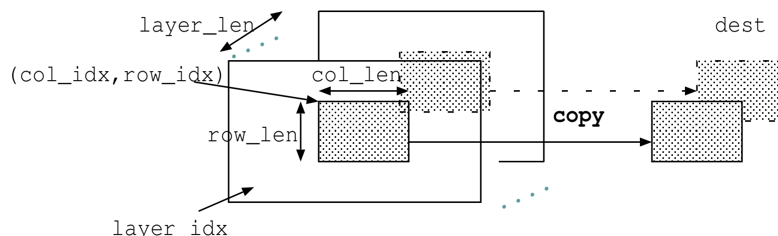
```
ssize_t copy( mdarray_type *dest,
              ssize_t col_idx, size_t col_len=MDARRAY_ALL,
              ssize_t row_idx=0, size_t row_len=MDARRAY_ALL,
              ssize_t layer_idx=0, size_t layer_len=MDARRAY_ALL ) const;
```

DESCRIPTION

画像データ向きのメンバ関数で，自身の内容の一部分を指定されたオブジェクト dest へコピーします．

自身 (コピー元) の配列に変化はありません．

copy() メンバ関数によって画像データがコピーされる様子を下図に示します．



図の網掛け部分が，2 番目以降の引数で指定した部分で，この領域が dest にコピーされます．

引数に，MDARRAY_ALL を明示的に与えないでください．

PARAMETER

[O]	dest	コピー先のオブジェクト
[I]	col_idx	コピー元の列位置
[I]	col_len	コピー元の列サイズ
[I]	row_idx	コピー元の行位置
[I]	row_len	コピー元の行サイズ
[I]	layer_idx	コピー元のレイヤ位置
[I]	layer_len	コピー元のレイヤサイズ

([I]：入力，[O]：出力)

RETURN VALUE

コピーした要素数 (列数 × 行数 × レイヤ数)

EXCEPTION

バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_cmdarr を my_fmdarr へコピーします．確認の為，要素の値を標準出力します．

```
stdstreamio sio;

double my_vals[] = {98, 99, 101, 102};
mdarray_double my_cmdarr(false, 2,2, my_vals);

double my_d[] = {-501, 501, -502, 502};
mdarray_double my_dmdarr(false, 2,2, my_d);

ssize_t ret_size = my_cmdarr.copy( &my_dmdarr, 1, 1, 1, 1 );
for ( size_t j = 0 ; j < my_dmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_dmdarr.length(0) ; i++ ) {
        sio.printf("my_dmdarr value(%zu,%zu)... [%g]\n",
                   i, j, my_dmdarr(i, j));
    }
}
```

実行結果

```
my_ldmdarr value(0,0)...  [102]
```

3.6.7にコピー&ペーストの例があります．

12.5.45 cut()

NAME

cut() — 配列の全値を切り出し，別オブジェクトへコピー

SYNOPSIS

```
mdarray_type &cut( mdarray_type *dest );
```

DESCRIPTION

自身の配列の全ての内容を切り出し，指定されたオブジェクト dest へコピーします.

自身の配列の全要素を切り出すので，自身 (取り出し元) の配列長は 0 になります.

PARAMETER

[O] dest コピー先のオブジェクト

([I] : 入力，[O] : 出力)

RETURN VALUE

自身の参照

EXCEPTION

バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_cmdarr をオブジェクト my_mdarr へ取り出します．確認の為，my_cmdarr の配列長を標準出力します.

```
stdstreamio sio;

mdarray_uchar my_mdarr;

unsigned char my_char[] = {51, 101, 52, 102};
mdarray_uchar my_cmdarr(false, 2, 2, my_char);

my_cmdarr.cut( &my_mdarr );
sio.printf("my_cmdarr length()... [%zu]\n", my_cmdarr.length());
```

実行結果

```
my_cmdarr length()... [0]
```

12.5.46 cut()

NAME

cut() — 配列の一部を切り出し，別オブジェクトへコピー (画像データ向き)

SYNOPSIS

```
mdarray_type &cut( mdarray_type *dest,
                   ssize_t col_idx, size_t col_len=MDARRAY_ALL,
                   ssize_t row_idx=0, size_t row_len=MDARRAY_ALL,
                   ssize_t layer_idx=0, size_t layer_len=MDARRAY_ALL );
```

