# Лекция

Boost. Part 2

## what is Boost?

*full name: Boost C++ Libraries*

http://boost.org/

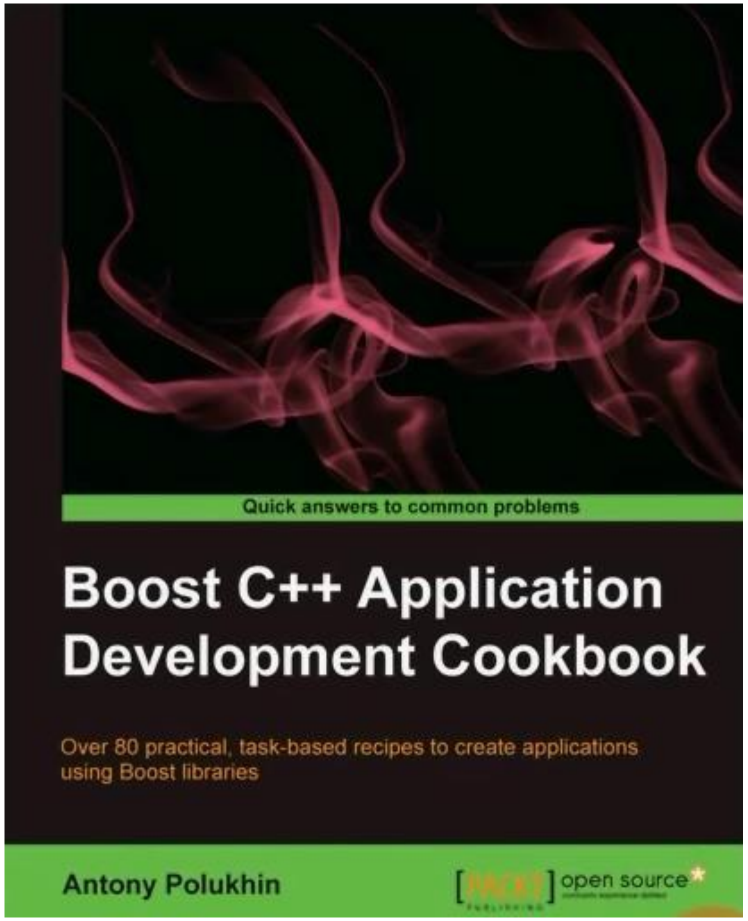
"The Boost C++ Libraries are a collection of free libraries that extend the functionality of C++"

»Wikipedia

## Help

- https://www.boost.org/doc
- https://theboostcpplibraries.com/raii-and-memory-management
- https://stackoverflow.com/questions

- Antony Polukhin. Boost C++ Application Development Cookbook. — Packt, 2013. — 348 с.


Quick answers to common problems
Boost C++ Application Development Cookbook
Over 80 practical, task-based recipes to create applications using Boost libraries
Antony Polukhin
[PACKT] open source

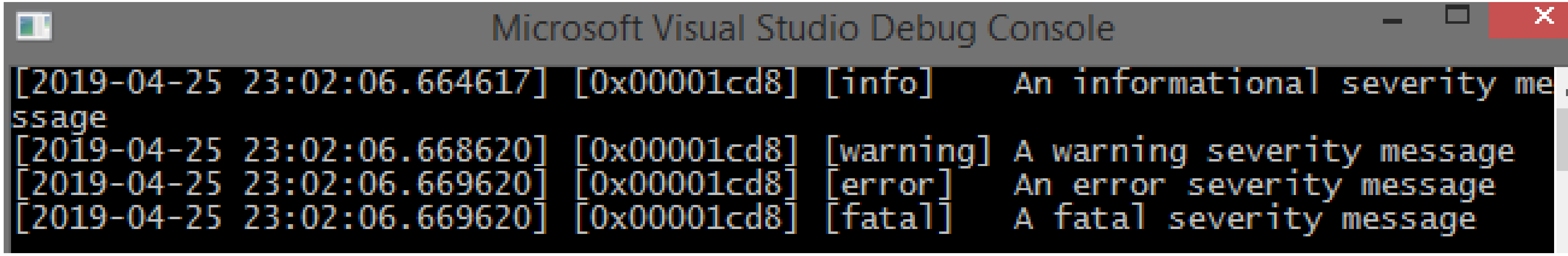
## Boost

- **Набор современных библиотек**, основанных на стандарте C++
- Лицензия позволяет использовать, изменять и распространять библиотеки **бесплатно**
- Библиотеки не зависят от платформы и поддерживают большинство **популярных компиляторов**
- **GitHub** используется в качестве хранилища кода
- **Сообщество Boost** появилось примерно в 1998 году, отвечает за разработку и публикацию библиотеки
- **Миссия сообщества** состоит в том, чтобы разрабатывать и собирать высококачественные библиотеки, которые дополняют стандартную библиотеку
- Boost часто представляет **ранний доступ** к новым разработкам по стандарту языка C++
- Благодаря отличной репутации библиотек Boost, их хорошее знание может быть **ценным навыком** для инженеров

