Simple and Light Interfaces for C and C++ users

SLLIB − Script-Like C-language library
User's Reference Guide 上級編

Version 1.1.0 日本語版

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*, *Sergio Pascual* and *Yukio Yamamoto*

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

目 次

1 TARRAY テンプレートクラス

tarray テンプレートクラスは，任意の型・クラスの配列を扱う事ができるクラスです.

機能的には，基本編の tarray_tstring とほとんど同じで，文字列に特化したメンバ関数を除いたものと言えるでしょう．ですので，この章ではコードの EXAMPLE は省略しています．EXAMPLE については，基本編の tarray_tstring のそれぞれのメンバ関数のリファレンスをご覧ください.

tarray テンプレート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/tarray.h>
```

また，namespace 宣言が必要な場合は,「`using namespace sli;`」もコードに書いてください.

1.1 オブジェクトの作り方

tarray テンプレートクラスのオブジェクトを作る時には，引数は必要ありません．次のように普通にオブジェクトを作成します.

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

int main()
{
    tarray<tarray_tstring> my_2d_array;
```

この例では，基本編の tarray_tstring クラスを配列化しています．これにより，次のように文字列の 2 次元配列が簡単に扱えるようになります.

```
    my_2d_array[0][0] = "SLLIB";
```

1.2 メンバ関数一覧

表 1 はメンバ関数一覧です.

	メンバ関数名	機能
§1.3.1	`[]`	指定要素の参照
§1.3.2	`=`	同じ型・クラスを扱う配列を代入
§1.3.3	`+=`	配列の追加
§1.3.4	`+=`	要素の追加
§1.4.1	`length()`	配列の長さ (個数)
§1.4.2	`at()`, `at_cs()`	指定要素の参照
§1.4.3	`copy()`	自身の配列 (の一部) を外部オブジェクトへコピー
§1.4.4	`swap()`	自身の配列と外部オブジェクトの配列との入替え
§1.4.5	`init()`	オブジェクトの完全初期化
§1.4.6	`assign()`	オブジェクトの初期化と代入
§1.4.7	`put()`	任意の要素位置へ要素を n 個セット
§1.4.8	`append()`	要素の追加
§1.4.9	`insert()`	要素の挿入
§1.4.10	`replace()`	要素の置換
§1.4.11	`erase()`	要素の削除
§1.4.12	`clean()`	既存の配列の要素値すべてを任意の値でパディング
§1.4.13	`resize()`	配列の長さを変更
§1.4.14	`resizeby()`	配列の長さを相対的に変更
§1.4.15	`crop()`	配列の切り抜き

表 1: tarray テンプレートクラスで利用可能なメンバ関数一覧.

1.3 演算子

1.3.1 []

NAME

`[]` — 指定要素の参照

SYNOPSIS

```
type &operator[]( size_t index );  ..... 1
const type &operator[]( size_t index ) const;  ..... 2
```

DESCRIPTION

添え字で指定された配列要素の参照を返します.「`[]`」の直後に「`.`」で接続し，type クラスのメンバ関数を使う事ができます

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 §1.4.2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)

1.3.2 =

NAME

= — 同じ型・クラスを扱う配列を代入

SYNOPSIS

```
tarray &operator=(const tarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定された同じ型・クラスを扱う配列を自身に代入します.

PARAMETER

[I] `obj` 同じ型・クラスを扱う配列を持つオブジェクト

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

1.3.3 +=

NAME
+= — 配列の追加

SYNOPSIS
```
tarray &operator+=(const tarray &obj);
```

DESCRIPTION
自身の配列に，演算子の右側 (引数) で指定された配列の追加を行います.

PARAMETER
[I] `obj` 同じ型・クラスを扱う配列を持つオブジェクト

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

1.3.4 +=

NAME
+= — 要素の追加

SYNOPSIS
```
tarray &operator+=(const type &one);
```

DESCRIPTION
自身の配列に，演算子の右側 (引数) で指定された 1 要素を追加します.

PARAMETER
[I] `one` 追加する要素

RETURN VALUE
自身の参照

EXCEPTION
内部バッファの確保に失敗した場合

1.4 メンバ関数

1.4.1 length()

NAME

length() — 配列の長さ (個数)

SYNOPSIS

```
size_t length() const;
```

DESCRIPTION

配列の長さ (個数) を返します.

RETURN VALUE

配列数

1.4.2 at(), at_cs()

NAME

at(), at_cs() — 指定要素の参照

SYNOPSIS

```
type &at( size_t index );  ..... 1
const type &at( size_t index ); const  ..... 2
const type &at_cs( size_t index ) const;  ..... 3
```

DESCRIPTION

index で指定された配列要素の参照を返します. これらメンバ関数の直後に「.」で接続し, type クラスのメンバ関数を使う事ができます

メンバ関数 1 は, 値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値または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)

1.4.3 copy()

NAME

copy() — 自身の配列 (の一部) を別の配列にコピー

SYNOPSIS

```
ssize_t copy( tarray *dest ) const;  ..... 1
ssize_t copy( size_t index, tarray *dest ) const;  ..... 2
ssize_t copy( size_t index, size_t n, tarray *dest ) const;  ..... 3
ssize_t copy( tarray &dest ) const;  ..... 4
ssize_t copy( size_t index, tarray &dest ) const;  ..... 5
ssize_t copy( size_t index, size_t n, tarray &dest ) const;  ..... 6
```

DESCRIPTION

自身の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．

引数 dest は，ポインタ変数 (メンバ関数 1〜3) と参照 (メンバ関数 4〜6) の 2 種類を用意していますが，動作はどちらも同じです．ポインタ変数を使うか参照を使うかは，ユーザのポリシーで決めてください．

メンバ関数 1,4 は，配列要素すべてを dest へコピーします．

メンバ関数 2,3 およびメンバ関数 5,6 は，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 3,6 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，要素番号 index 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 クラスのオブジェクト
[I]	index	コピー元オブジェクトの配列の開始要素番号
[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

1.4.4 swap()

NAME

swap() — 自身の配列と外部オブジェクトの配列との入替え

SYNOPSIS

```
tarray &swap( tarray &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 クラスのオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

1.4.5 init()

NAME

init() — オブジェクトの完全初期化

SYNOPSIS

```
tarray &init();  ..... 1
tarray &init(const tarray &obj);  ..... 2
```

DESCRIPTION

自身の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，配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ます.

メンバ関数 2 は，指定されたオブジェクト `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 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)

1.4.6 assign()

NAME

assign() — オブジェクトの初期化と代入

SYNOPSIS

```
tarray &assign( const type &one, size_t n );  ..... 1
tarray &assign( const tarray &src, size_t idx2 = 0 );  ..... 2
tarray &assign( const tarray &src, size_t idx2, size_t n2 );  ..... 3
```

DESCRIPTION

メンバ関数 1 は，自身の配列を one で指定された値を持つ要素 n 個で初期化します.

メンバ関数 2，3 は，配列 src の要素番号 idx2 から n2 個の要素を，自身の配列に代入します.

PARAMETER

[I] one 源泉となる値
[I] n 値 one を書き込む個数
[I] src 源泉となる配列要素を持つオブジェクト
[I] idx2 src の要素の開始番号
[I] n2 src から取り出す要素の個数
([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

1.4.7 put()

NAME

put() — 任意の要素位置へ値をセット

SYNOPSIS

```
tarray &put( size_t index, const type &one, size_t n );  ..................  1
tarray &put( size_t index, const tarray &src, size_t idx2 = 0 );  .........  2
tarray &put( size_t index, const tarray &src, size_t idx2, size_t n2 );  .  3
```

DESCRIPTION

メンバ関数 1 は，自身の配列の要素番号 index の位置から，one で指定された値を持つ要素 n 個で上書きします.

メンバ関数 2，3 は，自身の配列の要素番号 index の位置から，配列 src の要素番号 idx2 から n2 個の要素を上書きします.

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,value,6) と my_arr.put(2,value,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,value,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 value で指定されたものとなります.

PARAMETER

[I] index 自身の配列の書き込み位置
[I] one 源泉となる値
[I] n 値 one を書き込む個数
[I] src 源泉となる配列要素を持つオブジェクト
[I] idx2 src の要素の開始番号
[I] n2 src から取り出す要素の個数
([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

1.4.8 append()

NAME
append() — 要素の追加

SYNOPSIS
```
tarray &append( const type &one, size_t n );  ..... 1
tarray &append( const tarray &src, size_t idx2 = 0 );  ..... 2
tarray &append( const tarray &src, size_t idx2, size_t n2 );  ..... 3
```

DESCRIPTION
メンバ関数 1 は，自身の配列の最後に，one で指定された値を持つ要素 n 個を追加します.

メンバ関数 2，3 は，自身の配列の最後に，配列 src の要素番号 idx2 から n2 個の要素を追加します.

なお，配列の先頭の要素番号は 0 です.

PARAMETER
- [I] one 源泉となる値
- [I] n 値 one を書き込む個数
- [I] src 源泉となる配列要素を持つオブジェクト
- [I] idx2 src の要素の開始番号
- [I] n2 src から取り出す要素の個数

([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

1.4.9 insert()

NAME
insert() — 要素の挿入

SYNOPSIS
```
tarray &insert( size_t index, const type &one, size_t n );  ..... 1
tarray &insert( size_t index,
                const tarray &src, size_t idx2 = 0 );  ..... 2
tarray &insert( size_t index,
                const tarray &src, size_t idx2, size_t n2 );  ..... 3
```

DESCRIPTION

メンバ関数 1 は，自身の配列の要素番号 index の位置に，one で指定された値を持つ要素 n 個を挿入します．

メンバ関数 2，3 は，自身の配列の要素番号 index の位置に，配列 src の要素番号 idx2 から n2 個の要素を挿入します．

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．

PARAMETER

[I]	index	自身の配列の挿入位置
[I]	one	源泉となる値
[I]	n	値 one を書き込む個数
[I]	src	源泉となる配列要素を持つオブジェクト
[I]	idx2	src の要素の開始番号
[I]	n2	src から取り出す要素の個数

([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

1.4.10 replace()

NAME

replace() — 要素の置換

SYNOPSIS

```
tarray &replace( size_t idx1, size_t n1,
                 const type &one, size_t n );  ..............................  1
tarray &replace( size_t idx1, size_t n1,
                 const tarray &src, size_t idx2 = 0 );  .....................  2
tarray &replace( size_t idx1, size_t n1,
                 const tarray &src, size_t idx2, size_t n2 );  .............  3
```

DESCRIPTION

メンバ関数 1 は，自身の配列の要素番号 idx1 から n1 個の要素を，one で指定された値を持つ要素 n 個で置換します．

メンバ関数 2 は，自身の配列の要素番号 idx1 から n1 個の要素を，配列 src の要素番号 idx2 から n2 個の要素に置き換えます．

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 (§1.4.8) 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．

PARAMETER

[I] `idx1` 自身の配列の開始位置
[I] `n1` 置換される要素数
[I] `one` 源泉となる値
[I] `n` 値 `one` を書き込む個数
[I] `src` 源泉となる配列要素を持つオブジェクト
[I] `idx2` `src` の要素の開始番号
[I] `n2` `src` から取り出す要素の個数
([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

1.4.11 erase()

NAME

erase() — 要素の削除

SYNOPSIS

```
tarray &erase();  ..... 1
tarray &erase( size_t index, size_t num_elements = 1 );  ..... 2
```

DESCRIPTION

メンバ関数 1 は，自身の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.

