

Обработка исключений

- Исключение - это ошибка которая возникает во время выполнения программы.
- Давайте разделим **обнаружение ошибок** от **обработки ошибок**.

Обнаружение ошибки

```
// Открываем файл и возвращаем содержимое
string readFileToString(const string &filename) {

    // Попытка открыть файл
    ifstream fin(filename);
    if (!fin.is_open()) {
        // ОШИБКА! невозможно открыть
        // файл.
        // Что с этим делать?
    }
    ...
}
```

Игнорировать
и продолжить
работу?

Вывести
сообщение?

Обработка исключений

- Какими есть способы сообщить об ошибке в программе во внешний мир:
 - Глобальные коды ошибок
 - Возврат кода ошибки
 - Throw/Catch Exceptions (механизм исключений)

Глобальные коды ошибки

- Механизм:

1. Хранить код ошибки глобально, затем вернуть.
2. Вызывающая функция/программа получают код ошибки.

Такой подход плохо работает для сложных программ, должно гарантироваться что код ошибки проверяется и программа продолжает работать корректно.

Ошибки состояния объекта

- Если метод класса не отработал корректно, объект переходит в состояние ошибки
- Необходимо проверять состояние объекта после каждой операции с ним.

```
...  
// Попытка открыть файл  
ifstream fin(filename);  
if (!fin.is_open()) {  
    ...  
}  
...
```

Возврат кода ошибки

- Механизм:
 1. Вернуть код ошибки.
 2. Вызывающая функция/программа должна проверить код возврата.
- Лучше чем глобальный код ошибки, так как происходит локально и не может быть изменена извне.
- Однако, необходимо следить за кодом возврата...

Возврат кода ошибки

```
// Возвращает n! для положительных
// и -1 в противном случае.
int factorial(int n) {

    // Проверка
    if (n < 0) {
        return -1;
    }
    ...
}
```

Возврат кода ошибки

```
// Парсит число из строки.  
// Возвращает int. Возвращает ??? в  
// случае ошибки.  
int parseInt(const string &str) {  
    // Проверка  
    if (/*Недопустимый символ*/) {  
        return ???;  
    }  
    ...  
}
```

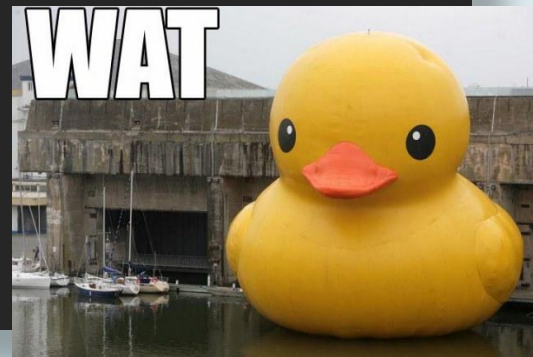
**Что вернуть в случае
ошибки?**

Возврат кода ошибки

```
// Создает Duck. Если что-то пошло не  
// так, возвращает duck WAT
```

```
Duck makeDuck(/*Duck Parameters*/) {  
  
    if (/*ОШИБКА*/) {  
        return Duck("WAT");  
    }  
    ...  
}
```

Как-то странно.



Возврат кода ошибки

- Вызывающая сторона может забыть проверить код возврата.

```
// Возвращает n! для положительных
// и -1 в противном случае.
int factorial(int n);

int main(int n) {

    int x = askUser();
    int f = factorial(x);

    // Использование кода ошибки в вычислениях.
    // Что может случиться??
}
```

Возврат кода ошибки

- Проверка кода ошибки чередуется с проверкой нормально значения.

```
int main() {  
    int x = askUser();  
    int f = factorial(x);  
    if (f < 0) {  
        cout << "ERROR" << endl;  
    }  
    else if (f < 100) {  
        cout << "Small factorial" << endl;  
    }  
    else {  
        cout << "Larger factorial" << endl;  
    }  
}
```

