# Курс «Проектирование больших систем на языке C++»

Лекция

**boost**

## what is Boost?

*full name: Boost C++ Libraries*

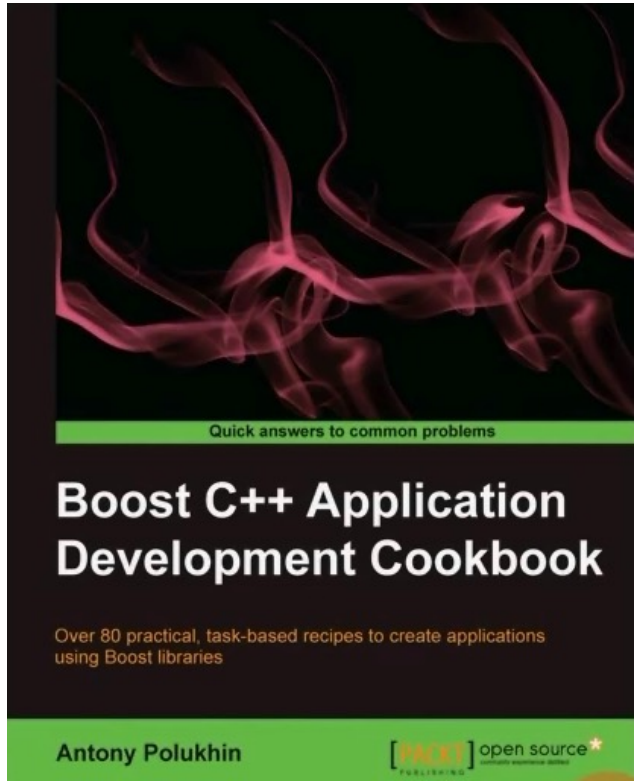
http://boost.org/

"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




- A. Polukhin. Boost C++ Application Development Cookbook. — Packt, 2013. — 348 c.

# Boost

- **Набор современных библиотек**, основанных на стандарте C++, часто представляет **ранний доступ** к новым разработкам по стандарту языка C++
- Библиотеки не зависят от платформы и поддерживают большинство **популярных компиляторов**
- **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
- Благодаря отличной репутации библиотек Boost, их хорошее знание может быть **ценным навыком** для инженеров
- Лицензия позволяет использовать, изменять и распространять библиотеки **бесплатно**
- **GitHub** используется в качестве хранилища кода

# Boost

- **Boost Core:** основа библиотеки
- **Boost.ASIO:** сетевое взаимодействие и асинхронное программирование
- **Boost.Concurrency:** многопоточные и асинхронные вычисления, HPC – high performance computing
- **Boost.GIL:** обработка изображений
- **Boost.Math, Boost.Interval, Boost.Random, Boost.Accumulators, Boost.Numeric:** вычисления, численные методы и т.п.
- **Boost.Python:** расширение и взаимодействие с языком Python
- **Boost.MPL, Boost.Fusion, Boost.Proto:** метапрограммирование
- **Boost.Spirit, Boost.Regex, Boost.String, Boost.Algorithm:** эффективные средства для работы с текстом

# Тестирование программ

- “A **program** that has **not** been **tested** does **not work**”
- “A systematic way to search for errors”
- “Test early and often”

Bjarne Stroustrup

# Тестирование программ

- Без написания *модульных тестов* нельзя достигнуть *приемлемого качества* в достаточно сложной системе
- *Повысить скорость разработки* за счет экономии времени на отладке
- *Критично взглянуть на интерфейсы*, выявить несостыковки, обеспечить простоту и логичность
- Если *писать и запускать тесты сложно*,
  то *покрытие тестами* будет слабое

  - https://github.com/google/googletest
  - https://github.com/unittest-cpp/unittest-cpp
  - https://www.boost.org/doc/libs/latest/libs/test/doc/html/index.html

# Тестирование программ
# организация юнит-теста

- **Юнит-тест** (*модульный тест*) – *отдельное приложение*, при запуске которого происходит выполнение набора тестов и возвращается результат, из которого становится очевидно выполнены ли все тесты без ошибок (txt/xml)
- Из выводимых сообщений можно узнать в каком месте теста произошло нарушение условий
- **Юнит-тест** может состоять из
  - *нескольких файлов с кодом* – физическая организация,
  - *нескольких наборов* (**suite**) *тестов* (**case**) – логическая организация
- **Тест-кейс** – несколько утверждений, нарушение которых означает ошибку
- например, *модульные тесты библиотеки* должны содержать несколько наборов тестов, *по набору* на каждый *тестируемый класс*, и в наборе *по тесту* на каждый *тестируемый метод*

# Boost.Test

**простой класс, единственная задача которого – деление и умножение**

```
1.  #include <stdexcept>
2.  using namespace std;

3.  class Calculator {
4.      int Value_;
5.  public:
6.      explicit Calculator(int value) : Value_(value) {}
7.      void Divide(int value) {
8.          if (value == 0)
9.              throw std::invalid_argument("Деление на ноль!");
10.         Value_ /= value;
11.     }
12.     void Multiply(int value) { Value_ *= value; }
13.     int Result() const { return Value_; }
14. };
```

# Boost.Test

**фреймворк**

```
1.  #include "calculator.h"
2.  #include <boost/test/unit_test.hpp>

3.  #define BOOST_TEST_MODULE testCalculator

4.  BOOST_AUTO_TEST_CASE(testCalculator)
5.  {
6.      Calculator calculator(12);
7.      BOOST_CHECK_EQUAL(calculator.Result(), 12);
8.      calculator.Divide(3);
9.      BOOST_CHECK_EQUAL(calculator.Result(), 4);
10.     calculator.Divide(2);
11.     BOOST_CHECK_EQUAL(calculator.Result(), 2);
12.     calculator.Multiply(2);
13.     BOOST_CHECK_EQUAL(calculator.Result(), 4);
14.     calculator.Multiply(3);
15.     BOOST_CHECK_EQUAL(calculator.Result(), 12);
16. }
```

# Boost.Test

**типы проверок**

- **BOOST_AUTO_TEST_SUITE**
  -
- **BOOST_CHECK(условие)**
  -
- **BOOST_CHECK_EQUAL(val _1, val_2)**
  -
- **BOOST_CHECK_CLOSE(val_1, val_2, точность)**
  -
- **BOOST_CHECK_BITWISE_EQUAL(val_1, val_2)**
  -
- **BOOST_CHECK_EQUAL_COLLECTIONS(begin1, end1, begin2, end2)**
  -
- **BOOST_CHECK_THROW(инструкция, исключение)**
  -
- **BOOST_CHECK_NO_THROW(инструкция)**
  -

