# Сторожевые объекты и операторы

2021

Сторожевые объекты

Операторы

```
struct MyGuard {

    X& _x;

    MyGuard(X& x): _x(x) {
        // изменить состояние x
    }

    ~MyGuard() {
        // восстановить состояние x
    }

    MyGuard(const MyGuard&) = delete;
    MyGuard& operator=(const MyGuard&) = delete;

};
```

# Пример: блокировки

```
template <class Mutex>
struct lock_guard {
    Mutex& mtx;
    lock_guard(Mutex& m): mtx(m) {
        mtx.lock(); // захват ресурса
    }
    ~lock_guard() {
        mtx.unlock(); // освобождение ресурса
    }
};
```

# Пример: блокировка в очереди

```
struct Pipeline {

    std::mutex mtx;
    std::vector<float> queue;

    void add_to_queue(float x) {
        std::lock_guard<std::mutex> lock(mtx);
        queue.push_back(x);
    }

};
```

# Пример: кэш

```
template <class T>
struct cache_guard {

    T& object;
    std::string filename;
    bool found = false;

    cache_guard(T& obj, std::string key): object(obj) {
        filename = myhash(key); // хэш от ключа
        std::ifstream in(filename);
        if (in.is_open()) { in >> object; found = true; }
    }

    ~cache_guard() {
        if (!found) { std::ofstream(filename) << object; }
    }
};
```

# Пример: кэш

```
int size = 1024;
MyVector vec(size);
std::string key = std::to_string(size);
cache_guard<MyVector> g(vec, key);
if (!g.found) {
    // сгенерировать
}
// использовать массив
```

# Пример: std::ostream::sentry

```
std::ostream& operator<<(std::ostream& out, const X& x) {
    std::ostream::sentry s(out);
    if (s) {
        // ...
    }
    return out;
}
```

# Пример: блокировка буфера OpenGL

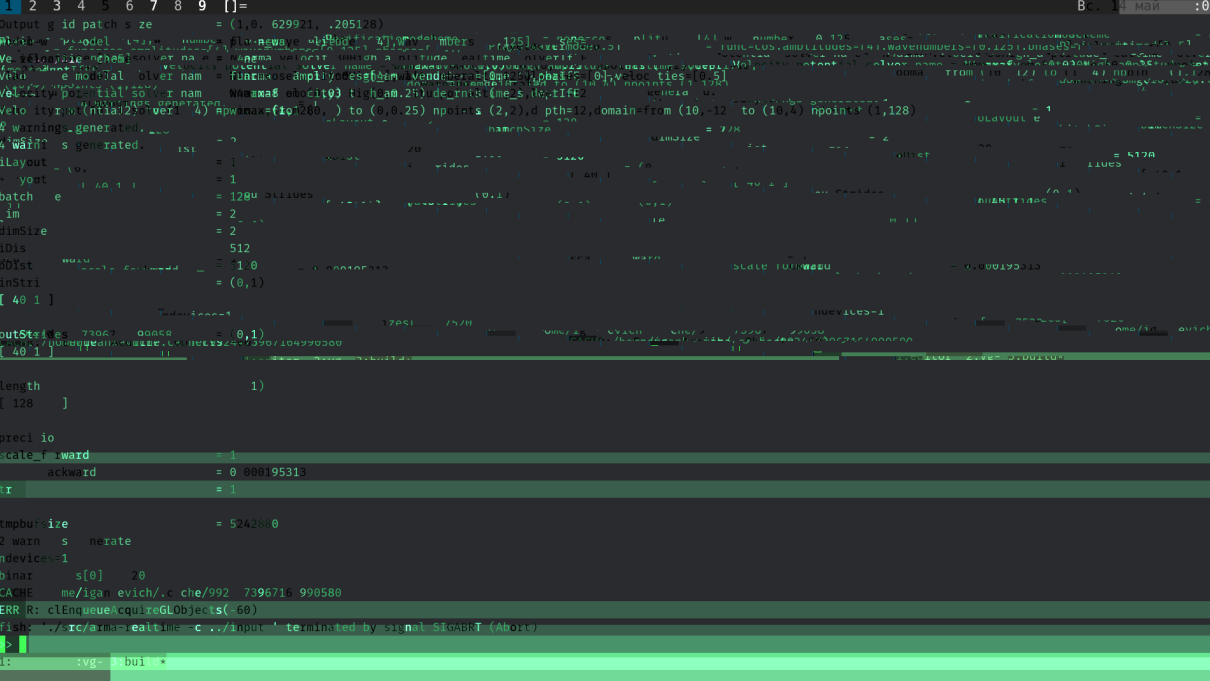
```
struct opengl_guard {
    cl::Memory mem;
    opengl_guard(cl::Memory m): mem(m) {
        clEnqueueAcquireGLObjects(mem); // блокировка
    }
    ~opengl_guard() {
        clEnqueueReleaseGLObjects(mem); // разблокировка
    }
};
```