`index` に配列の長さ以上の値が指定された場合，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

1.4.12 clean()

NAME

clean() — 既存の配列の要素値すべてを任意の値でパディング

SYNOPSIS

```
tarray &clean(const type &one);
```

DESCRIPTION

自身が持つ配列の全要素を値 one でパディングします. clean() を実行しても配列長は変化しません.

PARAMETER

[I] one 配列をパディングするための値

([I]：入力, [O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

要素内バッファの確保に失敗した場合

1.4.13 resize()

NAME

resize() — 配列の長さを変更

SYNOPSIS

```
tarray &resize( size_t new_num_elements );
```

DESCRIPTION

自身の配列の長さを new_num_elements に変更します. 配列長を収縮する場合, new_num_elements 以降の要素は削除されます.

PARAMETER

[I] new_num_elements 変更後の配列長

([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

1.4.14 resizeby()

NAME

resizeby() — 配列の長さを相対的に変更

SYNOPSIS

```
tarray &resizeby( ssize_t len );
```

DESCRIPTION

自身の配列の長さを `len` の長さ分だけ変更します.

配列長を収縮する場合，最後の abs(`len`) 個の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

1.4.15 crop()

NAME

crop() — 配列の切り抜き

SYNOPSIS

```
tarray &crop( size_t idx, size_t len );
tarray &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

---

2 ASARRAY テンプレートクラス

asarray テンプレートクラスは，任意の型・クラスの連想配列を扱う事ができるクラスです．

機能的には，基本編の asarray_tstring とほとんど同じで，文字列に特化したメンバ関数を除いたものと言えるでしょう．ですので，この章ではコードの EXAMPLE は省略しています．EXAMPLE については，基本編の asarray_tstring のそれぞれのメンバ関数のリファレンスをご覧ください．

asarray テンプレート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/asarray.h>
```

また，namespace 宣言が必要な場合は,「`using namespace sli;`」もコードに書いてください．

2.1 オブジェクトの作り方

asarray テンプレートクラスのオブジェクトを作る時には，引数は必要ありません．次のように普通にオブジェクトを作成します．

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

int main()
{
    asarray<asarray_tstring> my_2d_aarray;
```

この例では，基本編の asarray_tstring クラスを配列化しています．これにより，次のように文字列の 2 次元連想配列が簡単に扱えるようになります．

```
    my_2d_aarray["CONFIG"]["OS"] = "Linux";
```

2.2 メンバ関数一覧

表 2 はメンバ関数一覧です．

	メンバ関数名	機能
§2.3.1	`[]`	指定されたキーに該当する要素値の参照
§2.3.2	`=`	オブジェクトのコピー
§2.4.1	`length()`	連想配列の長さ (個数)
§2.4.2	`at()`, `atf()`	指定されたキー，または要素番号に該当する要素値の参照
§2.4.3	`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の参照 (読み取り専用)
§2.4.4	`index()`	キー文字列に該当する要素番号を取得
§2.4.5	`key()`	要素番号に該当するキー文字列を取得
§2.4.6	`keys()`	キー文字列の配列オブジェクトを参照 (読み取りのみ)
§2.4.7	`values()`	値の配列オブジェクトを参照 (読み取りのみ)
§2.4.8	`swap()`	オブジェクトの入替え
§2.4.9	`init()`	オブジェクトの完全初期化
§2.4.10	`assign()`	オブジェクトの初期化と要素の代入
§2.4.11	`assign_keys()`	複数の文字列または文字列配列をキーに設定
§2.4.12	`split_keys()`	文字列を分割し，キーに設定
§2.4.13	`append()`	要素を追加
§2.4.14	`insert()`	要素を挿入
§2.4.15	`erase()`	要素を削除
§2.4.16	`clean()`	既存の連想配列の要素値すべてを任意の値でパディング
§2.4.17	`rename_a_key()`	キー文字列の変更

表 2: asarray テンプレートクラスで利用可能なメンバ関数一覧.

2.3 演算子

2.3.1 []

NAME

`[]` — 指定されたキーに該当する要素値の参照

SYNOPSIS

```
type &operator[]( const char *key );  ..... 1
const type &operator[]( const char *key ) const;  ..... 2
```

DESCRIPTION

キーに対応する連想配列の要素値の参照を返します.「`[]`」の直後に「`.`」で接続し，type クラスのメンバ関数を使う事ができます

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デフォルト値 (type クラスに依存) 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 §2.4.2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)

2.3.2 =

NAME

= — asarray クラスのオブジェクトのコピー

SYNOPSIS

```
asarray &operator=(const asarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定された asarray クラスのオブジェクトを自身に代入します.

PARAMETER

[I] `obj` asarray 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

2.4 メンバ関数

2.4.1 length()

NAME
length() — 連想配列の長さ (個数)

SYNOPSIS
```
size_t length() const;
```

DESCRIPTION
自身の連想配列の長さ (個数) を返します.

RETURN VALUE
連想配列の要素数

2.4.2 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の参照

SYNOPSIS
```
type &at( const char *key );  ..... 1
type &atf( const char *fmt, ... );  ..... 2
type &vatf( const char *fmt, va_list ap );  ..... 3
type &at( size_t index );  ..... 4
const type &at( const char *key ) const;  ..... 5
const type &atf( const char *fmt, ... ) const;  ..... 6
const type &vatf( const char *fmt, va_list ap ) const;  ..... 7
const type &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) に対応する連想配列の要素値の参照を返します.

これらメンバ関数の直後に「.」で接続し，type クラスのメンバ関数を使う事ができますメンバ関数 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，libc の printf() 関数のマニュアル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デフォルト値 (type クラスに依存) 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)

2.4.3 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の参照 (読み取り専用)

SYNOPSIS

```
const type &at_cs( const char *key ) const;  ..................................  1
const type &atf_cs( const char *fmt, ... ) const;  ..........................  2
const type &vatf_cs( const char *fmt, va_list ap ) const;  ..................  3
const type &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) に対応する連想配列の要素値の参照を返します．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，libc の printf() 関数のマニュアル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)

2.4.4 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，libc の printf() 関数のマニュアル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)

2.4.5 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

2.4.6 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; 基本編マニュアル参照) 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

2.4.7 values()

NAME

values() — 値の配列オブジェクトを参照 (読み取りのみ)

SYNOPSIS

```
const tarray<type> &values() const;
```

DESCRIPTION

オブジェクト内で管理している値の配列オブジェクト (tarray テンプレートクラス; §1) の参照を返します．このメンバ関数の直後に「.」で接続し，tarray テンプレートクラスのメンバ関数を使う事ができます．利用可能な tarray テンプレートクラスのメンバ関数は，値を改変しないもの，すなわち const 属性を有するものに限ります．

RETURN VALUE

値の配列オブジェクト (tarray テンプレートクラス) の参照

2.4.8 swap()

NAME

swap() — オブジェクトの入替え

SYNOPSIS

```
asarray &swap( asarray &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 クラスのオブジェクト
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

2.4.9 init()

NAME

init() — オブジェクトの完全初期化

SYNOPSIS

```
asarray &init();  ..... 1
asarray &init( const asarray &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，自身を完全に初期化します．連想配列の配列バッファや要素等に割り当てられたメモリ領域は完全に開放されます．

メンバ関数 2 は，自身を 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 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)

2.4.10 assign()

NAME

assign() — オブジェクトの初期化と要素の代入

SYNOPSIS

```
asarray &assign( const char *key, const type &val );  ..... 1
asarray &assign( const asarray &src );  ..... 2
```

DESCRIPTION

メンバ関数 1 は，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 2 は，指定されたオブジェクト src の内容を，自身の連想配列に代入します.

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] src asarray 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

2.4.11 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 &assign_keys( const char *key0, ... );  ..... 1
```

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．tarray_tstring クラスについては，基本編マニュアルを参照してください．

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

2.4.12 split_keys()

NAME

split_keys() — 文字列を分割し，キーに設定

SYNOPSIS

```
asarray &split_keys( const char *src_str, const char *delims,
                     bool zero_str, const char *quotations,
                     int escape, bool rm_escape ); ..... 1
asarray &split_keys( const char *src_str, const char *delims,
                     bool zero_str = false ); ..... 2
asarray &split_keys( const tstring &src_str, const char *delims,
                     bool zero_str, const char *quotations,
                     int escape, bool rm_escape ); ..... 3
asarray &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 delims で分割し，自身の連想配列のキーに入れます．delims 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，基本編マニュアルの tstring クラスの trim() メンバ関数の解説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 (§2.4.11) 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

2.4.13 append()

NAME

append() — 要素を追加

SYNOPSIS

```
asarray &append( const char *key, const type &val );  ..... 1
asarray &append( const asarray &src );  ..... 2
```

DESCRIPTION

メンバ関数 1 は，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 2 は，指定されたオブジェクト src の内容を，自身の連想配列に代入します.

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` 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] `src` asarray 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

2.4.14 insert()

NAME

insert() — 要素を挿入

SYNOPSIS

```
asarray &insert( const char *key,
                 const char *newkey, const type &newval ); ..... 1
asarray &insert( const char *key, const asarray &src ); ..... 2
```

DESCRIPTION

メンバ関数 1 は，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 2 は，指定されたオブジェクト `src` の内容を，自身の連想配列のキー `key` の要素位置の前に挿入します.