DESCRIPTION

画像データ向きのメンバ関数で，自身の内容の一部分を切り出し，指定されたオブジェクト dest へコピーします．

自身 (取り出し元) の配列長は変わらず，第 2 引数以降で指定された領域の要素の値はデフォルト値で埋められます．

引数に，MDARRAY_ALL を明示的に与えないでください．

PARAMETER

[O]	dest	コピー先のオブジェクト
[I]	col_idx	コピー元の列位置
[I]	col_len	コピー元の列サイズ
[I]	row_idx	コピー元の行位置
[I]	row_len	コピー元の行サイズ
[I]	layer_idx	コピー元のレイヤ位置
[I]	layer_len	コピー元のレイヤサイズ

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

バッファの確保に失敗した場合
メモリ破壊を起こした場合

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_cmdarr の 0 列目をオブジェクト my_mdarr へ取り出します．確認の為，要素の値を標準出力します．

```
stdstreamio sio;

mdarray_uchar my_mdarr;

unsigned char my_char[] = {51, 101, 52, 102};
mdarray_uchar my_cmdarr(false, 2,2, my_char);

my_cmdarr.cut( &my_mdarr, 0, 1 );
for ( size_t j = 0 ; j < my_cmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_cmdarr.length(0) ; i++ ) {
        sio.printf("my_cmdarr value(%zu,%zu)... [%hhu]\n",
                   i, j, my_cmdarr(i, j));
    }
}
```

実行結果

```
my_cmdarr value(0,0)...  [0]
my_cmdarr value(1,0)...  [101]
```

```
my_cmdarr value(0,1)...  [0]
my_cmdarr value(1,1)...  [102]
```

12.5.47 clean()

NAME

clean() — 既存の配列要素をデフォルト値でパディング (画像データ向き)

SYNOPSIS

```
mdarray_type &clean( ssize_t col_index = 0, size_t col_size = MDARRAY_ALL,
                     ssize_t row_index = 0, size_t row_size = MDARRAY_ALL,
                     ssize_t layer_index = 0, size_t layer_size = MDARRAY_ALL );
```

DESCRIPTION

自身の配列の要素をデフォルト値でパディングします．引数は指定しなくても使用できます．引数を 1 つも指定しない場合は，全要素が対象となります．`clean()` を実行しても配列長は変化しません．

画像データ向きのメンバ関数です．

引数に，`MDARRAY_ALL` を明示的に与えないでください．

PARAMETER

[I]	`col_index`	列位置
[I]	`col_size`	列サイズ
[I]	`row_index`	行位置
[I]	`row_size`	行サイズ
[I]	`layer_index`	レイヤ位置
[I]	`layer_size`	レイヤサイズ

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト `my_smdarr` の各要素を設定後，1 要素を `clean()` します．確認の為，値を標準出力します．

```
stdstreamio sio;

mdarray_short my_smdarr(false, 2,2);
my_smdarr(0,0) = 1;
my_smdarr(1,0) = 3;
my_smdarr(0,1) = 2;
my_smdarr(1,1) = 4;

my_smdarr.clean(1,1,1,1);
for ( size_t j = 0 ; j < my_smdarr.length(1) ; j++ ) {
```

```
        for ( size_t i = 0 ; i < my_smdarr.length(0) ; i++ ) {
            sio.printf("my_smdarr value(%zu,%zu)... [%hd]\n",
                       i, j, my_smdarr(i, j));
        }
    }
```

実行結果

```
my_smdarr value(0,0)...  [1]
my_smdarr value(1,0)...  [3]
my_smdarr value(0,1)...  [2]
my_smdarr value(1,1)...  [0]
```

12.5.48 fill()

NAME

fill() — 要素値の書換え (画像データ向き)

