C++11

Аверчук Г.Ю.

1, 15 марта 2012

C++11: Дождались!

- Что было раньше
- Что появилось нового
- Как этим пользоваться
- Подводные камни
- Что нас ждёт впереди

Немного истории

http://img-fotki.yandex.ru/get/3504/alenacpp.0/0_2a6c6_9364879b_orig

Наше время...

http://softwaremaniacs.org/Images/alenacpp/cppmap-2012.png

Как компилировать?

- **GCC** версия от 4.6

  опция компиляции: **`-std=c++0x`**

  Eclipse: *Project -> Properties -> C/C++ Build -> Settings -> GCC C++ Compiler -> Miscellaneous -> Other flags*

  Qt Creator: `project_name`**`.pro`** ->

  QMAKE_CXXFLAGS += -std=c++0x

- **MSVS** 2010 и выше

Улучшения языка

- Улучшения в практическом использовании
- Улучшения в ядре языка
- Повышение производительности за счёт ядра языка
- Изменение стандартной библиотеки
- Ускорение компиляции

Улучшения в практическом использовании языка

Списки инициализации

```
// includes...

int main() {
    int c_arr[] = { 1, 2, 3, 4, 5 };
    std::deque<int> queue = { 1, 2, 3, 4, 5 };
    std::list<int> list = { 1, 2, 3, 4, 5 };
    std::set<int> set = { 1, 2, 3, 4, 5 };
    std::vector<int> vector = { 1, 2, 3, 4, 5 };
    std::map<int, int> map = { {1, 5}, {2, 4}, {3, 3}, {4, 2}, {5, 1} };

    for (int i = 0; i < 5; ++i) {
        std::cout << c_arr[i] << " = " << queue[i] << " = "
            << *(list.begin()) << " = " << *(set.begin()) << " = "
            << vector[i] << " | "
            << map.begin()->first << ": " << map.begin()->second
            << std::endl;

        list.pop_front();
        set.erase(set.begin());
        map.erase(map.begin());
    }

    return 0;
}
```

```
#include <iostream>
#include <deque>
using namespace std;

class Complex {
public:
    Complex(initializer_list<float> i_list) {
        initializer_list<float>::const_iterator p = i_list.begin();
        _rm = *(p++);
        _im = *p;
    }

    void print() {
        cout << _rm;
        if (_im > 0) cout << '+';
        cout << _im << 'i' << endl;
    }

private:
    float _rm, _im;
};

int main() {
    Complex c = { .5f, -2.f };
    c.print();

    return 0;
}
```

Универсальная инициализация

```
#include <iostream>
#include <string>

struct MyStruct {
    int _a;
    double _b;
    std::string _str;

    void print() const {
        std::cout << "a = " << _a << ", b = " << _b
            << ", str = \"" << _str << "\"" << std::endl;
    }
};

MyStruct get_struct_obj() {
    return {5, 0, "Structure Object"};
}

int main() {
    MyStruct mst{1, 2.71, "Hello, World!"};
    mst.print();
    get_struct_obj().print();

    return 0;
}
```

Вывод типов

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    auto &output = cout;

    auto two_pi = 2 * 3.14;
    output << two_pi << endl;

    decltype(output) console = output;

    vector<int> vec = { 1, 3, 2012 };
    for (auto p = vec.begin(); p != vec.end(); ++p) {
        console << ' ' << *p;
    }
    console << endl;

    return 0;
}
```

For-цикл по коллекции

```
#include <iostream>
#include <vector>
using namespace std;

class Int {
public:
    Int(int value) : _value(value) {}
    int value() const { return _value; }

private:
    int _value;
};

int main() {
    int c_arr[5] = { 1, 2, 3, 4, 5 };
    for (int x : c_arr) cout << ' ' << x;
    cout << endl;

    vector<Int> vec = { 1, 1, 2, 3, 5 };
    for (auto &i : vec) cout << ' ' << i.value();
    cout << endl;