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] `src` asarray 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

2.4.15 erase()

NAME

erase() — 要素 (キーと値のセット) の削除

SYNOPSIS

```
asarray &erase();  ..... 1
asarray &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

2.4.16 clean()

NAME

clean() — 既存の連想配列の要素値すべてを任意の値でパディング

SYNOPSIS

```
asarray &clean(const type &one);
```

DESCRIPTION

自身の連想配列の要素値すべてを値 `one` でパディングします．`clean()` を実行してもキーと配列長は変化しません.

PARAMETER

[I] `str` 連想配列の全要素をパディングする値
([I]：入力，[O]：出力)

RETURN VALUE

自身の参照

EXCEPTION

要素のバッファの確保に失敗した場合

2.4.17 rename_a_key()

NAME

rename_a_key() — キー文字列の変更

SYNOPSIS

```
asarray &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

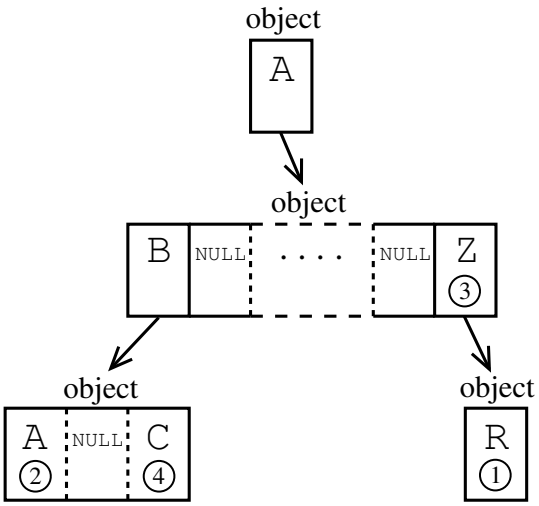
EXCEPTION

内部バッファの確保に失敗した場合

---

3 CTINDEX クラス

ctindex クラスは，キー文字列とインデックスとの関係を管理するクラスです．オブジェクトに文字列 (キー) と数字 (インデックス) との関係を登録すれば，キー文字列からインデックスを高速で取り出す事ができます．働きとしてはハッシュに似ていますが，キー文字列とインデックスとの関係は辞書で管理するため，コリジョンが発生しません．



key	index
"AZR"	1
"ABA"	2
"AZ"	3
"ABC"	4

図 1: ctindex クラスの動作原理．各ノードが ctindex クラスのオブジェクトで，ツリー構造をなす．丸付きの数字が保存されているインデックス．右の表は登録されているキー文字列とインデックスの組合せを示す．

図 1に ctindex クラスの動作原理を示します．図のように，キー文字列を登録すると複数の ctindex オブジェクトがツリー構造を作り，各ノード (オブジェクト) のバッファにインデックスを保存しています．したがって，キー文字列からインデックスを取り出すまでの時間は，例外なく文字列の長さに比例します．

ctindex 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/ctindex.h>
```

また，namespace 宣言が必要な場合は,「`using namespace sli;`」もコードに書いてください．

3.1 オブジェクトの作り方

ctindex クラスのオブジェクトを作る時には，引数は必要ありません．次のように普通にオブジェクトを作成します．

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

int main()
{
    ctindex my_index;
```

3.2 メンバ関数一覧

表 3 はメンバ関数一覧です.

	ctindex クラス	機能
§3.3.1	`=`	オブジェクトを代入
§3.4.1	`init()`	オブジェクトの完全初期化
§3.4.2	`append()`	キーとインデックスの追加
§3.4.3	`update()`	インデックスの更新
§3.4.4	`erase()`	キーとインデックスの削除
§3.4.5	`index()`	インデックスの取得

表 3: ctindex クラスで利用可能なメンバ関数一覧.

3.3 演算子

3.3.1 =

NAME

= — ctindex オブジェクトを代入

SYNOPSIS

```
ctindex &operator=(const ctindex &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した ctindex オブジェクトを代入をします.

PARAMETER

[I] `obj` ctindex クラスのオブジェクト

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，ctindex オブジェクト `my_idx` を `my_idx1` に代入し，キー文字列 `rose`，`cosmos` のインデックスを標準出力します．`index()`，`append()` に関しては §3.4.5，§3.4.2の解説を参照してください.

```
stdstreamio sio;

ctindex my_idx;
ctindex my_idx1;

my_idx.append("rose",0);
my_idx.append("cosmos",1);

my_idx1 = my_idx;
sio.printf("my_idx1.index... [%zd]\n", my_idx1.index("rose"));
sio.printf("my_idx1.index... [%zd]\n", my_idx1.index("cosmos"));
```

実行結果

```
my_idx1.index...  [0]
my_idx1.index...  [1]
```

3.4 メンバ関数

全体的な注意事項

1 つのキー文字列に対して，複数のインデックスを割り当てる事ができます．但し，その場合，

当該キー文字列については，インデックスの個数に比例して検索時のパフォーマンスが低下します.

インデックスを取得する際，該当キー文字列がない場合は，-1 が返ります.

キー文字列からインデックスを取得することはできますが，インデックスからキー文字列を取得することはできません.

3.4.1 init()

NAME

init() — オブジェクトの完全初期化

SYNOPSIS

```
ctindex &init();  ..... 1
ctindex &init(const ctindex &obj);  ..... 2
```

DESCRIPTION

オブジェクトの初期化を行います.

メンバ関数 1 は，オブジェクトを初期化し，オブジェクト内の辞書バッファに割り当てられたメモリ領域を完全に開放します.

メンバ関数 2 は ctindex クラスのオブジェクト obj の内容で初期化をします.

PARAMETER

[I] obj ctindex 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 (メンバ関数 2)

EXAMPLE

次のコードは，ctindex クラスのオブジェクトの my_idx で my_idx1 を初期化し，インデックスを標準出力します. index()，append() に関しては §3.4.5, §3.4.2の解説を参照してください.

```
stdstreamio sio;

ctindex my_idx;
ctindex my_idx1;

my_idx.append("Morning glory",0);
my_idx.append("Gentiana",1);

my_idx1.init(my_idx);
sio.printf("my_idx1.index... [%zd]\n", my_idx1.index("Morning glory"));
sio.printf("my_idx1.index... [%zd]\n", my_idx1.index("Gentiana"));
```

実行結果

```
my_idx1.index...  [0]
my_idx1.index...  [1]
```

3.4.2 append()

NAME

append() — 一対のキーとインデックスの追加

SYNOPSIS

```
int append( const char *key, size_t index );
```

DESCRIPTION

ctindex オブジェクトに，キーとなる文字列 key と対応するインデックス index を追加します.

PARAMETER

[I] key　キーとなる文字列
[I] index　キーに対応する任意のインデックス値
([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	同一のキーとインデックスの組合せが存在する場合
		キーが 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，ctindex オブジェクトの my_idx に"Pansy",2 と"Violet",4 を追加し，確認の為，"Pansy"と"Violet"のインデックス値を標準出力します．index() に関しては §3.4.5の解説を参照してください.

```
stdstreamio sio;

ctindex my_idx;

my_idx.append("Pansy",2);
my_idx.append("Violet",4);
sio.printf("my_idx.index... [%zd]\n", my_idx.index("Pansy"));
sio.printf("my_idx.index... [%zd]\n", my_idx.index("Violet"));
```

実行結果

```
my_idx.index...  [2]
my_idx.index...  [4]
```

3.4.3 update()

NAME

update() — インデックスの更新

SYNOPSIS

```
int update( const char *key, size_t current_index, size_t new_index );
```

DESCRIPTION

キーとなる文字列 key に対応するインデックス値を変更します.

PARAMETER

[I]	key	キーとなる文字列
[I]	current_index	現在のインデックス値
[I]	new_index	変更後のインデックス値

([I]：入力，[O]：出力)

RETURN VALUE

0	:	正常終了
負の値 (エラー)	:	キーが NULL の場合

EXAMPLE

次のコードは，ctindex オブジェクトの my_idx のキー"Pansy"のインデックス値を 2 から 1 へ変更し，確認の為，処理の前後のインデックス値を標準出力します．index() に関しては §3.4.5の解説を参照してください.

```
stdstreamio sio;

ctindex my_idx;

my_idx.append("Pansy",2);
my_idx.append("Violet",4);
sio.printf("my_idx.index... [%zd]\n", my_idx.index("Pansy"));

my_idx.update("Pansy", 2, 1);
sio.printf("my_idx.index... [%zd]\n", my_idx.index("Pansy"));
```

実行結果

```
my_idx.index...  [2]
my_idx.index...  [1]
```

3.4.4 erase()

NAME

erase() — 一対のキーとインデックスの削除

SYNOPSIS

```
int erase( const char *key, size_t index );
```

DESCRIPTION

キーとなる文字列 key と対応するインデックス値 index を削除します.

PARAMETER

[I] key キーとなる文字列
[I] index インデックス値
([I]：入力，[O]：出力)

RETURN VALUE

0 以上の値 (正常終了) ： 正常に削除処理が行なえた場合
負の値 (エラー) ： 指定されたキーとインデックスの組合せが存在しない場合
指定されたキーが存在しない場合
指定されたキーが 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，ctindex オブジェクトの my_idx のキー「"Pansy"」値「2」の組合せを削除します．確認の為，削除後にインデックス値を標準出力します．index()，append() に関しては §3.4.5, §3.4.2の解説を参照してください.

