不遇の標準ライブラリ

歌舞伎座.tech#8「C++初心者会」

山本 亮介 @Ryosuke839

自己紹介

- 山本 亮介 @Ryosuke839
- 某工大情報工学科4年生です
- 画像認識の研究始めました
- C++初心者です
  - 規格書暗記してません
  - コンパイラのバグ踏んだことありません

- (‘ω’)

今回のテーマ


STL
BOOST
The Great
New Standard War
external
libraries
seas
Exceptional swamps
Multiple inheritance forest
Alexandrescu
castle
Loki
Bjame
Stroustrup
tower
#include
fort
The C++ Lands

今回のテーマ


標準ライブラリ
このへん
STL
BOOST
The Great
New Standard War
external
libraries
seas
Exceptional swamps
Multiple inheritance forest
Alexandrescu
castle
Loki
Bjame
Stroustrup
tower
#include
fort
The C++ Lands

今回のテーマ


STL
BOOST
The Great New Standard War
ios lake
external libraries seas
標準ライブラリ
Exceptional swamps
Multiple inheritance forest
このへん
Alexandrescu castle
Loki
Templates maze
Bjame Stroustrup tower
HOLY STANDARD TERRITORY
#include fort
The C++ Lands

今回のテーマ

- 不遇の標準ライブラリ

今回のテーマ

- 不遇の標準ライブラリ
- 次期規格で追加されるライブラリなどのstate of the artな話題ではありません
- C++98(C++の初期規格)から存在するライブラリの話です

  – C++に詳しい諸兄はタイトルでピンときているはず

今回のテーマ

- valarray

- C++98から存在します
- 単独でヘッダファイルも持っています
  - #include <valarray>

- でも使われてません

今回のテーマ

- valarray

- 数値計算に特化した可変長配列を実現するテンプレートクラスvalarrayを含んでいます
  - vectorと被るような…
- ベクトル演算を簡単に記述できます