# Boost.Test

**типы проверок**

- **BOOST_AUTO_TEST_SUITE**
  - определения набора тестов
- **BOOST_CHECK(условие)**
  - простейшая проверка истинно условие или нет
- **BOOST_CHECK_EQUAL(val _1, val_2)**
  - проверка на равенство двух значений
- **BOOST_CHECK_CLOSE(val_1, val_2, точность)**
  - проверка на равенство чисел с плавающей точкой
- **BOOST_CHECK_BITWISE_EQUAL(val_1, val_2)**
  - проверит 2 значения побитово, сообщит место различия
- **BOOST_CHECK_EQUAL_COLLECTIONS(begin1, end1, begin2, end2)**
  - проверка 2 последовательностей (контейнеров) на равенство
- **BOOST_CHECK_THROW(инструкция, исключение)**
  - инструкция выбросит указанное исключение
- **BOOST_CHECK_NO_THROW(инструкция)**
  - наоборот, проверка, что исключений выброшено не будет

# Boost.Test

**фреймворк – пусть будет по тесту на каждый метод**

```
1.  #include "calculator.h"
2.  #include <boost/test/unit_test.hpp>
3.  #define BOOST_TEST_MODULE testCalculator

4.  BOOST_AUTO_TEST_CASE(testCalculator) {
5.      Calculator calculator(12);
6.      BOOST_CHECK_EQUAL(calculator.Result(), 12);
7.  }

8.  BOOST_AUTO_TEST_CASE(testCalculatorDivide) {
9.      Calculator calculator(12);
10.     calculator.Divide(3);
11.     BOOST_CHECK_EQUAL(calculator.Result(), 4);
12.     calculator.Divide(2);
13.     BOOST_CHECK_EQUAL(calculator.Result(), 2);
14. }

15. BOOST_AUTO_TEST_CASE(testCalculatorMultiply) {
16.     Calculator calculator(12);
17.     calculator.Multiply(2);
18.     BOOST_CHECK_EQUAL(calculator.Result(), 24);
19.     calculator.Multiply(3);
20.     BOOST_CHECK_EQUAL(calculator.Result(), 72);
21. }
```

# Boost.Test

**fixture – конструирование и настройка тестовых объектов**

```
1.  #include "calculator.h"
2.  #include <boost/test/unit_test.hpp>
3.  #define BOOST_TEST_MODULE testCalculator
4.  struct Fixture {
5.      Fixture() : calculator(12) { /* Здесь тестовый объект можно настроить */ }
6.      ~Fixture() { /* А здесь корректно завершить с ним работу */ }
7.      Calculator calculator; // тестовый объект
8.  };

9.  BOOST_FIXTURE_TEST_CASE(testCalculator, Fixture) {
10.     BOOST_CHECK_EQUAL(calculator.Result(), 12);
11. }
12. BOOST_FIXTURE_TEST_CASE(testCalculatorDivide, Fixture) {
13.     calculator.Divide(3);
14.     BOOST_CHECK_EQUAL(calculator.Result(), 4);
15.     calculator.Divide(2);
16.     BOOST_CHECK_EQUAL(calculator.Result(), 2);
17. }
18. BOOST_FIXTURE_TEST_CASE(testCalculatorMultiply, Fixture) {
19.     calculator.Multiply(2);
20.     BOOST_CHECK_EQUAL(calculator.Result(), 24);
21.     calculator.Multiply(3);
22.     BOOST_CHECK_EQUAL(calculator.Result(), 72);
23. }
```

# Boost.Test

**логическая организация**

```
1.  #include "calculator.h"
2.  #include <boost/test/unit_test.hpp>
3.  #define BOOST_TEST_MODULE testCalculator
4.  BOOST_AUTO_TEST_SUITE(testSuiteCalculator) // Начало набора тестов

5.  struct Fixture {
6.      // ...
7.  };

8.  BOOST_FIXTURE_TEST_CASE(testCalculator, Fixture) {
9.      // ...
10. }

11. BOOST_FIXTURE_TEST_CASE(testCalculatorDivide, Fixture) {
12.     // ...
13. }

14. BOOST_FIXTURE_TEST_CASE(testCalculatorMultiply, Fixture) {
15.     // ...
16. }

17. BOOST_AUTO_TEST_SUITE_END() // Конец набора тестов
```

# Boost.Test

**тестирование закрытых методов класса**

```
1.  namespace testSuiteCalculator { // имя набора тестов
2.  // А это имена конкретных тестов
3.  struct testCalculator;
4.  struct testCalculatorDivide;
5.  struct testCalculatorMultiply;
6.  }

7.  class Calculator {
8.      friend struct ::testSuiteCalculator::testCalculator;
9.      friend struct ::testSuiteCalculator::testCalculatorDivide;
10.     friend struct ::testSuiteCalculator::testCalculatorMultiply;
11. public:
12.     //...
13. };
```

# Boost.Test

## Интеграция сборки CMake & CTest

**CMakeLists.txt**

```
1. set (TESTS_SOURCES ../tests/файлы_с_тестами.cpp)
2. find_package (Boost COMPONENTS unit_test_framework REQUIRED)
3. include_directories(${Boost_INCLUDE_DIRS})
4. set (TEST test_${PROJECT})
5. add_executable (${TEST} ${TESTS_SOURCES})
6. target_link_libraries (${TEST} ${PROJECT} ${Boost_LIBRARIES})
7. enable_testing ()
8. add_test (${TEST} ${TEST})
```

**тесты можно запускать выполнив**

```
1. make test
```

# Boost.Test

Простая сборка и добавление тестов в проект

- Создается директория **test** в корне проекта
- В этой директории создаются cpp-файлы с тестами (не забываем, что один из них должен содержать определение **BOOST_TEST_MODULE**)

# Boost.Test

## Интеграция в сборку CDash

**Scan Build** *1 build* [view timeline]

| | | Update | Configure | | Build | | |
|---|---|---|---|---|---|---|---|
| Site | Build Name | Revision | Error | Warn | Error | Warn | Start Time ▼ |
| elysium-linux.kitware | Linux-clang-scanbuild | 0920c1 | 0 | 0 | 0 | 0 | 13 hours ago |

**Nightly Expected** *139 of 148 builds* [view timeline]

| | | Update | Configure | | Build | | Test | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Site | Build Name | Revision | Error | Warn | Error | Warn | Not Run | Fail ▼ | Pass | Start Time ▼ |
| dash3win7.kitware | vs14-64 | 0920c1 | 0 | 0 | 0 | 0 | 0 | $4^{+3}_{-4}$ | $507^{+4}_{-3}$ | 12 hours ago |
| vesper.kitware | vs12-64-ide-intl | 0920c1 | 0 | 0 | 0 | 0 | 0 | $3^{+3}_{-3}$ | $415^{+3}_{-3}$ | 11 hours ago |
| hythloth.kitware | Linux-bullseye-cov | 0920c1 | 0 | 0 | 0 | 0 | 0 | $2^{+2}_{-1}$ | $548^{+1}_{-2}$ | 14 hours ago |
| trinsic.kitware | vs14-64-ninja | 0920c1 | 0 | 0 | 0 | 0 | 0 | $2^{+2}$ | $535_{-2}$ | 14 hours ago |
| hythloth.kitware | Linux-gnu | 0920c1 | 0 | 0 | 0 | 0 | 0 | $1^{+1}$ | 660 | 10 hours ago |
| trinsic.kitware | icl-64-ninja | 0920c1 | 0 | 0 | 0 | 0 | 0 | 1 | 418 | 10 hours ago |
| unstable11s.opencsw.org | Solaris-11-sparc_SunStudio12 | 0920c1 | 0 | 0 | 0 | 0 | 0 | $1^{+1}$ | $425_{-1}$ | 14 hours ago |
| unstable11s.opencsw.org | Solaris-11-sparc_SolStudio12u2 | 0920c1 | 0 | 0 | 0 | 0 | 0 | $1^{+1}$ | $425_{-1}$ | 14 hours ago |
| unstable11s.opencsw.org | Solaris-11-sparc_SolStudio12u3 | 0920c1 | 0 | 0 | 0 | 0 | 0 | $1^{+1}$ | $425_{-1}$ | 14 hours ago |
| unstable11s.opencsw.org | Solaris-11-sparc_SolStudio12u4 | 0920c1 | 0 | 0 | 0 | 0 | 0 | $1^{+1}$ | $425_{-1}$ | 14 hours ago |