```
stdstreamio sio;

my_idx.append("Pansy",2);
my_idx.append("Violet",4);
my_idx.append("Pansy",6);

my_idx.erase("Pansy", 2);
sio.printf("my_idx.index... [%zd]\n", my_idx.index("Pansy"));
```

実行結果

```
my_idx.index...  [6]
```

3.4.5 index()

NAME

index() — インデックス値の取得

SYNOPSIS

```
ssize_t index( const char *key, int index_of_index = 0 ) const;
```

DESCRIPTION

キーとなる文字列 key のインデックス値を取得します.

キー文字列 key が複数のインデックス値を持つ場合，n 番目のインデックス値を取得する場合は，index_of_index に n を設定します．index_of_index の開始番号は 0 です.

PARAMETER

[I]	`key`	キーとなる文字列
[I]	`index_of_index`	インデックスの番号

([I]：入力，[O]：出力)

RETURN VALUE

正の値 (正常終了) ： インデックス値
負の値 (エラー) ： キーに該当するインデックスが存在しない場合

EXAMPLE

次のコードは，ctindex オブジェクトの `my_idx` に同一キーを異なるインデックス値で登録し，インデックス値を標準出力します．`append()` に関しては §3.4.2の解説を参照してください．

```
stdstreamio sio;

ctindex my_idx;

my_idx.append("Pansy",2);
my_idx.append("Violet",4);
my_idx.append("Pansy",6);

int i = 0 ;
while ( 0 < my_idx.index("Pansy",i) ) {
    sio.printf("my_idx.index[Pansy]... [%zd]\n", my_idx.index("Pansy",i));
    i++;
}
```

実行結果

```
my_idx.index[Pansy]...  [2]
my_idx.index[Pansy]...  [6]
```

4 MDARRAY クラス

mdarray クラスは，C 言語の「型」や構造体の多次元配列を簡単に扱えるクラスです.
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.
- 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).

mdarray 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>
using namespace sli;

int main()
{
    stdstreamio sio;
    mdarray my_mdarr(INT_ZT);    /* int 型の配列を扱う */

    my_mdarr.i(0) = 10;          /* 配列サイズは自動調整され，値が代入される */
    my_mdarr.i(1) = 20;          /* int 型なので「.i()」でアクセスする */

    my_mdarr.i(0,1) = 30;        /* ここで 2x2 (2 次元) の配列となる */

    return 0;
}
```

4.1 オブジェクトの作り方

次のコンストラクタを用いて mdarray クラスのオブジェクトを作成します.

自動リサイズモード

手動リサイズモード

メンバ関数 3〜5 はそれぞれ，1 次元，2 次元，3 次元の配列を作る時に使います．メンバ関数 6 は，n 次元の配列を使います．メンバ関数 1，2 の場合は，オブジェクト作成直後は次元数 0，配列長 0 ですが，要素にアクセスすると自動的に次元数と配列サイズが拡張されます．

引数 `sz_type` には，C 言語の型を表す整数 (型種別) を与えます．この整数は,「sli/size_types.h」の中で定数が定義されているので，それを用います (表 4)．コードでは実際の値を直接書かずに，定数名を使ってください．

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 (**未実装:使用不可**)
dcomplex	DCOMPLEX_ZT	-15	倍精度複素数型 (**未実装:使用不可**)
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

表 4: 「sli/size_types.h」の中で定義されている定数一覧.

構造体の配列を扱いたい場合は，`sz_type` に構造体のバイトサイズを与えます．なお，引数 `sz_type` を与えなかった場合 (すなわちメンバ関数 1 の場合) は，`UCHAR_ZT` を与えたものとみなされます．

mdarray クラスには 2 種類の動作モード,「自動リサイズモード」「手動リサイズモード」があります.「自動リサイズモード」では，`assign()` 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).

動作モードは，コンストラクタあるいは，init() メンバ関数 (§4.5.24) でオブジェクトを初期化する時に決まります．初期化時に，配列の個数を与えなかった場合は「自動リサイズモード」，配列の個数を与えた場合は「手動リサイズモード」で動作します．

4.2 数学関数

表 5 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 5: 利用可能な数学関数一覧.

4.3 メンバ関数一覧

表 6 はメンバ関数一覧です.

	メンバ関数名	機能	動作モードサポート
§4.4.1	=	配列を代入 (オブジェクトの初期化)	
§4.4.2	=	スカラー値を代入	
§4.4.3	+=	自身へ配列を加算	○
§4.4.4	+=	自身へスカラー値を加算	
§4.4.5	-=	自身から配列を減算	○
§4.4.6	-=	自身からスカラー値を減算	
§4.4.7	*=	自身へ配列を乗算	○
§4.4.8	*=	自身へスカラー値を乗算	
§4.4.9	/=	自身から配列を除算	○
§4.4.10	/=	自身からスカラー値を除算	
§4.4.11	+	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	
§4.4.12	+	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	
§4.4.13	-	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	
§4.4.14	-	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	
§4.4.15	*	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	
§4.4.16	*	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	
§4.4.17	/	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	
§4.4.18	/	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	
§4.4.19	==	比較	
§4.4.20	!=	比較 (否定形)	
§4.5.1	size_type()	型を表す整数 (型種別)	
§4.5.2	bytes()	1 要素のバイト数	
§4.5.3	dim_length()	配列の次元数	
§4.5.4	length()	(各次元の) 要素の個数	
§4.5.5	byte_length()	(各次元の) 配列の総バイト長	
§4.5.6	col_length()	配列の列の長さ	
§4.5.7	row_length()	配列の行の長さ	
§4.5.8	layer_length()	配列のレイヤ数	
§4.5.9	f(), ...	指定された 1 要素値の参照	○
§4.5.10	f_cs(), ...	指定された 1 要素値の読み出し	
§4.5.11	dvalue()	double 型に変換した 1 要素の値	
§4.5.12	lvalue(), llvalue()	long 型，long long 型に変換した 1 要素の値	
§4.5.13	default_value_ptr() assign_default()	サイズ拡張時の初期値の取得・設定	
§4.5.14	auto_resize(), set_auto_resize()	リサイズモードの取得・設定	

表 6: mdarray クラスで利用可能なメンバ関数一覧.

	メンバ関数名	機能	動作モードサポート
§4.5.15	rounding() set_rounding()	四捨五入の可否の取得・設定	
§4.5.16	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)	
§4.5.17	data_ptr()	指定要素のアドレスの取得	
§4.5.18	register_extptr()	ユーザポインタ変数の登録	
§4.5.19	get_elements()	自身の配列をユーザ・バッファへコピー	
§4.5.20	put_elements()	ユーザ・バッファの配列を自身へコピー	
§4.5.21	getdata()	自身の配列をユーザ・バッファへコピー	
§4.5.22	putdata()	ユーザ・バッファの配列を自身へコピー	
§4.5.23	reverse_endian()	必要に応じてエンディアンを反転	
§4.5.24	init()	配列の初期化	
§4.5.25	assign()	1 要素へ値を代入 (高レベル)	○
§4.5.26	put()	任意の要素位置へ値をセット	○
§4.5.27	swap()	要素間での値の入れ替え	
§4.5.28	move()	要素間での値のコピー	
§4.5.29	cpy()	要素間での値のコピー (自動拡張あり)	
§4.5.30	insert()	要素の挿入	
§4.5.31	crop()	要素の切り出し	
§4.5.32	erase()	要素の削除	
§4.5.33	resize()	配列の長さを変更	
§4.5.34	resizeby()	配列の長さを変更	
§4.5.35	increase_dim()	次元数の拡張	
§4.5.36	decrease_dim()	次元数の縮小	
§4.5.37	swap()	別オブジェクトとの内容の入れ替え	
§4.5.38	convert()	配列の型変換	
§4.5.39	ceil()	全要素値に対し小数部を切り上げる	
§4.5.40	floor()	全要素値に対し小数部を切り下げる	
§4.5.41	round()	全要素値に対し四捨五入	
§4.5.42	trunc()	全要素値に対し小数部を切り捨て	
§4.5.43	abs()	全要素値に対し絶対値をとる	
§4.5.44	compare()	配列オブジェクトの比較	

表 6: 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
§4.5.45	`copy()`	配列を別オブジェクトへコピー	
§4.5.46	`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)	
§4.5.47	`cut()`	配列の全値を切り出す	
§4.5.48	`cut()`	配列の一部を切り出す (画像データ向き)	
§4.5.49	`clean()`	要素をデフォルト値でパディング (画像データ向き)	
§4.5.50	`fill()`	要素値の書換え (画像データ向き)	
§4.5.51	`add()`	要素値の加算 (画像データ向き)	
§4.5.52	`multiply()`	要素値の乗算 (画像データ向き)	
§4.5.53	`paste()`	配列オブジェクトの貼り付け (画像データ向き)	
§4.5.54	`add()`	配列オブジェクトの加算 (画像データ向き)	
§4.5.55	`subtract()`	配列オブジェクトの減算 (画像データ向き)	
§4.5.56	`multiply()`	配列オブジェクトの乗算 (画像データ向き)	
§4.5.57	`divide()`	配列オブジェクトの除算 (画像データ向き)	

表 6: mdarray クラスで利用可能なメンバ関数一覧 (続き).

4.4 演算子

4.4.1 =

NAME

= — オブジェクトの初期化

SYNOPSIS

```
mdarray &operator=(const mdarray &obj);
```

DESCRIPTION

オブジェクトを初期化し，obj の内容をすべて (配列長，型など) 自身にコピーします.

PARAMETER

[I] obj mdarray 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 クラスのオブジェクト my_mdarr を area_mdarr で初期化し，その結果を標準出力します. ll(),length(), に関しては §4.5.9, §4.5.4の解説を参照してください.

```
stdstreamio sio;

mdarray my_mdarr;
mdarray area_mdarr(LLONG_ZT);
area_mdarr.ll(0) = 17090000;
area_mdarr.ll(1) = 9980000;
area_mdarr.ll(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.ll(i));
}
```

実行結果

```
my_mdarr value[0]...  [17090000]
my_mdarr value[1]...  [9980000]
my_mdarr value[2]...  [9620000]
```

4.4.2 =

NAME

= — スカラー値を代入

SYNOPSIS

```
mdarray &operator=(double v);  ..... 1
mdarray &operator=(long long v);  ..... 2
mdarray &operator=(long v);  ..... 3
mdarray &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(),c() に関しては，§4.5.4，§4.5.9の解説を参照してください．

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 2);

my_mdarr = 125;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%hhu]\n", i, my_mdarr.c(i));
}
```

実行結果

```
my_mdarr value[0]...  [125]
my_mdarr value[1]...  [125]
```

4.4.3 +=

NAME

+= — 自身へ配列を加算

SYNOPSIS

```
mdarray &operator+=(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray クラスのオブジェクトの配列を自身に加算します. 自身とは異なる型が設定されたオブジェクトが指定された場合，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 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

4.4.4 +=

NAME

+= — 自身へスカラー値を加算

SYNOPSIS

```
mdarray &operator+=(double 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 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()`,`c()` に関しては，§4.5.4，§4.5.9の解説を参照してください.

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 2);
```

```
my_mdarr = 25;
my_mdarr += 50;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%hhu]\n", i, my_mdarr.c(i));
}
```

実行結果

```
my_mdarr value[0]...  [75]
my_mdarr value[1]...  [75]
```

4.4.5 -=

NAME

−= — 自身から配列を減算

SYNOPSIS

```
mdarray &operator-=(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray クラスのオブジェクトの配列を自身から減算します. 自身とは異なる型が設定されたオブジェクトが指定された場合，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 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 クラスのオブジェクト `my_mdarr` から `subst_mdarr` を減算し，その結果を標準出力します. `length()`,`c()` に関しては，§4.5.4，§4.5.9の解説を参照してください.