- でも使われてません

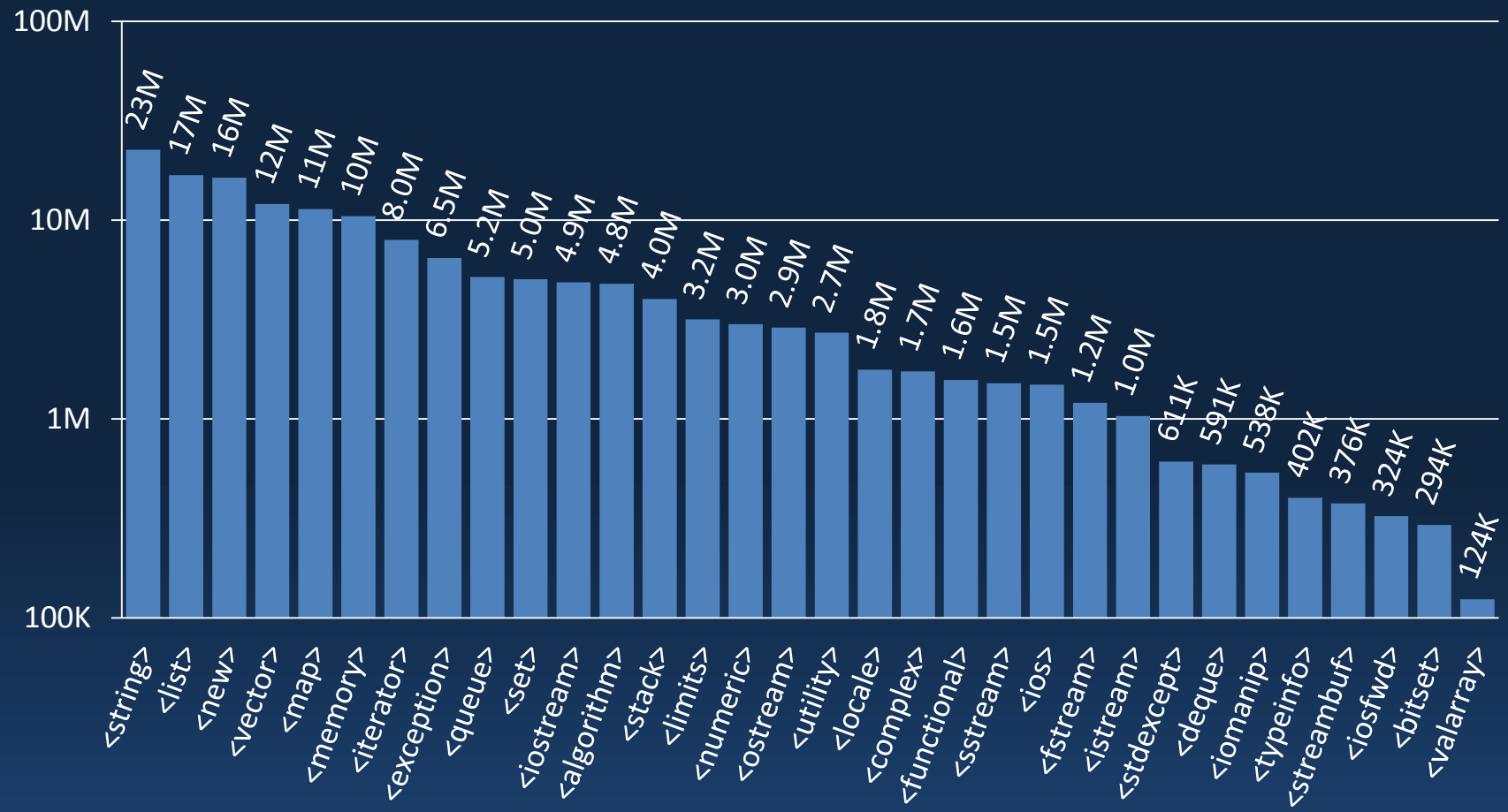
今回のテーマ

- valarray

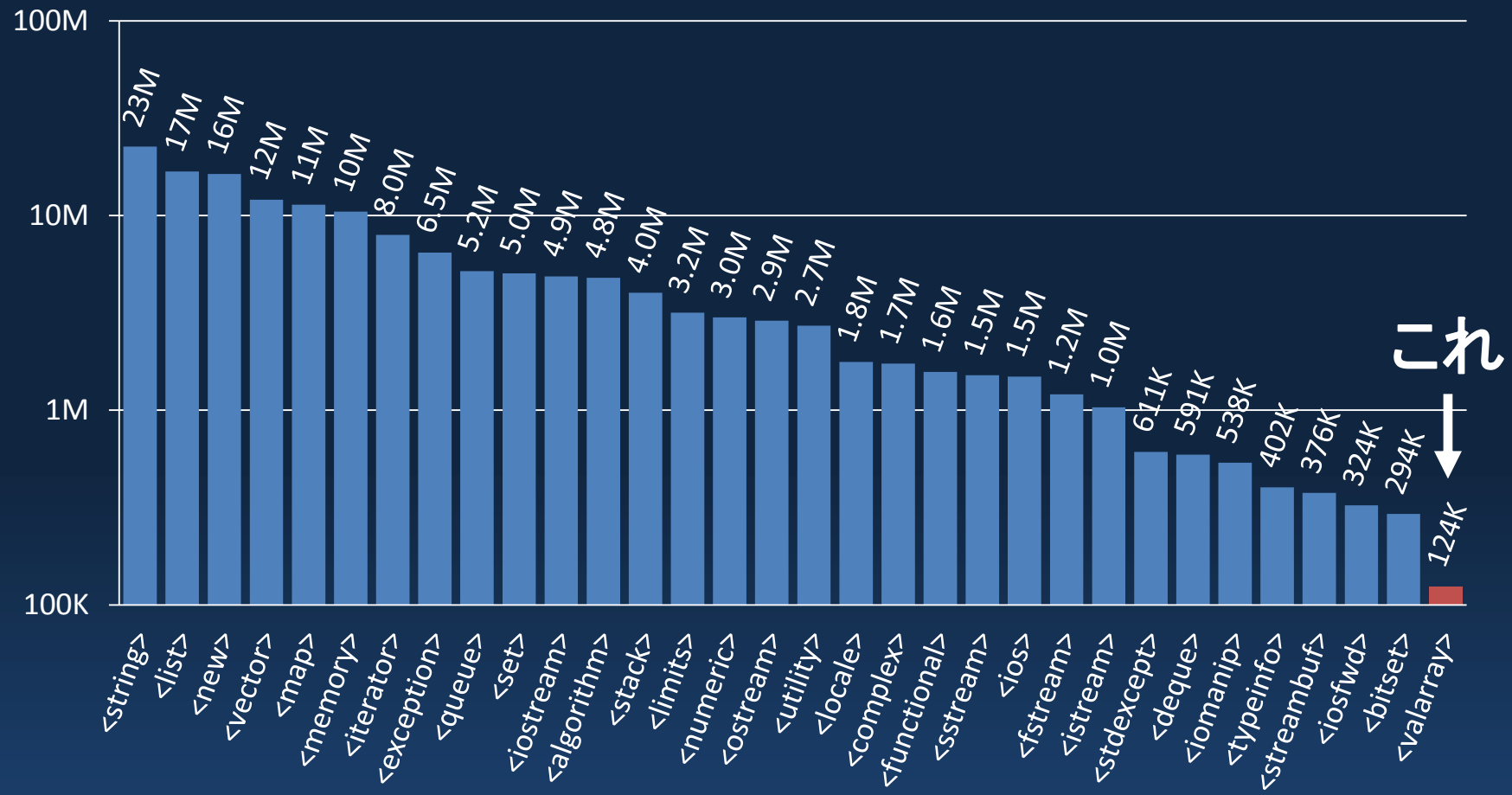
- どれだけ使われていないか

- 論文の世界では被引用数が論文の価値の目安になるらしいので、被include数を示します

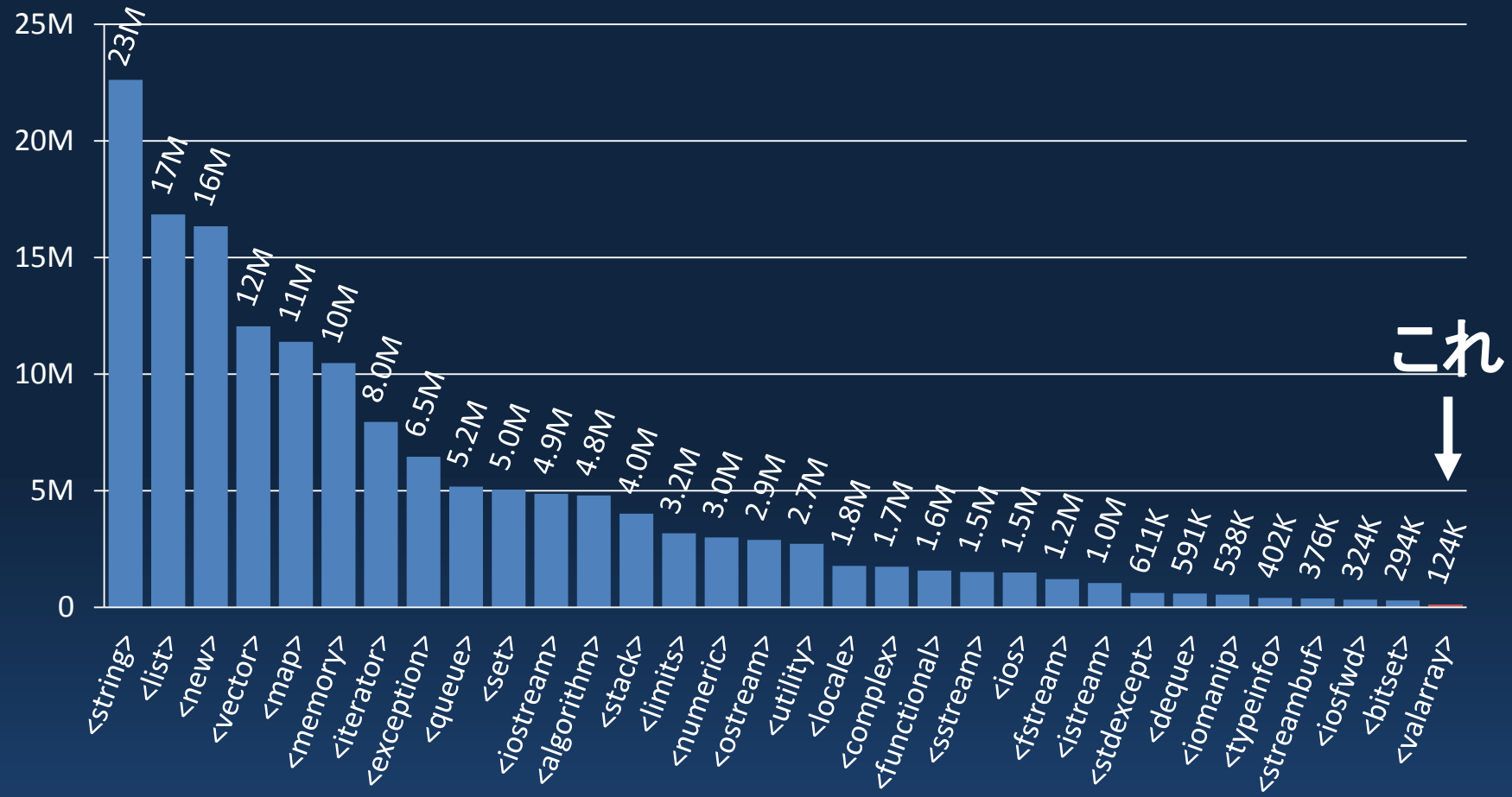
被include数@GitHub



被include数@GitHub



被include数@GitHub



今回のテーマ

- valarray

- これだけ使われていないのはかわいそう

- 今回は（無理して）使ってみます
  - 見た目でわかりやすい画像処理をします

使ってみる

- とりあえずIOを書きます

```
std::ifstream ifs(name, std::ios::binary);
std::valarray<unsigned char> res;