# Пример: блокировка буфера OpenGL

```
struct opengl_guard {
    cl::Memory mem;
    opengl_guard(cl::Memory m): mem(m) {
        clEnqueueAcquireGLObjects(mem); // блокировка
    }
    ~opengl_guard() {
        clEnqueueReleaseGLObjects(mem); // разблокировка
    }
};
```

```
void do_opencl_computations(cl::Memory mem) {
    opengl_guard g(mem); // заблокировать
    // какие-то вычисления с буфером
}
cl::Memory mem = ...; // создать буфер
do_opencl_computations(mem);
draw(mem);
```

А что будет, если не снять блокировку?



«Resource acquisition is initialization» (RAII)
Захват ресурса есть инициализация

- `std::shared_ptr`
- `std::unique_ptr`
- `std::vector`
- …

- `std::fstream`
- `std::thread`
- …

# std::ofstream

```
/**
The destructor does nothing.
The file is closed by the filebuf object,
not the formatting stream.
*/
~basic_ofstream() {}
```

# Функторы

```
struct Wave {
    float amplitude; // амплитуда
    float wave_number; // волновое число
    float velocity; // скорость
    float operator()(float x, float t) {
        return amplitude *
            std::cos(wave_number*x - velocity*t);
    }
};
```

# Функторы

```
struct Wave {
    float amplitude; // амплитуда
    float wave_number; // волновое число
    float velocity; // скорость
    float operator()(float x, float t) {
        return amplitude *
            std::cos(wave_number*x - velocity*t);
    }
};
```

```
Wave w{1.0f, 0.05f, 0.5f};
w(0, 10); // обычный вызов
w.operator()(0, 10); // явный вызов
```

# Ввод/вывод

```
struct MyStream {
    MyStream& operator<<(const MyObject& x) {
        // ...
    }
    MyStream& operator>>(MyObject& x) {
        // ...
    }
};

MyStream& operator<<(MyStream& mystr, const MyObject& x) {
    // ...
}
MyStream& operator>>(MyStream& mystr, MyObject& x) {
    // ...
}
```

# Ввод/вывод

```
class MyObject {
    int a, b;

    friend std::ostream&
    operator<<(std::ostream& out, const MyObject& obj);

    friend std::istream&
    operator>>(std::istream& in, MyObject& obj);
};

std::ostream& operator<<(std::ostream& out, const MyObject& obj) {
    return out << obj.a << ' ' << obj.b;
}

std::istream& operator>>(std::istream& in, MyObject& obj) {
    return in >> obj.a >> obj.b;
}
```

Ввод/вывод или сдвиг влево/вправо?

# Сдвиг влево/вправо

```
class MyInteger128 {
    // ...
};

MyInteger operator>>(const MyInteger128& i, int n) {
    // побитовый сдвиг i на n бит вправо
}

MyInteger128 i = 1000;
MyInteger128 j = i >> 1; // j=500
```

# Побитовые логические операторы

```
struct MyFlags {
    static const int Flag1 = 1, Flag2 = 2, Flag3 = 4;
    unsigned int value = 0;

    void set_flag(unsigned int f) {
        value = value | f;
    }

    void unset_flag(unsigned int f) {
        value = value & ~f;
    }
};

MyFlags flags;
flags.set_flag(MyFlags::Flag1);
flags.unset_flag(MyFlags::Flag1);
```

# Операторы сравнения

```
struct X {
    // для каждого типа свои
    bool operator==(const X& x) const { /* ... */ };
    bool operator<(const X& x) const { /* ... */ };
};
```

# Операторы сравнения для структур

```
struct X {
    // для каждого типа свои
    bool operator==(const X& x) const { /* ... */ };
    bool operator<(const X& x) const { /* ... */ };
    // для всех типов одинаковы
    bool operator!=(const X& x) const { return !operator==(x); }
    bool operator<=(const X& x) const {
        return operator==(x) || operator<(x);
    }
    bool operator>=(const X& x) const { return !operator<(x); }
    bool operator>(const X& x) const { return !operator<=(x); }
};
```

