お断り：
公開用に、
当日使用した画像を
一部加工してあります。

魔法言語C++☆0x

札幌C++勉強会/Sapporo.cpp

自己紹介

- @hotwatermorning
- 札幌C++勉強会を主催
- 大学生
- PStade.Ovenのコミッタ
- はてなid : heisseswasser
- DTMもやってます！

とあるプログラマ
鹿目まどかの苦悩

鹿目まどかの苦悩

普通の中学２年生、鹿目まどかはC++を使うプログラマ。
けれども、ちょっと便利なC言語としてしか使えない。

ある日、少女は夢を見る。
そこは、魔法で戦う異世界。
少女は謎の白い生物から告げられる
「僕と契約してC++erになってよ！」

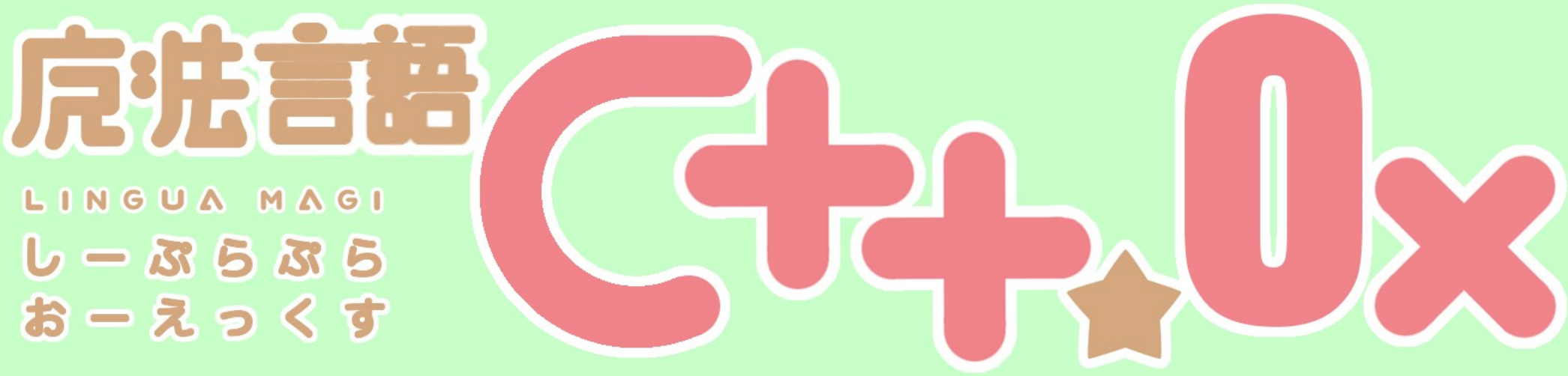
コンテンツ

- 第1話　“C++0x” 夢の中で逢った、ような……
- 第2話　“auto/decltype” それはとっても嬉しいなって
- 第3話　“smart pointer” もう何も恐くない
- 第4話　“lambda” 奇跡も、魔法も、あるんだよ
- 第5話　“initializer_list” 後悔なんて、あるわけない
- 第6話　“regex”こんなの絶対おかしいよ

コンテンツ

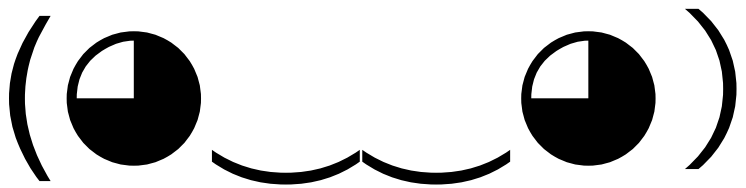
- 第7話　“random” 本当の気持ちと向き合えますか?
- 第8話　“thread” もう誰にも頼らない
- 第9話　“final” そんなの、あたしが許さない
- 第10話 “container” あたしって、ほんとバカ
- 第11話 “boost” 最後に残った道しるべ
- 最終話 “C++” わたしの、最高の友達

※本発表は某アニメとは全く関係がありません。



第1話　“C++0x”

「夢のなかで逢った、ような・・・」



まどかは夢を見ました

そこでは少女が、誰もが絶望するような敵と戦っていました。
その敵とは....

- newしたのにdeleteしていないソースコード
- 劣化STLともいうべき、氾濫するオレオレライブラリ
- std::vectorを使わずにnewで配列作ってるソースコード
- クラス？多態性？そんなのめんどくさいとか言い出す上司
- templateは難しいから、使ったらダメという上s(ry
- etc...

敵は２つに分類される

魔女(学習嫌い)とその使い魔(template嫌悪症)である

(◕‿‿◕)

「願いから産まれるのが魔法少女(C++er)だとすれば、
魔女は呪いから産まれた存在なんだ」

「魔法少女が希望を振りまくように、
魔女は絶望を蒔き散らす」

「しかもその姿は、
普通の人間(非C++er)には見えないからタチが悪い」

(◕‿‿◕)
「不安や猜疑心、過剰な怒りや憎しみ、

肥大化するコード、バグを産む設計、
意味不明な関数群、触れてはいけない謎関数
…

そういう
災い(デスマーチ♡)の種を世界にもたらしているんだ」

C++は難しいのか？

C++は難しいのか？

ぐーぐる先生の回答(2011/06/09)

- Python 難しい　　約　3,390,000 件
- Ruby 難しい　　約　5,020,000 件
- Lisp 難しい　　約　372,000 件
- C++ 難しい　　約　3,050,000 件
- Java 難しい　　約　5,890,000 件
- PHP 難しい　　約 20,000,000 件
- Haskell 難しい　　約　253,000 件

C++はPHPよりはるかに簡単！
そしてLisp, Haskellはさらに簡単！（違

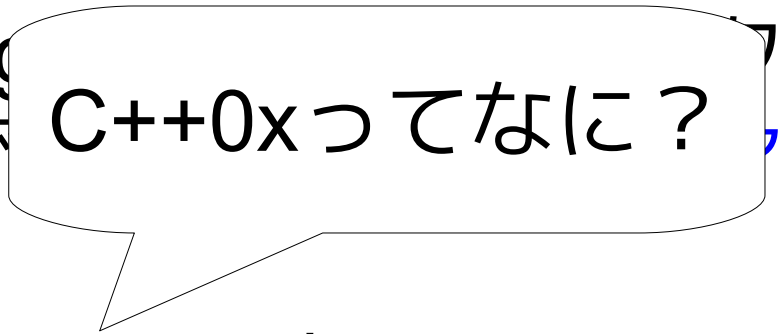

C++って簡単だね！
騙されないで！
そいつの思う壺よ！

C++は難しくない

- templateは慣れないうちは難しいと感じる。まずはSTL(Standard Template Library)を使いながら慣れよう！
- 訓練されたC++erは(ユーザーコードでは)ほとんどnew/deleteを使わない。スマートポインタとSTLのstd::vectorを使おう！
- STLのalgorithm形式のインターフェイスに慣れよう。まずはイテレータとファンクタの考え方を学ぼう！
- C++はC++0xで書きやすくなる。C++0x(の一部)を本セミナーで知っておこう。

C++は難しくない

- templateは慣れないうちは難しいと感じる。まずはSTL(Standard Template Library)を使いながら慣れよう！
- 訓練されたC++erは(ユーザーコードでは)ほとんどnew/deleteを使わない。スマートポインタとSTLのstd::vectorを使おう！
- STLのalg[illegible]ーフェイスに慣れよう。まず[illegible]ンクタの考え方を学ぼう！
- C++はC++0xで書きやすくなる。C++0x(の一部)を本セミナーで知っておこう。

