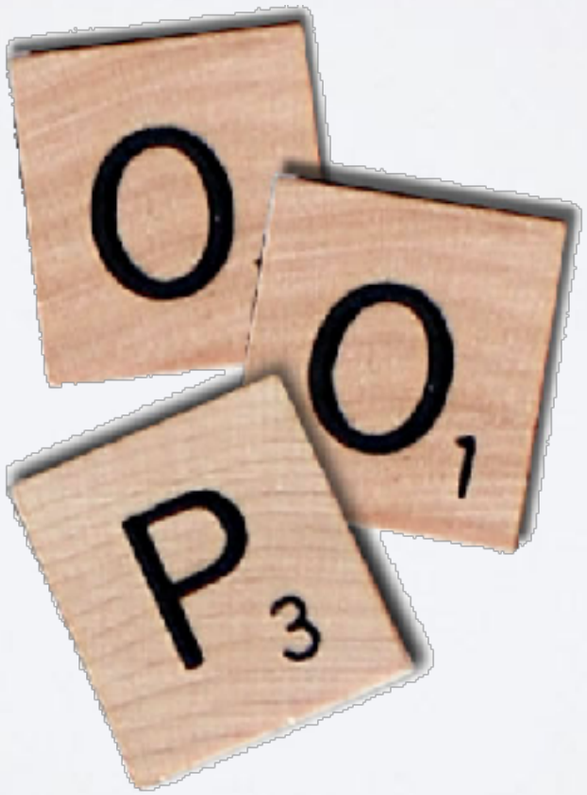
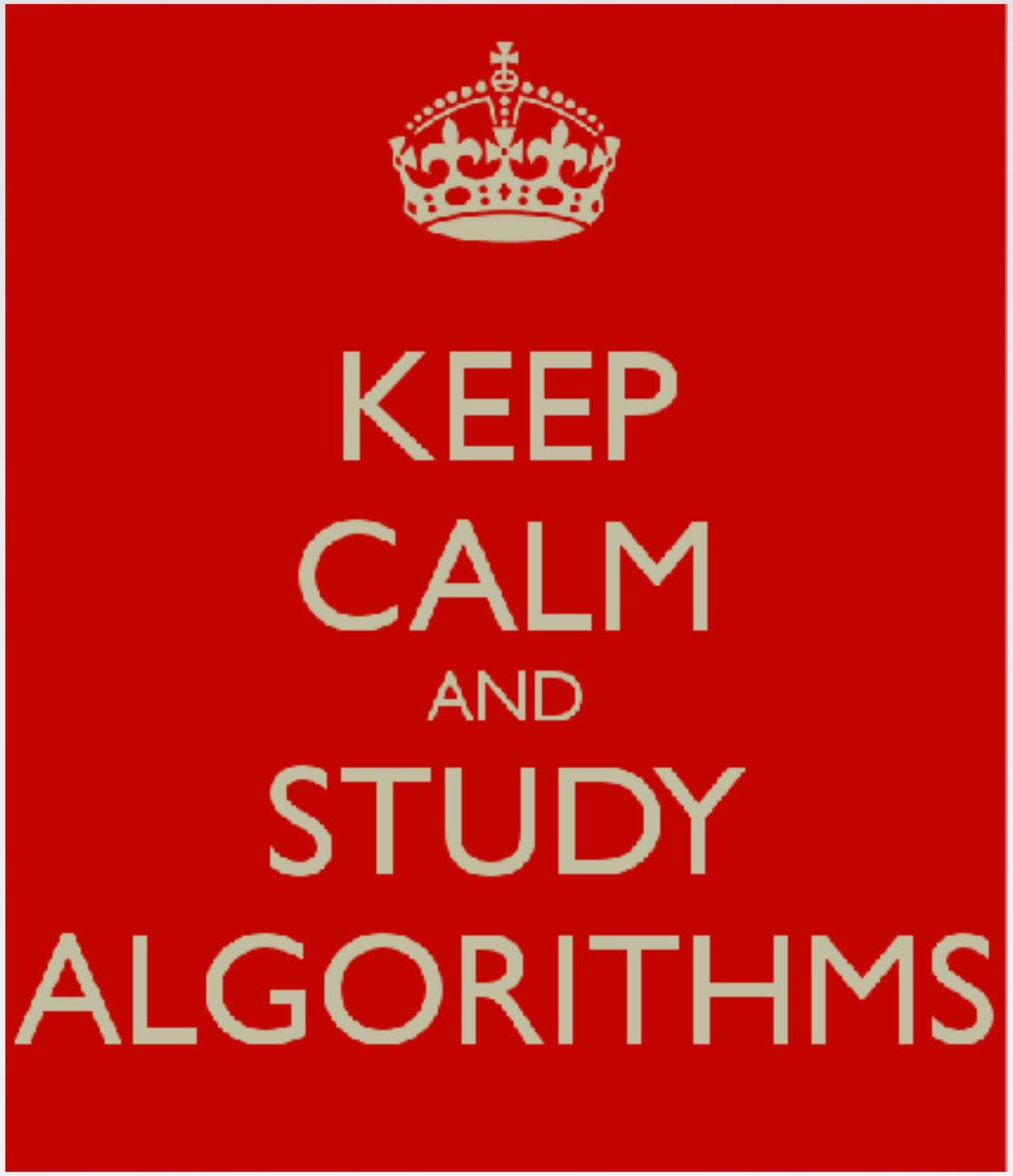
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ