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 2);
my_mdarr = 100;

mdarray subst_mdarr(UCHAR_ZT, 2);
subst_mdarr.c(0) = 10;
```

```
subst_mdarr.c(1) = 20;

my_mdarr -= subst_mdarr;
for ( size_t i=0 ; i < my_mdarr.length() ; i++ ) {
    sio.printf("my_mdarr value[%zu]... [%hhu]\n", i, my_mdarr.c(i));
}
```

実行結果

```
my_mdarr value[0]...  [90]
my_mdarr value[1]...  [80]
```

4.4.6 -=

NAME

−= — 自身からスカラー値を減算

SYNOPSIS

```
mdarray &operator-=(double v);
mdarray &operator-=(long long v);
mdarray &operator-=(long v);
mdarray &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

4.4.7 *=

NAME

∗= — 自身へ配列を乗算

SYNOPSIS

```
mdarray &operator*=(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray クラスのオブジェクトの配列を自身に乗算します．自身とは異なる型が設定されたオブジェクトが指定された場合，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 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 クラスのオブジェクト `my_mdarr` に `mdarrPlus` を乗算し，その結果を標準出力します．`length()`,`l()` に関しては，§4.5.4，§4.5.9の解説を参照してください．

```
stdstreamio sio;

mdarray my_mdarr(LONG_ZT, 2);
my_mdarr = 50;

mdarray multi_mdarr(LONG_ZT);
multi_mdarr.l(0) = 10;
multi_mdarr.l(1) = 20;
multi_mdarr.l(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.l(i));
}
```

実行結果

```
my_mdarr value[0]...  [500]
my_mdarr value[1]...  [1000]
```

4.4.8 *=

NAME

*= — 自身へスカラー値を乗算

SYNOPSIS

```
mdarray &operator*=(double 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

4.4.9 /=

NAME

/= — 自身から配列を除算

SYNOPSIS

```
mdarray &operator/=(const mdarray &obj);
```

DESCRIPTION

演算子の右側 (引数) で指定した mdarray クラスのオブジェクトの配列を自身から除算します．自身とは異なる型が設定されたオブジェクトが指定された場合，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 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 クラスのオブジェクト my_mdarr を div_mdarr で除算し，その結果を標準出力します．length(),l() に関しては，§4.5.4，§4.5.9の解説を参照してください．

```
stdstreamio sio;

mdarray my_mdarr(LONG_ZT, 2);
my_mdarr = 50;

mdarray div_mdarr(LONG_ZT);
div_mdarr.l(0) = 1;
div_mdarr.l(1) = 2;
div_mdarr.l(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.l(i));
}
```

実行結果

```
my_mdarr value[0]...  [50]
my_mdarr value[1]...  [25]
```

4.4.10 /=

NAME

/= — 自身からスカラー値を除算

SYNOPSIS

```
mdarray &operator/=(double v);
mdarray &operator/=(long long v);
mdarray &operator/=(long v);
mdarray &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

4.4.11 +

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 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，自身とは異なる型が設定された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 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

4.4.12 +

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

4.4.13 -

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 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，自身とは異なる型が設定された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 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

4.4.14 -

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

4.4.15 *

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 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，自身とは異なる型が設定された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 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

4.4.16 *

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

4.4.17 /

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 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，自身とは異なる型が設定された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 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

4.4.18 /

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

4.4.19 ==

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 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 (§4.5.44) を使っています.

PARAMETER

[I] `obj` mdarray 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 クラスのオブジェクト `my_mdarr` と `comp_mdarr` を比較し，その結果を標準出力します.

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 3);
my_mdarr = 20;

mdarray comp_mdarr(LONG_ZT);
comp_mdarr = 20;
```

```
if (my_mdarr == comp_mdarr) {
    sio.printf("true\n");
} else {
    sio.printf("false\n");
}
```

実行結果

```
false
```

4.4.20 !=

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 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 (§4.5.44) を使っています.

PARAMETER

[I] `obj` mdarray 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 クラスのオブジェクト `my_mdarr` と `comp_mdarr` を比較し，その結果を標準出力します.

```
stdstreamio sio;

mdarray my_mdarr(UCHAR_ZT, 3);
my_mdarr = 20;

mdarray comp_mdarr(LONG_ZT);
comp_mdarr = 20;
if (my_mdarr != comp_mdarr) {
    sio.printf("true\n");
} else {
    sio.printf("false\n");
}
```

実行結果

```
true
```

4.5 メンバ関数

4.5.1 size_type()

NAME

size_type() — 型を表す整数 (型種別)

SYNOPSIS

```
ssize_t size_type() const;
```

DESCRIPTION

型を表す整数 (型種別) を取得します．これにより，自身に設定された配列の型を調べる事ができます．

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 4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，オブジェクト my_mdarr を int32_t 型の要素を持つ配列として定義し，my_mdarr の size_type を標準出力します．

```
stdstreamio sio;

mdarray my_mdarr(INT32_ZT);
sio.printf("*** my_mdarr size_type... [%zd]\n", my_mdarr.size_type());
```

実行結果

```
*** my_mdarr size_type... [4]
```

4.5.2 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，オブジェクト my_mdarr を double 型の要素をもつ配列として定義し，要素のバイト長を標準出力します.

```
stdstreamio sio;

mdarray my_mdarr(DOUBLE_ZT);
sio.printf("*** my_mdarr bytes... [%zu]\n", my_mdarr.bytes());
```

実行結果

```
*** my_mdarr bytes... [8]
```

4.5.3 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 my_mdarr3dim(FLOAT_ZT, 3, 4, 5);
sio.printf("*** my_mdarr3dim dim... [%zu]\n", my_mdarr3dim.dim_length());
```

実行結果

```
*** my_mdarr3dim dim... [3]
```

4.5.4 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 my_mdarr3dim(FLOAT_ZT, 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]
```

4.5.5 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 my_mdarr3dim(FLOAT_ZT, 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]
```

4.5.6 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 my_mdarr3dim(FLOAT_ZT, 3, 4, 5);
sio.printf("*** my_mdarr3dim col... [%zu]\n", my_mdarr3dim.col_length());
```

実行結果

```
*** my_mdarr3dim col... [3]
```

4.5.7 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 my_mdarr3dim(FLOAT_ZT, 3, 4, 5);
sio.printf("*** my_mdarr3dim row... [%zu]\n", my_mdarr3dim.row_length());
```

実行結果
```
*** my_mdarr3dim row... [4]
```

4.5.8 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 my_mdarr3dim(FLOAT_ZT, 3, 4, 5);
sio.printf("*** my_mdarr3dim layer... [%zu]\n", my_mdarr3dim.layer_length());
```

実行結果
```
*** my_mdarr3dim layer... [5]
```

4.5.9 f(), d(), ld(), c(), s(), i(), l(), ll(), i16(), i32(), i64(), z(), sz(), b(), p()

NAME
f(), d(), ld(), etc. — 指定された 1 要素値の参照

SYNOPSIS

```
float &f( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
          ssize_t idx2 = MDARRAY_INDEF );  ..... 1
double &d( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..... 2
long double &ld( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..... 3
unsigned char &c( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
          ssize_t idx2 = MDARRAY_INDEF );  ..... 4
short &s( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..... 5
int &i( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..... 6
long &l( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
          ssize_t idx2 = MDARRAY_INDEF );  ..... 7
long long &ll( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..... 8
int16_t &i16( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..... 9
int32_t &i32( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
          ssize_t idx2 = MDARRAY_INDEF );  ..... 10
int64_t &i64( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..... 11
size_t &z( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..... 12
ssize_t &sz( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
          ssize_t idx2 = MDARRAY_INDEF );  ..... 13
bool &b( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..... 14
uintptr_t &p( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
           ssize_t idx2 = MDARRAY_INDEF );  ..... 15
```

メンバ関数名は同一で,「const」属性付きのものもあります．その場合，§4.5.10にあるメンバ関数と同じ動作です.

DESCRIPTION

idx0，idx1，idx2 で指定された配列要素の値を設定・取得します．設定された型によって，使用するメンバ関数が異なります.

これらのメンバ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 §4.5.33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．

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

使用するメンバ関数に従った型での要素の値の参照

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)
オブジェクト内の要素のバイトサイズが，メンバ関数の戻り値の型のバイトサイズより小さい場合

EXAMPLE

次のコードは，浮動小数点型の配列を持つオブジェクト my_fmdarr の要素に f() メンバ関数を使用して値を設定した後，f() 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 my_fmdarr(FLOAT_ZT);

my_fmdarr.f(0) = 1000.1;
my_fmdarr.f(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.f(i));
}
```

実行結果

```
my_fmdarr value[0]...  [1000.1]
my_fmdarr value[1]...  [2000.2]
```

4.5.10 f_cs(), d_cs(), ld_cs(), c_cs(), s_cs(), i_cs(), l_cs(), ll_cs(), i16_cs(), i32_cs(), i64_cs(), z_cs(), sz_cs(), b_cs(), p_cs()

NAME

f_cs(), d_cs(), ld_cs(), etc. — 指定された 1 要素値の読み出し

SYNOPSIS

```
const float &f_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                   ssize_t idx2 = MDARRAY_INDEF ) const;  ..................  1
const double &d_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
```

「const」属性付きオブジェクトの場合は，メンバ関数名の「_cs」を省略できます.

DESCRIPTION

idx0，idx1，idx2 で指定された配列要素の値を取得します．値を設定することはできません.

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.

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

使用するメンバ関数に従った型での要素の値の参照

EXCEPTION

オブジェクト内の要素のバイトサイズが，メンバ関数の戻り値の型のバイトサイズより小さい場合

EXAMPLE

次のコードは，long long 型の配列を持つオブジェクト my_llmdarr の要素に値を設定した後，各要素の値を標準出力します.