## Boost

- **Boost Core:** generally-useful libraries and “vocabulary components and idioms”
- **Boost.ASIO:** networking and async. services
- **Boost.Concurrency:** suite of libraries to solve the issues in concurrent systems (HPC – high performance computing)
- **Boost.GIL:** powerful image processing
- **Boost.Math, Boost.Interval, Boost.Random, Boost.Accumulators, Boost.Numeric:** numerical computing
- **Boost.Python:** extending and embedding flow between Python and C++
- **Boost.MPL, Boost.Fusion, Boost.Proto:** metaprogramming concepts and frameworks
- **Boost.Spirit, Boost.Regex, Boost.String, Boost.Algorithm:** effective text processing libraries

## Содержание



```
[2019-04-25 23:02:06.664617] [0x00001cd8] [info]    An informational severity me
ssage
[2019-04-25 23:02:06.668620] [0x00001cd8] [warning] A warning severity message
[2019-04-25 23:02:06.669620] [0x00001cd8] [error]   An error severity message
[2019-04-25 23:02:06.669620] [0x00001cd8] [fatal]   A fatal severity message
```

**supports numerous back-ends to log data in various formats**

```cpp
#include <boost/log/core.hpp>
#include <boost/log/trivial.hpp>
#include <boost/log/expressions.hpp>
namespace logging = boost::log;
void init() {
    logging::core::get()->set_filter(
        logging::trivial::severity >= logging::trivial::info);
}
int main() {
    init();

    BOOST_LOG_TRIVIAL(trace) << "A trace severity message";
    BOOST_LOG_TRIVIAL(debug) << "A debug severity message";
    BOOST_LOG_TRIVIAL(info) << "An informational severity message";
    BOOST_LOG_TRIVIAL(warning) << "A warning severity message";
    BOOST_LOG_TRIVIAL(error) << "An error severity message";
    BOOST_LOG_TRIVIAL(fatal) << "A fatal severity message";
    return 0;
}
```

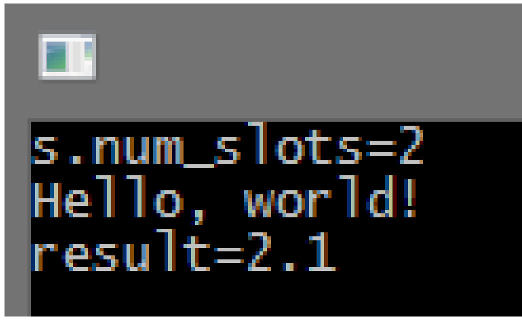
# Boost.Signals2

- Шаблон проектирования Наблюдатель (Observer) **process events flexibly**
- Событийное программирование supporting **event-driven development** **std::function** can also be used for **event handling**
- Механизм publish-subscribe
  - Компонент A хочет быть уведомлён о изменениях в компоненте B
  - Класс B публикует набор событий, о происхождении которых внутри него, он может уведомлять (multicast)
  - Остальные компоненты могут выбрать к каким событиям из списка подключиться
- Build-in C#, Java

# Boost.Signals2

**сигнал может быть отправлен компонентам приёмникам**

```cpp
#include <iostream>
#include <boost/signals2.hpp>
using namespace boost::signals2;
using namespace std;

void world() { cout << ", world!\n"; };

int main() {
    signal<void()> s;
    s.connect(1, world);
    s.connect(0, [] { cout << "Hello"; });

    cout << "s.num_slots=" << s.num_slots() << endl;
    if (!s.empty()) { s(); } // notify!

    s.disconnect(world);
    s.disconnect_all_slots();

    signal<float(float, float)> sig;
    sig.connect(0, [] (float a, float b) { return a + b; });
    cout << "result=" << *sig(1.4f, 0.7f) << endl;

    sig.disconnect(0);
    return 0;
}
```



```
s.num_slots=2
Hello, world!
result=2.1
```

## RAII and Memory Management

- **Boost.SmartPointers**
- **Boost.PointerContainer**
  - containers to store dynamically allocated
  - containers destroy objects with **delete** in the destructor
- **Boost.ScopeExit**
  - RAII idiom for any resources
  - no resource-specific classes need to be used
- **Boost.Pool**
  - not RAII
  - provide memory to your program faster
  - Object Usage vs. Singleton Usage
  - Out-of-Memory Conditions: Exceptions vs. Null Return
  - Ordered versus unordered