    return 0;
}
```

Лямбда-функции и выражения

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main() {
    auto add = [](int a, int b) { return a + b; };
    cout << add(2, 5) << endl;

    auto pow = [](int a, int b) -> int {
        int result = 1;
        for (int i = 0; i < b; ++i) result *= a;
        return result;
    };
    cout << pow(2, 16) << endl;

    vector<int> vec = { 5, 2, 21, 13, 1, 8, 3, 1 };
    int max = vec[0];
    for_each(vec.begin() + 1, vec.end(), [&max](int x) {
        if (x > max) max = x;
    });
    cout << max << endl;

    return 0;
}
```

```
#include <iostream>
#include <vector>
using namespace std;

template <typename LambdaType>
void print_elements(LambdaType lambda_print, int n) {
    for (int i = 0; i < n; ++i) lambda_print();
}

int main() {
    vector<int> arr1 = { 5, 4, 3, 2, 1 };
    vector<int> arr2 = { 0, 9, 8, 7, 6 };

    auto direct_print = [&]() {
        for (const auto &x : arr1) cout << ' ' << x;
        for (const auto &x : arr2) cout << ' ' << x;
        cout << endl;
    };

    auto reverse_print = [=]() {
        for (int i = arr2.size() - 1; i >= 0; --i) cout << ' ' << arr2[i];
        for (auto p = arr1.crbegin(); p != arr1.crend(); ++p) {
            cout << ' ' << *p;
        }
        cout << endl;
    };

    print_elements(direct_print, 3);
    print_elements(reverse_print, 2);

    return 0;
}
```

```
#include <iostream>
#include <vector>
using namespace std;

class Ring {
public:
    Ring(initializer_list<int> list) : _vec(list), _seek(0) {}
    void crank() {
        auto cycle_print = [](Ring *ring) {
            ring->print();
            if (++ring->_seek == ring->_vec.size()) ring->_seek = 0;
        };
        for (uint i = 0; i <= _vec.size(); ++i) cycle_print(this);
    }

private:
    void print() const {
        auto element = [this](int i) { cout << ' ' << _vec[i]; };
        for (uint i = _seek; i < _vec.size(); ++i) element(i);
        for (uint i = 0; i < _seek; ++i) element(i);
        cout << endl;
    }

    vector<int> _vec;
    uint _seek;
};

int main() {
    Ring ring = { 0, 1, 1, 2, 3, 5, 8, 13 };
    ring.crank();
    return 0;
}
```

Альтернативный синтаксис функций

```
// Проблема:
template <typename Lhs, typename Rhs>
RETURN_TYPE add_func(const Lhs &lhs, const Rhs &rhs) {
    return lhs + rhs;
}

// Решение:
template <typename Lhs, typename Rhs>
auto add_func(const Lhs &lhs, const Rhs &rhs) -> decltype(lhs + rhs) {
    return lhs + rhs;
}

// Пример попроще:
struct SomeStruct {
    auto foo(int x, int y) -> int;
};

auto SomeStruct::foo(int x, int y) -> int {
    return x + y;
}
```

Улучшение конструкторов объектов

```
class Member {
public:
    Member() {}
    explicit Member(const string &name) : _name(name) {}
private:
    string _name = "Undefined";
};

class Robot {
public:
    Robot(string model) : _model(model) {}
    Robot() : Robot("R2D2") {}
private:
    string _model;
};

class BaseClass {
public:
    BaseClass(int value);
};
class DerivedClass : public BaseClass {
public:
    using BaseClass::BaseClass;
};
```

Явное замещение виртуальных функций и финальность*

```
// Проблема:
struct Base {
    virtual void some_func();
};

struct Derived : Base {
    void sone_func();
};

// Пути решения:
struct Base {
    virtual void some_func();
    virtual void f(int);
    virtual void g() const;
};

struct Derived : public Base {
    void sone_func() override;              // ошибка
    void f(int) override;                   // OK
    virtual void f(long) override;          // ошибка
    virtual void f(int) const override;     // ошибка
    virtual int f(int) override;            // ошибка
    virtual void g() const final;           // OK
    virtual void g(long);                   // OK
};
```

Константа нулевого указателя

```
// Например:
void foo(char *);
void foo(int);

foo(0);     // какая из функций будет вызвана?
```

```
// Новое ключевое слово:
char *pc = nullptr;      // верно
int  *pi = nullptr;      // верно
bool   b = nullptr;      // верно. b = false
int    i = nullptr;      // ошибка
```

```
foo(nullptr);            // вызывает foo(char *), а не foo(int)
```