# Именование библиотек boost

**single letter is the tag**

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

5.  // -mt Threading tag: the library was built with multithreading support enabled
6.  // -d  ABI tag: the library's interoperability with other compiled code
7.  //     For each such feature, a single letter is added to the tag:
8.  //
9.  //  Key   Use this library when (Boost.Build option)
10. //
11. //  s     linking statically to the C++ standard library
12. //        and compiler runtime support libraries
13. //        (runtime-link=static)
14. //  g     using debug versions of the standard and runtime support libraries
15. //        (runtime-debugging=on)
16. //  y     using a special debug build of Python
17. //        (python-debugging=on)
18. //  d     building a debug version of your code
19. //        (variant=debug)
20. //  p     using the STLPort standard library rather than
21. //        the default one supplied with your compiler
22. //        (stdlib=stlport)
```

# #pragma comment

**фишка исключительно компилятора Microsoft**

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

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

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

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

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

# Boost.Log

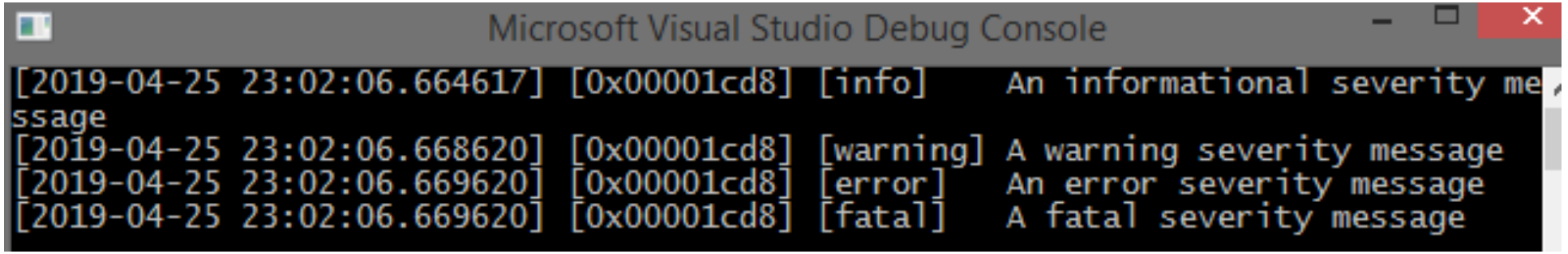
**supports numerous back-ends to log data in various formats**

```
1.  #include <boost/log/core.hpp>
2.  #include <boost/log/trivial.hpp>
3.  #include <boost/log/expressions.hpp>
4.  namespace logging = boost::log;

5.  void init() {
6.      logging::core::get()->set_filter(
7.          logging::trivial::severity >= logging::trivial::info);
8.  }
9.  int main() {
10.     init();

11.     BOOST_LOG_TRIVIAL(trace)   << "A trace severity message";
12.     BOOST_LOG_TRIVIAL(debug)   << "A debug severity message";
13.     BOOST_LOG_TRIVIAL(info)    << "An informational severity message";
14.     BOOST_LOG_TRIVIAL(warning) << "A warning severity message";
15.     BOOST_LOG_TRIVIAL(error)   << "An error severity message";
16.     BOOST_LOG_TRIVIAL(fatal)   << "A fatal severity message";
17.     return 0;
18. }
```

# Boost.Log


Microsoft Visual Studio Debug Console
[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

# Boost.Signals2

- Шаблон проектирования Наблюдатель (Observer)

- Поддерживает событийное программирование, где **std::function** может также быть использована для обработки событий
- 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 (**broadcast**)
  - Остальные компоненты могут выбрать к каким событиям из списка подключиться
- Build-in C#, Java

# Boost.Signals2

**сигнал может быть отправлен компонентам приёмникам**

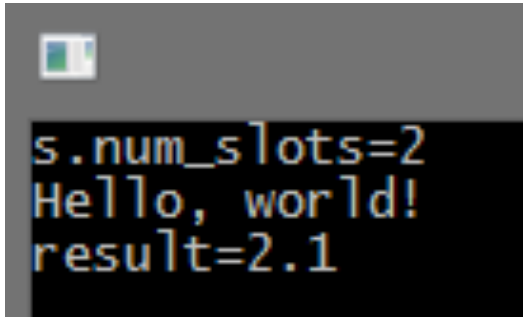
```
1.  #include <iostream>
2.  #include <boost/signals2.hpp>
3.  using namespace boost::signals2;
4.  using namespace std;

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

6.  int main() {
7.      signal<void()> s;
8.      s.connect(1, world);
9.      s.connect(0, [] { cout << "Hello"; });
10.     cout << "s.num_slots=" << s.num_slots() << endl;
11.     if (!s.empty()) { s(); } // notify!
12.     s.disconnect(world);
13.     s.disconnect_all_slots();

14.     signal<float(float, float)> sig;
15.     sig.connect(0, [] (float a, float b) { return a + b; });
16.     cout << "result=" << *sig(1.4f, 0.7f) << endl;
17.     sig.disconnect(0);
18.     return 0;
19. }
```

# Boost.Signals2



# boost::algorithm::string

**преобразование из строкового вида**

```
1.  #include <boost/algorithm/string.hpp>
2.  #include <boost/algorithm/string/trim_all.hpp>
3.  using namespace std;
4.  using namespace boost;
5.  using namespace boost::algorithm;

6.  void f() {
7.      std::string test = "hello  world\r\n";
8.      trim(test);
9.      trim_all(test);
10.     to_upper(test);
11. }
```

# boost::algorithm::string

**преобразование из строкового вида**

```
1.  #include <boost/algorithm/string.hpp>
2.  #include <boost/algorithm/string/trim_all.hpp>
3.  using namespace std;
4.  using namespace boost;
5.  using namespace boost::algorithm;