# Boost.Pool

**a fast memory allocator, and guarantees proper alignment of all allocated chunks**

```
#include <boost/pool/pool.hpp>
#include <boost/pool/object_pool.hpp>
#include <boost/pool/singleton_pool.hpp>
#include <vector>
class my { int i; };

struct my_sole_pool_tag {};
using sole_pool = boost::singleton_pool<my_sole_pool_tag, sizeof(int)>;

int main() {
    boost::pool<> pool(sizeof(int));
    boost::object_pool<my> obj_pool;
    std::vector<int, boost::pool_allocator<int>> vec;

    for (int i = 0; i < 10000; ++i) {
        int* const ptr = static_cast<int*>(pool.malloc());
        my* const obj_ptr = static_cast<my* const>(obj_pool.malloc());
        int* const i_ptr = static_cast<int* const>(sole_pool::malloc());
        // ... // do something with ptrs, don't take the time to free them
        vec.push_back(13 - i);
        // in order to force freeing the system memory of vector, you should call
    }//boost::singleton_pool<boost::pool_allocator_tag,sizeof(int)>::release_memory();
    return 0;
} // on function exit, pools are destroyed, all malloced ints are implicitly freed
// all destructors for the my objects are called
```

# Boost.Pool

```cpp
template <typename UserAllocator = default_user_allocator_new_delete>
class pool {
    pool(const pool &) = delete;      void operator=(const pool &) = delete;
    pool(pool&&) = default;           void operator=(pool&&) = default;
public:
    typedef typename UserAllocator                   user_allocator;
    typedef typename UserAllocator::size_type        size_type;
    typedef typename UserAllocator::difference_type  difference_type;

    explicit pool(size_type requested_size);
    ~pool();

    bool release_memory();
    bool purge_memory();

    bool is_from(void* chunk) const;
    size_type get_requested_size() const;

    void* malloc();
    void* ordered_malloc();
    void* ordered_malloc(size_type n);
    void  free(void* chunk);
    void  ordered_free(void * chunk);
    void  free(void* chunks, size_type n);
    void  ordered_free(void * chunks, size_type n);
};
```

# Boost.Pool

**extends and generalizes the framework provided by the Simple Segregated Storage solution**

```
struct default_user_allocator_new_delete {
    typedef std::size_t     size_type;
    typedef std::ptrdiff_t  difference_type;

    static char* malloc(const size_type bytes) {
        return new (std::nothrow) char[bytes];
    }
    static void free(char * const block) { delete[] block; }
};

struct default_user_allocator_malloc_free {
    typedef std::size_t     size_type;
    typedef std::ptrdiff_t  difference_type;

    static char* malloc(const size_type bytes) {
        return static_cast<char*>(std::malloc(bytes));
    }
    static void free(char * const block) { std::free(block); }
};
```

# Boost.PointerContainer

**specialized to manage dynamically allocated objects**

```
#include <boost/ptr_container/indirect_fun.hpp>
#include <boost/ptr_container/ptr_inserter.hpp>
#include <boost/ptr_container/ptr_vector.hpp>
#include <boost/ptr_container/ptr_set.hpp>
int main() {
    std::array<int, 3> arr{ 0, 1, 2 };
    boost::ptr_vector<int> vec; // works like std::vector<std::unique_ptr<int>>
    std::copy(arr.begin(), arr.end(), boost::ptr_container::ptr_back_inserter(vec));
    // vec expects addresses of dynamically allocated int objects,
    // inserter creates copies on the heap and adds the addresses to the container
    vec.push_back(new int{ 3 });
    std::cout << vec.size() << ' ' << vec.back() << '\n';

    boost::ptr_set<int> s;
    s.insert(new int{ 2 }), s.insert(new int{ 1 });
    std::cout << *s.begin() << '\n';

    std::set<std::unique_ptr<int>, //  together with resource manager
            boost::indirect_fun<std::less<int>>// must be told how to compare elements
    > v; // non-specialized container
    v.insert(std::unique_ptr<int>(new int{ 2 }));
    v.insert(std::unique_ptr<int>(new int{ 1 }));
    std::cout << **v.begin() << '\n';
    return 0;
}
```

# Boost.ScopeExit

**makes it possible to use RAII without resource-specific classes**

```cpp
#include <iostream>
#include <utility>

template <typename T>
struct scope_exit {
    T t;
    scope_exit(T &&t) : t{ std::move(t) } {}
    ~scope_exit() { t(); }
};
template <typename T>
scope_exit<T> make_scope_exit(T &&t) {
    return scope_exit<T>{ std::move(t) };
}

int* foo() {
    int *i = new int{ 10 };
    auto cleanup = make_scope_exit([&i]() mutable { delete i; i = 0; });
    std::cout << *i << '\n';
    return i;
}

int main() {
    int *j = foo();
    std::cout << j << '\n';
}
```