Перечисления со строгой типизацией

```
enum Enum1;                         // неверно для C++ и C++11
enum Enum2 : unsigned int;          // верно для C++11
enum class Enum3;                   // верно для C++11
enum class Enum4 : unsigned int;    // верно для C++11


// Например:
#include <iostream>
using namespace std;

int main() {
    enum OldEnum { ONE, TWO, NINE = 9, MINUS_ONE = -1 };
    OldEnum old_enum_value = NINE;
    if (old_enum_value == 9) cout << NINE << endl;

    enum class NewEnum : unsigned int { ONE, TWO, NINE = 9 };
    NewEnum new_enum_value = NewEnum::ONE;
    if (new_enum_value != NewEnum::TWO) {
        cout << "Is not NewEnum::TWO" << endl;
    }

    return 0;
}
```

Угловые скобки

```
template <int Number>
class X {
public:
    X() : _value(Number) {}

private:
    int _value;
};

template <class Type>
class Y {
public:
    Y() {}

private:
    Type _value;
};

int main() {
    X<2> var_x;
    Y<X<5>> var_y;
    Y<X<(8 >> 2)>> another_y;

    // ...

    return 0;
}
```

Операторы явного преобразования

```
class Array {
public:
    Array(initializer_list<int> list) : _vec(list) {}
    explicit operator bool() const {
        return _vec.size() != 0;
    }

private:
    vector<int> _vec;
};

void check_expressions(const char *name, const Array &arr) {
    cout << name << " is " << (arr ? "not empty" : "empty") << endl;
    cout << name << ' ' << (arr + 1) << endl; // ошибка!
}

int main() {
    Array first_arr = { 1, 2, 3 };
    Array second_arr = {};

    check_expressions("first", first_arr);
    check_expressions("second", second_arr);

    return 0;
}
```

"typedef" для шаблонов

```
// Раньше нельзя было делать так:
template <typename First, typename Second, int third>
class SomeType;

template <typename Second>
typedef SomeType<OtherType, Second, 5> TypedefName;


// В новом стандарте можно вот так:
template <typename First, typename Second, int third>
class SomeType;

template <typename Second>
using TypedefName = SomeType<OtherType, Second, 5>;


// using вместо typedef:
typedef void (*Type)(double);          // Старый стиль
using OtherType = void (*)(double);    // Новый синтаксис
```

Улучшения в ядре языка

Шаблоны с переменным числом аргументов

```
// для класса (например):
template<typename... Values> class Tuple;

Tuple<int, vector<int>, map<string, vector<int>>> some_object;

// можно так:
template<typename First, typename... Rest> class Tuple;

// переменное число базовых классов:
template <typename... Bases>
class ClassName : public Bases... {
public:
    ClassName (Bases &&... bases) : Bases(bases)... {}
};



// для функции (например):
template<typename... Params>
void printf(const string &format, Params... parameters);
```


http://en.wikipedia.org/wiki/Variadic_templates

```
#include <iostream>
using namespace std;

class Cat {
public:
    Cat(const string &name, int age) : _name(name), _age(age) {}
    friend ostream &operator << (ostream &os, const Cat &cat) {
        os << "Cat's name: " << cat._name << ", age: " << cat._age;
        return os;
    }

private:
    string _name;
    int _age;
};

void print_values() { cout << "The end" << endl; }

template <typename T, typename... Args>
void print_values(T value, Args... args) {
    cout << "tuple size: " << sizeof...(Args) << ", "
        << "current value: " << value << endl;
    print_values(args...);
}

int main() {
    int i = 555;
    double d = 1.618;
    Cat cat("Murka", 3);
    print_values(i, &i, d, cat, &cat);
    return 0;
}
```

```
class ICell { /* ... */ };
class SimpleCell : public ICell { /* ... */ };

class IArea { /* ... */ };
class CurveArea : public IArea { /* ... */ };

struct ICellsFactory {
    virtual ICell *createCell() = 0;
};
struct SimpleCellsFactory : public ICellsFactory {
    ICell *createCell() { return new SimpleCell; }
};

struct IAreasFactory {
    virtual IArea *createArea() = 0;
};
struct CurveAreasFactory : public IAreasFactory {
    IArea *createArea() { return new CurveArea; }
};

template <class... Factories>
class MultiFactory : public Factories... {};

int main() {
    MultiFactory<SimpleCellsFactory, CurveAreasFactory> mf;
    ICell *cell = mf.createCell();
    IArea *area = mf.createArea();
    // ...
    delete cell;
    delete area;
    return 0;
}
```