```
stdstreamio sio;

mdarray my_llmdarr(LLONG_ZT);
my_llmdarr.ll(0) = 60000;
my_llmdarr.ll(1) = 70000000;
for ( size_t i = 0 ; i < my_llmdarr.length() ; i++ ) {
    sio.printf("my_llmdarr value[%zu]... [%lld]\n", i, my_llmdarr.ll_cs(i));
}
```

実行結果

```
my_llmdarr value[0]...  [60000]
my_llmdarr value[1]...  [70000000]
```

4.5.11 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，float 型の配列を持つオブジェクト 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 my_mdarry(FLOAT_ZT);
my_mdarry.f(0) = 123.456;
sio.printf("my_mdarry dvalue... [%6.3f]\n", my_mdarry.dvalue(0));
```

実行結果

```
my_mdarry dvalue...  [123.456]
```

4.5.12 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 §4.5.15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，float 型の配列を持つオブジェクト 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 my_mdarry(FLOAT_ZT);
my_mdarry.f(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]
```

4.5.13 default_value_ptr(), assign_default()

NAME

default_value_ptr(), assign_default() — サイズ拡張時の初期値の取得・設定

SYNOPSIS

```
const void *default_value_ptr() const;  ..... 1
mdarray &assign_default( const void *value_ptr );  ..... 2
mdarray &assign_default( double value );  ..... 3
```

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．初期値が未設定の場合は，NULL を返します．

メンバ関数 2，3 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格納されているアドレスを返します (初期値が未設定の場合は NULL が返ります)．

メンバ関数 2，3 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 2，3)

EXAMPLE

次のコードは，float 型の配列を持つオブジェクト my_mdarr のサイズ拡張時の初期値を標準出力します．

```
stdstreamio sio;

mdarray my_mdarr;
my_mdarr.init(FLOAT_ZT);
my_mdarr.assign_default(50);
sio.printf("*** my_mdarr defval... [%f]\n",
            *((const float *)my_mdarr.default_value_ptr()));
```

実行結果

```
*** my_mdarr defval... [50.000000]
```

4.5.14 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 &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 6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 my_mdarr0dim;
sio.printf("*** my_mdarr0dim auto_resize... [%d]\n",
           (int)(my_mdarr0dim.auto_resize()));
mdarray my_mdarr3dim(FLOAT_ZT, 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]
```

4.5.15 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 &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．

mdarray クラスの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 §4.5.24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 (§4.5.12)，assign_default() メンバ関数 (§4.5.13)，assign() メンバ関数 (§4.5.25)，convert() メンバ関数 (§4.5.38)，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，long long 型の配列を持つオブジェクト 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 my_mdarr(LLONG_ZT);
my_mdarr.assign(1.618, 0);
sio.printf("my_mdarr value[0]... [%lld]\n", my_mdarr.ll(0));

my_mdarr.set_rounding(true);
my_mdarr.assign(1.618, 1);
sio.printf("my_mdarr value[1]... [%lld]\n", my_mdarr.ll(1));
```

実行結果

```
my_imdarr value[0]...  [1]
my_imdarr value[1]...  [2]
```

4.5.16 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 my_array(INT_ZT, 3,2,1);
my_array.i(2,0,0) = 100;
my_array.i(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 } }
}
```

4.5.17 data_ptr()

NAME

data_ptr() — 指定要素のアドレスの取得

SYNOPSIS

```
void *data_ptr();  ..... 1
void *data_ptr( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                ssize_t idx2 = MDARRAY_INDEF );  ..... 2
const void *data_ptr() const;  ..... 3
const void *data_ptr( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                      ssize_t idx2 = MDARRAY_INDEF ) const;  ..... 4
const void *data_ptr_cs( ssize_t idx0, ssize_t idx1 = MDARRAY_INDEF,
                         ssize_t idx2 = MDARRAY_INDEF ) const;  ..... 5
const void *data_ptr_cs() const;  ..... 6
```

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 3〜6 は，読み取り専用のアドレスを取得します.

data_ptr() メンバ関数の場合,「const」属性なしのメンバ関数 1,2・「const」属性ありのメンバ関数 3,4 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 1,2 が，有る場合にはメンバ関数 3,4 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 my_fmdarr(FLOAT_ZT, 2,2);
my_fmdarr.f(0,0) = 1000;
my_fmdarr.f(1,0) = 2000;
my_fmdarr.f(0,1) = 3000;
my_fmdarr.f(1,1) = 4000;

const float *mycarray_ptr = (float *)my_fmdarr.data_ptr_cs(0, 1);
sio.printf("*** my_fmdarr carray[0] ---> [%f] *** \n",  mycarray_ptr[0]);
```

実行結果

```
*** my_fmdarr carray[0] ---> [3000.000000] ***
```

4.5.18 register_extptr()

NAME

register_extptr() — ユーザポインタ変数の登録

SYNOPSIS

```
mdarray &register_extptr(void *extptr_ptr);
```

DESCRIPTION

ユーザのポインタ変数をオブジェクトに登録します.

このメンバ関数を使ってユーザのポインタ変数のアドレスを登録すれば，オブジェクトが管理するバッファの先頭アドレスを常にユーザのポインタ変数に保持させておく事ができます.

引数を与える時に，ポインタ変数に「&」をつけるのを忘れないようにしましょう (EXAMPLE を参照).

PARAMETER

[I] extptr_ptr ユーザのポインタ変数のアドレス
([I] : 入力, [O] : 出力)

RETURN VALUE

自身の参照

EXAMPLE

次のコードは，オブジェクト tokyoStock_buf に tokyoStock_ptr のアドレスを登録し，バッファを確保しています．これにより，tokyoStock_ptr を使って直接バッファにアクセスできるようになります．確認の為，tokyoStock_buf の値を標準出力します．

```
stdstreamio sio;

struct tokyoStock {
    short id;
    long stock;
};
struct tokyoStock *tokyoStock_ptr;
mdarray tokyoStock_buf(sizeof(struct tokyoStock));

tokyoStock_buf.register_extptr(&tokyoStock_ptr);    /* 「&」を忘れずに! */
tokyoStock_buf.resize(6);

tokyoStock_ptr[0].id = 3407;
tokyoStock_ptr[0].stock = 438;
tokyoStock_ptr[1].id = 4951;
tokyoStock_ptr[1].stock = 1058;
for ( size_t i = 0 ; i < tokyoStock_buf.length(0) ; i++ ) {
    sio.printf("tokyoStock_buf stock[%zu]... [%ld]\n", i,
                tokyoStock_ptr[i].stock);
}
```

実行結果

```
tokyoStock_buf stock[0]...  [438]
tokyoStock_buf stock[1]...  [1058]
```

4.5.19 get_elements()

NAME

get_elements() — 配列の内容をユーザ・バッファへコピー

SYNOPSIS

```
ssize_t get_elements( void *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;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
my_fmdarr.f(0,0) = 1000;
my_fmdarr.f(1,0) = 2000;
my_fmdarr.f(0,1) = 3000;
my_fmdarr.f(1,1) = 4000;

float myfloat[4];
my_fmdarr.get_elements((void *)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 value[0]...  [1000.000000]
myfloat value[1]...  [2000.000000]
myfloat value[2]...  [3000.000000]
myfloat value[3]...  [4000.000000]
```

4.5.20 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 void *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 my_fmdarr(FLOAT_ZT, 2,2);

float my_float[] = {1000, 2000, 3000, 4000};
my_fmdarr.put_elements((const void *)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.f(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]
```

4.5.21 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 my_fmdarr(FLOAT_ZT, 2,2);
my_fmdarr.f(0,0) = 1000;
my_fmdarr.f(1,0) = 2000;
my_fmdarr.f(0,1) = 3000;
my_fmdarr.f(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]
```

4.5.22 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 my_fmdarr(FLOAT_ZT, 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.f(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]
```

4.5.23 reverse_endian()

NAME

reverse_endian() — 必要に応じてエンディアンを反転

SYNOPSIS

```
mdarray &reverse_endian( bool is_little_endian, ssize_t sz_type = 0 );
```

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，data_ptr() メンバ関数 (§4.5.17) 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，data_ptr() メンバ関数 (§4.5.17) 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．

また，`sz_type` に型種別 (表 4を参照) を指定することによって，その型のバイト単位でのエンディアン反転処理を行うことができます．

PARAMETER

[I]	`is_little_endian`	1 回目の処理後のメモリ上のエンディアン
[I]	`sz_type`	型種別 (省略可)

([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 my_mdarr(INT_ZT, 2,2);
my_mdarr.i(0,0) = 10;
my_mdarr.i(1,0) = 20;
my_mdarr.i(0,1) = 30;
my_mdarr.i(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"
```

4.5.24 init()

NAME

init() — 配列の初期化

SYNOPSIS

```
mdarray &init();  ..... 1
mdarray &init( ssize_t sz_type );  ..... 2
mdarray &init( ssize_t sz_type, const size_t naxisx[], size_t ndim );  .... 3
mdarray &init( ssize_t sz_type, size_t naxis0 );  ..... 4
mdarray &init( ssize_t sz_type, size_t naxis0, size_t naxis1 );  ..... 5
mdarray &init( ssize_t sz_type, size_t naxis0, size_t naxis1,
```

DESCRIPTION

自身の配列を初期化します.

メンバ関数 1 と 2 は,「自動リサイズモード」で初期化します. メンバ関数 3, 4, 5 と 6 は,「手動リサイズモード」で初期化します. メンバ関数 7 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 は, 型情報はそのまま, 配列サイズ 0 としてオブジェクトを初期化します.

メンバ関数 2 は, 配列の要素の型を sz_type で指定します.

メンバ関数 3 は, sz_type で要素の型, ndim で次元数, naxisx[] で各次元の要素数を指定します.

メンバ関数 4 は, sz_type で要素の型, naxis0 で 1 次元目の要素数を指定し, 1 次元の配列を持つオブジェクトを作成します.

メンバ関数 5 は, sz_type で要素の型, naxis0 で 1 次元目の要素数, naxis1 で 2 次元目の要素数指定し, 2 次元の配列を持つオブジェクトを作成します.

メンバ関数 6 は, sz_type で要素の型, naxis0 で 1 次元目の要素数, naxis1 で 2 次元目の要素数, naxis2 で 3 次元目の要素数を指定し, 3 次元の配列を持つオブジェクトを作成します.

引数 sz_type に与える値については, 表 4を参照してください.

PARAMETER

[I]	sz_type	変数の型種別
[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]	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, int 型の 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 my_mdarr;
my_mdarr.init(INT_ZT, 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] ***
```

4.5.25 assign()

NAME

assign() — 1 要素へ値を代入 (高レベル)

SYNOPSIS

```
mdarray &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 §4.5.15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 §4.5.33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，float 型の配列を持つオブジェクト 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 my_mdarr(FLOAT_ZT);
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]
```

4.5.26 put()

NAME

put() — 任意の要素位置へ値をセット

SYNOPSIS

```
mdarray &put( const void *value_ptr, ssize_t idx, size_t len );  ..... 1
mdarray &put( const void *value_ptr,
              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，アドレス value_ptr で示された 1 つの値を 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，アドレス value_ptr で示された 1 つの値を要素番号 idx から len 個に書き込みます．

メンバ関数 2 は，アドレス value_ptr で示された 1 つの値を，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_ptr	値への間接参照
[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)
メモリ破壊を起こした場合

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;

short s12 = 12;
const void *ptr_s12 = (const void *)&s12;

mdarray my_smdarr(SHORT_ZT, 3);
my_smdarr.put(ptr_s12, 1,2);
for ( size_t i = 0 ; i < my_smdarr.length() ; i++ ) {
    sio.printf("my_smdarr value[%zu]... [%hd]\n", i, my_smdarr.s(i));
}
```

実行結果

```
my_smdarr value[0]...  [0]
my_smdarr value[1]...  [12]
my_smdarr value[2]...  [12]
```

4.5.27 swap()

NAME

swap() — 要素間での値の入れ替え

SYNOPSIS

```
mdarray &swap( ssize_t idx_src, size_t len, ssize_t idx_dst );  ..... 1
mdarray &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，unsigned char 型の配列を持つオブジェクト `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. `putdata()` に関しては §4.5.22の解説を参照してください.