## Boost.Spirit

  - practical parsing tool, significantly reduces development time
- Develop parsers for text formats
- Formats are described with rules
- **Parsing Expression Grammar** (**PEG**) that is related to **Extended Backus-Naur-Form** (**EBNF**)
- Apply complex processes, and better scale, than
  - primitive tools (scanf)
  - pattern searching (regexp)
  - scanners (tokenizers)
- Have not to write code to implement parsing
  - **boost::spirit::qi** is a component to develop **parsers**
  - **boost::spirit::karma** is a component to develop **generators**
  - **boost::spirit::lex** is a component to develop **lexers**

## Boost.Spirit

**includes and aliases**

```cpp
#include <boost/config/warning_disable.hpp>
#include <boost/spirit/include/qi.hpp>
#include <boost/spirit/include/phoenix_core.hpp>
#include <boost/spirit/include/phoenix_operator.hpp>
#include <boost/spirit/include/phoenix_fusion.hpp>
#include <boost/spirit/include/phoenix_stl.hpp>
#include <boost/fusion/include/adapt_struct.hpp>
#include <boost/variant/recursive_variant.hpp>
#include <boost/foreach.hpp>

#include <iostream>
#include <fstream>
#include <string>
#include <vector>

namespace client {
namespace fusion = boost::fusion;
namespace phoenix = boost::phoenix;
namespace qi = boost::spirit::qi;
namespace ascii = boost::spirit::ascii;
}
```

# Boost.Spirit

**mini_xml declaration**

```
namespace client {
struct mini_xml;

typedef boost::variant<boost::recursive_wrapper<mini_xml>, std::string>
mini_xml_node;

struct mini_xml {
    std::string name;                            // tag name
    std::vector<mini_xml_node> children;         // children
};
}
// tell fusion about our mini_xml struct to make it a first-class fusion citizen
BOOST_FUSION_ADAPT_STRUCT(client::mini_xml, (std::string, name)
                          (std::vector<client::mini_xml_node>, children))
namespace client {
constexpr int tabsize = 4;
void tab(int indent) { for (int i = 0; i < indent; ++i) std::cout << ' '; }

struct mini_xml_printer {
    int indent;
    mini_xml_printer(int indent = 0) : indent(indent) {}
    void operator()(mini_xml const& xml) const;
};
}
```

# Boost.Spirit

**Printer xml nodes and all file**

```
namespace client {
struct mini_xml_node_printer : boost::static_visitor<> {
    int indent;
    mini_xml_node_printer(int indent = 0) : indent(indent) {}

    void operator()(mini_xml const& xml) const {
        mini_xml_printer(indent + tabsize)(xml);
    }
    void operator()(std::string const& text) const {
        tab(indent + tabsize);
        std::cout << "text: \"" << text << '"' << std::endl;
    }
};
void mini_xml_printer::operator()(mini_xml const& xml) const {
    tab(indent);
    std::cout << "tag: " << xml.name << std::endl;
    tab(indent);
    std::cout << '{' << std::endl;
    for (mini_xml_node const& node : xml.children)
        boost::apply_visitor(mini_xml_node_printer(indent), node);
    tab(indent);
    std::cout << '}' << std::endl;
}
}
```