Лекция № | / **07**
19.03.2018 г.



АЛГОРИТМЫ

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

inline bool isEven(int x) {
    return x % 2 == 0;
}

template <int N>
inline bool greaterThan(int x) {
    return x > N;
}

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

    if (!all_of(x.begin(), x.end(), isEven))
        cout << "Not all are even!" << endl;

    if (any_of(x.begin(), x.end(), isEven))
        cout << "But there is at least one even!" << endl;

    if (none_of(x.begin(), x.end(), greaterThan<10>))
        cout << "No number is > 10!" << endl;

    return 0;
}
```

Проверка по предикатам

```cpp
#include <string>
#include <iostream>
#include <algorithm>
#include <cctype>

using namespace std;

int main() {
    string w("Dolly"), e("  \t\t \n   "), s("Hello Dolly!");

    if (all_of(w.begin(), w.end(), ::isalnum))
        cout << w << " is alphanumeric" << endl;

    if (all_of(e.begin(), e.end(), ::isspace))
        cout << "e is completely whitespace" << endl;

    cout << "Space in " << s << ": " <<
         count_if(s.begin(), s.end(), ::isspace) << endl;

    return 0;
}
```

То же для строк

```cpp
#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;

inline bool isEven(int x) { return x % 2 == 0; }

int main() {
    vector<int> x = { 1, 2, 3, 4 }, y;
    copy_if(x.begin(), x.end(), back_inserter(y), isEven);

    for (auto val: y)
        cout << val << endl;          // 2… 4

    return 0;
}
```

copy_if

```cpp
#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;

struct less_than {
    less_than(int _value)
        : value(_value) {}

    bool operator()(int x) const {
        return x < value;
    }

    int value;
};

int main() {
    vector<int> x = { 1, 2, 3, 4 }, y;
    copy_if(x.begin(), x.end(), back_inserter(y), less_than(3));

    for (auto val: y)
        cout << val << endl;          // 1… 2

    return 0;
}
```

Функтор

```cpp
#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;

inline int increase(int i) { return i + 1; }

int main() {
    vector<int> x = { 1, 2, 3, 4 }, y;

    transform(x.begin(), x.end(), back_inserter(y), increase);

    for (auto val: y)
        cout << val << endl;          // 2… 3… 4… 5

    // in-place transform
    transform(y.begin(), y.end(), y.begin(), increase);

    for (auto val: y)
        cout << val << endl;          // 3… 4… 5… 6

    // binary op
    transform(x.begin(), x.end(),     // источник 1-го аргумента
              y.begin(),              // источник 2-го аргумента
              y.begin(),              // куда писать результат
              plus<int>());           // операция (функция или функтор)

    for (auto val: y)
        cout << val << endl;          // 4… 6… 8… 10

    return 0;
}
```

transform

```cpp
#include <string>
#include <algorithm>
#include <iostream>
#include <random>

using namespace std;

int randomLength() {
    static default_random_engine generator;
    static uniform_int_distribution<int> distribution(1, 10);
    return 1 + 2*distribution(generator);
}

string makeSpaceship(int len) {
    if (len < 3)
        return string();

    string ss(len, '<');
    auto it = ss.begin() + len / 2;