Новые строковые литералы

```
#include <iostream>
using namespace std;

int main() {
    char u8_str[] = u8"\u00a9 Привет, Мир!";
    char16_t u16_str[] = u"\u2665 Привет, Весна!";
    char32_t u32_str[] = U"\u2620 Привет, 2012!";

    cout << u8_str << endl;
    cout << u16_str << endl;
    cout << u32_str << endl;

    char delimeted_str[] = R"(Some stirng with "" or \ or ())";
    char example2_str[] = u8R"ddd(Можно указывать любой ограничитель)ddd";

    cout << delimeted_str << endl;
    cout << example2_str << endl;

    return 0;
}
```

Литералы определяемые пользователем

```
#include <iostream>
#include <cstdlib>
#include <cstring>
using namespace std;

class ExtendedInt {
public:
    ExtendedInt(int value, int size) : _value(value), _size(size) {}
    friend ostream &operator << (ostream &os, const ExtendedInt &ei) {
        os << "value: " << ei._value << ", size: " << ei._size;
        return os;
    }
private:
    int _value, _size;
};

ExtendedInt operator "" _ex_int(const char *literal) {
    return ExtendedInt(atoi(literal), strlen(literal));
}

int main() {
    ExtendedInt variable = 12345_ex_int;
    cout << variable << endl;
    return 0;
}
```

Явное умолчание и удаление специальных функций членов

```
struct OtherType { /* ... */ };
struct SomeType {
    SomeType() = default;
    SomeType(OtherType value);
};

struct NonCopyable {
    NonCopyable() = default;
    NonCopyable(const NonCopyable&) = delete;
    NonCopyable &operator= (const NonCopyable&) = delete;
};

struct NoInt {
    void f(double i);
    void f(int) = delete;
};

struct OnlyDouble {
    void f(double d);
    template<class T> void f(T) = delete;
};
```

Тип long long int

```
#include <iostream>
using namespace std;

void find_max() {
    long long int var = 1;
    while (var > 0) {
        cout << var << endl;
        var <<= 1;
    }

    cout << "max: " << var - 1 << endl;
}

int main() {
    find_max();

    cout << "sizeof(long long int) = "
        << sizeof(long long int) << endl;

    return 0;
}
```

Статическая диагностика

```
// пример #1:
#define N 5

int main() {
    static_assert(N > 9, "N must not less that 10");

    // ...

    return 0;
}


// пример #2:
template <typename ... Types>
void run(Types... params) {
    static_assert(sizeof...(Types) > 1,
        "Should be used with at least two arguments");

    // ...
}

int main() {
    run(33600, 8.31);

    return 0;
}
```

Работа `sizeof` с данными класса без создания объекта

```
#include <iostream>
using namespace std;

struct OtherType {
    char _name[20];
    int _value;
};

struct SomeType {
    OtherType _concrete_inner_object;
};

int main() {
    cout << "sizeof(SomeType::_concrete_inner_object) = "
         << sizeof(SomeType::_concrete_inner_object) << endl;

    return 0;
}
```

Прочие улучшения ядра языка

- Многонитевая модель памяти
- Локальное хранилище нити
- Разрешена реализация сборщика мусора

Повышение производительности за счёт ядра языка

Rvalue и семантика перемещения

- Возможность распознавания временных объектов
- Не копировать временные объекты, а сразу перемещать их в нужное место
- Перемещаются не сами объекты, а указатели на динамически выделенную память

```
move semantics
reference semantics
reference semantics
move semantics
move semantics
move semantics
reference semantics
```

```
#include <iostream>
#include <string>
using namespace std;

void print_string(const string &str) {
    cout << "reference semantics" << endl;
}

void print_string(string &&str) {
    cout << "move semantics" << endl;
}

string get_string() { return "Function string"; }
const string get_const_string() { return "Function const string"; }

int main() {
    char c_str[100] = "C string";
    string str = "std::string";
    const string const_str = "const std::string";

    print_string(c_str);
    print_string(str);
    print_string(const_str);
    print_string("Another C string");
    print_string(string("Another std::string"));
    print_string(get_string());
    print_string(get_const_string());

    return 0;
}
```

```
#include <iostream>
using namespace std;

#define N 5

class Array {
public:
    Array() {
        _arr = new int[N];
        for (int i = 0; i < N; ++i) _arr[i] = N - i;
    }

    Array(Array &&other) {
        cout << "move semantics constructor" << endl;
        _arr = other._arr;
        other._arr = 0;
    }

    ~Array() { /* if (_arr) */ delete [] _arr; }

    void info(const char *label) {
        cout << '[' << label << "] " << _arr << ':';
        for (int i = 0; i < N; ++i) cout << ' ' << _arr[i];
        cout << endl;
    }

private:
    int *_arr;
};
```

```
void inspect_move(Array &&a) {
    a.info("function argument");
    Array moved_a = move(a);
    moved_a.info("moved array");
    a.info("crush test");                // ошибка
}

int main() {
    inspect_move(Array());
    return 0;
}
```

```
[function argument] 0x11f1010: 5 4 3 2 1
move semantics constructor
[moved array] 0x11f1010: 5 4 3 2 1
```