# Boost.Spirit

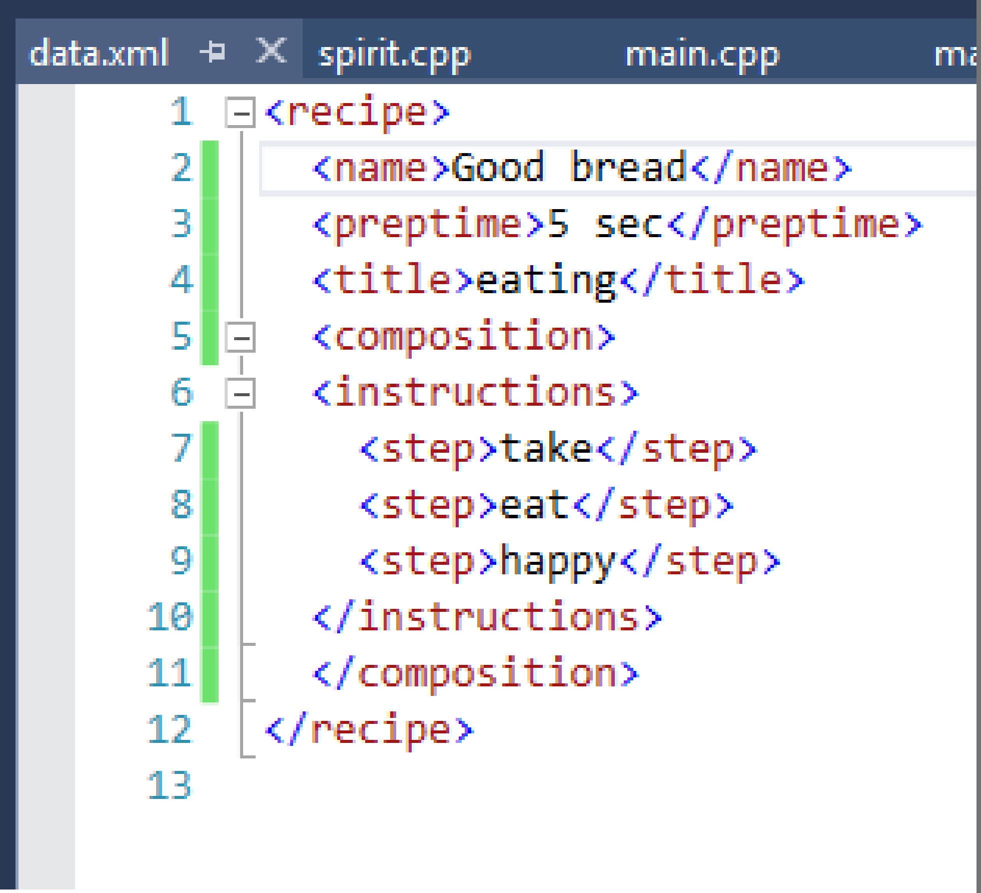
**parser instance**

```
namespace client {
template <typename Iterator>
struct mini_xml_grammar : qi::grammar<Iterator, mini_xml(), ascii::space_type> {
    mini_xml_grammar() : mini_xml_grammar::base_type(xml) {
        using namespace qi::labels;
        text = qi::lexeme[+(ascii::char_ - '<')[_val += _1]];
        node = (xml | text)[_val = _1];

        start_tag = '<' >> !qi::lit('/')
                        >> qi::lexeme[+(ascii::char_ - '>')[_val += _1]] >> '>';

        end_tag = "</" >> qi::lit(_r1) >> '>';

        xml = start_tag[phoenix::at_c<0>(_val) = _1]
            >> *node[phoenix::push_back(phoenix::at_c<1>(_val), _1)]
            >> end_tag(phoenix::at_c<0>(_val));
    }
    qi::rule<Iterator, mini_xml(), ascii::space_type> xml;
    qi::rule<Iterator, mini_xml_node(), ascii::space_type> node;
    qi::rule<Iterator, std::string(), ascii::space_type> text;
    qi::rule<Iterator, std::string(), ascii::space_type> start_tag;
    qi::rule<Iterator, void(std::string), ascii::space_type> end_tag;};
}
}
```