    *it++ = '*';
    fill(it, ss.end(), '>');

    return ss;
}

int main() {
    vector<int> lengths(10);
    vector<string> spaceships;

    generate(lengths.begin(), lengths.end(), randomLength);
    transform(lengths.begin(), lengths.end(), back_inserter(spaceships),
              makeSpaceship);
    for (const auto &s: spaceships)
        cout << s << endl;
    return 0;
}
```

fill и generate

```cpp
#include <string>
#include <algorithm>
#include <iostream>
#include <sstream>

using namespace std;

int main() {
    string s("Hello, World!");
    istringstream is(s);

    while (is.good()) {
        string word;

        is >> word;
        cout << "Word: '" << word << "'" << endl;
    }

    return 0;
}
```

```
Word: 'Hello,'
Word: 'World!'
```

Разбивка на слова

```
#include <string>
#include <algorithm>
#include <iostream>
#include <sstream>
#include <cctype>

using namespace std;

void depunct(string &s) {
    auto new_end =
        remove_if(s.begin(), s.end(), ::ispunct);

    s.erase(new_end, s.end());
}

int main() {
    string s("Hello, World!");
    istringstream is(s);

    while (is.good()) {
        string word;

        is >> word;
        depunct(word);
        cout << "Word: '" << word << "'" << endl;
    }

    return 0;
}
```

```
Word: 'Hello'
Word: 'World'
```

remove_if

Входные итераторы

Выходной итератор

Абстракция входных данных

Абстракция выходных данных

Алгоритм

Абстракция действия над одним элементом

Предикат или операция

ИСКЛЮЧЕНИЯ (EXCEPTIONS)

- Сигнализирование об ошибках в C:
  - Код возврата функции.
  - Глобальная переменная (**errno**).
  - **Можно проигнорировать!**
- Исключения **нельзя** проигнорировать! Muahahaha

```cpp
#include <iostream>

class A {
public:
    ~A() { std::cout << "~A()" << std::endl; }
};

class B {
public:
    ~B() { std::cout << "~B()" << std::endl; }
};

void cc() {
    throw 123;
}

void bb() {
    B b;
    cc();
}

void aa() {
    A a;
    bb();
}

int main() {
    try {
        aa();
    }
    catch (int v) {
        std::cout << "Caught " << v << std::endl;
    }
}
```

```
~B()
~A()
Caught 123
```

Работа исключений

КАК ЭТО РАБОТАЕТ?

- При генерации исключения с помощью **throw** начинается раскрутка стека *(stack unrolling)*.

- У всех объектов всех функций на стеке вызываются деструкторы.

- Раскрутка останавливается, если найден подходящий обработчик исключения.

```cpp
#include <iostream>
#include <string>
#include <cctype>
#include <exception>

using namespace std;

class INNException          : public std::exception {};
class WrongLengthException   : public INNException {};
class WrongCharsException    : public INNException {};
class WrongChecksumException : public INNException {};

unsigned long long int parseINN(const string &s) {
    if (s.length() != 10)
        throw WrongLengthException();

    if (!all_of(s.begin(), s.end(), ::isdigit))
        throw WrongCharsException();

    static int coeffs[] = { 2, 4, 10, 3, 5, 9, 4, 6, 8 };

    int res = 0;
    for (int i = 0; i < 9; ++i)
        res += (int(s[i]) - int('0')) * coeffs[i];

    if (int(s[9]) - int('0') != (res % 11) % 10)
        throw WrongChecksumException();

    return stoull(s);
}
```

Пример: валидация ИНН

```
int main() {
    string s;
    if (getline(cin, s)) {
        try {
            auto inn = parseINN(s); // например, 5445264092

            cout << "Хороший, годный ИНН: " << inn << endl;
        }
        catch (WrongLengthException &) {
            cerr << "ИНН имеет неверную длину!" << endl;
        }
        catch (WrongCharsException &) {
            cerr << "ИНН содержит недопустимые символы!" << endl;
        }
        catch (WrongChecksumException &) {
            cerr << "У ИНН неверная контрольная сумма!" << endl;
        }
        catch (INNException &) {
            cerr << "ИНН никуда не годится!" << endl;
        }
    }
}
```

Обработка исключений