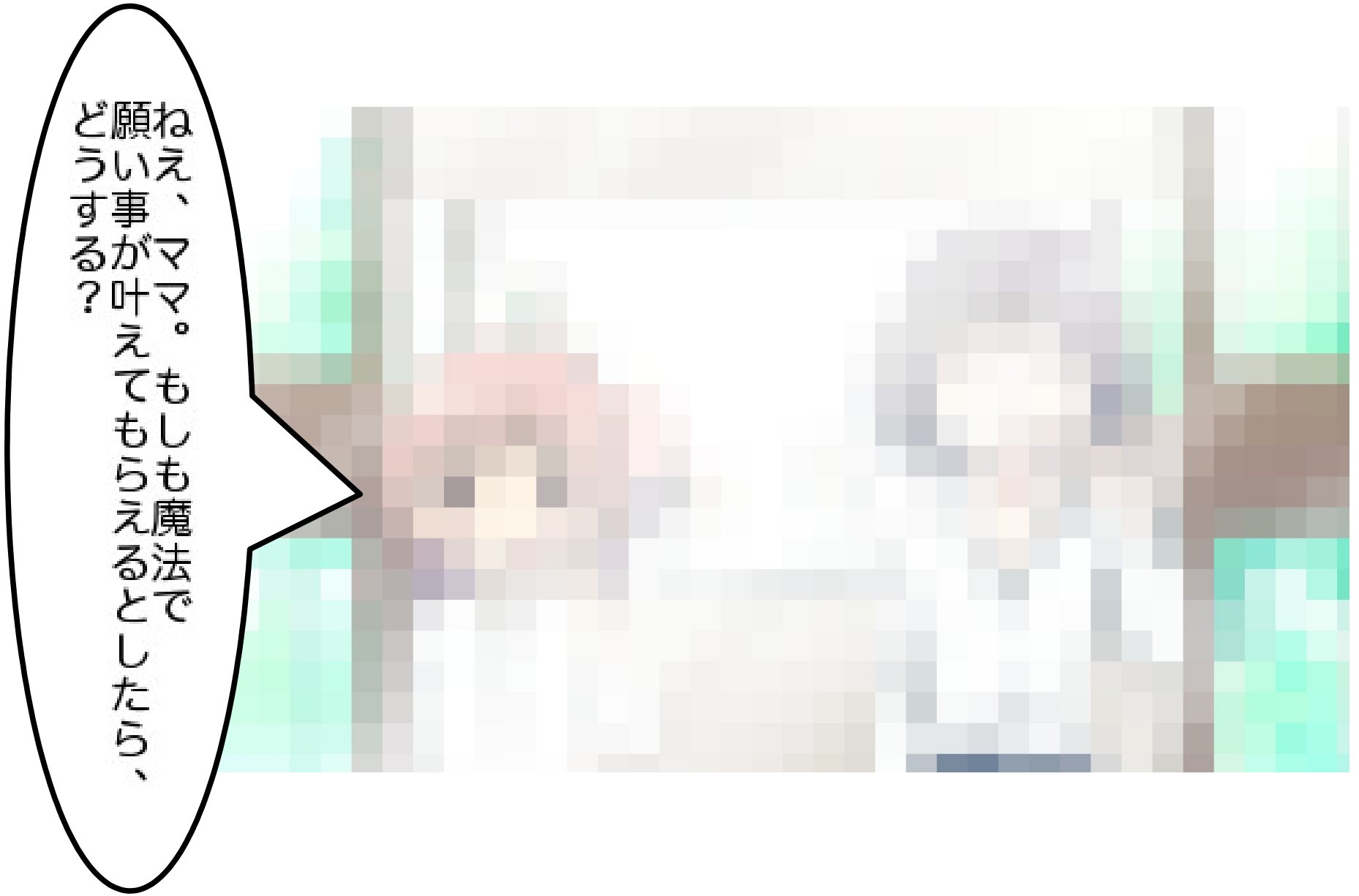


C++0xとは

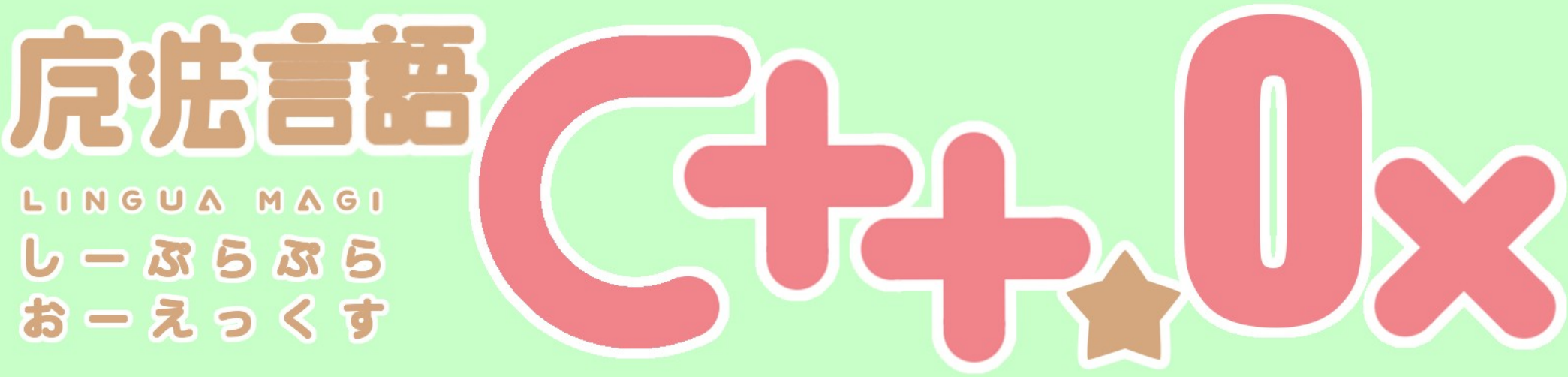
- C++の次期規格の通称(現在のはC++03と呼ばれている)
- C++03とほぼ100%の互換性をもつ
- C++03で使いにくかった部分や不満があった部分を大幅に改善
- ライブラリも拡充されます！

C++0xとは

- C++の次期規格の通称(現在のはC++03と呼ばれている)
- C++03とほぼ100%の互換性をもつ
- C++03で使いにくかった部分や不満があった部分を大幅に改善
- ライブラリも拡充されます！
- 今年度あたりには国際規格として承認される予定・・・


ねえ、ママ。もしも魔法で
願い事が叶えてもらえるとしたら、
どうする？


使えないプログラマを全員よそに飛ばしてもらうわ



第2話　“auto/decltype”

「それはとっても嬉しいなって」

C++0xの目玉機能その１　auto

- これまでのC++ではautoは記憶クラス(自動変数)を指定するためのキーワードだった。(通常は省略される)

```
auto int a;        // 自動変数
static int a;       //静的変数
```

- C++0xでは型推論のためのキーワードとして使われる。

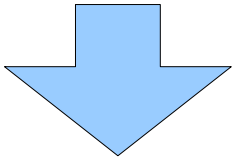
- 従来の用途でautoを使用することはできないが、互換性についてはおそらく問題ない。（これまでautoを明示的に指定しているコードはほとんどないため）

型推論

- 変数の型をコンパイル時に、コンパイラが推論。
- 煩雑な型指定を簡略化できる。

C++03

```
std::vector<int> vec(N);
std::vector<int>::iterator itr = vec.begin();
```