res.resize(256 * 256);
for (auto& c : res)
    ifs.read(reinterpret_cast<char*>(&c), 1);
```

```
std::ofstream ofs(name, std::ios::binary);
for (auto& c : data)
    ofs.write(reinterpret_cast<const char*>(&c), 1);
```

使ってみる

- とりあえずIOを書きます

```
std::ifstream ifs(name, std::ios::binary);
std::valarray<unsigned char> res;
res.resize(256 * 256);
for (auto& c : res)
    ifs.read(reinterpret_cast<char*>(&c), 1);
```

可変長だけど可変長じゃない！

```
std::ofstream ofs(name, std::ios::binary);
for (auto& c : data)
    ofs.write(reinterpret_cast<const char*>(&c), 1);
```

使ってみる

- あとは処理を書くだけ

```
auto data = readimg("nico.bmp");
data = (unsigned char)(192) - data / (unsigned char)(2);
writeimg("result.bmp", data);
```

使ってみる

- あとは処理を書くだけ

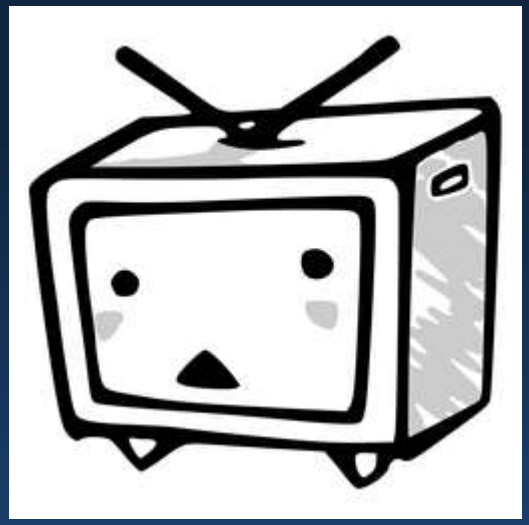
```
auto data = readimg("nico.bmp");
data = (unsigned char)(192) - data / (unsigned char)(2);
writeimg("result.bmp", data);
```