6.  void f() {
7.      std::string test = "hello  world\r\n";
8.      trim(test);      // <hello  world>
9.      trim_all(test);  // <hello world>
10.     to_upper(test);  // <HELLO WORLD>
11. }
```

# boost::algorithm::string

**преобразование из строкового вида**

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

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

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

7.    for (string part : t)
8.      cout << "<" << part << ">" << endl;
9.  }
```

**Output**

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

# boost::tokenizer

**custom characters separation**

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

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

8.    for (string part : t)
9.      cout << "<" << part << ">" << endl;
10. }
```

**Output**

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

# boost::lexical_cast

**преобразование из строкового вида полиморфным образом**

```
1.  #include <boost/lexical_cast.hpp>
2.  using namespace std;
3.  using namespace boost;

4.  void f() {
5.      std::string s = "2.1";
6.      // atoi? std::stolf?
7.      double d = lexical_cast<double>(s);

8.      try {
9.          lexical_cast<int>("abcde");
10.     } catch (const bad_lexical_cast& e) {
11.         cout << e.what() << endl;
12.     }
13. }
```

# boost::type_traits → std::type_traits

**разнообразные свойства типов**

1. is_const
2. is_standard_layout
3. is_pod
4. is_literal_type
5. is_polymorphic
6. is_abstract
7. is_constructible
8. is_trivially_constructible
9. is_nothrow_constructible
10. is_same
11. is_base_of
12. result_of
13. is_integral
14. is_floating_point
15. is_class
16. is_function
17. is_member_function_pointer
18. is_arithmetic
19. is_reference

# std::type_traits

**как получить специфичные для типа значения?**

```
1.  template <typename C>
2.  auto findMax(const C &container) -> typename C::value_type {
3.      auto _max = std::numeric_limits<C::value_type>::min();
4.      for (const auto &item : container)
5.          if (_max < item)
6.              _max = item;
7.      return _max;
8.  }
```

# std::type_traits

**хотим реализовать изменение порядка битов в памяти для POD типов**

```
1.  template <typename T>
2.  T byte_inplace_swap(T value) {
3.      std::byte* bytes = reinterpret_cast<std::byte*>(&value);
4.      constexpr size_t n = sizeof(T);
5.      for (size_t i = 0; i < (n / 2); ++i)
6.          std::swap(bytes[i], bytes[n - 1 - i]);
7.      return value;
8.  }
```

# std::type_traits

**допустим, что эта операция допустима не для любого POD типа**

```
1.  template <typename T>
2.  T byte_inplace_swap(T value) {
3.      std::byte* bytes = reinterpret_cast<std::byte*>(&value);
4.      constexpr size_t n = sizeof(T);
5.      for (size_t i = 0; i < (n / 2); ++i)
6.          std::swap(bytes[i], bytes[n - 1 - i]);
7.      return value;
8.  }
9.  template <>
10. double byte_inplace_swap(double value) {
11.   assert(false && "Illegal to swap doubles");
12.   return value;
13. }
14. template <>
15. char byte_inplace_swap(char value) {
16.   assert(false && "Illegal to swap chars");
17.   return value;
18. }
```

# std::type_traits

**напишем специализированные флаги, чтобы определить допустимые типы**

```
1.  template <typename T>
2.  struct is_swapable { static const bool value = false; };

3.  template <>
4.  struct is_swapable<unsigned short> { static const bool value = true; };

5.  template <>
6.  struct is_swapable<short> { static const bool value = true; };
7.
8.  template <>
9.  struct is_swapable<unsigned long> { static const bool value = true; };
10.
11. template <>
12. struct is_swapable<long> { static const bool value = true; };
13.
14. template <>
15. struct is_swapable<unsigned long long> { static const bool value = true; };
16.
17. template <>
18. struct is_swapable<long long> { static const bool value = true; };
```

# std::type_traits

**теперь хорошо?**

```
1.  #include <type_traits>
```

```
2.  template <typename T>
3.  T byte_inplace_swap(T value) {
4.      assert(is_swapable<T>::value && "Cannot swap this type");
5.      std::byte *bytes = reinterpret_cast<std::byte*>(&value);
6.      constexpr size_t n = sizeof(T);
7.      for (size_t i = 0; i < (n / 2); ++i)
8.          std::swap(bytes[i], bytes[n - 1 - i]);
9.      return value;
10. }
```

# std::type_traits

**лучше**

```
1.  #include <type_traits>
```

```
2.  template <typename T, typename = std::enable_if<is_swapable<T>::value, T>::type>
3.  T byte_inplace_swap(T value) {
4.
5.      std::byte *bytes = reinterpret_cast<std::byte*>(&value);
6.      constexpr size_t n = sizeof(T);
7.      for (size_t i = 0; i < (n / 2); ++i)
8.          std::swap(bytes[i], bytes[n - 1 - i]);
9.      return value;
10. }
```

# std::type_traits

**идеально**

```
1.  #include <type_traits>

2.  template <typename T>
3.  constexpr bool is_swapable() {
4.      return (std::is_integral<T>::value && (sizeof(T) > 1));
5.  }

6.  template <typename T, typename = std::enable_if<is_swapable<T>(), T>::type>
7.  T byte_inplace_swap(T value) {
8.
9.      std::byte *bytes = reinterpret_cast<std::byte*>(&value);
10.     constexpr size_t n = sizeof(T);
11.     for (size_t i = 0; i < (n / 2); ++i)
12.         std::swap(bytes[i], bytes[n - 1 - i]);
13.     return value;
14. }
```

# boost::type_traits

**есть некоторые конструкционные блоки, из которых можно создавать флаги**

```
1.  template <typename T>
2.  struct is_void : public false_type {};

3.  template <>
4.  struct is_void<void> : public true_type {};


5.  template <typename T>
6.  struct is_pointer : public false_type {};

7.  template <typename T>
8.  struct is_pointer<T*> : public true_type {};
```

# Boost.Filesystem

- Делает возможной лёгкую работу с файлами и директориями.
- Предоставляет класс пути **boost::filesystem::path** в файловой системе.
- Пересматривалась несколько раз, текущая версия есть **Boost.Filesystem 3**, начиная с **Boost C++ Libraries 1.46.0**

- Ошибки имеют хорошо читаемые описания:

terminate called after throwing an instance of 'boost::filesystem::filesystem_error'

- what(): **boost::filesystem::file_size: Operation not permitted: "."**
- what(): **boost::filesystem::file_size: No such file or directory: "foo"**
- what(): **boost::filesystem::status: Permission denied: "/home/jane/foo"** //disk was not ready

# boost::filesystem → std::filesystem

**path**

```
1.  #include <clocale>
2.  #include <iostream>
3.  #include <cstdlib>
4.  #include <boost/locale.hpp>
5.  #include <boost/filesystem.hpp>
6.  #include <boost/filesystem/path.hpp>
7.  #include <boost/filesystem/fstream.hpp>

