Лекция 7. После boost::mpl

Метапрограммирование в C++

Список литературы

метапрограммирование

 *Abrahams D.*, *Gurtovoy A.* — C++ template metaprogramming : concepts, tools, and techniques from boost and beyond. — Addison-Wesley, 2005. — c. 373. — ISBN 0321227255. — URL: https://www.oreilly.com/library/view/c-template-metaprogramming/0321227255/.

 *Czarnecki K.*, *Eisenecker U.* — Generative programming : methods, tools, and applications. — Addison Wesley, 2000. — c. 832. — ISBN 0201309777.

Список литературы

C++11-14-17

 *Meyers S.* — Overview of the New C++ (C++11/14).

 *Stroustrup B.* — Programming : principles and practice using C++. — c. 1274. — ISBN 9780321992789.

Список литературы (продолжение)

Шаблоны по-новому


Joel Falcou E. A. — Practical C++ Metaprogramming. — OŔeilly, 2016. — ISBN 9781492042778.

Boost. Fusion

`https://www.boost.org/doc/libs/1_62_0/libs/fusion/doc/html/fusion/introduction.html`
требует:

- Гетерогенные контейнеры
- Обобщенные алгоритмы над ними
- Как tuple но с ленивым transform()

Примеры

Пример

```
#include <boost/fusion/sequence.hpp>
#include <boost/fusion/include/sequence.hpp>
vector<int, char, std::string> stuff(1, 'x', "howdy");
int i = at_c<0>(stuff);
char ch = at_c<1>(stuff);
std::string s = at_c<2>(stuff);
```

Примеры

Пример

```
#include <boost/fusion/algorithm.hpp>
#include <boost/fusion/include/algorithm.hpp>
struct print_xml
{
    template <typename T>
    void operator()(T const& x) const
    {
        std::cout
        << '<' << typeid(x).name() << '>'
        << x
        << "</" << typeid(x).name() << '>'
    ;
}
};
```

Boost.Hana

`https://boostorg.github.io/hana/`
требует:

- Generic lambdas
- Generalized constexpr
- Variable templates
- Automatically deduced return type
- type traits

Примеры

Пример

```
auto animals = hana::make_tuple(Fish{"Nemo"},
         Cat{"Garfield"}, Dog{"Snoopy"});
using namespace hana::literals;
// Access tuple elements with operator[] instead of std::get.
Cat garfield = animals[1_c];
// Perform high level algorithms on tuples (this is like   std::transform
auto names = hana::transform(animals, [](auto a) {
return a.name;
});
assert(hana::reverse(names) ==
   hana::make_tuple("Snoopy", "Garfield", "Nemo"));
```

Примеры

Пример

```
// 1. Give introspection capabilities to 'Person'
struct Person {
BOOST_HANA_DEFINE_STRUCT(Person,
   (std::string, name),
   (int, age)
 );
};
// 2. Write a generic serializer (bear with std::ostream for the example)
auto serialize = [](std::ostream& os, auto const& object) {
  hana::for_each(hana::members(object), [&](auto member) {
    os << member << std::endl;
  });
};
// 3. Use it
Person john{"John", 30};
```

Brigand

https://github.com/edouarda/brigand
https://github.com/edouarda/brigand/wiki/Introduction
требует:

What can you do with brigand

- Create a tuple from a list of types and then transform it into a variant
- Look for the presence of a type in a tuple and get its index
- Sort a list of types
- Advanced static assertion with arithmetics and complex functions
- Go through a list of types and perform a runtime action for each type

Примеры

Пример

```
#include <brigand/sequences/list.hpp>
#include <brigand/algorithms/find.hpp>
#include <type_traits>

using my_list = brigand::list<int, bool, char>;

// find_result will be 'bool, char'
using find_result = brigand::find<my_list,
std::is_same<brigand::_1, bool>>;
```

Примеры

Пример

```
#include <brigand/sequences/list.hpp>
#include <brigand/algorithms/transform.hpp>
#include <type_traits>

using brigand::_1;
using vanilla_list = brigand::list<char, bool, int>;
// ptr list will be 'char *, bool *, int *'
using ptr_list = brigand::transform<vanilla_list,
brigand::bind<std::add_pointer_t,_1>>;
```