C++0x

```
std::vector<int> vec(N);
auto itr = vec.begin();
```

型推論

templateとの併用で威力を発揮！

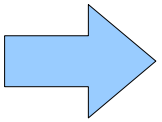
```
struct A {
    int   get() { return 1; }
};
struct B {
    std::string
          get() { return “ほむ”  ; }
};
template<class T>
void func(T arg) {
    auto tmp = arg.get();
}
...
func(A()); func(B());
```

arg.get()が返す型は
わからない。
でも、autoとしておけば
コンパイラが
推論してくれる！

もう１つの型推論 decltype

- autoは変数の宣言時に型推論させる方法
- templateに型を渡したい場合などに使うのがdecltype
- decltypeを使うと式の型を知ることができる

```
Int var = 3;
std::vector<decltype(var + 3.0)> vec(N);
```



vecはstd::vector<double>型

型の後置記法

```
struct A { };
struct B { };
struct C { };
C operator+(const A& a, const B& b)
{ return C(); }

template<class T1, class T2>
auto func(T1 t1, T2 t2) -> decltype(t1+t2) {
    return t1+t2;
}

func(A(), B());
```

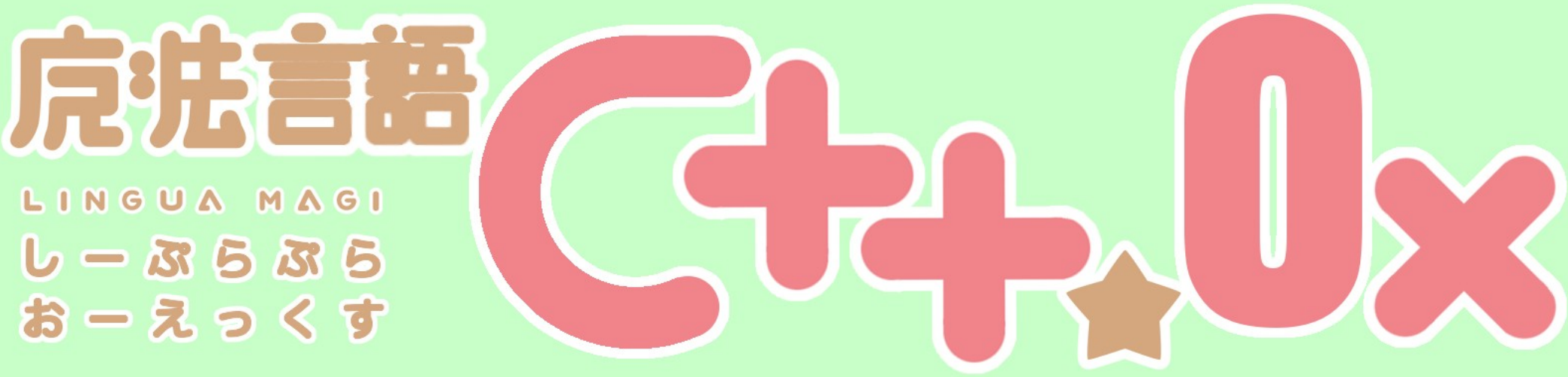
C++0xでは
関数の返り値の型を後置できる

```
void SomeFunction(
        const ExpressionA  a,
        const ExpressionB  b
        const Operator op)
{

    const Expression<
        typename get_return_type<
                    ExpressionA,
                    ExpressionB
        >::type,
        typename ExpressionA::expr_type,
        typename ExpressionB::expr_type>   ab = op(a,b);

        ...
        return ....;
}
```

型名

もう長い型名を書かなくてよくて、
それはとっても嬉しいなって思うのでした
(マジで....)

※ソースコードはフィクションです



第3話　“smart pointer”

「もうなにも怖くない」

スマート
ポインタ

スマートポインタ

Smart pointer

- C++03のスマートポインタ auto_ptrが使えない子だったために、長らく標準化が望まれていたライブラリ。
- Boostにshared_ptrというスマートポインタがあり、「shared_ptrを使うためだけにBoost使う」とまで言われ続けた。
- C++0xで必ず使って欲しい機能の１つ！
  （ってか使ってない人は早くBoostのでもいいので使って下さい＞＜)
- 本セミナーではC++0xのstd::shared_ptrを紹介します

Smart pointer(shared_ptr)の利点

- Smart pointer＝ポインタをラップするクラス
- shared_ptr＝Smart pointerのひとつ。
- shared_ptrでラップされたポインタは、shared_ptrの参照カウントが０になったときにdeleteされる。
- プログラマはdelete し忘れの不安から解放される！
- shared_ptrはvectorなどのコンテナに格納できる(C++03のauto_ptrではできなかった！)

もう

なにも

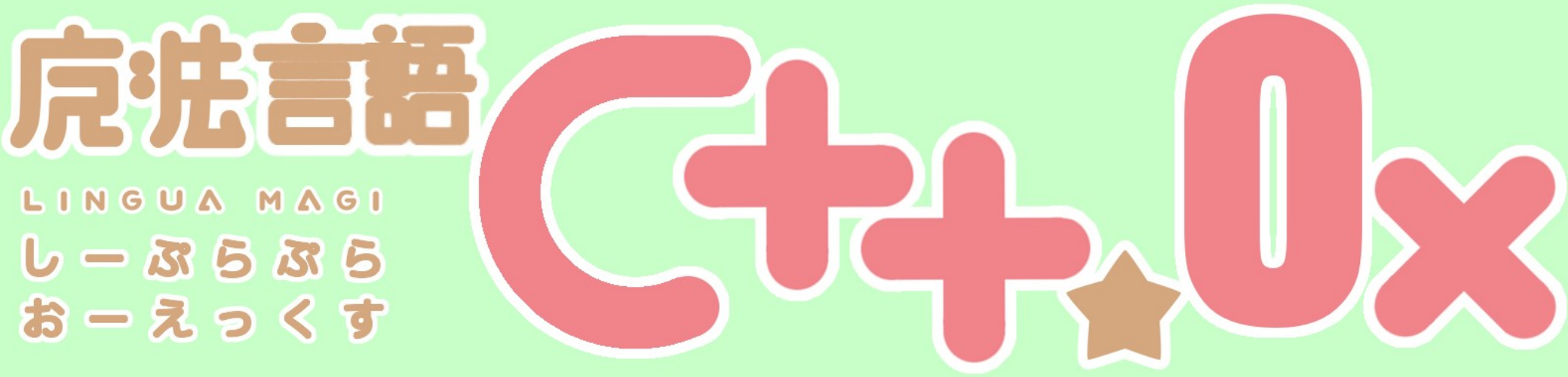
怖くない！

（嘘です。闇の軍団とか怖いです・・・）

使用例

```
int main() {

    {
        std::shared_ptr<Person> person(new Person);
        std::cout << person->name() << std::endl;

    } // ここでデストラクタが呼ばれリソースが解放される

}
```

deleteいらず。ぜひ使おう！(◕‿‿◕)



第4話　“lambda”

「奇跡も、魔法も、あるんだよ」

無名関数
ラムダ式

C++にもlambdaが登場しました

- 関数内で簡易的な無名関数を作れる。
- 特にファンクタを多用するSTLのalgorithmライブラリで活躍。
- これを機に、STLのalgorithmライブラリを使いこなそう！

C++にもlambdaが登場しました

- 関数内で簡易的な無名関数を作れる。
- 特にファンクタを多用するSTLのalgorithmライブラリで活躍。
- これを機に、STLのalgorithmライブラリを使いこなそう！