Обработка исключений

- Системные средства, с помощью которых программа может справиться с ошибками времени выполнения

ИСПОЛЬЗОВАНИЕ ИСКЛЮЧЕНИЙ

Код которым
может
обнаружить
ошибку
помещается
в try блок.

```
int main() {  
    int x = askUser();  
    try {  
        int f = factorial(x);  
        if (f < 100) {  
            cout << "Small" << endl;  
        }  
        else {  
            cout << "Larger" << endl;  
        }  
    }  
    catch (FactorialError &e) {  
        cout << "ERROR" << endl;  
    }  
}
```

Код который, выполняется в
случае если произошла
ошибка.

```
class FactorialError { };  
  
// Возвращает n! для  
// положительных  
// Генерирует исключение для  
// отрицательных чисел  
int factorial(int n) {  
  
    // Check for error  
    if (n < 0) {  
        throw FactorialError();  
    }  
    ...  
}
```

Если что-то пошло не
так генерируется
исключения

Использование ИСКЛЮЧЕНИЙ

- Обработка исключений основывается на трех понятиях:
 1. Обнаружение ошибки. Код который необходимо контролировать на предмет ошибки помещается в блок **try**
 2. Выброс исключения **throw**. Сообщение о возникшей ошибке определенного типа
 3. Обработка ошибки. Перехват кода ошибки и обработка ошибки соответствующим образом. Блок **catch**

Выброс исключения `throw`

```
class FactorialError { };

// Возвращает n! для
// положительных
// Генерирует исключение для
// отрицательных чисел
int factorial(int n) {

    if (n < 0) {
        throw FactorialError();
    }
    ...
}
```

- Когда встречается **throw** дальнейшее выполнение кода в блоке прекращается.
- Программа пытается найти соответствующий блок **catch**.
- Тип кода ошибки может быть любым, включая пользовательские типы.
 - например `FactorialError`
- Только один объект может выбросить исключение в один момент времени

try-catch блок

```
int main() {  
    int x = askUser();  
    try {  
        int f = factorial(x);  
        if (f < 100) {  
            cout << "Small" << endl;  
        }  
        else {  
            cout << "Larger" << endl;  
        }  
    }  
    catch (FactorialError &e) {  
        cout << "ERROR" << endl;  
    }  
}
```

- Каждому **try** блоку должен соответствовать один или несколько блоков **catch**.
- Если исключение выбрасывается внутри блока **try**, проверяются соответствующие блоки **catch**
- Если блок **catch** соответствует типу исключения, то выполняется код внутри этого блока
- Если соответствий не находится, исключение выбрасывается выше
- Необработанное исключение == крах программы.



Exercise: average

- Напишите функцию, которая усредняет числа в последовательности, заданной двумя итераторами. Если последовательность пуста, выбросите исключение.

```
class AverageException { };

// РЕЗУЛЬТАТ: Возвращает усредненное значение в
// последовательности [begin, end). Если
// последовательность пустая выбросить исключение типа
// AverageException

template <typename Iter_type>
double average(Iter_type begin, Iter_type end) {

    // РЕАЛИЗАЦИЯ

}
```

Solution: average

```
class AverageException { };

// РЕЗУЛЬТАТ: Возвращает усредненное значение в
// последовательности [begin, end). Если
// последовательность пустая выбросить исключение типа
// AverageException
template <typename Iter_type>
double average(Iter_type begin, Iter_type end) {
    int count = 0;
    double total = 0;
    if (begin == end) {
        throw AverageException();
    }
    while (begin != end) {
        ++count;
        total += *begin;
        ++begin;
    }
    return total / count;
}
```

catch параметры

- catch блок выполняется когда тип исключения совпадает с типом параметра блока **catch**
- **catch-by-value** или **catch-by-reference**.
- Неявные преобразования недопустимы, но полиморфизм работает

```
int main() {  
    try {  
        throw 2;  
    }  
    catch (int e) { ... }  
}
```

Типы int.

```
int main() {  
    try {  
        throw 2;  
    }  
    catch (double e) { ... }  
}
```

Неявное преобразование недопустимо.

```
int main() {  
    try {  
        throw Gorilla();  
    }  
    catch (Gorilla &e) {  
        ...  
    }  
}
```

Норм,
передача по
ссылке.

```
int main() {  
    try {  
        throw Duck();  
    }  
    catch (Bird &e) {  
        ...  
    }  
}
```

Полиморфизм!