# Операторы сравнения для векторов

```
template <class T>
struct Vector {
    Vector<bool> operator==(const Vector& x) const { /* ... */ };
    Vector<bool> operator<(const Vector& x) const { /* ... */ };
    Vector<bool> operator==(const T& x) const { /* ... */ };
    Vector<bool> operator<(const T& x) const { /* ... */ };
};

Vector<float> x{1,2,3,4};
Vector<bool> half = x < 3;
std::cout << std::count(half.begin(), half.end(), true) << '\n';
```

# Пример: массивы и векторы Blitz++

```
using namespace blitz;
TinyVector<int,3> shape(10,10,10);
if (all(shape > 0)) {
    // ...
}

Array<float,3> x(shape);
// инициализация x
std::cout << count(x > 0) << std::endl;
```

# Запятая

```
// запятая-разделитель
std::vector<float>
    x(100),      // ок
    y(x.size()), // ошибка: произвольный порядок
    z(x.size()); // ошибка: инициализации

// запятая-оператор (Blitz++)
Array<float,3> m(3,3);
m = 1, 0, 0,
    0, 1, 0,
    0, 0, 1;

// плохой пример
int n = (std::cout << "hello", 10); // n=10
```

# Запятая

```
// еще один плохой пример
std::complex<float> operator,(float re, float im) {
    return std::complex<float>{re, im};
}

std::complex<float> z;
z = (1.0f,2.0f); // 1+2i
z = (1,2); // 2
```

# Пример: путь к файлу

```
struct MyPath {
    std::string str;
    operator const char* () const {
        return str.data();
    }
};

MyPath operator/(const MyPath& a, const MyPath& b) {
    return MyPath{a.str + "/" + b.str};
}

MyPath p1("/home/myuser"), p2("garbage.txt");
p3 = p1 / p2; // p3=/home/myuser/garbage.txt
std::remove(p3);
```

# Пример: директория

```
struct MyEntry: public dirent {
    bool operator<(const MyEntry& e) {
        return std::strcmp(d_name, e.d_name) < 0;
    }
};
struct MyDirectory {
    DIR* dir = nullptr;
    MyDirectory(const char* path) { dir = opendir(path); }
    ~MyDirectory() { if (dir) { closedir(dir); } }
    MyDirectory& operator>>(MyEntry& entry) {
        MyEntry* result = static_cast<MyEntry*>(readdir(dir));
        if (result) { entry = *result; }
        else { closedir(dir); dir = nullptr; }
        return *this;
    }
    explicit operator bool() const { return dir != nullptr; }
};
```

# Пример: директория

```
// вывод в произвольном порядке
MyDirectory home("/home/myuser");
MyEntry entry;
while (home >> entry) {
    std::cout << entry.d_name << std::endl;
}

// вывод в алфавитном порядке
MyDirectory pictures("/home/myuser/pictures");
std::vector<MyEntry> entries;
MyEntry entry;
while (home >> entry) { entries.push_back(entry); }
std::sort(entries.begin(), entries.end());
for (const auto& ent : entries) {
    std::cout << ent.d_name << std::endl;
}
```

# Литералы

```
long double operator"" _deg(long double deg) {
    return deg*M_PI/180;
}

std::cout << 90_deg << std::endl; // 1.570796
```

# Пример: комплексные числа

```
std::complex<double> operator""i(unsigned long long d) {
    return std::complex<double>{0.0, static_cast<double>(d)};
}

std::complex<double> operator""i(long double d) {
    return std::complex<double>{0.0, static_cast<double>(d)};
}

std::complex<double> c = 1.0 + 1_i;    // 1+1i в c++14
std::cout << std::abs(c) << std::endl; // 1.41421
```

# Пример: целое 128 бит

```
template<char ... Chars>
uint128_t operator"" _u128() {
    const char str[] = {Chars...};
    // ...
}

uint128_t min = 0_u128;
uint128_t max = 0xffffffffffffffffffffffffffffffff_u128;
```

# Все операторы

```
Логические          &&  ||  !   ^
Побитовые           &   |   ~   ^   >>  <<
Арифметические      +   -   *   /   %
Сравнение           <   >   <=  ==  >=  !=
Указатели           *   ->  &   ++  --
Другие              ,   =   ()  []  ""
Приведение типов    T
```