まずはファンクタの使い方を覚えよう！

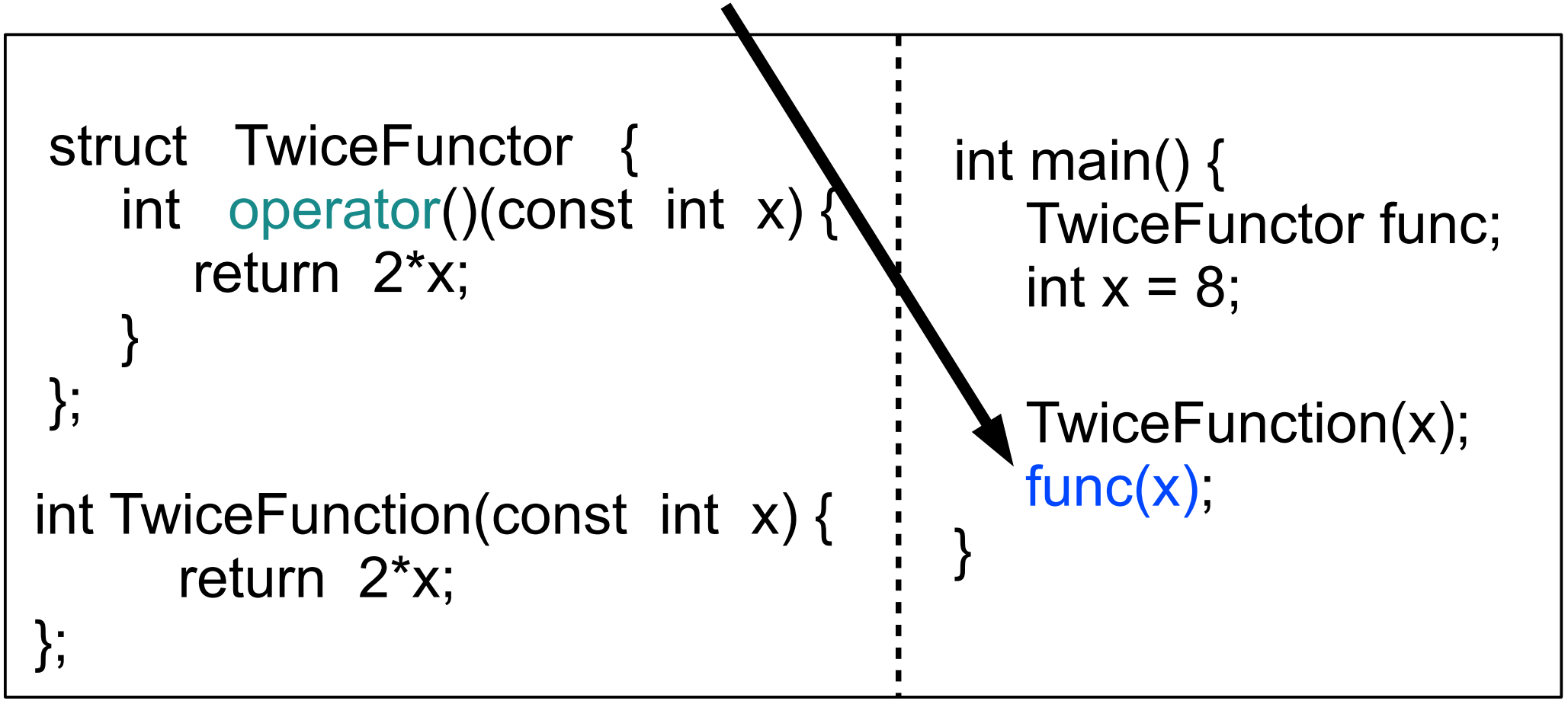
functor
ファンクタ
関数オブジェクト
Predicate
述語関数

呼び方は様々だけど、同じ意味

※厳密には同じものではありません。
http://d.hatena.ne.jp/Flast/20110612/1307873074

ファンクタとは？

- operator()をオーバーロードしているクラスのこと
- あたかも関数のように振る舞う



```
struct   TwiceFunctor   {
    int   operator()(const  int  x) {
        return  2*x;
    }
};

int TwiceFunction(const  int  x) {
        return  2*x;
};
```

```
int main() {
    TwiceFunctor func;
    int x = 8;

    TwiceFunction(x);
    func(x);
}
```

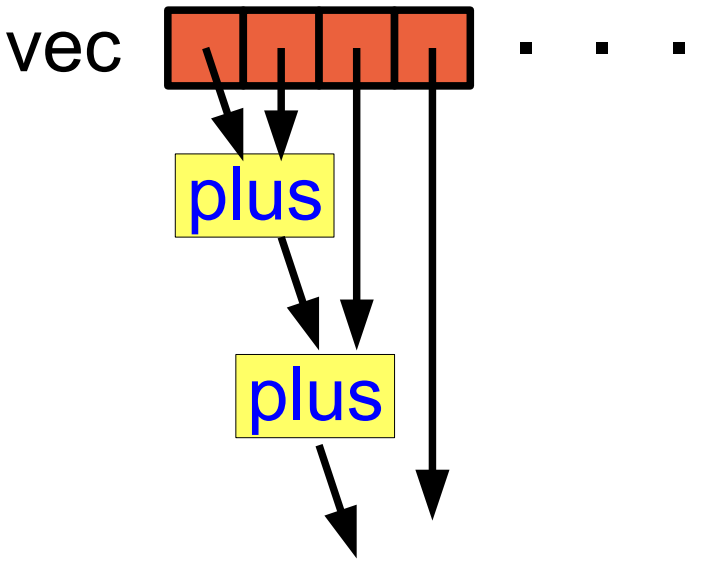
STLのalgorithmでは
ファンクタを多く使う

- std::for_each(vec.begin(), vec.end(), functor);
- std::sort(vec.begin(), vec.end(), functor);
- std::generate(vec.begin(), vec.end(), functor);
- ....

ファンクタを使うことで アルゴリズムの用途が広がる！

- 例えば、配列の和を計算するstd::accumulateさん。
  - accumulate(vec.begin(),vec.end(),0,plus<int>());

    配列vec内の要素をplusという演算にかけ、それを繰り返している



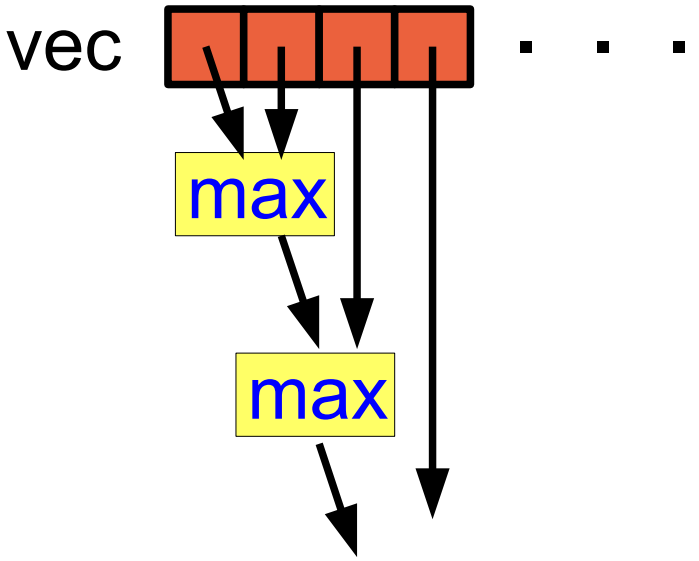
Question

plus以外のファンクタを渡したらどうなるか？

ex) multiplies, max

デフォルトでは配列の和を求めるaccumulateさんだが・・・

- 例えば、accumulateのファンクタにmaxを指定してみよう！(maxは２つの引数のうち大きい方の値を返す)
  - accumulate(vec.begin(),vec.end(),vec[0],max<int>());