SYNOPSIS

```
mdarray_type &fill( double value,
    ssize_t col_index = 0, size_t col_size = MDARRAY_ALL,
    ssize_t row_index = 0, size_t row_size = MDARRAY_ALL,
    ssize_t layer_index = 0, size_t layer_size = MDARRAY_ALL );  .......... 1
mdarray_type &fill( double value,
    void (*func)(double [],double,size_t,
                 ssize_t,ssize_t,ssize_t,mdarray_type *,void *),
    void *user_ptr,
    ssize_t col_index = 0, size_t col_size = MDARRAY_ALL,
    ssize_t row_index = 0, size_t row_size = MDARRAY_ALL,
    ssize_t layer_index = 0, size_t layer_size = MDARRAY_ALL );  .......... 2
```

DESCRIPTION

自身の配列の指定された範囲の要素を，指定された値 (メンバ関数 1)，またはユーザ定義関数経由 (メンバ関数 2) で書換えます．

ユーザ定義関数 func の引数には順に，ユーザプログラムで書き換えるべき要素値の配列 (一時バッファ上の double 型に変換された自身の要素値の配列)，value で与えられた値，書き換えるべき個数 (一時バッファ上の配列の個数)，列位置，行位置，レイヤ位置，自身のオブジェクトのアドレス，ユーザポインタ user_ptr の値，が与えられます．ユーザ定義関数の指定の方法については，§12.5.51の EXAMPLE を参照してください．

画像データ向きのメンバ関数です．

引数に，MDARRAY_ALL を明示的に与えないでください．

PARAMETER

[I]	`value`	書き込む値
[I]	`user_ptr`	func の最後に与えられるユーザのポインタ
[I]	`col_index`	列位置
[I]	`col_size`	列サイズ
[I]	`row_index`	行位置
[I]	`row_size`	行サイズ
[I]	`layer_index`	レイヤ位置
[I]	`layer_size`	レイヤサイズ
[I]	`func`	値変換の為のユーザ関数のアドレス

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_smdarr の全要素を値 100 に書換えます．確認の為，要素の値を標準出力します.

```
stdstreamio sio;

mdarray_short my_smdarr(false, 2,2);
my_smdarr.fill(100);

for ( size_t j = 0 ; j < my_smdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_smdarr.length(0) ; i++ ) {
        sio.printf("my_smdarr value(%zu,%zu).. [%hd]\n",
                   i, j, my_smdarr(i, j));
    }
}
```

実行結果

```
my_smdarr value(0,0)..  [100]
my_smdarr value(1,0)..  [100]
my_smdarr value(0,1)..  [100]
my_smdarr value(1,1)..  [100]
```

12.5.49 add()

NAME

add() — 要素値の加算 (画像データ向き)

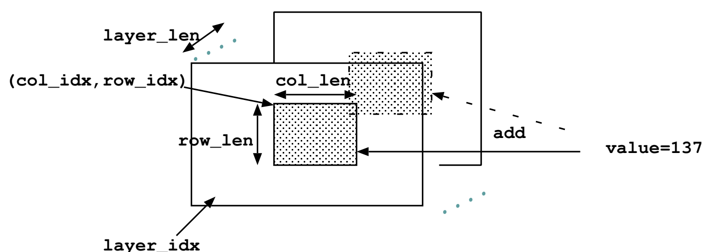
SYNOPSIS

```
mdarray_type &add( double value,
                   ssize_t col_index = 0, size_t col_size = MDARRAY_ALL,
                   ssize_t row_index = 0, size_t row_size = MDARRAY_ALL,
                   ssize_t layer_index = 0, size_t layer_size = MDARRAY_ALL );
```

DESCRIPTION

自身の配列の指定された範囲の要素に value を加算します．画像データ向きのメンバ関数です．

add() メンバ関数によって，画像データの一部に値 137 が加算される様子を下図に示します．



図の網掛け部分が，2 番目以降の引数で指定した部分で，この領域に value が加算されます．

引数に，MDARRAY_ALL を明示的に与えないでください．

PARAMETER

[I]	value	加算する値
[I]	col_index	列位置
[I]	col_size	列サイズ
[I]	row_index	行位置
[I]	row_size	行サイズ
[I]	layer_index	レイヤ位置
[I]	layer_size	レイヤサイズ