暗黙の変換はしてくれません

使ってみる

- あとは処理を書くだけ

```
auto data = readimg("nico.bmp");
data = (unsigned char)(192) - data / (unsigned char)(2);
writeimg("result.bmp", data);
```



使ってみる

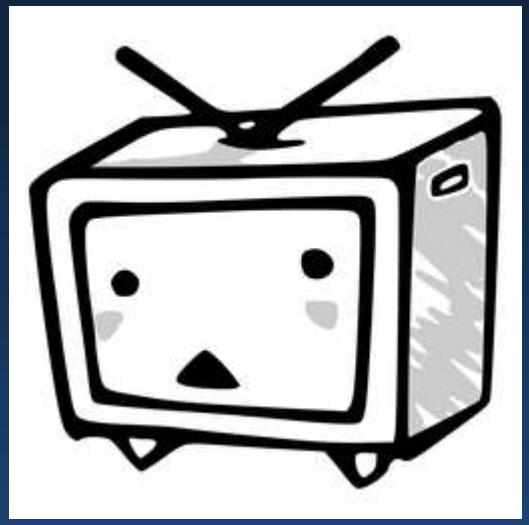
- あとは処理を書くだけ

```
auto data = readimg("nico.bmp");
data = (unsigned char)(192) - data / (unsigned char)(2);
writeimg("result.bmp", data);
```

使ってみる

- Sliceも取れます

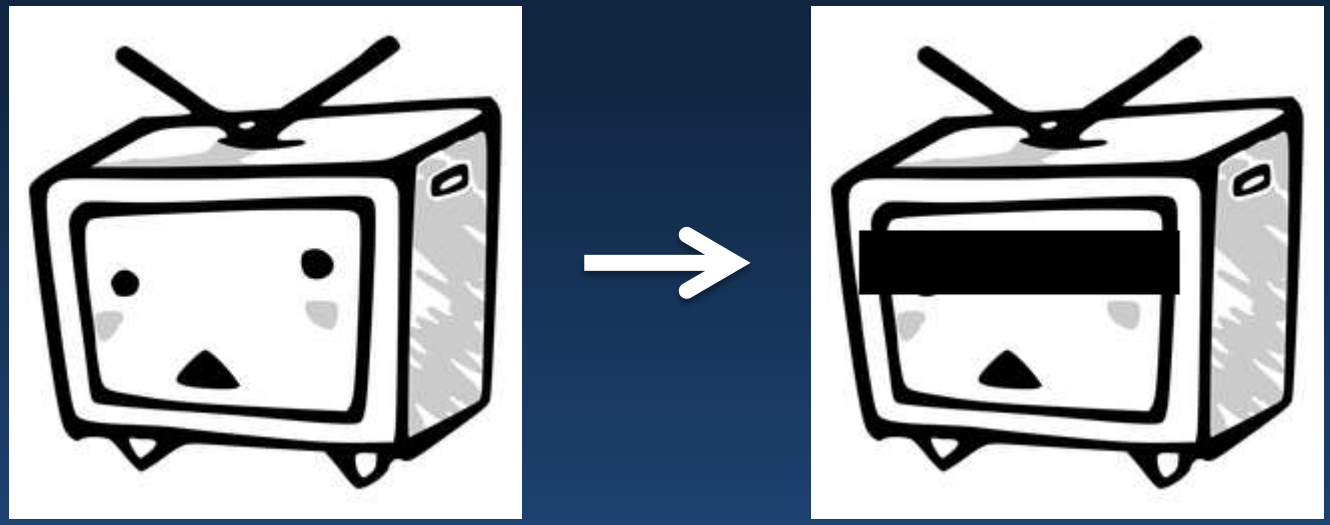
```
auto data = readimg("nico.bmp");
data[std::gslice(112 * 256 + 24,
    std::valarray<size_t>{32, 160},
    std::valarray<size_t>{256, 1})] = (unsigned char)(0);
writeimg("result.bmp", data);
```