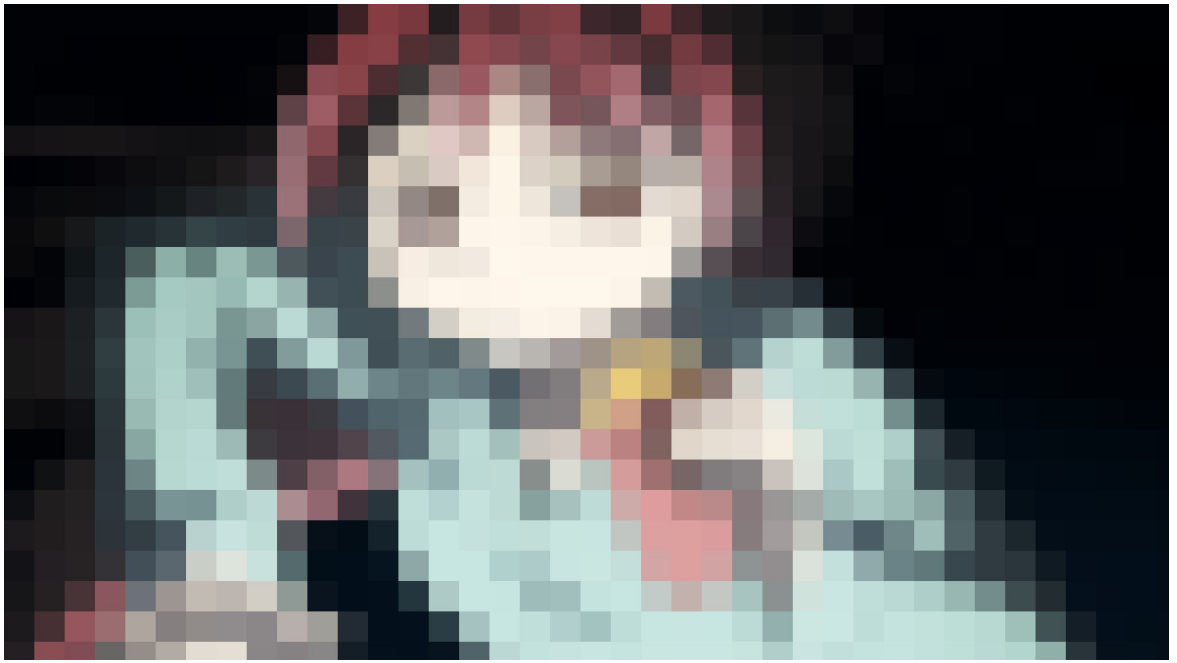


vec配列の最大値が求まる！

ファンクタを適切に選ぶことでアルゴリズムの使い方が広がる！

※ただし、STLには配列の最大値を求める
max_elementという関数があります。

適切な関数がある場合は
必ずそれを使いましょう。



関数名と合わない用法は非推奨です。
杏子さんに怒られます。

本題

「私、すべての一時的な関数を、
その場で定義したい！」

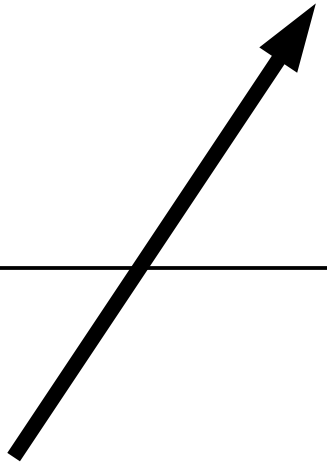
ラムダ式

ファンクタはoperator()をもつクラスでした

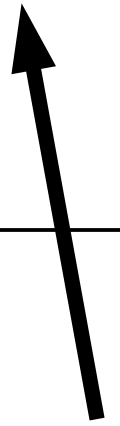
- C++03までの問題点
  - ファンクタを自作する場合、クラスとして新しく定義する必要があった。
  - 簡単なファンクタ(2倍するとか。√をとるとか)のために、わざわざ新しいクラスを作りたくない！
- C++0xでは
  - ラムダ式によって関数内で、即座に無名関数を作れる

使用例

```
std::for_each( vec.begin(), vec.end(),
                []( int n ) {
                    std::cout << n << std::endl;
                } );
```



ファンクタを渡すところに、
ラムダ式を記述



配列vecの値は
nとして使える

こうなると
とても便利

魔法言語 C++0x

LINGUA MAGI
しーぷらぷら
おーえっくす

第5話　“initializer_list”

「後悔なんてあるわけ無い」

initializer_list

- C++03ではvectorの初期化が組み込み配列のようにできなかった。
  - 組み込み配列での初期化

    int array[] = { 1, 2 ,3 ,4 ,5 ,6 };
  - std::vectorの初期化

    std::vector<int> vec = { 1,2,3,4,5 }; // エラー
- C++0xではvectorでも組み込み配列のように初期化するための構文initializer_listが追加された。

C++0xでのvectorの初期化

```
// 組み込み配列と同じように初期化できる
std::vector<int> vec = { 1,2,3,4 } ;
```

- 初期化のインターフェイスが組み込み配列と統一できる
- もちろん自作クラスでもinitializer_listは利用できます

自作クラスの場合

```
struct MyType {
    MyType(initializer_list<int> params){
            for(int* param: params)
                cout << "Parameter: " << *param << endl;
    }
};

int main(void){
    MyType mt = {1, 2, 3}; // ←こういう表記が可能に！
    return 0;
}
```

自作クラスの場合

```
struct MyType {
    MyType(initializer_list<int> params){
            for(int* param: params)
                cout << "Parameter: " << *param << endl;
    }
};

int main(void){
    MyType mt = {1, 2, 3}; // ←こういう表記が可能に！
    return 0;
}
```

あれ？

自作クラスの場合

```
struct MyType {
    MyType(initializer_list<int> params){
            for(int* param: params)
                cout << "Parameter: " << *param << endl;
    }
};

int main(void){
    MyType mt = {1, 2, 3}; // ←こういう表記が可能に！
    return 0;
}
```

あれ？

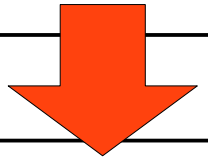
```
for(int* param: params) {

}
```
って書き方あったっけ？

あるんです！

- C++0xからrange-based for文という、範囲アクセス構文が追加されました。
- range-based for文を使うとコンテナの要素に対する処理が簡単に書けます。

C++03

```
for(vector<int>::iterator itr=vec.begin();itr!=vec.end();++itr) {
    cout << *itr << endl;
}
```



C++0x

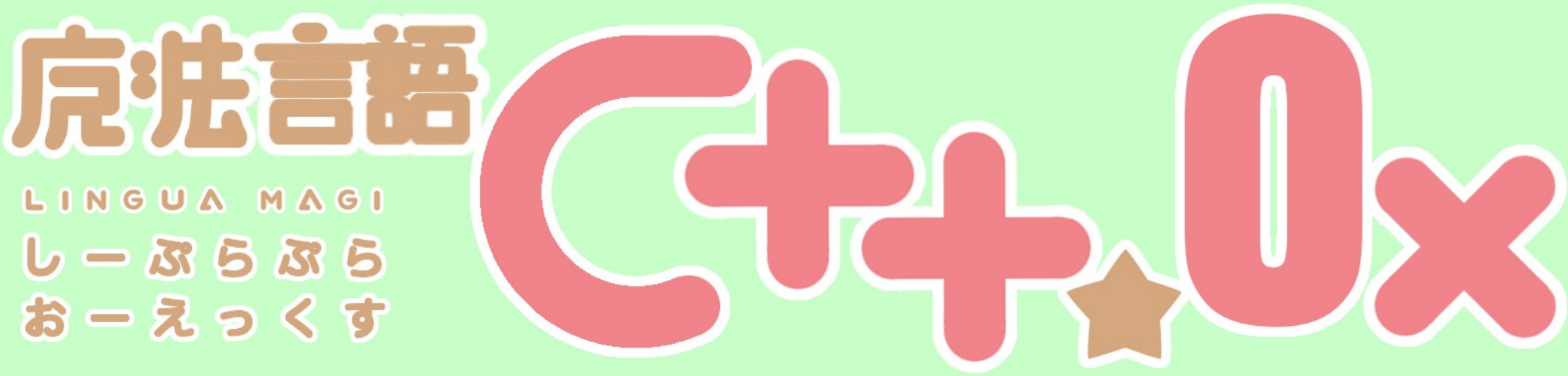
```
for(int n : vec) {
    cout << n << endl;
}
```