Несколько catch блоков

- try блок может иметь несколько catch блоков.
- Будет использоваться тот блок, для которого тип совпадет
- Используйте "..." для того чтобы обработать любой тип.

```
int main() {  
    try {  
        if (/*something*/) {  
            throw 4;  
        }  
        if (/*something*/) {  
            throw 2.0;  
        }  
        if (/*something*/) {  
            throw 'a';  
        }  
        if (/*something*/) {  
            throw false;  
        }  
    }  
    catch (int x) { }  
    catch (double d) { }  
    catch (char c) { }  
    catch (...) { }  
}
```



21

Exercise: Exceptions 1

- ЧТО ВЫВЕДЕТ ЭТОТ КОД?

```
class GoodbyeError { };  
void goodbye() {  
    cout << "goodbye called\n";  
    GoodbyeError e; throw e;  
    cout << "goodbye returns\n";  
}
```

```
class HelloError { };  
void hello() {  
    cout << "hello called\n";  
    goodbye();  
    cout << "hello returns\n";  
}
```

```
int main() {  
    try {  
        hello();  
        cout << "done\n";  
    }  
    catch (HelloError &he) {  
        cout << "caught hello\n";  
    }  
    catch (GoodbyeError &ge) {  
        cout << "caught goodbye\n";  
    }  
    cout << "main returns\n";  
}
```

Solution: Exceptions 1

- ЧТО ВЫВЕДЕТ ЭТОТ КОД?

```
class GoodbyeError { };  
void goodbye() {  
    cout << "goodbye called\n";  
    GoodbyeError e; throw e;  
    cout << "goodbye returns\n";  
}
```

```
class HelloError { };  
void hello() {  
    cout << "hello called\n";  
    goodbye();  
    cout << "hello returns\n";  
}
```

```
int main() {  
    try {  
        hello();  
        cout << "done\n";  
    }  
    catch (HelloError &he) {  
        cout << "caught hello\n";  
    }  
    catch (GoodbyeError &ge) {  
        cout << "caught goodbye\n";  
    }  
    cout << "main returns\n";  
}
```

```
hello called  
goodbye called  
caught goodbye  
main returns
```



23

Exercise: Exceptions 2

- ЧТО ВЫВЕДЕТ ЭТОТ КОД?

```
class GoodbyeError { };  
void goodbye() {  
    cout << "goodbye called\n";  
    GoodbyeError e; throw e;  
    cout << "goodbye returns\n";  
}
```

```
class HelloError { };  
void hello() {  
    cout << "hello called\n";  
    try { goodbye(); }  
    catch (GoodbyeError &ge) {  
        throw HelloError();  
    }  
    cout << "hello returns\n";  
}
```

```
int main() {  
    try {  
        hello();  
        cout << "done\n";  
    }  
    catch (HelloError &he) {  
        cout << "caught hello\n";  
    }  
    catch (GoodbyeError &ge) {  
        cout << "caught goodbye\n";  
    }  
    cout << "main returns\n";  
}
```

Solution: Exceptions 2

- ЧТО ВЫВЕДЕТ ЭТОТ КОД?

```
class GoodbyeError { };  
void goodbye() {  
    cout << "goodbye called\n";  
    GoodbyeError e; throw e;  
    cout << "goodbye returns\n";  
}
```

```
class HelloError { };  
void hello() {  
    cout << "hello called\n";  
    try { goodbye(); }  
    catch (GoodbyeError &ge) {  
        throw HelloError();  
    }  
    cout << "hello returns\n";  
}
```

```
int main() {  
    try {  
        hello();  
        cout << "done\n";  
    }  
    catch (HelloError &he) {  
        cout << "caught hello\n";  
    }  
    catch (GoodbyeError &ge) {  
        cout << "caught goodbye\n";  
    }  
    cout << "main returns\n";  
}
```

```
hello called  
goodbye called  
caught hello  
main returns
```