8.  // path: "D:\repos\test\x64\Release\test.exe"
9.  // file: "test.exe" size is 99328
10. int main(int argc, char **argv) {
11.     namespace fs = boost::filesystem;
12.     std::setlocale(LC_ALL, "English_USA.1251");
13.
14.     fs::path exec(argv[0]);
15.     std::cout << "path: " << exec.make_preferred()
16.               << " file: " << exec.filename()
17.               << " size is " << fs::file_size(exec) << '\n';

18.     fs::path test_path{ "./test/" };
19.     fs::path note_path{ "C:\\Windows\note.txt" };
20.     fs::path unicode_path{ L"C:\\Boost C++ \u5E93" };
21.     return 0;
22. }
```

# std::filesystem

**is_regular_file exists**

```
1.  // ".." = "C:\Users\favorart\source\repos\test\test\.."
2.  // is a directory false
3.  //     "..\.vs"
4.  //     "..\Debug"
5.  //     "..\test"
6.  //     "..\test.sln"
7.  //     "..\x64"
8.  int main(int argc, char **argv) {
9.     fs::path path(argv[1]);

10.    try {
11.       if (!fs::exists(path)) { std::cout << path << " does not exist\n"; return 1; }

12.       if (fs::is_regular_file(path))
13.          std::cout << path << " size is " << fs::file_size(path) << '\n';
14.       else if (fs::is_directory(path)) {
15.          std::cout << path << " = " << fs::absolute(path) << "\nis a directory "
16.                    << std::boolalpha << path.is_absolute() << "\n";

17.          for (const fs::directory_entry &x : fs::directory_iterator(path))
18.              std::cout << "    " << x.path() << '\n';
19.       }
20.       else std::cout << path << " exists, but what is it?\n";
21.    } catch (const fs::filesystem_error& ex) { std::cout << ex.what() << '\n'; }
22.    return 0;
23. }
```

# std::filesystem

**operator/ to concatenate paths**

```
1.  int main(int argc, char **argv) {
2.      try {
3.          fs::path complex{ "." };
4.          complex = complex / "complex" / "path" / "to" / "test.txt";
5.          std::cout << complex << " = " << fs::absolute(complex) << '\n';
6.          // ".\complex\path\to\test.txt" = "D:\repos\test\.\complex\path\to\test.txt"

7.          fs::create_directories(complex.parent_path());
8.          //fs::permissions(complex, fs::perms::remove_perms | fs::perms::others_all);

9.          fs::ofstream myofs{ complex };
10.         myofs << "Hello, world!\n";
11.         myofs.close();

12.         fs::path onemoredir = complex / ".." / ".." / "from";
13.         fs::create_directory(onemoredir);
14.         std::system(("dir " + onemoredir.string()).c_str());

15.         fs::remove_all("complex");
16.     }
17.     catch (const fs::filesystem_error& ex) { std::cout << ex.what() << '\n'; }
18.     return 0;
19. }
```

# Boost.Filesystem

**directory_iterator**

```
1.  //  Том в устройстве D имеет метку DATA
2.  //  Серийный номер тома: 6A2E-F91F
3.  //
4.  //  Содержимое папки D:\repos\test\complex\path\to
5.  //
6.  // 19-Apr-19  14:43    <DIR>          .
7.  // 19-Apr-19  14:43    <DIR>          ..
8.  //                0 файлов              0 байт
9.  //                2 папок  51 899 994 112 байт свободно

10. // boost::filesystem::space().capacity
11. // boost::filesystem::space().available

12. // boost::filesystem::current_path()
13. // boost::filesystem::current_path(new)

14. // boost::filesystem::create_symlink() or copy_file()
15. // boost::filesystem::last_write_time()

16. // boost::filesystem::directory_iterator
17. // boost::filesystem::recursive_directory_iterator
```

# Boost.Serialization

**save-load**

```
1.  #include <boost/archive/binary_oarchive.hpp>
2.  #include <boost/archive/text_oarchive.hpp>

3.  class Data {
4.      std::shared_ptr<std::vector<double>> pv{};
5.      unsigned long num{}, seed{};
6.      std::stack<double> mean{};
7.      friend class boost::serialization::access;
8.      template <class Archive> void serialize(Archive & ar, const unsigned int version) {
9.          ar & pv & num & seed & mean;
10.     }
11. public:
12.     Data() = default;
13.     void save_text(const char *filename) const {
14.         boost::archive::text_oarchive toa(std::ofstream(filename));
15.         toa & *this;
16.     }
17.     void save_binary(const char *filename) const {
18.         boost::archive::binary_oarchive boa(std::ofstream(filename, std::ios::binary));
19.         boa & *this;
20.     }
21. }; // end class Data
22. BOOST_CLASS_VERSION(Data, 2 /*version*/)
```

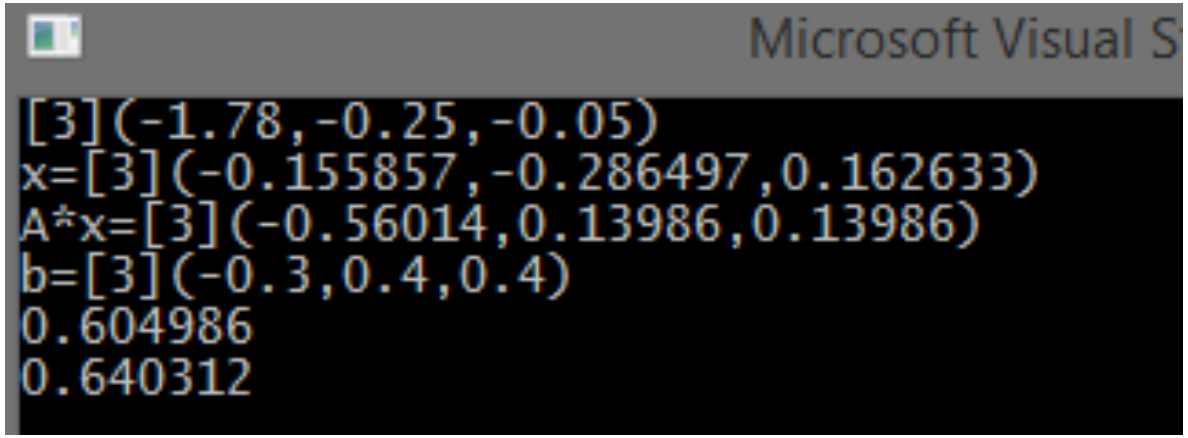
# Boost.Numeric

**линейная алгебра**