驚くほど
スッキリします！

なんだか
ねじ込んだかのような
range-based forの説明ですが、

後悔なんて！

あるわけない！



第6話“regex”

「こんなの絶対おかしいよ」

C++での文字列処理

- C++は昔から、文字列処理が苦手な子だった。
- 文字列処理が出来る子なら、きっとC++サーブレットができたはず！（？）
- そんな、そんなC++に待望の！

C++での文字列処理

- C++は昔から、文字列処理が苦手な子だった。
- 文字列処理が出来る子なら、きっとC++サーブレットができたはず！（？）
- そんな、そんなC++に待望の！

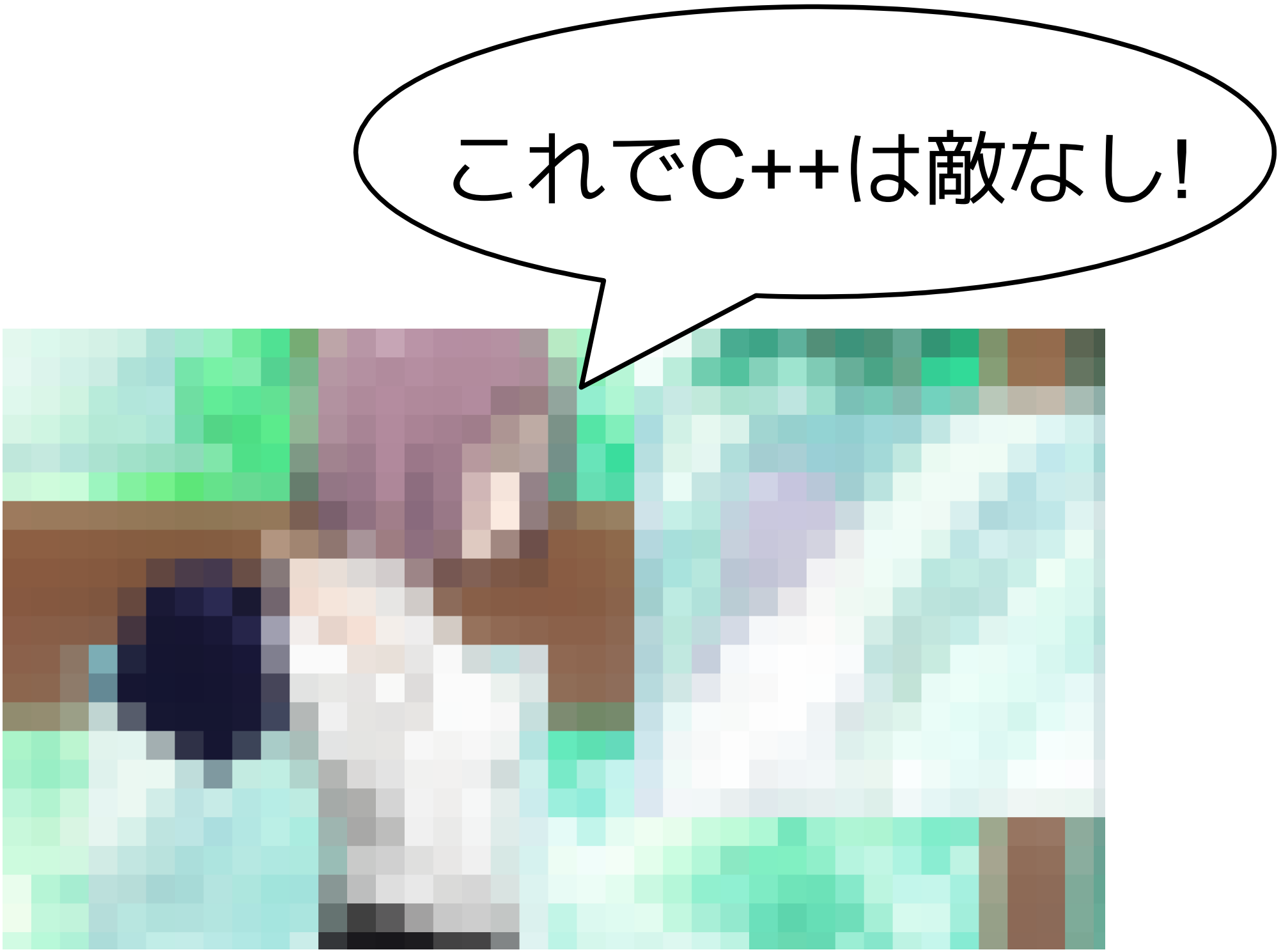
正規表現が！

正規表現 regex

- ついにC++にも、強力な文字列操作ライブラリが導入されました。
- ECMAScript、basic、extended、awk、grep、egrep等の方言が使えます。

```
std::regex r("<[^>]+>");
std::string str = "template <class T> hoge";
std::string after = "<censored>";

std::cout <<
    std::regex_replace(str, r, after)
<< std::endl;
    // "template <censored> hoge"
```

※ ソースコードはhttp://www.slideshare.net/faithandbrave/dev10-c0x より引用


これでC++は敵なし!

札幌C++勉強会メンバーの反応

-
-
-

札幌C++勉強会メンバーの反応

- lapisさん「regexよく知らない」
-
-

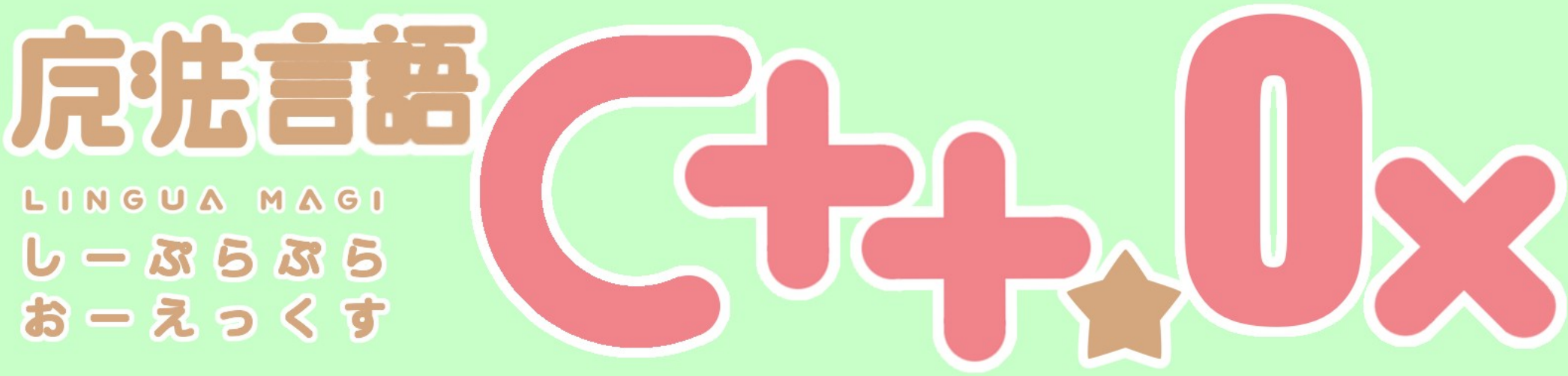
札幌C++勉強会メンバーの反応

- lapisさん「regexよく知らない」
- h.hiroさん「regexはちょっと」
-

札幌C++勉強会メンバーの反応

- lapisさん「regexよく知らない」
- h.hiroさん「regexはちょっと」
- ignisさん「regexとかほむほむ ほむほむ....」

こんなの絶対おかしいよ！！



第7話“random”

