Библиотека Boost

Александр Смаль

Академический университет
10 апреля 2014
Санкт-Петербург

Boost

- Boost — собрание библиотек, расширяющих функциональность C++.
- Свободно распространяются по лицензии Boost Software License вместе с исходным кодом.
- Библиотеки boost являются кандидатами на включение в следующий стандарт C++.
- Некоторые библиотеки boost были включены стандарт C++ 2011 года.
- При включении библиотеки в boost она проходит несколько этапов рецензирования.
- Имеет собственную систему сборки (Boost Build).
- Количество библиотек в текущей версии (1.55) — более сотни.
- Исходный код и документация доступны на `boost.org`.

Категории библиотек

- String and text processing
- Containers
- Iterators
- Algorithms
- Function objects and higher-order programming
- Generic Programming
- Template Metaprogramming
- Concurrent Programming
- Math and numerics
- Correctness and testing
- Data structures
- Domain Specific
- System
- Input/Output
- Memory
- Other: Image processing, Inter-language support, Language Features Emulation, Parsing, Patterns and Idioms, Programming Interfaces, State Machines, Broken compiler workarounds, Preprocessor Metaprogramming

```
#include <list>
#include <boost/any.hpp>

using boost::any_cast;
typedef std::list<boost::any> many;

void append_int(many & values, int value) {
    boost::any to_append = value;
    values.push_back(to_append);
}

void append_string(many & values, const std::string & value) {
    values.push_back(value);
}

void append_char_ptr(many & values, const char * value) {
    values.push_back(value);
}

void append_any(many & values, const boost::any & value) {
    values.push_back(value);
}

void append_nothing(many & values) {
    values.push_back(boost::any());
}
```

assign

```
#include <boost/assign/list_inserter.hpp> // for 'insert()'
#include <boost/assert.hpp>
using namespace std;
using namespace boost::assign;

int main() {
    vector<int> v;
    v += 1,2,3,4,5,6,7,8,9;

    map<string,int> months;
    insert( months )
        ( "january",   31 )( "february", 28 )
        ( "march",     31 )( "april",    30 )
        ( "may",       31 )( "june",     30 )
        ( "july",      31 )( "august",   31 )
        ( "september", 30 )( "october",  31 )
        ( "november",  30 )( "december", 31 );

    typedef pair< string,string > str_pair;
    deque<str_pair> deq;
    push_front( deq )( ``foo'', ``bar'')( ``boo'', ``far'' );
}
```

function

```
boost::function<float (int x, int y)> f;

struct int_div {
  float operator()(int x, int y) const { return ((float)x)/y; };
};

f = int_div();

std::cout << f(5, 3) << std::endl;

boost::function<void (int values[], int n, int& sum, float& avg)>
    sum_avg;

void do_sum_avg(int values[], int n, int& sum, float& avg) {}

sum_avg = &do_sum_avg;

struct X { int foo(int); };

boost::function<int (X*, int)> f;
f = &X::foo;

X x;
f(&x, 5);
```

```
struct image;

struct animation {
    void advance(int ms);
    bool inactive() const;
    void render(image & target) const;
};

std::vector<animation> anims;

template<class C, class P> void erase_if(C & c, P pred) {
    c.erase(std::remove_if(c.begin(), c.end(), pred), c.end());
}

void update(int ms) {
    std::for_each(anims.begin(), anims.end(),
        boost::bind(&animation::advance, _1, ms));
    erase_if(anims, boost::mem_fn(&animation::inactive));
}

void render(image & target) {
    std::for_each(anims.begin(), anims.end(),
        boost::bind(&animation::render, _1, boost::ref(target)));
}
```

bind(&string::size, _1) <
bind(&string::size, _2)

bind(, _1)

bind and function

```
struct button
{
    boost::function<void()> onClick;
};

struct player
{
    void play();
    void stop();
};

button playButton, stopButton;
player thePlayer;

void connect()
{
    playButton.onClick = boost::bind(&player::play, &thePlayer);
    stopButton.onClick = boost::bind(&player::stop, &thePlayer);
}
```

lexical_cast

```
#include <boost/lexical_cast.hpp>
#include <vector>

int main(int /*argc*/, char * argv[])
{
    using boost::lexical_cast;
    using boost::bad_lexical_cast;

    std::vector<short> args;
    while (*++argv) {
        try {
            args.push_back(lexical_cast<short>(*argv));
        }
        catch(const bad_lexical_cast &) {
            args.push_back(0);
        }
    }
}

void log_message(const std::string &);

void log_errno(int yoko) {
    log_message(''Error `` +
        boost::lexical_cast<std::string>(yoko) +
        ``: `` +  strerror(yoko));
}
```

```
optional<char> get_async_input() {
    if ( !queue.empty() )
        return optional<char>(queue.top());
    else return optional<char>(); // uninitialized
}

void receive_async_message() {
    optional<char> rcv ;
    // The safe boolean conversion from 'rcv' is used here.
    while ( (rcv = get_async_input()) && !timeout() )
        output(*rcv);
}
////////////////////////////////////
optional<string> name ;
if ( database.open() )
    name.reset ( database.lookup(employer_name) ) ;
else
    if ( can_ask_user )
        name.reset ( user.ask(employer_name) ) ;

if ( name )
    print(*name);
else
    print("employer's name not found!");
```

```
#include <climits>
#include <limits>
#include <cwchar>
#include <iterator>
#include <boost/static_assert.hpp>
#include <boost/type_traits.hpp>

BOOST_STATIC_ASSERT(std::numeric_limits<int>::digits >= 32);
BOOST_STATIC_ASSERT(WCHAR_MIN >= 0);

template <class RandomAccessIterator >
RandomAccessIterator foo(RandomAccessIterator from,
                         RandomAccessIterator to)
{
   // this template can only be used with
   // random access iterators...
   typedef typename std::iterator_traitsi<
            RandomAccessIterator >::iterator_category cat;
   BOOST_STATIC_ASSERT(
      (boost::is_convertible<
        (cat,
         const std::random_access_iterator_tag&>::value));
   //
   // detail goes here...
   return from;
}
```

String Algo

```
#include <boost/algorithm/string.hpp>
using namespace std;
using namespace boost;

string str1(" hello world! ");
to_upper(str1);  // str1 == " HELLO WORLD! "
trim(str1);      // str1 == "HELLO WORLD!"

string str2=
   to_lower_copy(
      ireplace_first_copy(
         str1,"hello","goodbye")); // str2 == "goodbye world!"

string str3("hello abc-*-ABC-*-aBc goodbye");
typedef vector< iterator_range<string::iterator> >
                                    find_vector_type;
find_vector_type FindVec; // #1: Search for separators
ifind_all( FindVec, str3, "abc" ); // { [abc],[ABC],[aBc] }

typedef vector< string > split_vector_type;

split_vector_type SplitVec; // #2: Search for tokens
split( SplitVec, str3, is_any_of("-*"), token_compress_on );
// { "hello abc","ABC","aBc goodbye" }
```

```cpp
#include "boost/variant.hpp"
#include <iostream>

struct my_visitor : public boost::static_visitor<int>
{
    int operator()(int i) const
    {
        return i;
    }

    int operator()(const std::string & str) const
    {
        return str.length();
    }
};

int main() {
    boost::variant< int, std::string > u("hello world");
    std::cout << u; // output: hello world

    int result = boost::apply_visitor( my_visitor(), u );
    std::cout << result;
    // output: 11 (i.e., length of "hello world")
}
```