http://indifferent.me/post/2011/08/02/rvalue-ссылки-и-изменения-которые-они-привносят-в-C++.aspx

Обобщённые константные выражения

- Возможность использовать константные функции, чьи значения определяются на этапе компиляции

```
#include <iostream>
using namespace std;

constexpr int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

constexpr int totient(int x, int i) {
    return i == 0 ? 0 : (gcd(x, i) == 1) + totient(x, i - 1);
}

constexpr int totient(int x) { return totient(x, x - 1); }

int main() {
    constexpr int x = totient(1000);
    cout << x << endl;
    return 0;
}
```

Изменения в определении простых типов

Класс — простой тип, если он:

- Тривиальный
- Со стандартным размещением
- Типы всех его нестатических свойств также являются типами простых данных

Изменения стандартной библиотеки

Обновления в стандартной библиотеке

Поддержка:

- Rvalue и семантики перемещения
- UTF-16 и UTF-32 символов
- Шаблонов с переменным количеством аргументов (в купе с Rvalue)
- Константных выражений времени компиляции
- Автоматического определения типа
- Операторов явного преобразования
- Умолчальных / удалённых функций

Связывание и удержание

```
#include <iostream>
#include <functional>
using namespace std;
using namespace placeholders;

int calc_a_number(int some_n) {
    return some_n % 5;
}

void print(const char *name, int number) {
    for (int i = 0; i < number; ++i) {
        cout << (i + 1) << ". " << name << endl;
    }
}

int main() {
    auto delegat = std::bind(&print, _1, bind(&calc_a_number, _2));

    delegat("Hoho, peoples!!", 7);
    delegat("Tomorow is friday :)", 13);

    return 0;
}
```

Обёртка ссылок

```
#include <iostream>
#include <functional> // #include <utility>
using namespace std;

void foo(int &r) {
    r++;
}

template <typename Func, typename Param>
void bar(Func f, Param t)  {
    f(t);
}

int main() {
    int i = 0;
    bar(foo, i);
    cout << i << endl; // получим: 0

    bar(foo, std::ref(i));
    cout << i << endl; // получим: 1

    return 0;
}
```

Поддержка нитей и средства работы с ними

```
#include <iostream>
#include <thread>
using namespace std;

int main() {
    try {
        std::thread thr([]() {
            cout << "lambda calling from thread" << endl;
        });

        thr.join();

    } catch (std::system_error &e) {
        cerr << "Error: " << e.what() << endl;
        cerr << "try to use -lpthread compiler option" << endl;
    }

    return 0;
}
```

http://en.cppreference.com/w/cpp/thread

http://channel9.msdn.com/Events/GoingNative/GoingNative-2012/Threads-and-Shared-Variables-in-C-11?format=html5

```
#include <iostream>
#include <vector>
#include <thread>
using namespace std;

void run_on_thread(int tid) {
    cout << "Calling from thread " << tid << endl;
}

int main() {
    vector<thread> threads;
    for (int i = 0; i < 10; ++i) {
        threads.push_back(thread(run_on_thread, i));
    }

    cout << "Launched from the main" << endl;

    for (auto &thr : threads) thr.join();

    return 0;
}
```

```
// includes...
vector<int> saw(int parts, int number) {
    vector<int> vec;
    for (int i = 0; i < number; i += number / parts) vec.push_back(i);
    vec.push_back(number);
    return vec;
}

void calc_portion(const vector<int> &v1, const vector<int> &v2,
    int &result, int L, int R)
{
    for(int i = L; i < R; ++i) result += v1[i] * v2[i];
}

int main(){
    int n_elements = 100000;
    int n_threads = 2;
    int result = 0;

    vector<thread> threads;
    vector<int> v1(n_elements, 1), v2(n_elements, 2);
    vector<int> limits = saw(n_threads, n_elements);
    for (int i = 0; i < n_threads; ++i) {
        threads.push_back(thread(calc_portion, ref(v1), ref(v2),
           ref(result), limits[i], limits[i+1]));
    }

    for(auto &t : threads) t.join();
    cout << result << endl;

    return 0;
}
```