「本当の気持ちと向き合えますか？」

乱数　random

- これまでのC++では、乱数生成にC言語由来のrand()関数しかなかった。
- rand()は乱数の質が悪く、使いにくい。
- C++0xでは複数の乱数生成器と、分布クラスが利用出来るようになる。

乱数　random

- 乱数生成器
  - メルセンヌ・ツイスター法などが利用可
- 分布(distribution)クラス
  - 一様分布(整数 or 実数)、ベルヌーイ分布、
    幾何分布、ポアソン分布、二項分布、
    指数分布、正規分布、ガンマ分布
    が用意されている。

使い方

```
std::mt19937  gen;
std::uniform_int_distribution<> dst(0,9);

for(int i=0;i<5;++i) {
    const int random_number = dst(gen);
    std::cout << random_number << std::endl;
}
```

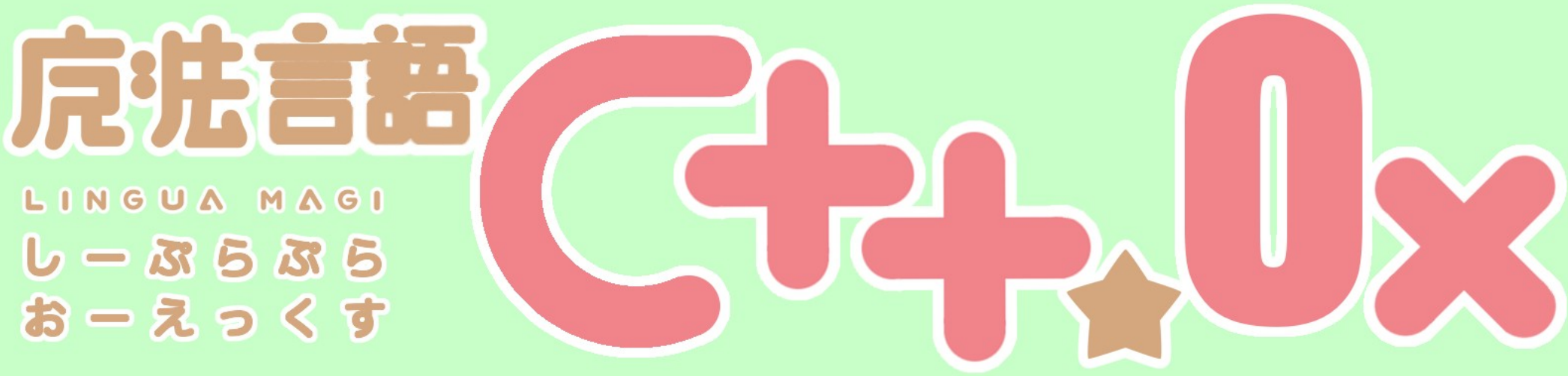
使い方

メルセンヌツイスターの生成器

```
std::mt19937  gen;
std::uniform_int_distribution<> dst(0,9);

for(int i=0;i<5;++i) {
    const int random_nu
    std::cout << random_
}
```

一様分布(整数)

ndl;



第8話　“thread”

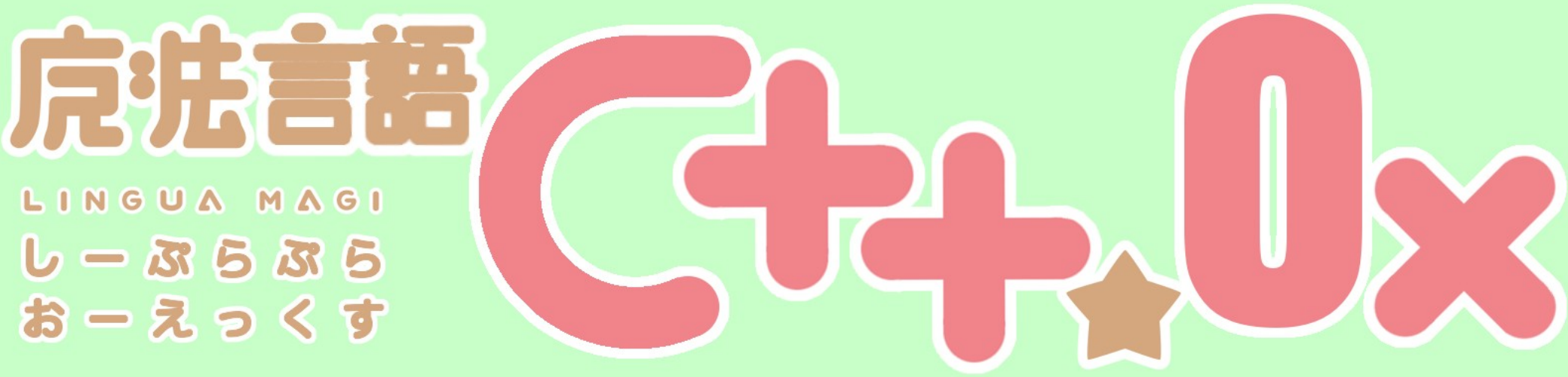
「あたしって、ほんとバカ」

thread

C++0xから、標準のスレッドが
導入されます！

使い方

```
int main() {
    std::thread th(
        [](){
            for(int i = 0; i < 10; ++i) {
                std::cout << i << std::endl;
            }
        });
    th.join();
}
```



第9話“final”

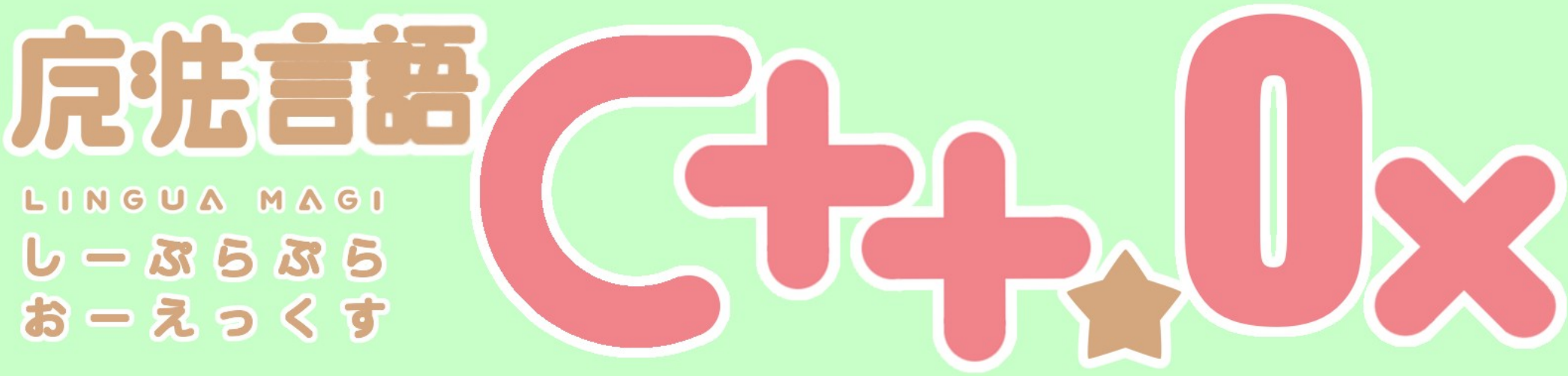
「そんなの、あたしが許さない」

final/override

```
struct Base {
    virtual void f() const final;
};
struct Derived : Base {
    void f() const;
    // エラー：Derived::fがfinal Base::fを
    //オーバーライドしようとする。
};
```

final/override

```
struct Base {
    virtual void some_func(float);
};
struct Derived : Base {
    virtual void some_func(int) override;
    // 不正：基底クラスの仮想関数を
    //オーバーライドしてない
};
```