```
stdstreamio sio;

mdarray my_cmdarr(UCHAR_ZT, 2, 2);
unsigned char my_char[] = {51, 52, 101, 102};
my_cmdarr.putdata((const void *)my_char, sizeof(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.c(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]
```

4.5.28 move()

NAME

move() — 要素間での値のコピー

SYNOPSIS

```
mdarray &move( ssize_t idx_src, size_t len, ssize_t idx_dst,
               bool clr );  ..................................................  1
mdarray &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 (§4.5.29) 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，unsigned char 型の配列を持つオブジェクト 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 my_cmdarr(UCHAR_ZT, 3);
my_cmdarr.c(0) = 99;
my_cmdarr.c(1) = 98;
my_cmdarr.c(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.c(i));
}
```

実行結果

```
my_cmdarr value[0]...  [97]
my_cmdarr value[1]...  [98]
my_cmdarr value[2]...  [0]
```

4.5.29 cpy()

NAME

cpy() — 要素間での値のコピー (自動拡張あり)

SYNOPSIS

```
mdarray &cpy( ssize_t idx_src, size_t len, ssize_t idx_dst,
              bool clr );  ..................................................... 1
mdarray &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，long long 型の配列を持つオブジェクト 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 my_lmdarr(LLONG_ZT);
my_lmdarr.ll(0) = -2147483646;
my_lmdarr.ll(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.ll(i));
}
```

実行結果

```
my_lmdarr value[0]...  [-2147483646]
my_lmdarr value[1]...  [2147483647]
my_lmdarr value[2]...  [2147483647]
```

4.5.30 insert()

NAME

insert() — 要素の挿入

SYNOPSIS

```
mdarray &insert( ssize_t idx, size_t len );  ..... 1
mdarray &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，long 型の配列を持つオブジェクト 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 my_mdarr(LONG_ZT, 2);
my_mdarr.l(0) = -2147483646;
my_mdarr.l(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.l(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]
```

4.5.31 crop()

NAME

crop() — 要素の切り出し

SYNOPSIS

```
mdarray &crop( ssize_t idx, size_t len );  ..................................  1
mdarray &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，unsigned char 型の配列を持つオブジェクト 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 my_cmdarr(UCHAR_ZT, 2, 3);
my_cmdarr.c(0,0) = 124;
my_cmdarr.c(1,0) = 125;
my_cmdarr.c(0,1) = 126;
my_cmdarr.c(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.c(i, j));
    }
}
```

実行結果

```
my_cmdarr value(0, 0)...  [125]
my_cmdarr value(0, 1)...  [127]
my_cmdarr value(0, 2)...  [0]
```

4.5.32 erase()

NAME

erase() — 要素の削除

SYNOPSIS

```
mdarray &erase( ssize_t idx, size_t len );  ..... 1
mdarray &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 my_mdarr(LLONG_ZT, 3);
my_mdarr.ll(0) = 0;
my_mdarr.ll(1) = 2147483646;
my_mdarr.ll(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.ll(i));
}
```

実行結果

```
my_mdarr value[0]...  [0]
my_mdarr value[1]...  [2147483647]
```

4.5.33 resize()

NAME

resize() — 配列の長さを変更

SYNOPSIS

```
mdarray &resize( size_t len );  ..... 1
mdarray &resize( size_t dim_index, size_t len );  ..... 2
mdarray &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 my_cmdarr(UCHAR_ZT, 2, 2);
my_cmdarr.c(0,0) = 70;
my_cmdarr.c(1,0) = 71;
my_cmdarr.c(0,1) = 36;
my_cmdarr.c(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.c(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]
```

4.5.34 resizeby()

NAME

resizeby() — 配列の長さを相対的に変更

SYNOPSIS

```
mdarray &resizeby( ssize_t len );  ..... 1
mdarray &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 my_cmdarr(UCHAR_ZT, 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.c(i));
}
```

実行結果

```
my_cmdarr value[0]...  [0]
```

4.5.35 increase_dim()

NAME

increase_dim() — 次元数の拡張

SYNOPSIS

```
mdarray &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 my_mdarr(UCHAR_ZT, 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]
```

4.5.36 decrease_dim()

NAME

decrease_dim() — 次元数の縮小

SYNOPSIS

```
mdarray &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 my_mdarr(UCHAR_ZT, 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]
```

4.5.37 swap()

NAME

swap() — 別オブジェクトとの内容の入れ替え

SYNOPSIS

```
mdarray &swap( mdarray &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 mdarray 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 my_fmdarr(FLOAT_ZT, 2);
my_fmdarr.f(0) = 1000;
my_fmdarr.f(1) = 2000;

mdarray swap_mdarr(DOUBLE_ZT, 2);
```

```
swap_mdarr.d(0) = 100;
swap_mdarr.d(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]
```

4.5.38 convert()

NAME

convert() — 配列の型変換

SYNOPSIS

```
mdarray &convert( ssize_t sz_type );  ..................................... 1
mdarray &convert( ssize_t sz_type, void (*func)(const void *,void *,void *),
                  void *user_ptr );  ..................................... 2
```

DESCRIPTION

自身の配列の型を sz_type が示す型へ変換します．変換前後の型によっては，桁落ちや桁あふれが発生しますので注意して下さい．

メンバ関数 2 では，ユーザ定義関数を与えて変換時の挙動を変えることができます．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 user_ptr が与えられます．

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

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_fmdarr を float 型から int 型へ変更します．確認の為，値を標準出力します．

```
stdstreamio sio;
```

```
mdarray my_fmdarr(FLOAT_ZT);

my_fmdarr.f(0) = 1000.1;
my_fmdarr.f(1) = -2000.6;
my_fmdarr.convert(INT_ZT);
for ( size_t i = 0 ; i < my_fmdarr.length() ; i++ ) {
    sio.printf("my_fmdarr value[%zu]... [%d]\n", i, my_fmdarr.i(i));
}
```

実行結果

```
my_fmdarr value[0]...  [1000]
my_fmdarr value[1]...  [-2000]
```

4.5.39 ceil()

NAME

ceil() — 浮動小数点型の全要素値の小数部の切り上げ

SYNOPSIS

```
mdarray &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 my_fmdarr(FLOAT_ZT);

my_fmdarr.f(0) = 1000.1;
my_fmdarr.f(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.f(i));
}
```

実行結果

```
my_fmdarr value[0]...  [1001.000000]
my_fmdarr value[1]...  [2001.000000]
```

4.5.40 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 &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 my_fmdarr(FLOAT_ZT);

my_fmdarr.f(0) = 1000.1;
my_fmdarr.f(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.f(i));
}
```

実行結果

```
my_fmdarr value[0]...  [1000.000000]
my_fmdarr value[1]...  [2000.000000]
```

4.5.41 round()

NAME

round() — 浮動小数点型の全要素値の四捨五入

SYNOPSIS

```
mdarray &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 my_fmdarr(FLOAT_ZT);
my_fmdarr.f(0) = 1000.5;
my_fmdarr.f(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.f(i));
}
```

実行結果

```
my_fmdarr value[0]...  [1001.000000]
my_fmdarr value[1]...  [-1001.000000]
```

4.5.42 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 &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 my_fmdarr(FLOAT_ZT);
my_fmdarr.f(0) = 1.7;
my_fmdarr.f(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.f(i));
}
```

実行結果

```
my_fmdarr value[0]...  [1.000000]
my_fmdarr value[1]...  [-1.000000]
```

4.5.43 abs()

NAME

abs() — 全要素値に対し絶対値をとる

SYNOPSIS

```
mdarray &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 my_fmdarr(FLOAT_ZT);

my_fmdarr.f(0) = -1000.1;
my_fmdarr.f(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.f(i));
}
```

実行結果

```
my_fmdarr value[0]...  [1000.1]
my_fmdarr value[1]...  [2000.6]
```

4.5.44 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．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 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 my_fmdarr(FLOAT_ZT, 2,2);
my_fmdarr.f(0,0) = 1000;

mdarray my_i64mdarr(INT64_ZT, 2,2);
my_i64mdarr.i64(0,0) = 1000;

sio.printf("*** my_fmdarr compare [%d] *** \n",
            (int)my_fmdarr.compare(my_i64mdarr));
```

実行結果

```
*** my_fmdarr compare [1] ***
```

4.5.45 copy()

NAME

copy() — 配列を別オブジェクトへコピー

SYNOPSIS

```
ssize_t copy( mdarray *dest ) const;
ssize_t copy( mdarray &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，コピー元の型，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 mdarray クラスのオブジェクト (コピー先)
([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 my_fmdarr(FLOAT_ZT);
mdarray my_cmdarr(UCHAR_ZT, 2, 2);
unsigned char my_char[] = {99, 101, 98, 102};
my_cmdarr.putdata((const void *)my_char, sizeof(my_char));

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)... [%hhu]\n",
                    i, j, my_fmdarr.c(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]
```