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_smdarr の 2 列 2 行目の値に 10 を加算します．確認の為，値を標準出力します．

```
stdstreamio sio;

short my_short[] = {1, 2, 3, 4};
mdarray_short my_smdarr(false, 2,2, my_short);

my_smdarr.add(10.0, 1,1,1,1);

for ( size_t j = 0 ; j < my_smdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_smdarr.length(0) ; i++ ) {
        sio.printf("my_smdarr value(%zu,%zu)... [%hd]\n",
                   i, j, my_smdarr(i, j));
    }
}
```

実行結果

```
my_smdarr value(0,0)...  [1]
my_smdarr value(1,0)...  [2]
my_smdarr value(0,1)...  [3]
my_smdarr value(1,1)...  [14]
```

12.5.50 multiply()

NAME

multiply() — 要素値の乗算 (画像データ向き)

SYNOPSIS

```
mdarray_type &multiply( double value,
                        ssize_t col_index = 0, size_t col_size = MDARRAY_ALL,
                        ssize_t row_index = 0, size_t row_size = MDARRAY_ALL,
                        ssize_t layer_index = 0, size_t layer_size = MDARRAY_ALL );
```

DESCRIPTION

自身の配列の指定された範囲の要素の値に value を乗算します．画像データ向きのメンバ関数です．

引数に，MDARRAY_ALL を明示的に与えないでください．

PARAMETER

[I]	value	乗算する値
[I]	col_index	列位置
[I]	col_size	列サイズ
[I]	row_index	行位置
[I]	row_size	行サイズ
[I]	layer_index	レイヤ位置
[I]	layer_size	レイヤサイズ

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_fmdarr の全要素に 50 を乗算します．確認の為，値を標準出力します．

```
stdstreamio sio;

float my_float[] = {1, 3, 2, 4};
mdarray_float my_fmdarr(false, 2,2, my_float);

my_fmdarr.multiply(50);
for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
```

```
        for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
            sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
                       i, j, my_fmdarr(i, j));
        }
    }
```

実行結果

```
my_fmdarr value(0,0)...  [50.000000]
my_fmdarr value(1,0)...  [150.000000]
my_fmdarr value(0,1)...  [100.000000]
my_fmdarr value(1,1)...  [200.000000]
```

12.5.51 paste()

NAME

paste() — 配列オブジェクトの貼り付け (画像データ向き)

SYNOPSIS

```
mdarray_type &paste( const mdarray &src,
    ssize_t dest_col = 0, ssize_t dest_row = 0, ssize_t dest_layer = 0 );   1
mdarray_type &paste( const mdarray &src,
    void (*func)(double [],double [],size_t,
                 ssize_t,ssize_t,ssize_t,mdarray_type *,void *),
    void *user_ptr,
    ssize_t dest_col = 0, ssize_t dest_row = 0, ssize_t dest_layer = 0 );   2
```

DESCRIPTION

自身の配列の指定された範囲の要素値に，src で指定されたオブジェクトの各要素値，またはユーザ定義関数経由で変換された各要素値を貼り付けます.

1 番目の引数は mdarray クラスなので，自身とは異なる型の配列を持つオブジェクトを指定できます.

メンバ関数 2 では，ユーザ定義関数を与えて貼り付け時の挙動を変えることができます．ユーザ定義関数 func の引数には順に，ユーザプログラムで書き換えるべき要素値の配列 (一時バッファ上の double 型に変換された自身の要素値の配列)，一時バッファ上の double 型に変換されたオブジェクト src の要素値の配列，一時バッファの長さ (書き換えるべき配列の個数)，列位置，行位置，レイヤ位置，自身のオブジェクトのアドレス，ユーザポインタ user_ptr の値，が与えられます.

画像データ向きのメンバ関数です.