使ってみる

- Sliceも取れます

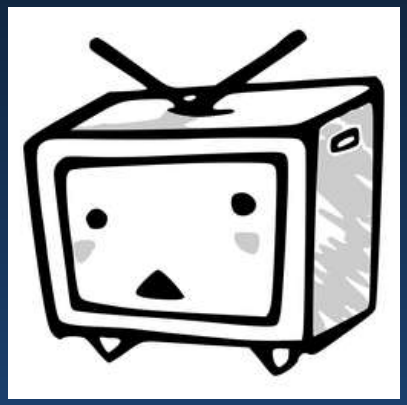
```
auto data = readimg("nico.bmp");
data[std::gslice(112 * 256 + 24,
    std::valarray<size_t>{32, 160},
    std::valarray<size_t>{256, 1})] = (unsigned char)(0);
writeimg("result.bmp", data);
```

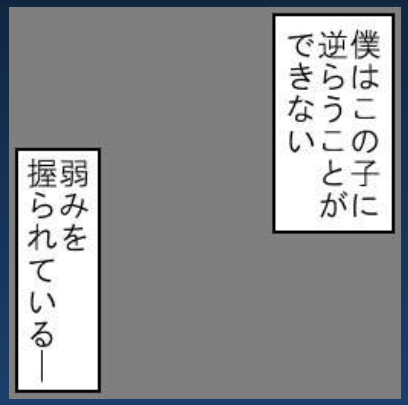


使ってみる

- 比較結果でsliceを作ることもできます

```
auto data = readimg("nico.bmp");
data[std::gslice(...)] = (unsigned char)(0);
auto text = readimg("text.bmp");
data[text != byte(127)] =
    std::valarray<byte>(text[text != byte(127)]);
writeimg("result.bmp", data);
```

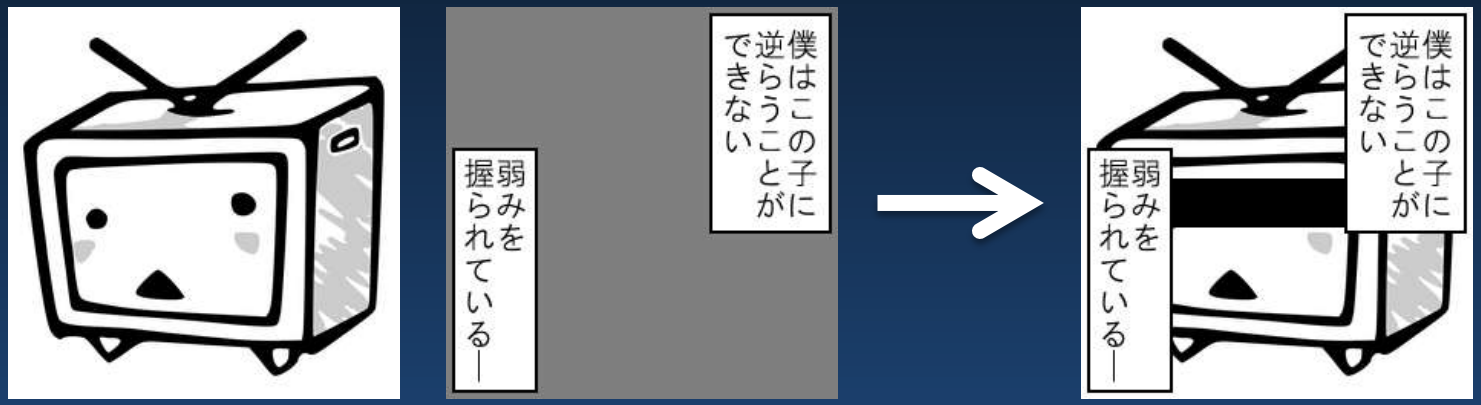




使ってみる

- 比較結果でsliceを作ることもできます

```
auto data = readimg("nico.bmp");
data[std::gslice(...)] = (unsigned char)(0);
auto text = readimg("text.bmp");
data[text != byte(127)] =
    std::valarray<byte>(text[text != byte(127)]);
writeimg("result.bmp", data);
```



その他の用法

- 内積

```
(a * b).sum()
```

- 外積

```
a.cshift(1) * b.cshift(-1) - a.cshift(-1) * b.cshift(1)
```

- ノルム
  - L1 `std::abs(a).sum()`
  - L2 `std::sqrt((a * a).sum())`
  - L∞ `std::abs(a).max()`