race condition!!

```
// решение #1:
#include <thread>
#include <mutex>

static std::mutex barrier;

void calc_portion(const vector<int> &v1, const vector<int> &v2,
    int &result, int L, int R)
{
    int partial_sum = 0;
    for(int i = L; i < R; ++i) partial_sum += v1[i] * v2[i];
    std::lock_guard<mutex> block_threads_until_finish(barrier);
    result += partial_sum;
}

// решение #2:
#include <thread>
#include <atomic>

void calc_portion(const vector<int> &v1, const vector<int> &v2,
    std::atomic<int> &result, int L, int R)
{
    int partial_sum = 0;
    for(int i = L; i < R; ++i) partial_sum += v1[i] * v2[i];
    result += partial_sum;
}

int main(){
    // ...
    std::atomic<int> result(0);
    // ...
}
```

Кортеж типов

```
#include <iostream>
#include <tuple>
using namespace std;

int main() {
    typedef std::tuple<int, double, long &, const char *> test_tuple;
    cout << std::tuple_size<test_tuple>::value << endl;

    long length = 18;
    test_tuple proof(123, 9.8, length, "Hi!");

    length = std::get<0>(proof);
    get<3>(proof) = "Hello!";
    cout << get<2>(proof) << endl;
    cout << get<3>(proof) << endl;

    std::tuple_element<1, test_tuple>::type d = get<1>(proof);
    cout << "d = " << d << endl;

    tuple<float, char> t1;
    tuple<float, char> t2(0.01, 'c');
    t1 = t2;
    cout << get<0>(t1) << endl;
    cout << get<1>(t1) << endl;

    return 0;
}
```

Неотсортированные хеш таблицы

Тип хеш таблицы	Ассоциировано значение	Эквивалентные ключи
`std::unsortered_set`	Нет	Нет
`std::unsortered_multiset`	Нет	Да
`std::unsortered_map`	Да	Нет
`std::unsortered_multimap`	Да	Да

http://en.wikipedia.org/wiki/Hash_tables

Регулярные выражения

```
#include <iostream>
#include <string>
#include <regex>
using namespace std;

int main() {
    try {
        std::regex date_exp(R"(\d{1,2}[./-]\d{1,2}[./-]\d{4})");
        std::cmatch match;

        char some_text[] = "Today 15.03.2012\nTomorrow 16/3/2012";
        if (std::regex_search(some_text, match, date_exp)) {
            const size_t n = match.size();
            for (size_t i = 0; i < n; ++i) {
                string str(match[i].first, match[i].second);
                cout << str << endl;
            }
        }
    } catch (std::regex_error &e) {
        cerr << "Error: " << e.what() << endl;
    }

    return 0;
}
```

Умные указатели общего назначения

- `std::auto_ptr` — устарел!

Отныне есть:

- `std::unique_ptr`
- `std::shared_ptr` и `std::weak_ptr`

```
// уникальный указатель:
#include <iostream>
#include <memory>

int main() {
    std::unique_ptr<int> p1(new int(5));
    std::unique_ptr<int> p2 = p1;        // ошибка!
    std::unique_ptr<int> p3 = std::move(p1);

    p3.reset();    // отчищает память
    p1.reset();    // ничего не происходит

    return 0;
}
```

```
// общий указатель:
std::shared_ptr<int> p1(new int(5));
std::shared_ptr<int> p2 = p1;

p1.reset();    // сбрасывает первый указатель
p2.reset();    // удаляет выделенную память



// слабый указатель:
std::shared_ptr<int> p1(new int(5));
std::weak_ptr<int> wp1 = p1;  // память принадлежит p1

{
    std::shared_ptr<int> p2 = wp1.lock();
    // теперь память принадлежит p1 и p2

    if (p2) { // общий указатель следует проверять перед использованием
        // используем p2
    }
} // p2 уничтожен, память принадлежит только p1

p1.reset(); // выделенная память удаляется

std::shared_ptr<int> p3 = wp1.lock();
// p3 пустой, и не указывает на какую-либо память

if (p3) {
  // данный код не будет выполнен
}
```