PARAMETER

[I]	src	源泉となる配列を持つオブジェクト
[I]	func	値変換のためのユーザ関数のアドレス
[I]	user_ptr	func の最後に与えられるユーザのポインタ
[I]	dest_col	列位置
[I]	dest_row	行位置
[I]	dest_layer	レイヤ位置

([I] : 入力, [O] : 出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは, 2 次元配列を持つオブジェクト my_fmdarr に, 2 次元配列を持つオブジェクト mypaste_mdarr を貼り付けます. 貼り付けの際に, 両者の値を加えてさらに 500 を加算するユーザ定義関数を指定しています. 確認の為, 値を標準出力します.

```
void my_func(double self[], double src[], size_t len, ssize_t x,
             ssize_t y, ssize_t z, mdarray_float *myptr, void *p)
{
    size_t i;
    for ( i=0 ; i < len ; i++ ) self[i] += src[i] + 500;
}

/* 処理 */
stdstreamio sio;

float my_float[] = {100, 0, 200, 0};
mdarray_float my_fmdarr(false, 2,2, my_float);

float mypaste_float[] = {1000, 3000, 2000, 4000};
mdarray_float mypaste_mdarr(false, 2,2, mypaste_float);

my_fmdarr.paste(mypaste_mdarr, &my_func, NULL);
for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
        sio.printf("my_fmdarr value[%zu][%zu]... [%f]\n", i, j,
                   my_fmdarr(i, j));
    }
}
```

実行結果

```
my_fmdarr value(0,0)...  [1600.000000]
my_fmdarr value(1,0)...  [2500.000000]
my_fmdarr value(0,1)...  [3700.000000]
my_fmdarr value(1,1)...  [4500.000000]
```

3.6.7にコピー&ペーストの例があります.

12.5.52 add()

NAME

add() — 配列オブジェクトの加算 (画像データ向き)

SYNOPSIS

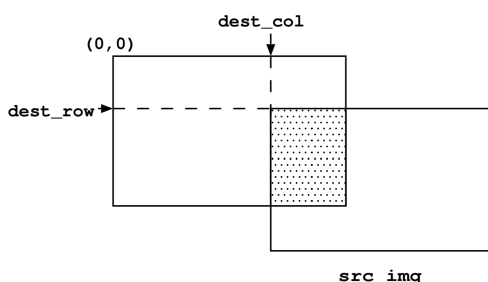
```
mdarray_type &add( const mdarray &src_img, ssize_t dest_col = 0,
                   ssize_t dest_row = 0, ssize_t dest_layer = 0 );
```

DESCRIPTION

自身の要素にオブジェクト src_img が持つ配列を加算します．列・行・レイヤについてそれぞれの加算適用開始位置を指定できます．画像データ向きのメンバ関数です．

1 番目の引数は mdarray クラスなので，自身とは異なる型の配列を持つオブジェクトを指定できます．

add() メンバ関数によって画像データが加算される様子を下図に示します．



図の網掛け部分が，2 番目以降の引数で指定した部分で，この領域に src_img が加算されます．

PARAMETER

[I]	src_img	演算に使う配列を持つオブジェクト
[I]	dest_col	加算開始位置 (列)
[I]	dest_row	加算開始位置 (行)
[I]	dest_layer	加算開始位置 (レイヤ)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_smdarr に 2 次元配列を持つオブジェクト add_smdarr を加算します．確認の為，要素の値を標準出力します．

```
stdstreamio sio;

short my_short[] = {1, 2, 3, 4};
mdarray_short my_smdarr(false, 2,2, my_short);

short myadd_short[] = {9, 8, 7, 6};
mdarray_short myadd_smdarr(false, 2,2, myadd_short);

my_smdarr.add(myadd_smdarr);
for ( size_t j = 0 ; j < my_smdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_smdarr.length(0) ; i++ ) {
```

```
            sio.printf("my_smdarr value(%zu,%zu)... [%hd]\n",
                       i, j, my_smdarr(i, j));
        }
    }
```

実行結果

```
my_smdarr value(0,0)...  [10]
my_smdarr value(1,0)...  [10]
my_smdarr value(0,1)...  [10]
my_smdarr value(1,1)...  [10]
```

12.5.53 subtract()

NAME

subtract() — 配列オブジェクトの減算 (画像データ向き)

SYNOPSIS

```
mdarray_type &subtract( const mdarray &src_img, ssize_t dest_col = 0,
                        ssize_t dest_row = 0, ssize_t dest_layer = 0 );
```

DESCRIPTION

自身の配列の要素値からオブジェクト src_img が持つ配列の要素値を減算します．列・行・レイヤについてそれぞれの減算適用開始位置を指定できます．画像データ向きのメンバ関数です．