残念な仕様

- 他にもsin, cos, exp, log等の関数を使えます
  - この場合でも引数と戻り値の型は同じです
  - 他の型のvalarrayにキャストもできません

```
template<class T> valarray<T> valarray<T>::apply(T func(const T&)) const;
template<class T> valarray<T> operator* (const valarray<T>&, const T&);
template<class T> valarray<T> pow(const valarray<T>&, const valarray<T>&);
```

- Sliceで元のvalarrayへの参照を取れますが、sliceのsliceを取ることはできません
  - 一旦sliceから新しいvalarrayを生成する必要あり

残念な仕様

- valarrayがC++に導入されたそもそもの経緯
  - C++が開発された80年代はベクトル型計算機の華の時代
  - Fortranではベクトル化最適化が実装されていた

残念な仕様

- valarrayがC++に導入されたそもそもの経緯
  - C++が開発された80年代はベクトル型計算機の華の時代
  - Fortranではベクトル化最適化が実装されていた

↓

  - C++でもFortran並みの最適化を簡単に実現できるよう狙った？[要出典]
  - 型の制約がきついのも最適化のため？[要出典]

残念な仕様

- 現在はPC向けCPUでもSSEなどのベクトル命令が実装されている
  - では、valarrayもそれらに最適化されるのでは…？

残念な仕様

- 現在はPC向けCPUでもSSEなどのベクトル命令が実装されている
  - では、valarrayもそれらに最適化されるのでは…？
- ベンチマークしてみます
  - Cスタイルの配列とforループ
  - std::vectorとstd::transform
  - std::valarray

ベンチマーク

Add, mul, sum, sqrt, sin, maxを求めるコード
いずれのコンパイラにも高速演算オプション適用

	gcc	clang
array + for loop	606ms FPU	660ms SSE2
vector + algorithm	603ms FPU	641ms SSE2
valarray	603ms FPU	648ms SSE2

ベンチマーク

Add, mul, sum, sqrt, sin, maxを求めるコード
いずれのコンパイラにも高速演算オプション適用

	gcc	clang	msvc
array + for loop	606ms FPU	660ms SSE2	
vector + algorithm	603ms FPU	641ms SSE2	281ms SSE2
valarray	603ms FPU	648ms SSE2	256ms SSE2

ベンチマーク

Add, mul, sum, sqrt, sin, maxを求めるコード
いずれのコンパイラにも高速演算オプション適用

	gcc	clang	msvc	icc	icc+ipp
array + for loop	606ms FPU	660ms SSE2			
vector + algorithm	603ms FPU	641ms SSE2	281ms SSE2		184ms AVX
valarray	603ms FPU	648ms SSE2	256ms SSE2	45ms AVX	41ms AVX(ipp)

ベンチマーク

Add, mul, sum, sqrt, sin, maxを求めるコード
いずれのコンパイラにも高速演算オプション適用

	gcc	clang	msvc	icc	icc+ipp
array + for loop	606ms FPU	660ms SSE2	47ms SSE4		27ms AVX
vector + algorithm	603ms FPU	641ms SSE2	281ms SSE2		184ms AVX
valarray	603ms FPU	648ms SSE2	256ms SSE2	45ms AVX	41ms AVX(ipp)

まとめ

- valarrayは使われてない割には便利です！
  - 簡単なベクトル演算をしたい時に俺々ライブラリを書くよりはずっと便利　ただし謎の制約多数
  - 仕様が独特なので、本格的な演算をしたい場合は本格的なライブラリを使いましょう
- vector以上に最適化が効きます
- iccでコンパイルするとvectorの数倍速いです

まとめ

- valarrayは使われてない割には便利です！
  - 簡単なベクトル演算をしたい時に俺々ライブラリを書くよりはずっと便利　ただし謎の制約多数
  - 仕様が独特なので、本格的な演算をしたい場合は本格的なライブラリを使いましょう
- vector以上に最適化が効きます
- iccでコンパイルするとvectorの数倍速いです
  - インテルのコンパイラ今日限り特価140,000円！