```
1.  #include <boost/numeric/ublas/io.hpp>
2.  #include <boost/numeric/ublas/matrix.hpp>
3.  #include <boost/numeric/ublas/lu.hpp>
4.  using namespace boost::numeric::ublas;
5.  void main() {
6.      matrix<double> A(3, 3, -0.5);
7.      A(0, 0) = A(2, 2) = 1.8;
8.      A(0, 2) = -2.6; A(2, 0) = 1.9;
9.      matrix<double> A1 = A + matrix<double>(3, 3, -0.93);
10.     vector<double> b(3, 0.4); b(0) = -0.3;
11.     vector<double> x = b;
12.     matrix_row<matrix<double>> mr(A, 2);
13.     matrix_column<matrix<double>> mc(A, 2);
14.     std::cout << prod(A, b) << std::endl;
15.     permutation_matrix<double> P1(A1.size1());
16.     lu_factorize(A1, P1);
17.     lu_substitute(A1, P1, x);
18.     std::cout << "x=" << x << std::endl;
19.     std::cout << "A*x=" << prod(A, x) << std::endl;
20.     std::cout << "b=" << b << std::endl;
21.     std::cout << norm_1(x) << std::endl;
22.     std::cout << norm_2(b) << std::endl;
23. }
```

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

# Boost.Numeric

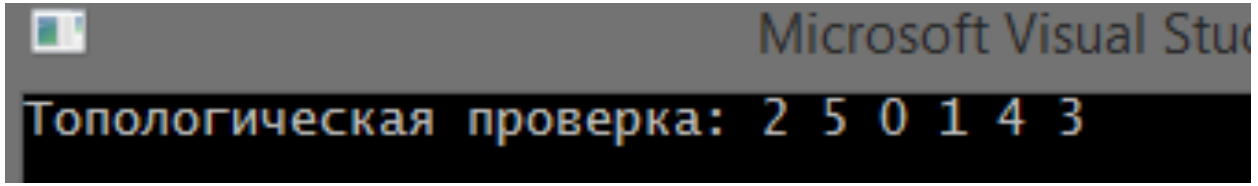


# Boost.Graph (BGL)

**предоставляет гибкую и эффективную реализацию концепции графов**

```
1.  #include <boost/graph/adjacency_list.hpp>
2.  #include <boost/graph/topological_sort.hpp>
3.  int main() {
4.      using namespace boost;
5.      std::setlocale(LC_ALL, "English_USA.1251");
6.      typedef adjacency_list<vecS, vecS, directedS,
7.          property<vertex_color_t, default_color_type> > Graph; // тип графа
8.      typedef boost::graph_traits<Graph>::vertex_descriptor Vertex;// дескриптор вершин
9.      typedef std::vector<Vertex> container; // контейнер для цепочки вершин
10.     typedef std::pair<std::size_t, std::size_t> Pair; // тип представления дуг графа
11.     Pair edges[6] = { Pair(0,1), Pair(2,4), Pair(2,5),
12.                       Pair(0,3), Pair(1,4), Pair(4,3) }; // Дуги графа
13.     Graph G(edges, edges + 6, 6); // Граф
14.     // словарь для получения номеров вершин по дескриптору вершин
15.     boost::property_map<Graph, vertex_index_t>::type id = get(vertex_index, G);
16.     container c; // контейнер для хранения отсортированных вершин
17.     topological_sort(G, std::back_inserter(c)); // выполнение алгоритма
18.     // Вывод результата: перебор дескрипторов графа в контейнере, получение порядковых номеров вершин
19.     std::cout << "Топологическая проверка: ";
20.     for (container::reverse_iterator ii = c.rbegin(); ii != c.rend(); ++ii)
21.         std::cout << id[*ii] << " ";
22.     std::cout << std::endl;
23. }
```

# Boost.Graph (BGL)



# boost::regexp → std::regexp

- Экземпляры класса **std::regex** определяют *регулярные выражения*

- **std::regex_search** используется для поиска
- **std::regex_replace** используется для операции «*найти и заменить*», вернёт строку после выполнения замены

- Получают на вход регулярное выражение и строку, возвращают найденные результаты в виде экземпляров шаблона **std::match_results**

# std::regexp

**удобно использовать с «сырыми» строками (можно с обычными)**

```
1.  #include <regex>
2.  #include <string>
3.  #include <iostream>

4.  int main() {
5.      const char *reg_exp = R"([ ,.\t\n;:])";  // символы-разделители

6.      std::regex rgx(reg_exp);
7.      std::smatch match;

8.      std::string target{ "\tMoscow State University - in-Tashkent;;" };

9.      while (std::regex_search(target, match, rgx))
10.     {
11.         std::cout << '"' << match.str() << '"' << std::endl;
12.         target = match.suffix();
13.     }
14.     return 0;
15. }
```

# RegExp

| Якоря | |
|---|---|
| ^ | Начало строки + |
| \A | Начало текста + |
| $ | Конец строки + |
| \Z | Конец текста + |
| \b | Граница слова + |
| \B | Не граница слова + |
| \< | Начало слова |
| \> | Конец слова |

| Кванторы | |
|---|---|
| * | 0 или больше + |
| *? | 0 или больше, нежадный + |
| + | 1 или больше + |
| +? | 1 или больше, нежадный + |
| ? | 0 или 1 + |
| ?? | 0 или 1, нежадный + |
| {3} | Ровно 3 + |
| {3,} | 3 или больше + |
| {3,5} | 3, 4 или 5 + |
| {3,5}? | 3, 4 или 5, нежадный + |

| Символьные классы | |
|---|---|
| \c | Управляющий символ |
| \s | Пробел |
| \S | Не пробел |
| \d | Цифра |
| \D | Не цифра |
| \w | Слово |
| \W | Не слово |
| \xhh | Шестнадцатиричный символ hh |
| \Oxxx | Восьмиричный символ xxx |

| Специальные символы | |
|---|---|
| \ | Экранирующий символ + |
| \n | Новая строка + |
| \r | Возврат каретки + |
| \t | Табуляция + |
| \v | Вертикальная табуляция + |
| \f | Новая страница + |
| \a | Звуковой сигнал |
| [\b] | Возврат на один символ |
| \e | Escape-символ |
| \N{name} | Именованый символ |

| Образцы шаблонов | |
|---|---|
| ([A-Za-z0-9-]+) | Буквы, числа и знаки переноса |
| (\d{1,2}\/\d{1,2}\/\d{4}) | Дата (напр., 21/3/2006) |
| ([^\s]+(?=\.(jpg\|gif\|png))\.\2) | Имя файла jpg, gif или png |
| (^[1-9]{1}$\|^[1-4]{1}[0-9]{1}$\|^50$) | Любое число от 1 до 50 включительно |
| (#?([A-Fa-f0-9]){3}(([A-Fa-f0-9]){3})?) | Шестнадцатиричный код цвета |
| ((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,15}) | От 8 до 15 символов с минимум одной цифрой, одной заглавной и одной строчной буквой (полезно для паролей). |
| (\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6}) | Адрес email |
| (\<(/?[^\>]+)\>) | HTML теги |

# Boost.MetaStateMachine

- **Конечные автоматы** описывают объекты через их **состояния**. Смена состояний описывается функцией перехода.
- Библиотека предоставляет три способа определения конечных автоматов.
- **Basic front-end** (через указатели на фукнции) или **Function front-end** (через функторы) позволяют определять конечные автоматы традиционным образом:
  - создавая классы состояний
  - наследуясь от базовых классов в библиотеке
  - определяя все необходимые поля и методы
  - описывая всю логику самостоятельно
- **eUML front-end** (через макросы)
  - основан на DSL – domain-specific language
  - можно переиспользовать формальные описания из схемы поведения в UML

# boost::msm

**the simplest state machine possible**