Больше возможностей при работе со случайными числами

Движок	Целое/дробное	Качество	Скорость
`linear_congruential`	целое	среднее	средняя
`subtract_with_carry`	оба	среднее	высокая
`mersenne_twister`	целое	хорошее	высокая

Распределители:

```
uniform_int_distribution, bernoulli_distribution,
geometric_distribution, poisson_distribution,
binomial_distribution, uniform_real_distribution,
exponential_distribution, normal_distribution,
gamma_distribution
```


http://en.cppreference.com/w/cpp/numeric/random

```cpp
#include <iostream>
#include <string>
#include <map>
#include <random>
#include <functional>
using namespace std;

template <typename DistFunc>
void show_distribution(DistFunc dist_f) {
    map<int, int> hist;
    for(int i = 0; i < 10000; ++i) ++hist[dist_f()];
    for(auto &p : hist) {
        cout << p.first << ' ' << string(p.second / 100, '*') << '\n';
    }
    cout << endl;
}

int main() {
    std::random_device rd;
    std::mt19937 engine(rd());

    std::uniform_int_distribution<int> uniform_distr(0, 9);
    auto uniform_gen = bind(uniform_distr, engine);
    show_distribution(uniform_gen);

    std::poisson_distribution<> poisson_distr(5);
    auto poisson_gen = bind(poisson_distr, engine);
    show_distribution(poisson_gen);

    return 0;
}
```

Полиморфные обёртки для объектов функций

```
#include <iostream>
#include <functional>
using namespace std;

struct Equal {
    bool operator()(short x, short y) { return x == y; }
};

bool adjacent(long x, long y) { return x + 1 == y || x - 1 == y; }

int main() {
    std::function<int (int, int)> func;
    std::plus<int> add;
    func = add;
    cout << func(1, 2) << endl;

    function<bool (short, short)> func2;
    if (!func2) {
        func2 = &adjacent;
        Equal eq;
        func = ref(eq);
    }
    func = func2;
    cout << (func(2, 3) ? "true" : "false") << endl;
    return 0;
}
```

Проверка типа для метапрограммирования

```
#include <iostream>
#include <type_traits>

template <bool B> struct Algorithm {
    template<class T1, class T2>
    static int do_it(T1 &p1, T2 &p2) { return p1 + p2; }
};

template <> struct Algorithm<true> {
    template<class T1, class T2>
    static int do_it(T1 p1, T2 p2) { return p1 * p2; }
};

template<class T1, class T2> int elaborate (T1 a, T2 b) {
    return Algorithm<std::is_integral<T1>::value &&
        std::is_floating_point<T2>::value>::do_it(a, b) ;
}

int main() {
    std::cout << elaborate(5, 5) << std::endl;
    std::cout << elaborate(10, 1.618) << std::endl;

    return 0;
}
```


http://en.cppreference.com/w/cpp/types

Методы для определения типа возвращаемого объектом-функцией

```
#include <iostream>
using namespace std;

struct Confused {
    double operator()(int p) const { return p * 2.71; }
    int operator()(double p) const { return (int)(p * 2.71 + 0.5); }
};

template <class Obj>
class Calculus {
public:
    template <class Arg>
    typename std::result_of<Obj(Arg)>::type operator()(Arg a) const {
        return _member(a);
    }
private:
    Obj _member;
};

int main() {
    Calculus<Confused> cc;
    cout << cc(10) << endl;
    cout << cc(10.0) << endl;
    return 0;
}
```

Ускорение компиляции

Внешние шаблоны

В библиотеке:

```
template class std::vector<MyClass>;
```

Там где используете:

```
extern template class std::vector<MyClass>;
```

Напоследок

Что нас ждёт в будущем

- Модули
- Десятичные типы
- Специальные математические функции
- Концепты
- Более полная поддержка сборки мусора (если потребуется)
- Отражения
- Макро скопы

Спасибо за внимание!

- **Глеб Аверчук** <*altermn@gmail.com*>
- Презентация доступна по адресу: **http://newmen.pro/C++11.pdf**