1 番目の引数は mdarray クラスなので，自身とは異なる型の配列を持つオブジェクトを指定できます．

PARAMETER

[I]	src_img	演算に使う配列を持つオブジェクト
[I]	dest_col	減算開始位置 (列)
[I]	dest_row	減算開始位置 (行)
[I]	dest_layer	減算開始位置 (レイヤ)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_fmdarr から 2 次元配列を持つオブジェクト mysubtract_mdarr の値を減算します．確認の為，値を標準出力します．

```
    stdstreamio sio;

    float my_float[] = {1000, 2000, 3000, 4000};
    mdarray_float my_fmdarr(false, 2,2, my_float);

    float mysubt_float[] = {100, 200, 300, 400};
```

```
mdarray_float mysubtract_mdarr(false, 2,2, mysubt_float);

my_fmdarr.subtract(mysubtract_mdarr);
for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
        sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
                   i, j, my_fmdarr(i, j));
    }
}
```

実行結果

```
my_fmdarr value(0,0)...  [900.000000]
my_fmdarr value(1,0)...  [1800.000000]
my_fmdarr value(0,1)...  [2700.000000]
my_fmdarr value(1,1)...  [3600.000000]
```

12.5.54 multiply()

NAME

multiply() — 配列オブジェクトの乗算 (画像データ向き)

SYNOPSIS

```
mdarray_type &multiply( const mdarray &src_img, ssize_t dest_col = 0,
                        ssize_t dest_row = 0, ssize_t dest_layer = 0 );
```

DESCRIPTION

自身の配列の要素値にオブジェクト `src_img` が持つ配列を乗算します．列・行・レイヤについてそれぞれの乗算適用開始位置を指定できます．画像データ向きのメンバ関数です．

1 番目の引数は mdarray クラスなので，自身とは異なる型の配列を持つオブジェクトを指定できます．

PARAMETER

[I]	`src_img`	演算に使う配列を持つオブジェクト
[I]	`dest_col`	乗算開始位置 (列)
[I]	`dest_row`	乗算開始位置 (行)
[I]	`dest_layer`	乗算開始位置 (レイヤ)

([I] : 入力，[O] : 出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト `my_fmdarr` に 2 次元配列を持つオブジェクト `mymulti_fmdarr` を乗算します．確認の為，要素の値を標準出力します．

```
stdstreamio sio;
```

```
float my_float[] = {1, 2, 3, 4};
mdarray_float my_fmdarr(false, 2,2, my_float);

float mymulti_float[] = {10, 20, 30, 40};
mdarray_float mymulti_fmdarr(false, 2,2, mymulti_float);

my_fmdarr.multiply(mymulti_fmdarr);
for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
        sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
                   i, j, my_fmdarr(i, j));
    }
}
```

実行結果

```
my_fmdarr value(0,0)...  [10.000000]
my_fmdarr value(1,0)...  [40.000000]
my_fmdarr value(0,1)...  [90.000000]
my_fmdarr value(1,1)...  [160.000000]
```

12.5.55 divide()

NAME

divide() — 配列オブジェクトの除算 (画像データ向き)

SYNOPSIS

```
mdarray_type &divide( const mdarray &src_img, ssize_t dest_col = 0,
                      ssize_t dest_row = 0, ssize_t dest_layer = 0 );
```

DESCRIPTION

自身の配列の要素値からオブジェクト src_img が持つ配列を除算します．列・行・レイヤについてそれぞれの除算適用開始位置を指定できます．画像データ向きのメンバ関数です．

1 番目の引数は mdarray クラスなので，自身とは異なる型の配列を持つオブジェクトを指定できます．

PARAMETER

[I]	src_img	演算に使う配列を持つオブジェクト
[I]	dest_col	除算開始位置 (列)
[I]	dest_row	除算開始位置 (行)
[I]	dest_layer	除算開始位置 (レイヤ)