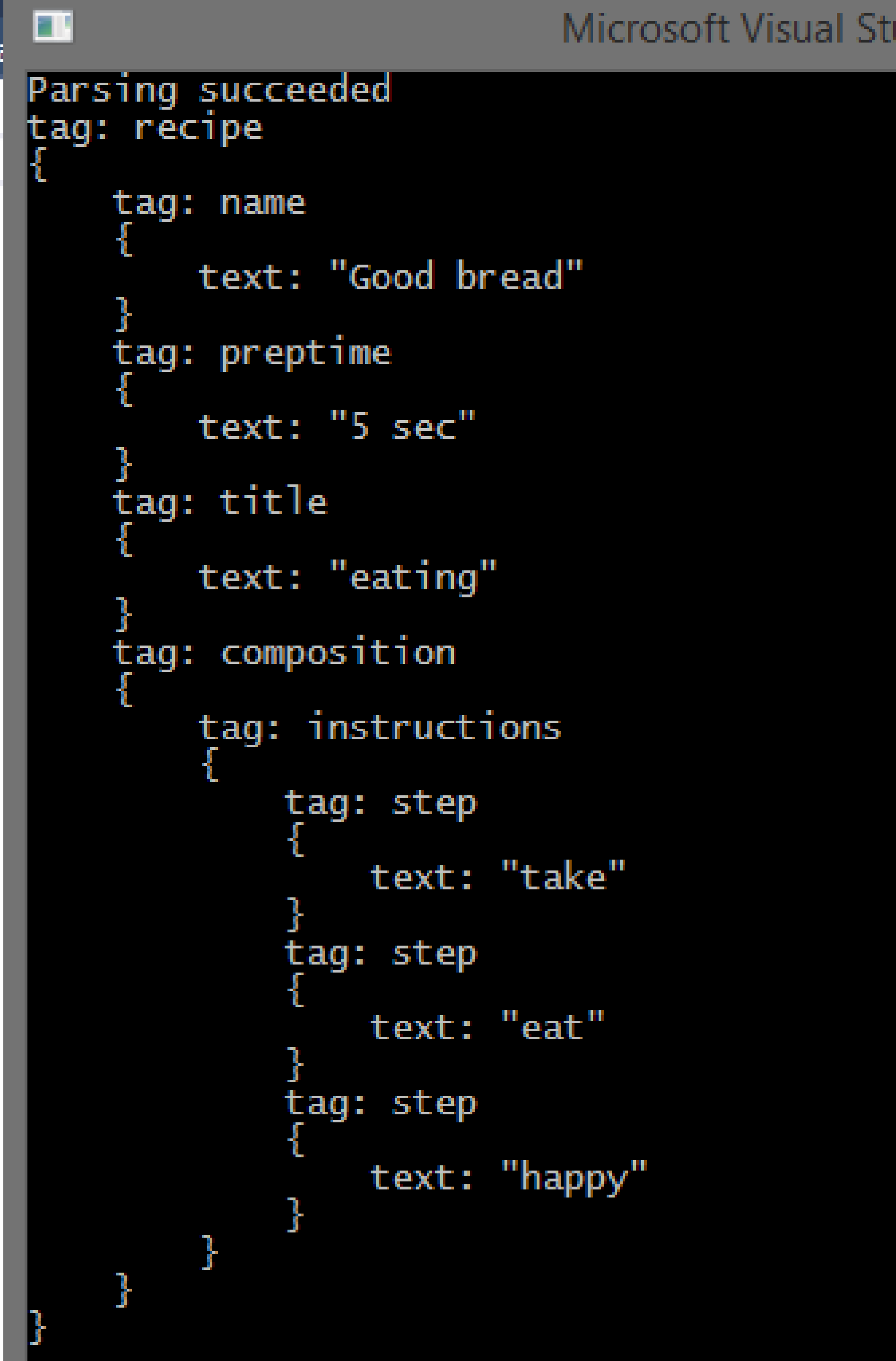
# Boost.Spirit

**how it works**

```
int main(int argc, char **argv) {
    std::ifstream in(argv[1], std::ios_base::in);
    std::string storage; // We will read the contents here.
    in.unsetf(std::ios::skipws); // No white space skipping!
    std::copy(std::istream_iterator<char>(in),
              std::istream_iterator<char>(), std::back_inserter(storage));

    typedef client::mini_xml_grammar<std::string::const_iterator> mini_xml_grammar;
    mini_xml_grammar xml; // Our grammar
    client::mini_xml ast; // Our tree

    std::string::const_iterator iter = storage.begin(), end = storage.end();
    bool r = phrase_parse(iter, end, xml, boost::spirit::ascii::space, ast);

    if (r && iter == end) {
        std::cout << "Parsing succeeded\n";
        client::mini_xml_printer{}(ast);
    } else {
        std::string::const_iterator some = iter + std::min(30, int(end - iter));
        std::string context(iter, (some > end) ? end : some);
        std::cout << "Parsing failed\n" << "stopped at: \"" << context << "...\"\n";
    }
    return 0;
}
```



```
<recipe>
  <name>Good bread</name>
  <preptime>5 sec</preptime>
  <title>eating</title>
  <composition>
  <instructions>
    <step>take</step>
    <step>eat</step>
    <step>happy</step>
  </instructions>
  </composition>
</recipe>
```



```
Parsing succeeded
tag: recipe
{
    tag: name
    {
        text: "Good bread"
    }
    tag: preptime
    {
        text: "5 sec"
    }
    tag: title
    {
        text: "eating"
    }
    tag: composition
    {
        tag: instructions
        {
            tag: step
            {
                text: "take"
            }
            tag: step
            {
                text: "eat"
            }
            tag: step
            {
                text: "happy"
            }
        }
    }
}
```

## boost::algorithm::string

**преобразование из строкового вида**

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

void f()
{
    string test = "hello  world\r\n";
    trim(test);       // <hello  world>
    trim_all(test);   // <hello world>
    to_upper(test);   // <HELLO WORLD>
}
```

# boost::tokenizer

**default behavior**

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

void f()
{
  string s = "To be, or not to be?";

  tokenizer<char_separator<char>> t(s);

  for (string part : t)
    cout << "<" << part << ">" << endl;
}
```

**Output**

```
<To>
<be>
<or>
<not>
<to>
<be>
```

# boost::tokenizer

**custom characters separation**

```cpp
#include <boost/tokenizer.hpp>
using namespace std;
using namespace boost;

void f()
{
  string s = "To be, or not to be?";
  char_separator<char> sep("o", " ", keep_empty_tokens);
  tokenizer<char_separator<char>> t(s, sep);

  for (string part : t)
    cout << "<" << part << ">" << endl;
}
```