```
1.  #include <boost/msm/front/euml/euml.hpp>
2.  #include <boost/msm/front/euml/state_grammar.hpp>
3.  #include <boost/msm/back/state_machine.hpp>
4.  #include <iostream>

5.  namespace msm = boost::msm;
6.  using namespace boost::msm::front::euml;

7.  BOOST_MSM_EUML_STATE((), Off);
8.  BOOST_MSM_EUML_STATE((), On);
9.  BOOST_MSM_EUML_EVENT(press);
10. BOOST_MSM_EUML_TRANSITION_TABLE((((Off + press) == On),((On + press) == Off)),
11.                                 light_transition_table);
12. BOOST_MSM_EUML_DECLARE_STATE_MACHINE((light_transition_table, init_ << Off),
13.                                      light_state_machine);

14. int main() {
15.     msm::back::state_machine<light_state_machine> light;
16.     std::cout << *light.current_state() << '\n';
17.     light.process_event(press);
18.     std::cout << *light.current_state() << '\n';
19.     light.process_event(press);
20.     std::cout << *light.current_state() << '\n';
21.     return 0;
22. }
```

# Boost.Spirit

- Инструмент для парсинга произвольного текста, существенно снижает время разработки
- Можно разрабатывать парсеры под специфический формат текста
- Формат описывается правилами
- **Parsing Expression Grammar** (**PEG**), **Extended Backus-Naur-Form** (**EBNF**)
- Скрывает множество деталей реализации, лучше масштабируется чем
  - парсинг вручную (scanf)
  - поиск паттернов (regexp)
  - сканнеры (tokenizers)
- Компоненты библиотеки
  - **boost::spirit::qi -** для разработки **парсеров**
  - **boost::spirit::karma -** для разработки **генераторов**
  - **boost::spirit::lex** - для разработки **лексеров**

# Boost.Spirit

**includes and aliases**

```
1.  #include <boost/config/warning_disable.hpp>
2.  #include <boost/spirit/include/qi.hpp>
3.  #include <boost/spirit/include/phoenix_core.hpp>
4.  #include <boost/spirit/include/phoenix_operator.hpp>
5.  #include <boost/spirit/include/phoenix_fusion.hpp>
6.  #include <boost/spirit/include/phoenix_stl.hpp>
7.  #include <boost/fusion/include/adapt_struct.hpp>
8.  #include <boost/variant/recursive_variant.hpp>
9.  #include <boost/foreach.hpp>

10. #include <iostream>
11. #include <fstream>
12. #include <string>
13. #include <vector>

14. namespace client {
15. namespace fusion = boost::fusion;
16. namespace phoenix = boost::phoenix;
17. namespace qi = boost::spirit::qi;
18. namespace ascii = boost::spirit::ascii;
19. }
```

# Boost.Spirit

**mini_xml declaration**

```
20. namespace client {
21. struct mini_xml;

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

24. struct mini_xml {
25.     std::string name;                           // tag name
26.     std::vector<mini_xml_node> children;        // children
27. };
28. }
29. // tell fusion about our mini_xml struct to make it a first-class fusion citizen
30. BOOST_FUSION_ADAPT_STRUCT(client::mini_xml, (std::string, name)
31.                           (std::vector<client::mini_xml_node>, children))
32. namespace client {
33. constexpr int tabsize = 4;
34. void tab(int indent) { for (int i = 0; i < indent; ++i) std::cout << ' '; }

35. struct mini_xml_printer {
36.     int indent;
37.     mini_xml_printer(int indent = 0) : indent(indent) {}
38.     void operator()(mini_xml const& xml) const;
39. };
40. }
```

# Boost.Spirit

**Printer xml nodes and all file**

```
41. namespace client {
42. struct mini_xml_node_printer : boost::static_visitor<> {
43.     int indent;
44.     mini_xml_node_printer(int indent = 0) : indent(indent) {}

45.     void operator()(mini_xml const& xml) const {
46.         mini_xml_printer(indent + tabsize)(xml);
47.     }
48.     void operator()(std::string const& text) const {
49.         tab(indent + tabsize);
50.         std::cout << "text: \"" << text << '"' << std::endl;
51.     }
52. };
53. void mini_xml_printer::operator()(mini_xml const& xml) const {
54.     tab(indent);
55.     std::cout << "tag: " << xml.name << std::endl;
56.     tab(indent);
57.     std::cout << '{' << std::endl;
58.     for (mini_xml_node const& node : xml.children)
59.         boost::apply_visitor(mini_xml_node_printer(indent), node);
60.     tab(indent);
61.     std::cout << '}' << std::endl;
62. }
63. }
```

# Boost.Spirit

**parser instance**

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

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

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

74.         xml = start_tag[phoenix::at_c<0>(_val) = _1]
75.             >> *node[phoenix::push_back(phoenix::at_c<1>(_val), _1)]
76.             >> end_tag(phoenix::at_c<0>(_val));
77.     }
78.     qi::rule<Iterator, mini_xml(), ascii::space_type> xml;
79.     qi::rule<Iterator, mini_xml_node(), ascii::space_type> node;
80.     qi::rule<Iterator, std::string(), ascii::space_type> text;
81.     qi::rule<Iterator, std::string(), ascii::space_type> start_tag;
82.     qi::rule<Iterator, void(std::string), ascii::space_type> end_tag;};
83. }
84. }
```

# Boost.Spirit

**how it works**

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

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

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

96.      if (r && iter == end) {
97.          std::cout << "Parsing succeeded\n";
98.          client::mini_xml_printer{}(ast);
99.      } else {
100.         std::string::const_iterator some = iter + std::min(30, int(end - iter));
101.         std::string context(iter, (some > end) ? end : some);
102.         std::cout << "Parsing failed\n" << "stopped at: \"" << context << "...\"\n";
103.     }
104.     return 0;
105. }
```

data.xml | spirit.cpp | main.cpp | ma

```xml
<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>
```

Microsoft Visual St

```
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.PropertyTree

Предоставляет возможности работы с древовидной структурой пар ключ-значение. Обрабатывает подразделы, позволяет множественное ветвление

Активно используется для конфигурационных файлов

- **json_parser**
- **ini_parser**

Для отделения подразделов использует символ «точка»

- **get_child**(key) – доступ к подразделу
  - возвращает объект типа, как у родителя
  - т.к. каждый подраздел имеет право иметь свои подразделы, разницы в их типе нет
- **get_value**<type>(key) – получения листового значения нужного типа
- **get_value_or**<type>(key, val) – вернуть значение нужного типа, если в конфиг.файле ключ представлен, иначе вернуть переданное val
- **put**(key, val) – положить значение по ключу
- **put_value**<type>(key, val) – положить подраздел целиком по ключу
- **add_child**(key, child) – добавить подраздел по ключу (даже копией)