25

Exercise: Exceptions 3

- ЧТО ВЫВЕДЕТ ЭТОТ КОД?

```
class Error {  
    string msg;  
public:  
    Error(const string &s) : msg(s) { }  
    const string &get_msg() { return msg; }  
};
```

```
void goodbye() {  
    cout << "goodbye called\n";  
    throw Error("bye");  
    cout << "goodbye returns\n";  
}
```

```
void hello() {  
    cout << "hello called\n";  
    try { goodbye(); }  
    catch (Error &e) { throw Error("hey"); }  
    cout << "hello returns\n";  
}
```

```
int main() {  
    try {  
        hello();  
        cout << "done\n";  
    }  
    catch (Error &e) {  
        cout << e.get_msg();  
        cout << endl;  
    }  
    catch (...) {  
        cout << "unknown error\n";  
    }  
    cout << "main returns\n";  
}
```

Solution: Exceptions 3

- ЧТО ВЫВЕДЕТ ЭТОТ КОД?

```
class Error {  
    string msg;  
public:  
    Error(const string &s) : msg(s) { }  
    const string &get_msg() { return msg; }  
};
```

```
void goodbye() {  
    cout << "goodbye called\n";  
    throw Error("bye");  
    cout << "goodbye returns\n";  
}
```

```
void hello() {  
    cout << "hello called\n";  
    try { goodbye(); }  
    catch (Error &e) { throw Error("hey"); }  
    cout << "hello returns\n";  
}
```

```
int main() {  
    try {  
        hello();  
        cout << "done\n";  
    }  
    catch (Error &e) {  
        cout << e.get_msg();  
        cout << endl;  
    }  
    catch (...) {  
        cout << "unknown error\n";  
    }  
    cout << "main returns\n";  
}
```

```
hello called  
goodbye called  
hey  
main returns
```



27

Exercise: Exceptions 4

- Что выведет этот код?

```
class Error {  
    string msg;  
public:  
    Error(const string &s) : msg(s) { }  
    const string &get_msg() { return msg; }  
};
```

```
void goodbye() {  
    cout << "goodbye called\n";  
    throw GoodbyeError();  
    cout << "goodbye returns\n";  
}
```

```
void hello() {  
    cout << "hello called\n";  
    try { goodbye(); }  
    catch (Error &e) { throw Error("hey"); }  
    cout << "hello returns\n";  
}
```

```
int main() {  
    try {  
        hello();  
        cout << "done\n";  
    }  
    catch (Error &e) {  
        cout << e.get_msg();  
        cout << endl;  
    }  
    catch (...) {  
        cout << "unknown error\n";  
    }  
    cout << "main returns\n";  
}
```

Solution: Exceptions 4

- ЧТО ВЫВЕДЕТ ЭТОТ КОД?

```
class Error {  
    string msg;  
public:  
    Error(const string &s) : msg(s) { }  
    const string &get_msg() { return msg; }  
};
```

```
void goodbye() {  
    cout << "goodbye called\n";  
    throw GoodbyeError();  
    cout << "goodbye returns\n";  
}
```

```
void hello() {  
    cout << "hello called\n";  
    try { goodbye(); }  
    catch (Error &e) { throw Error("hey"); }  
    cout << "hello returns\n";  
}
```

```
int main() {  
    try {  
        hello();  
        cout << "done\n";  
    }  
    catch (Error &e) {  
        cout << e.get_msg();  
        cout << endl;  
    }  
    catch (...) {  
        cout << "unknown error\n";  
    }  
    cout << "main returns\n";  
}
```

```
hello called  
goodbye called  
unknown error  
main returns
```