**Output**

```
<T>
< >
<be,>
< >
<>
<r>
< >
<n>
<t>
< >
<t>
<>
< >
<be?>
```

## boost::lexical_cast

**преобразование из строкового вида**

```cpp
#include <boost/lexical_cast.hpp>
using namespace std;
using namespace boost;

void f()
{
    // std::to_string??
    // atoi??
    string s = "2.1";
    double d = lexical_cast<double>(s);

    try {
        lexical_cast<int>("abcde");
    } catch (const bad_lexical_cast& e) {
        cout << e.what() << endl;
    }
}
```

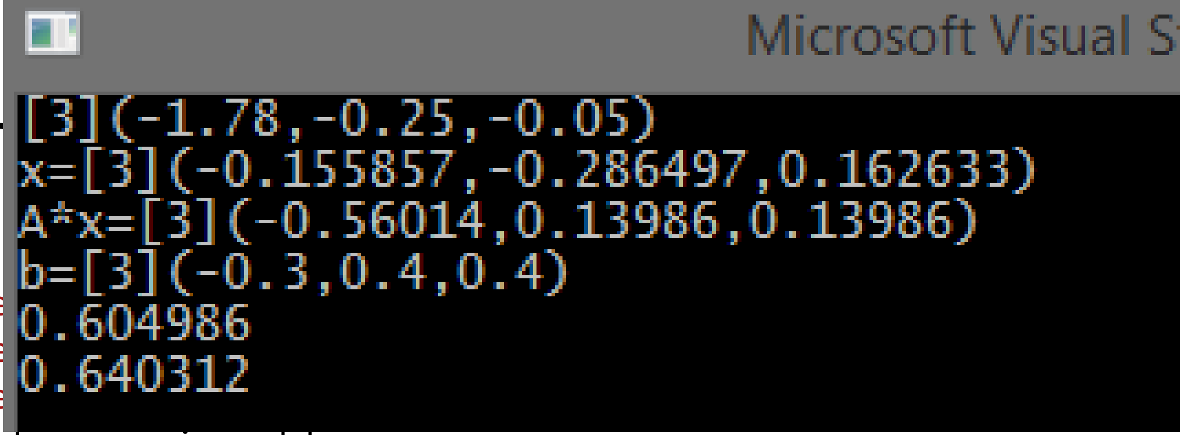
# Boost.Serialization

```cpp
#include <boost/archive/binary_oarchive.hpp>
#include <boost/archive/text_oarchive.hpp>
class Data {
    std::shared_ptr<std::vector<double>> pv{};
    unsigned long num{}, seed{};
    std::stack<double> mean{};

    friend class boost::serialization::access;
    template <class Archive>
    void serialize(Archive & ar, const unsigned int version) {
        ar & pv & num & seed & mean;
    }
public:
    Data() = default;
    void save_text(const char *filename) const {
        std::ofstream f(filename);
        boost::archive::text_oarchive toa(f);
        toa & *this;
    }
    void save_binary(const char *filename) const {
        std::ofstream f(filename, std::ios::binary);
        boost::archive::binary_oarchive boa(f);
        boa & *this;
    }
}; // end class Data
BOOST_CLASS_VERSION(Data, 2 /*version*/)
```

# Boost.Num

**линейная алгебра**

```
#include <boost/nume
#include <boost/nume
#include <boost/nume
using namespace boost::numeric::ublas;
void main() {
    matrix<double> A(3, 3, -0.5);
    A(0, 0) = A(2, 2) = 1.8;
    A(0, 2) = -2.6; A(2, 0) = 1.9;
    vector<double> b(3, 0.4); b(0) = -0.3;
    matrix<double> A1 = A + matrix<double>(3, 3, -0.93);
    vector<double> x = b;
    matrix_row<matrix<double>> mr(A, 2);
    matrix_column<matrix<double>> mc(A, 2);
    std::cout << prod(A, b) << std::endl;
    permutation_matrix<double> P1(A1.size1());
    lu_factorize(A1, P1);
    lu_substitute(A1, P1, x);
    std::cout << "x=" << x << std::endl;
    std::cout << "A*x=" << prod(A, x) << std::endl;
    std::cout << "b=" << b << std::endl;
    std::cout << norm_1(x) << std::endl;
    std::cout << norm_2(b) << std::endl;
}
```