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，2 次元配列を持つオブジェクト my_fmdarr から 2 次元配列を持つオブジェクト mydiv_mdarrf を除算します．確認の為，値を標準出力します．

```
stdstreamio sio;

float my_float[] = {1000, 2000, 3000, 4000};
mdarray_float my_fmdarr(false, 2,2, my_float);

float mydiv_float[] = {2, 4, 6, 8};
mdarray_float mydiv_mdarrf(false, 2,2, mydiv_float);

my_fmdarr.divide(mydiv_mdarrf);
for ( size_t j = 0 ; j < my_fmdarr.length(1) ; j++ ) {
    for ( size_t i = 0 ; i < my_fmdarr.length(0) ; i++ ) {
        sio.printf("my_fmdarr value(%zu,%zu)... [%f]\n",
        i, j, my_fmdarr(i, j));
    }
}
```

実行結果

```
my_fmdarr value(0,0)...  [500.000000]
my_fmdarr value(1,0)...  [500.000000]
my_fmdarr value(0,1)...  [500.000000]
my_fmdarr value(1,1)...  [500.000000]
```

version 1.0 系からの API の変更について

名称変更したメンバ関数

- `tstring::substr()` → `tstring::crop()`

  substr() メンバ関数は自身の文字列を，自身の文字列の一部に置き換えるメンバ関数でしたが，一般の substr() 関数と挙動が異なるので，`crop()` に名称変更しました．動作は以前の `substr()` と同じです．

  一般の substr() 関数に相当するものは，copy() メンバ関数です．`copy()` により，自身の文字列の一部を，外部のオブジェクトにコピーする事ができます．

- `tstring::strltrim()` → `tstring::ltrim()`
  `tstring::strrtrim()` → `tstring::rtrim()`
  `tstring::strtrim()` → `tstring::trim()`

  `strrtrim()` が長くて読みずらいので，変更しました．

  `tstring::strtrim()` は残していますが，今後は `tstring::trim()` を使ってください．

引数を変更したメンバ関数

- `tstring::strtol()`, `tstring::strtoll()`, `tstring::strtoul()`, `tstring::strtoull()`

  引数「`size_t *endpos`」が常に最後になるように変更しました．引数「`int base`」と「`size_t *endpos`」とを入れ替えてください．

  コードを適切に変更せずにコンパイルするとエラーが報告されます．

- `tarray_tstring::split()`,
  `asarray_tstring::split_keys()`,
  `asarray_tstring::split_values()`

  最後に 1 つだけ引数「`bool rm_escape`」が追加されました．この新しい引数により，分割後の文字列におけるエスケープ文字を削除するかどうかを指定する事ができます．例えば，この引数が `true` の場合，元の文字列「`program\ files`」は分割後には「`program files`」となります．ただし，クォーテーションで囲まれた部分については，`rm_escape` に関係なくエスケープ文字の削除を行ないません．

  version 1.0 系と同じ動作にするには，`rm_escape` に `true` をセットします．

  コードを適切に変更せずにコンパイルするとエラーが報告されます．

  引数 `delims` には `"[A-Z]"` のような表現を与えられるようになりました．表現の詳細は tstring::trim() (§9.5.26) を参照してください．

場合によっては別の API を使う事を考えた方が良いメンバ関数

- `tstring::regmatch( const char *pat )`,
  `tstring::regmatch( size_t pos, const char *pat )`

  後方参照の情報を取り出し，それらをさらに加工する場合には，`tarray_tstring::regassign()` の方が便利です．

tarray_tstring::regassign() は，引数で与えられた文字列について，正規表現にマッチした部分と，その後方参照部分文字列を，文字列配列として格納します．

オーバーロードにより強化されたメンバ関数

- 各クラスの [] 演算子，at() メンバ関数に，const 版が追加されました．
- tstring::find()，tstring::strpbrk()，tstring::regmatch() などに，次の検索位置を得るためのポインタ引数を与えられるようにしました．

追加された主なメンバ関数

- tarray_tstring クラス，asarray_tstring クラスに strreplace()，chomp()，trim()，tolower()，toupper()，regreplace() などが追加されました．これらにより，配列の全要素を一括して変更する事ができます．
- tarray_tstring クラスに，find()，find_elem()，regmatch() などの検索のための API を追加しました．

  asarray_tstring クラスでも，values() メンバ関数，keys() メンバ関数を経由してこれらの検索用の API が利用できます．