第10話“Container”

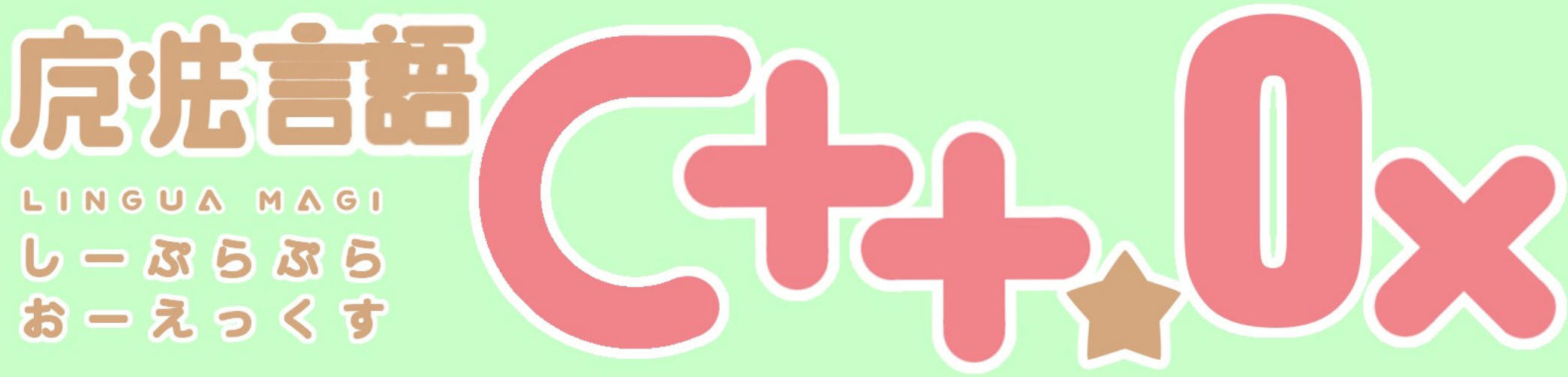
「もう誰にも頼らない」

Container

- C++0xから新たに連想配列コンテナが追加されました！
- unordered_set
- unordered_map
- unordered_multiset
- unordered_multimap

Container

```
unordered_map<string,int> um {
    {"Dijkstra",1972}, {"Scott",1976},
    {"Wilkes",1967}, {"Hamming",1968}
};
um["Ritchie"] = 1983;
for(auto x : um) {
    cout << '{' << x.first << ',' << x.second << '}';
}
```

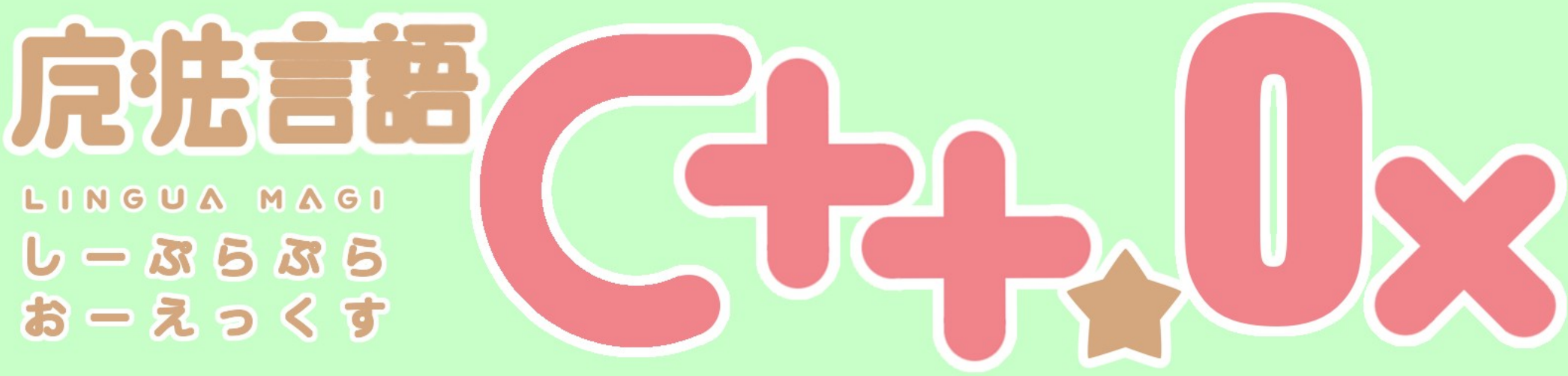


第11話　“Boost”

「最後に残った道しるべ」

Boost

- C++における準標準的なライブラリ
- C++0xの実験場として、C++標準化委員会のメンバがスタートさせた、オープンソースプロジェクトです
- C++の真髄ともいうべき変態的な（ｒｙ
- C++0xが使えない！そんな時はBoostで！
- C++0xが物足りない！そんな時もBoostで！！



最終話　“C++”

「わたしの、最高の友達」

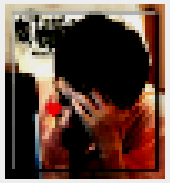
C++

- C++は世界中の標準化委員（ボランティア）によって規格化されています。
- 愛です。
- 0xによってC++はさらに便利になります
- みなさんも使いやすくなったC++を今以上に愛してやってください！

C++

- C++は世界中の標準化委員（ボランティア）によって規格化されています。
- 愛です。
- 0xによってC++はさらに便利になります
- みなさんも使いやすくなったC++を今以上に愛してやってください！
- こんな風に

@PG_kura



C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++ はぁはぁ C++..

2010-09-03 18:40:09 via ラーメン大陸

@SubaruG

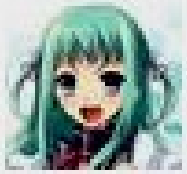
「C++知らない」「C++使い始めた」「C++分からない」「C++分かり始めた」「C++分かってきた」「C++分かった」「C++分かってなかった」「C++分からなくなってきた」「C++分からない」「C++分からない」「C++可愛い」「C++可愛くなんかなかった」「C++やっぱ可愛

2011-05-09 01:58:15 via みについ

@Flast_RO

夢のなかでもtemplateを書くようになってしまった・・・。起床の第一声が「コンパイルできるっ」なのはどういうことなんだ・・・

2010-10-05 10:38:02 via Ubuntu



@Cryolite
C6e

腹減ったから夜食に C++ でも食うか.

5月17日 webから　☆ お気に入り　リツイート　返信

tyruと他15人がリツイート

勉強会のお知らせ

- 7/3に札幌C++勉強会 #2を開催します
- 場所はここ、産業振興センターです
- C++の濃ゆい話が聞けます
- スピーカー募集中！
- ぜひお越しください！
- http://atnd.org/events/16805

```
std::cout <<
      “ありがとうございました！！”
<< std::endl;
```