4.5.46 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 *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;
ssize_t copy( mdarray &dest,
              ssize_t col_idx, size_t col_len=MDARRAY_ALL,
```

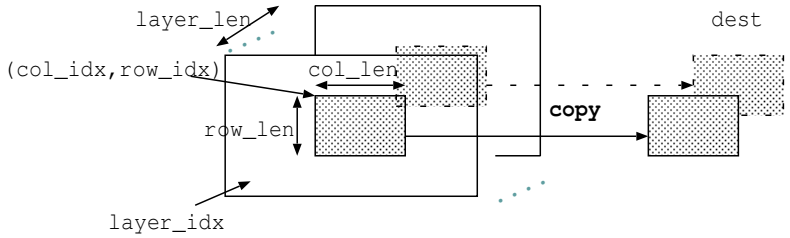
```
ssize_t row_idx=0, size_t row_len=MDARRAY_ALL,
ssize_t layer_idx=0, size_t layer_len=MDARRAY_ALL ) const;
```

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．

コピー先へは，各要素値とコピー元の型をコピーします．自身 (コピー元) の配列に変化はありません．

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．putdata() に関しては §4.5.22の解説を参照してください．

```
stdstreamio sio;

mdarray my_cmdarr(UCHAR_ZT, 2,2);
unsigned char my_char[] = {98, 99, 101, 102};
my_cmdarr.putdata((const void *)my_char, sizeof(my_char));
```

```
mdarray my_dmdarr(DOUBLE_ZT ,2,2);
double my_d[] = {-501, 501, -502, 502};
my_dmdarr.putdata((const void *)my_d, sizeof(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++ ) {
        /* copy により my_dmdarr の型は unsigned char になります */
        sio.printf("my_dmdarr value(%zu,%zu)... [%hhu]\n",
                    i, j, my_dmdarr.c(i, j));
    }
}
```

実行結果

```
my_ldmdarr value(0,0)...  [102]
```

4.5.47 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 &cut( mdarray *dest );
mdarray &cut( mdarray &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．`putdata()` に関しては §4.5.22の解説を参照してください.

```
stdstreamio sio;

mdarray my_mdarr;

mdarray my_cmdarr(UCHAR_ZT, 2, 2);
unsigned char my_char[] = {51, 101, 52, 102};
my_cmdarr.putdata((const void *)my_char, sizeof(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]
```

4.5.48 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 &cut( mdarray *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 );
mdarray &cut( mdarray &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．putdata() に関しては §4.5.22の解説を参照してください．

```
stdstreamio sio;

mdarray my_mdarr;

mdarray my_cmdarr(UCHAR_ZT, 2, 2);
unsigned char my_char[] = {51, 101, 52, 102};
my_cmdarr.putdata((const void *)my_char, sizeof(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.c(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]
```

4.5.49 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 &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 my_smdarr(SHORT_ZT, 2,2);
my_smdarr.s(0,0) = 1;
my_smdarr.s(1,0) = 3;
my_smdarr.s(0,1) = 2;
my_smdarr.s(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.s(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]
```

4.5.50 fill()

NAME

fill() — 要素値の書換え (画像データ向き)

SYNOPSIS

```
mdarray &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 &fill( double value,
     double (*func)(double,double,ssize_t,ssize_t,ssize_t,mdarray *,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
mdarray &fill( double value,
     void (*func_dest2d)(const void *,void *,void *), void *user_ptr_dest2d,
     void (*func_d2dest)(const void *,void *,void *), void *user_ptr_d2dest,
     double (*func)(double,double,ssize_t,ssize_t,ssize_t,mdarray *,void *),
     void *user_ptr_func,
     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 ); ..... 3
```

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 2，3) で書換えます.

ユーザ定義関数 `func` の引数には順に，自身の要素値，`value` で与えられた値，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，§4.5.53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]	user_ptr_dest2d	func_dest2d の最後に与えられるユーザのポインタ
[I]	user_ptr_d2dest	func_d2dest 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]	func_dest2d	自身の生データを double 型へ変換するためのユーザ関数のアドレス
[I]	func_d2dest	double 型の値から自身の生データへ変換するため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 my_smdarr(SHORT_ZT, 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.s(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]
```

4.5.51 add()

NAME

add() — 要素値の加算 (画像データ向き)

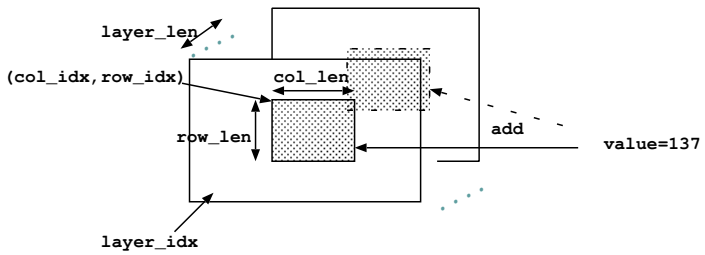
SYNOPSIS

```
mdarray &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．putdata() に関しては §4.5.22の解説を参照してください．

```
stdstreamio sio;

mdarray my_smdarr(SHORT_ZT, 2,2);
short my_short[] = {1, 2, 3, 4};
my_smdarr.putdata((const void *)my_short, sizeof(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.s(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]
```

4.5.52 multiply()

NAME

multiply() — 要素値の乗算 (画像データ向き)

SYNOPSIS

```
mdarray &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;
```

```
mdarray my_fmdarr(FLOAT_ZT, 2,2);
float my_float[] = {1, 3, 2, 4};
my_fmdarr.putdata((const void *)my_float, sizeof(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.f(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]
```

4.5.53 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 &paste( const mdarray &src,
  ssize_t dest_col = 0, ssize_t dest_row = 0, ssize_t dest_layer = 0 ); ... 1
mdarray &paste( const mdarray &src,
  double (*func)(double,double,ssize_t,ssize_t,ssize_t,mdarray *,void *),
  void *user_ptr,
  ssize_t dest_col = 0, ssize_t dest_row = 0, ssize_t dest_layer = 0 ); ... 2
mdarray &paste( const mdarray &src,
  void (*func_src2d)(const void *,void *,void *), void *user_ptr_src2d,
  void (*func_dest2d)(const void *,void *,void *), void *user_ptr_dest2d,
  void (*func_d2dest)(const void *,void *,void *), void *user_ptr_d2dest,
  double (*func)(double,double,ssize_t,ssize_t,ssize_t,mdarray *,void *),
  void *user_ptr,
  ssize_t dest_col = 0, ssize_t dest_row = 0, ssize_t dest_layer = 0 ); ... 3
```

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.

メンバ関数 2 と 3 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，自身の要素値，オブジェクト src の要素値，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_src2d	オブジェクト src の生データを double 型へ変換するためのユーザ関数へのアドレス
[I]	user_ptr_src2d	func_src2d の最後に与えられるユーザのポインタ
[I]	func_dest2d	自身の生データを double 型へ変換するためのユーザ関数へのアドレス
[I]	user_ptr_dest2d	func_dest2d の最後に与えられるユーザのポインタ
[I]	func_d2dest	double の値から自身の生データへ変換するためのユーザ関数へのアドレス
[I]	user_ptr_d2dest	func_d2dest の最後に与えられるユーザのポインタ
[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．putdata() に関しては §4.5.22の解説を参照してください.

```
double my_func(double self, double src, ssize_t x,
               ssize_t y, ssize_t z, mdarray *myptr, void *p)
{
    return self + src + 500;
}

/* 処理 */
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
float my_float[] = {100, 0, 200};
my_fmdarr.putdata((const void *)my_float, sizeof(my_float));

mdarray mypaste_mdarr(FLOAT_ZT, 2,2);
float mypaste_float[] = {1000, 3000, 2000, 4000};
mypaste_mdarr.putdata((const void *)mypaste_float, sizeof(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.f(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]
```

4.5.54 add()

NAME

add() — 配列オブジェクトの加算 (画像データ向き)

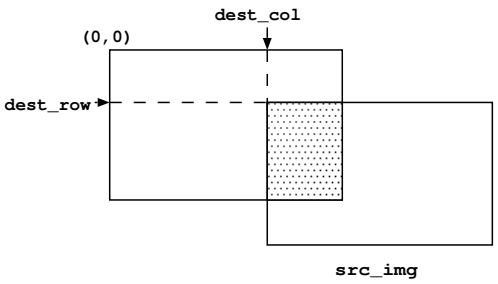
SYNOPSIS

```
mdarray &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．

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．putdata() に関しては §4.5.22の解説を参照してください．

```
stdstreamio sio;

mdarray my_smdarr(SHORT_ZT, 2,2);
short my_short[] = {1, 2, 3, 4};
my_smdarr.putdata((const void *)my_short, sizeof(my_short));

mdarray myadd_smdarr(SHORT_ZT, 2,2);
short myadd_short[] = {9, 8, 7, 6};
myadd_smdarr.putdata((const void *)myadd_short, sizeof(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.s(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]
```

4.5.55 subtract()

NAME

subtract() — 配列オブジェクトの減算 (画像データ向き)

SYNOPSIS

```
mdarray &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．

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．putdata() に関しては §4.5.22の解説を参照してください．

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
mdarray mysubtract_mdarr(FLOAT_ZT, 2,2);

float my_float[] = {1000, 2000, 3000, 4000};
my_fmdarr.putdata((const void *)my_float, sizeof(my_float));
float mysubt_float[] = {100, 200, 300, 400};
mysubtract_mdarr.putdata((const void *)mysubt_float, sizeof(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.f(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]
```

4.5.56 multiply()

NAME

multiply() — 配列オブジェクトの乗算 (画像データ向き)

SYNOPSIS

```
mdarray &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．

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;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
float my_float[] = {1, 2, 3, 4};
my_fmdarr.putdata((const void *)my_float, sizeof(my_float));

mdarray mymulti_fmdarr(FLOAT_ZT, 2,2);
float mymulti_float[] = {10, 20, 30, 40};
mymulti_fmdarr.putdata((const void *)mymulti_float, sizeof(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.f(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]
```

4.5.57 divide()

NAME

divide() — 配列オブジェクトの除算 (画像データ向き)

SYNOPSIS

```
mdarray &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．

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．putdata() に関しては §4.5.22の解説を参照してください．

```
stdstreamio sio;

mdarray my_fmdarr(FLOAT_ZT, 2,2);
float my_float[] = {1000, 2000, 3000, 4000};
my_fmdarr.putdata((const void *)my_float, sizeof(my_float));

mdarray mydiv_mdarrf(FLOAT_ZT, 2,2);
float mydiv_float[] = {2, 4, 6, 8};
mydiv_mdarrf.putdata((const void *)mydiv_float, sizeof(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.f(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]
```