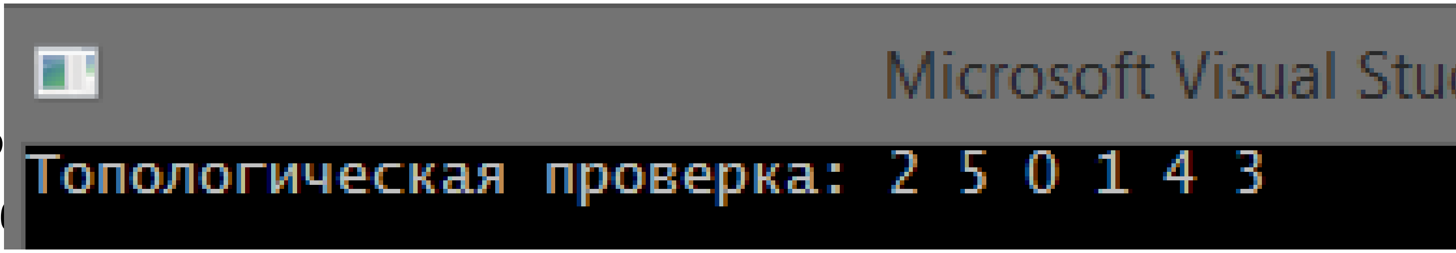


```
[3](-1.78,-0.25,-0.05)
x=[3](-0.155857,-0.286497,0.162633)
A*x=[3](-0.56014,0.13986,0.13986)
b=[3](-0.3,0.4,0.4)
0.604986
0.640312
```

**uBLAS is a C++ version of Fortran package BLAS with a STL conforming iterator interface**

# Boos

**предоставляет ги**



```
Microsoft Visual Stu
Топологическая проверка: 2 5 0 1 4 3
```

```
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/topological_sort.hpp>
int main() {
    using namespace boost;
    std::setlocale(LC_ALL, "English_USA.1251");
    typedef adjacency_list<vecS, vecS, directedS,
        property<vertex_color_t, default_color_type> > Graph; // тип графа
    typedef boost::graph_traits<Graph>::vertex_descriptor Vertex;// дескриптор вершин
    typedef std::vector<Vertex> container; // контейнер для цепочки вершин
    typedef std::pair<std::size_t, std::size_t> Pair; // тип представления дуг графа
    Pair edges[6] = { Pair(0,1), Pair(2,4), Pair(2,5),
                      Pair(0,3), Pair(1,4), Pair(4,3) }; // Дуги графа
    Graph G(edges, edges + 6, 6); // Граф
    // словарь для получения номеров вершин по дескриптору вершин
    boost::property_map<Graph, vertex_index_t>::type id = get(vertex_index, G);
    container c; // контейнер для хранения отсортированных вершин
    topological_sort(G, std::back_inserter(c)); // выполнение алгоритма
    // Вывод результата: перебор дескрипторов графа в контейнере,
    // получение порядковых номеров вершин
    std::cout << "Топологическая проверка: ";
    for (container::reverse_iterator ii = c.rbegin(); ii != c.rend(); ++ii)
        std::cout << id[*ii] << " ";
    std::cout << std::endl;
    return 0;
}
```

# Именование lib

**single letter is the tag**

```
#pragma comment( lib, "libboost_test_exec_monitor-vc141-mt-x64-1_66.lib" )
#pragma comment( lib, "libboost_test_exec_monitor-vc141-mt-s-x64-1_66.lib")
#pragma comment( lib, "libboost_unit_test_framework-vc141-mt-sgd-x64-1_66.lib")
#pragma comment( lib, "libboost_unit_test_framework-vc141-mt-gd-x64-1_66.lib")

// -mt Threading tag: the library was built with multithreading support enabled
// -d  ABI tag: the library's interoperability with other compiled code
//     For each such feature, a single letter is added to the tag:
//
//  Key   Use this library when (Boost.Build option)
//
//  s     linking statically to the C++ standard library
//        and compiler runtime support libraries
//        (runtime-link=static)
//  g     using debug versions of the standard and runtime support libraries
//        (runtime-debugging=on)
//  y     using a special debug build of Python
//        (python-debugging=on)
//  d     building a debug version of your code
//        (variant=debug)
//  p     using the STLPort standard library rather than
//        the default one supplied with your compiler
//        (stdlib=stlport)
```

## #pragma comment

**фишка исключительно компилятора Microsoft**

```
// автоматическая линковка статических библиотек
#pragma comment( lib, "libname.lib" )
#pragma comment( lib, "emapi" )

// при вызове линкера будет использован доп.параметр /include:__mySymbol
#pragma comment( linker, "/include:__mySymbol" )

// в OBJ файл будет записано имя и версия компилятора, просто текст
#pragma comment( compiler )

// строка "Compiled on <compile-date> at <compile-time>"
// будет записана в OBJ файл просто в виде текста
#pragma comment( exestr, "Compiled on " __DATE__ " at " __TIME__ )
#pragma comment( exestr, "Ваша строка, просто будет болтаться в EXE файле" )

// GCC так не умеет, а MSDN также говорит, что
#pragma( exestr, "ваш комментарий" )
// устаревшая и в будущих версиях компилятора поддерживаться не будет
// вместо неё нужно использовать:
#pragma( user, "ваша строка коммента" )
```

СПАСИБО ЗА ВНИМАНИЕ!