# boost::property_tree

**file system is a good example of a tree structure**

```
1.  #include <iostream>
2.  #include <boost/property_tree/ptree.hpp>
3.  #include <boost/property_tree/json_parser.hpp>
4.  //#include <boost/property_tree/ini_parser.hpp>
5.  using namespace boost::property_tree;
6.  int main() {
7.      ptree root;
8.      root.put("C:.Windows.System", "20 files");
9.      root.put("C:.Windows.Cursors", "50 files");
10.     boost::optional<std::string> opt = root.get_optional<std::string>("C:");
11.     std::cout << std::boolalpha << opt.get() << '\n';
12.     root.put_child("D:.Program Files", ptree{ "40 files" });
13.     root.put_child("D:.Program Files", ptree{ "50 files" });
14.     root.add_child("D:.Program Files", ptree{ "60 files" });
15.     for (const std::pair<std::string, ptree> &p : root.get_child("D:"))
16.         std::cout << p.second.get_value<std::string>() << '\n';
17.     json_parser::write_json("file.json", root);
18.     ptree copy;
19.     json_parser::read_json("file.json", copy);
20.     std::cout << std::boolalpha << (root == copy) << '\n';
21.     return 0;
22. }
```

# boost::property_tree

**file system is a good example of a tree structure**

```
1.  template <typename  T> struct Point {
2.     using value_type = T;
3.     void save(ptree &node) const {
4.        ptree xelem, yelem;
5.        xelem.put_value<Point::value_type>(x); node.push_back(std::make_pair("", xelem));
6.        yelem.put_value<Point::value_type>(y); node.push_back(std::make_pair("", yelem));
7.     }
8.     value_type x, y;
9.  };

10. void write_joints() {
11.    ptree root, joints;
12.    for (const auto &i : { 1, 2, 3, 4 }) {
13.       ptree node;
14.       node.put<unsigned>("joint", 19 + i);
15.       node.put<bool>("show", i % 2);
16.       node.put<float>("frames", 0.4 * i);
17.       Point<double> pt{ i,i }; ptree p; pt.save(p); node.add_child("base", p);
18.       joints.push_back(std::make_pair("", node));
19.    }
20.    root.put<std::string>("pt", "name");
21.    root.add_child("joints", joints);
22.    json_parser::write_json("joints.json", root);
23. }
```

# RAII and Memory Management

- **Boost.SmartPointers**
- **Boost.PointerContainer**
  - контейнер из указателей на объекты
  - при удалении контейнера удаляются и объекты
- **Boost.ScopeExit**
  - RAII повсюду
  - не обязательно память, сеть или мьютексы
- **Boost.Pool**
  - не RAII
  - память выделяется быстрее
  - с объектами классов или чистой памятью
  - настраиваемое поведение при ошибках
  - Ordered vs. Unordered

# Boost.Pool

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

```
1.  #include <boost/pool/pool.hpp>
2.  #include <boost/pool/object_pool.hpp>
3.  #include <boost/pool/singleton_pool.hpp>
4.  #include <vector>
5.  class my { int i; };
6.  struct my_sole_pool_tag {};
7.  using sole_pool = boost::singleton_pool<my_sole_pool_tag, sizeof(int)>;

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

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

# Boost.Pool

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

```
1.  template <typename UserAllocator = default_user_allocator_new_delete>
2.  class pool {
3.      pool(const pool &) = delete;      void operator=(const pool &) = delete;
4.      pool(pool&&) = default;           void operator=(pool&&) = default;
5.  public:
6.      typedef typename UserAllocator                    user_allocator;
7.      typedef typename UserAllocator::size_type         size_type;
8.      typedef typename UserAllocator::difference_type   difference_type;

9.      explicit pool(size_type requested_size);
10.     ~pool();
11.     bool release_memory();
12.     bool purge_memory();
13.     bool is_from(void* chunk) const;
14.     size_type get_requested_size() const;

15.     void* malloc();
16.     void* ordered_malloc();
17.     void* ordered_malloc(size_type n);
18.     void  free(void* chunk);
19.     void  ordered_free(void * chunk);
20.     void  free(void* chunks, size_type n);
21.     void  ordered_free(void * chunks, size_type n);
22. };
```

# Boost.Pool

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

```
1.  struct default_user_allocator_new_delete {
2.      typedef std::size_t     size_type;
3.      typedef std::ptrdiff_t  difference_type;

4.      static char* malloc(const size_type bytes) {
5.          return new (std::nothrow) char[bytes];
6.      }
7.      static void free(char * const block) { delete[] block; }
8.  };

9.  struct default_user_allocator_malloc_free {
10.     typedef std::size_t     size_type;
11.     typedef std::ptrdiff_t  difference_type;

12.     static char* malloc(const size_type bytes) {
13.         return static_cast<char*>(std::malloc(bytes));
14.     }
15.     static void free(char * const block) { std::free(block); }
16. };
```

# Boost.PointerContainer

**specialized to manage dynamically allocated objects**

```
1.  #include <boost/ptr_container/indirect_fun.hpp>
2.  #include <boost/ptr_container/ptr_inserter.hpp>
3.  #include <boost/ptr_container/ptr_vector.hpp>
4.  #include <boost/ptr_container/ptr_set.hpp>
5.  int main() {
6.      std::array<int, 3> arr{ 0, 1, 2 };
7.      boost::ptr_vector<int> vec; // works like std::vector<std::unique_ptr<int>>
8.      std::copy(arr.begin(), arr.end(), boost::ptr_container::ptr_back_inserter(vec));
9.      // vec expects addresses of dynamically allocated int objects,
10.     // inserter creates copies on the heap and adds the addresses to the container
11.     vec.push_back(new int{ 3 });
12.     std::cout << vec.size() << ' ' << vec.back() << '\n';
13.
14.     boost::ptr_set<int> s;
15.     s.insert(new int{ 2 }), s.insert(new int{ 1 });
16.     std::cout << *s.begin() << '\n';

17.     std::set<std::unique_ptr<int>, //  together with resource manager
18.             boost::indirect_fun<std::less<int>>// must be told how to compare elements
19.     > v; // non-specialized container
20.     v.insert(std::unique_ptr<int>(new int{ 2 }));
21.     v.insert(std::unique_ptr<int>(new int{ 1 }));
22.     std::cout << **v.begin() << '\n';
23.     return 0;
24. }
```

# Boost.ScopeExit

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

```
1.  #include <iostream>
2.  #include <utility>

3.  template <typename T>
4.  struct scope_exit {
5.      T t;
6.      scope_exit(T &&t) : t{ std::move(t) } {}
7.      ~scope_exit() { ~t(); }
8.  };
9.  template <typename T>
10. scope_exit<T> make_scope_exit(T &&t) {
11.     return scope_exit<T>{ std::move(t) };
12. }

13. int* foo() {
14.     int *i = new int{ 10 };
15.     auto cleanup = make_scope_exit([&i]() mutable { delete i; i = 0; });
16.     std::cout << *i << '\n';
17.     return i;
18. }
19. int main() {
20.     int *j = foo();
21.     std::cout << j << '\n';
22. }
```