```cpp
// ...
using namespace std;

class INNException           : public std::exception {};

class WrongLengthException   : public INNException {
public:
    const char *what() const noexcept { return "неверная длина"; }
};

class WrongCharsException    : public INNException {
public:
    const char *what() const noexcept { return "неверные символы"; }
};

class WrongChecksumException : public INNException {
public:
    const char *what() const noexcept { return "неверная контр. сумма"; }
};

unsigned long long int parseINN(const string &s);

int main() {
    string s;
    if (getline(cin, s)) {
        try {
            auto inn = parseINN(s); // например, 5445264092

            cout << "Хороший, годный ИНН: " << inn << endl;
        }
        catch (INNException &exc) {
            cerr << "Ошибочный ИНН (" << exc.what() << ")!" << endl;
        }
    }
}
```

Другой вариант

```
void h() {
    try {
        // …
    }
    catch (Matherr &) {
        if (can_handle_it) {    // можем полностью обработать ошибку?
            // …обработка…
            return;
        } else {
            // делаем что можем
            throw;  // и кидаем исключение дальше
        }
    }
}
```

Повторная генерация

```
void process_file(const char *fn) {
    FILE *fp = fopen(fn, "r");

    try {
        // работаем с файлом
    }
    catch (...) {
        fclose(fp);
        throw;
    }

    fclose(fp);
}
```

Перехват всего

```
class FilePtr {
    FILE *fp;
public:
    FilePtr(const char *fn, const char *access) {
        fp = fopen(fn, access);
    }
    FilePtr(FILE *fp) { this->fp = fp; }
    ~FilePtr()        { if (fp) fclose(fp); }

    operator FILE *() { return fp; }
};
```

```
void process_file(const char *fn) {
    // Принцип RAII:
    // Resource Acquisition is Initialization
    // (выделение ресурса есть инициализация)
    FilePtr f(fn, "r");

    // …просто используем f
    // файл в любом случае будет закрыт автоматически
}
```

«Обёртка»

- Деструктор вызывается только для полностью сконструированного объекта!
- Если в конструкторе было исключение, то будет утечка памяти:

```
class Y {
    int *p;
    void init() { /* здесь бабах! */ }

public:
    Y(int s) { p = new int[s]; init(); }
    ~Y() { delete p; }
};
```

Исключения в конструкторах

```
class FileProcessor {
    FilePtr fp;
    MemoryHeap mh;
public:
    FileProcessor(const char *fn, size_t sz) :
        fp(fn, "rw"),                  // открываем файл
        mh(sz)                         // выделяем память
    {
        // Сначала конструируется fp, потом mh
        // Если конструктор MemoryHeap генерирует исключение,
        // то будет вызван деструктор для fp
    }
};
```

Защита от исключений в конструкторах

```
Getting value 10
0x7fe169c03990, 0x0
Getting value 100
Losing value 10
0x0, 0x7fe169c039a0
Losing value 100
```

```cpp
#include <memory>
#include <iostream>

using namespace std;

class Valuable {
    int value;
public:
    Valuable(int val) : value(val) {
        cout << "Getting value " << value << endl;
    }
    ~Valuable() { cout << "Losing value " << value << endl; }
};

int main() {
    unique_ptr<Valuable> v1(new Valuable(10)), v2;

    cout << v1.get() << ", " << v2.get() << endl;

    v1 = unique_ptr<Valuable>(new Valuable(100));
    v1.swap(v2);

    cout << v1.get() << ", " << v2.get() << endl;

    return 0;
}
```

unique_ptr

BEST PRACTICES

- Использовать иерархию классов исключений: не кидать **int**, **const char** * и т.п.
- Делать классы исключений простыми.
- Использовать исключения только для *исключительных* ситуаций.
- Генерируя исключение, понимать где и кем оно будет обработано.

КОНЕЦ СЕДЬМОЙ ЛЕКЦИИ

```
throw EndOfLecture();
```