Практическая работа по теме "Обработка ошибок в C++: throw, try, catch"

Задача 1: Деление на ноль

Напишите функцию `divide`, которая принимает два числа и возвращает результат деления. Если делитель равен нулю, бросьте исключение `std::runtime_error`.

```cpp
#include <iostream>
#include <stdexcept>

double divide(int a, int b) {
    // Ваш код здесь
}

int main() {
    try {
        std::cout << divide(10, 0) << std::endl;
    } catch (const std::runtime_error& e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}
```

Задача 2: Вектор и выход за пределы

Напишите функцию `getElement`, которая возвращает элемент вектора по индексу. Если индекс выходит за пределы вектора, бросьте исключение `std::out_of_range`.

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

int getElement(const std::vector<int>& vec, size_t index) {
    // Ваш код здесь
}

int main() {
    std::vector<int> vec = {1, 2, 3};
    try {
        std::cout << getElement(vec, 5) << std::endl;
    } catch (const std::out_of_range& e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}
```

Задача 3: Пользовательское исключение

Создайте пользовательское исключение NegativeNumberException, которое будет бросаться, если число отрицательное. Напишите функцию sqrt, которая вычисляет квадратный корень числа и бросает это исключение, если число отрицательное.

```
#include <iostream>
#include <cmath>
#include <stdexcept>

class NegativeNumberException : public std::exception {
    // Ваш код здесь
};

double sqrt(double x) {
    // Ваш код здесь
}

int main() {
    try {
        std::cout << sqrt(-1) << std::endl;
    } catch (const NegativeNumberException& e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}
```

Задача 4: Множественные исключения

Напишите функцию processNumber, которая принимает число. Если число меньше 0, бросьте исключение NegativeNumberException. Если число больше 100, бросьте исключение std::out_of_range.

```
#include <iostream>
#include <stdexcept>

class NegativeNumberException : public std::exception {
    // Ваш код здесь
};

void processNumber(int x) {
    // Ваш код здесь
}

int main() {
    try {
        processNumber(-5);
    } catch (const NegativeNumberException& e) {
        std::cerr << e.what() << std::endl;
    } catch (const std::out_of_range& e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}
```

Задача 5: Обработка всех исключений

Напишите программу, которая ловит все исключения с помощью блока `catch (...)`. Бросьте исключение любого типа и обработайте его.

```
#include <iostream>

void riskyFunction() {
    // Ваш код здесь (бросьте исключение)
}

int main() {
    try {
        riskyFunction();
    } catch (...) {
        std::cerr << "Поймано неизвестное исключение!" << std::endl;
    }
    return 0;
}
```

Задача 6: Исключение в конструкторе

Создайте класс `Rectangle`, который принимает ширину и высоту в конструкторе. Если ширина или высота отрицательные, бросьте исключение `std::invalid_argument`.

```
#include <iostream>
#include <stdexcept>

class Rectangle {
public:
    Rectangle(int width, int height) {
        // Ваш код здесь
    }
};

int main() {
    try {
        Rectangle rect(-10, 20);
    } catch (const std::invalid_argument& e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}
```

Задача 7: Исключение в деструкторе

Создайте класс `Resource`, который в деструкторе бросает исключение. Объясните, почему это плохая практика.

```
#include <iostream>
#include <stdexcept>

class Resource {
public:
    ~Resource() {
        // Ваш код здесь (бросьте исключение)
    }
```

```
};

int main() {
    try {
        Resource res;
    } catch (const std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}
```

Задача 8: Исключение в функции

Создайте класс Calculator с методом add, который складывает два числа. Если результат сложения превышает 1000, бросьте исключение std::overflow_error.

```
#include <iostream>
#include <stdexcept>

class Calculator {
public:
    int add(int a, int b) {
        // Ваш код здесь
    }
};

int main() {
    Calculator calc;
    try {
        std::cout << calc.add(500, 600) << std::endl;
    } catch (const std::overflow_error& e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}
```

Задача 9: Исключение в операторе new

Напишите программу, которая пытается выделить большой объем памяти с помощью оператора new. Если выделение памяти не удалось, поймайте исключение std::bad_alloc.

```
#include <iostream>
#include <new>

int main() {
    try {
        // Ваш код здесь (попытка выделить много памяти)
    } catch (const std::bad_alloc& e) {
        std::cerr << "Ошибка выделения памяти: " << e.what() << std::endl;
    }
    return 0;
}
```

Задача 10: Исключение в лямбда-функции

Напишите лямбда-функцию, которая бросает исключение, если переданное ей число равно нулю. Вызовите эту функцию и обработайте исключение.

```
#include <iostream>
#include <stdexcept>

int main() {
    auto checkZero = [](int x) {
        // Ваш код здесь
    };

    try {
        checkZero(0);
    } catch (const std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
    return 0;
}
```
