

Программирование на языке C++

Лекция 10

Обработка ошибок

Александр Смаль

Логические ошибки и исключительные ситуации

- **Логические ошибки.**

Ошибки в логике работы программы, которые происходят из-за неправильно написанного кода, т.е. это ошибки программиста:

- выход за границу массива,
- попытка деления на ноль,
- обращение по нулевому указателю,
- ...

Логические ошибки и исключительные ситуации

- **Логические ошибки.**

Ошибки в логике работы программы, которые происходят из-за неправильно написанного кода, т.е. это ошибки программиста:

- выход за границу массива,
- попытка деления на ноль,
- обращение по нулевому указателю,
- ...

- **Исключительные ситуации.**

Ситуации, которые требуют особой обработки.

Возникновение таких ситуаций — это „нормальное“ поведение программы.

- ошибка записи на диск,
- недоступность сервера,
- неправильный формат файла,
- ...

Выявление логических ошибок на этапе разработки

- Оператор `static_assert`.

```
#include<type_traits>

template<class T>
void countdown(T start) {
    static_assert(std::is_integral<T>::value
                  && std::is_signed<T>::value,
                  "Requires signed integral type");

    while (start >= 0) {
        std::cout << start-- << std::endl;
    }
}
```

Выявление логических ошибок на этапе разработки

- Оператор `static_assert`.
- Макрос `assert`.

```
#include<type_traits>
// #define NDEBUG
#include <cassert>

template<class T>
void countdown(T start) {
    static_assert(std::is_integral<T>::value
                  && std::is_signed<T>::value,
                  "Requires signed integral type");
    assert(start >= 0);
    while (start >= 0) {
        std::cout << start-- << std::endl;
    }
}
```

Способы сообщения об ошибке

```
size_t write(string file, string data);
```

Способы сообщения об ошибке

```
size_t write(string file, string data);
```

- Возврат статуса операции:

```
bool write(string file, string data, size_t & bytes);
```

Способы сообщения об ошибке

```
size_t write(string file, string data);
```

- Возврат статуса операции:

```
bool write(string file, string data, size_t & bytes);
```

- Возврат кода ошибки:

```
int const OK = 0, IO_WRITE_FAIL = 1, IO_OPEN_FAIL = 2;  
int write(string file, string data, size_t & bytes);
```


Способы сообщения об ошибке

```
size_t write(string file, string data);
```

- Возврат статуса операции:

```
bool write(string file, string data, size_t & bytes);
```

- Возврат кода ошибки:

```
int const OK = 0, IO_WRITE_FAIL = 1, IO_OPEN_FAIL = 2;  
int write(string file, string data, size_t & bytes);
```

- Глобальная переменная для кода ошибки:

```
size_t write(string file, string data);
```

```
size_t bytes = write(f, data);  
if (errno) {  
    cerr << strerror(errno);  
    errno = 0;  
}
```

Способы сообщения об ошибке

```
size_t write(string file, string data);
```

- Возврат статуса операции:

```
bool write(string file, string data, size_t & bytes);
```

- Возврат кода ошибки:

```
int const OK = 0, IO_WRITE_FAIL = 1, IO_OPEN_FAIL = 2;  
int write(string file, string data, size_t & bytes);
```

- Глобальная переменная для кода ошибки:

```
size_t write(string file, string data);
```

```
size_t bytes = write(f, data);  
if (errno) {  
    cerr << strerror(errno);  
    errno = 0;  
}
```

- Исключения.

Исключения

```
size_t write(string file, string data) {  
    if (!open(file)) throw FileOpenError(file);  
    //...  
}  
  
double safediv(int x, int y) {  
    if (y == 0) throw MathError("Division by zero");  
    return double(x) / y;  
}  
  
void write_x_div_y(string file, int x, int y) {  
    try {  
        write(file, to_string(safediv(x, y)));  
    } catch (MathError & s) {  
        // обработка ошибки в safediv  
    } catch (FileError & e) {  
        // обработка ошибки в write  
    } catch (...) {  
        // все остальные ошибки  
    }  
}
```

Stack unwinding

При возникновении исключения объекты на стеке уничтожаются в естественном (обратном) порядке.

```
void foo() {
    D d;
    E e(d);
    if (!e) throw F();
    G g(e);
}

void bar() {
    A a;
    try {
        B b;
        foo();
        C c;
    } catch (F & f) {
        // обработка и пересылка
        throw f;
    }
}
```

Почему не стоит бросать встроенные типы

```
int foo() {  
    if (...) throw -1;  
    if (...) throw 3.1415;  
}  
  
void bar(int a) {  
    if (a == 0) throw string("Not my fault!");  
}  
  
int main () {  
    try { bar(foo());  
    } catch (string & s) {  
        // только текст  
    } catch (int a) {  
        // мало информации  
    } catch (double d) {  
        // мало информации  
    } catch (...) {  
        // нет информации  
    }  
}
```

Стандартные классы исключений

Базовый класс для всех исключений (в <exception>):

```
struct exception {  
    virtual ~exception();  
    virtual const char* what() const;  
};
```

Стандартные классы ошибок (в <stdexcept>):

- logic_error: domain_error, invalid_argument, length_error, out_of_range
- runtime_error: range_error, overflow_error, underflow_error

```
int main() {  
    try { ... }  
    catch (std::exception const& e) {  
        std::cerr << e.what() << '\n';  
    }  
}
```

Исключения в стандартной библиотеке

- Метод `at` контейнеров `array`, `vector`, `deque`, `basic_string`, `bitset`, `map`, `unordered_map` бросает `out_of_range`.
- Оператор `new T` бросает `bad_alloc`.
Оператор `new (std::nothrow) T` возвращает `0`.
- Оператор `typeid` от разыменованного нулевого указателя бросает `bad_typeid`.
- Потоки ввода-вывода.

```
std::ifstream file;  
file.exceptions( std::ifstream::failbit  
                | std::ifstream::badbit );  
  
try {  
    file.open ("test.txt");  
    cout << file.get() << endl;  
    file.close();  
}  
  
catch (std::ifstream::failure const& e) {  
    cerr << e.what() << endl;  
}
```

Как обрабатывать ошибки?

Есть несколько „правил хорошего тона“.

- Разделяйте „ошибки программиста“ и „исключительные ситуации“.
- Используйте `assert` и `static_assert` для выявления ошибок на этапе разработки.
- В пределах одной логической части кода обрабатывайте ошибки централизованно и единообразно.
- Обрабатывайте ошибки там, где их можно обработать.
- Если в данном месте ошибку не обработать, то пересылайте её выше при помощи исключения.
- Бросайте только стандартные классы исключений или производные от них.
- Бросайте исключения по значению, а отлавливайте по ссылке.
- Отлавливайте все исключения в точке входа.