Пермский филиал федерального государственного автономного образовательного учреждения высшего образования  
«Национальный исследовательский университет   
Высшая школа экономики»

*Факультет социально-экономических и компьютерных наук*

Мусихин Данил Михайлович

**Обработка исключений**

*Лабораторная работа №3*

студента образовательной программы «Разработка информационных систем для бизнеса» по направлению подготовки *09.03.04 Программная инженерия*

Руководитель, к.т.н., Доцент кафедры ИТБ.

Л.Н. Лядова

Пермь, 2023 год

**Оглавление**

[1 Постановка задачи 4](#_Toc152166223)

[2 Анализ 5](#_Toc152166224)

[3 Код 9](#_Toc152166241)

# Постановка задачи

*Цель работы* – изучение возможностей обработки исключений в VS.

При выполнении работы необходимо рассмотреть исключения:

1. При работе с целочисленными данными.
2. При работе с вещественными числами.
3. При работе с динамическими структурами данных.

Для каждой категории исключений определить (и описать в отчёте):

* какие исключения распознаются и возбуждаются процессором, к какой категории они относятся (нарушения, ловушки);
* какие исключения можно активизировать и обрабатывать, используя средства системы программирования (VS C++, C#);
* какие исключения и какими способами можно обработать, используя SEH,
* какие возможности имеются для определения и обработки исключений пользователей.

Для описанных средств привести примеры их использования для обработки исключений различных типов (нарушений, ловушек, пользовательских исключений для операций с целыми и вещественными числами, динамическими структурами данных) различными способами, используя средства систем программирования и операционной системы (SEH).

Исходные коды программ с обработкой исключений нужно включить в отчёт.

Текст отчёта нужно структурировать – выделить заголовки: 1–3 – описания исключений при работе с данными перечисленных типов, исключения и средства их обработки, примеры описать для каждой категории данных. Описания можно оформить с помощью таблиц, включить в текст ссылки на источники, справочные материалы. Выполнения программ с обработкой исключений необходимо проиллюстрировать скриншотами, показывающими порядок обработки исключений.

# Анализ

## При работе с целочисленными данными

### Процессорные исключения:

Деление на ноль: Процессор может сгенерировать исключение при попытке деления на ноль.

Переполнение: Процессор может генерировать исключение при арифметическом переполнении.

### Обработка в C++

try, catch, throw: Используются стандартные механизмы C++, такие как блоки try, catch и оператор throw.

std::overflow\_error, std::divide\_by\_zero: Можно использовать стандартные исключения C++ для переполнения (std::overflow\_error) и деления на ноль (std::divide\_by\_zero).

### SEH в C++

try, \_\_except: Для более низкоуровневой обработки исключений, можно использовать Structured Exception Handling (SEH) с ключевыми словами \_\_try и \_\_except.

Полную информацию можно посмотреть в официальной документации MircoSoft [2]

### Исключения пользователей в C++

Создание пользовательских исключений путем наследования от стандартных классов (например, std::exception).

## При работе с вещественными числами:

### Процессорные исключения

Деление на ноль: Процессор может сгенерировать исключение при попытке деления на ноль.

Потеря точности: Вещественные числа могут иметь потерю точности при выполнении арифметических операций.

### Обработка в C++

try, catch, throw: Используются стандартные механизмы C++, такие как блоки try, catch и оператор throw.

std::overflow\_error, std::divide\_by\_zero: Можно использовать стандартные исключения C++ для переполнения (std::overflow\_error) и деления на ноль (std::divide\_by\_zero).

std::underflow\_error: Для случаев недостаточной точности.

### SEH в C++

\_\_try, \_\_except: Для более низкоуровневой обработки исключений, можно использовать Structured Exception Handling (SEH) с ключевыми словами \_\_try и \_\_except.

### Исключения пользователей в C++

Создание пользовательских исключений путем наследования от стандартных классов (например, std::exception).

## При работе с динамическими структурами данных

### Процессорные исключения

Процессор не генерирует специфических исключений для динамических структур данных.

### Обработка в C++

try, catch, throw: Используются стандартные механизмы C++ для обработки исключений.

std::bad\_alloc: Для обработки исключений, связанных с нехваткой памяти при динамическом выделении.

### SEH в C++

\_\_try, \_\_except: Для более низкоуровневой обработки исключений, можно использовать Structured Exception Handling (SEH) с ключевыми словами \_\_try и \_\_except.

Не специфично для динамических структур данных.

### Исключения пользователей в C++

Создание пользовательских исключений путем наследования от стандартных классов (например, std::exception). Можно определить свои исключения для ошибок в работе с динамическими структурами данных.

## Обработка исключений в C#

В C# обработка исключений является важной частью парадигмы управления ошибками. В языке предусмотрены конструкции для создания, генерации и обработки исключений. Ключевые элементы в этом контексте включают в себя блоки try, catch, finally и ключевое слово throw.

## Стандартные исключения С++

Основные из них:

* **runtime\_error**: общий тип исключений, которые возникают во время выполнения
* **range\_error**: исключение, которое возникает, когда полученный результат превосходит допустимый диапазон
* **overflow\_error**: исключение, которое возникает, если полученный результат превышает допустимый диапазон
* **underflow\_error**: исключение, которое возникает, если полученный в вычислениях результат имеет недопустимое отрицательное значение (выход за нижнюю допустимую границу значений)
* **logic\_error**: исключение, которое возникает при наличии логических ошибок к коде программы
* **domain\_error**: исключение, которое возникает, если для некоторого значения, передаваемого в функцию, не определен результат
* **invalid\_argument**: исключение, которое возникает при передаче в функцию некорректного аргумента
* **length\_error**: исключение, которое возникает при попытке создать объект большего размера, чем допустим для данного типа
* **out\_of\_range**: исключение, которое возникает при попытке доступа к элементам вне допустимого диапазона

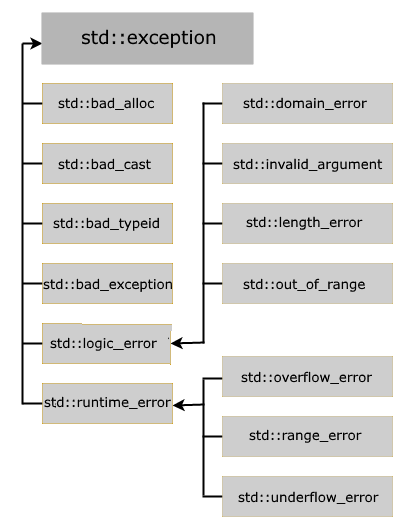


Рисунок - Полная схема стандартных исключений С++ [1]

# Код

## Код на С++

// РАБОТА С ИСКЛЮЧЕНИЯМИ

#include <iostream>

#include <excpt.h>

#include <limits>

#include <cmath>

#include <vector>

using namespace std;

// целочисленная арифметика

void integer\_arithmentic()

{

try

{

short result = 32767;

if (result == numeric\_limits<short>::max())

throw exception("Сложение с максимальным числом.");

result += 1;

}

catch (const exception& e)

{

cerr << "Переполнение целого числа: " << e.what() << endl;

}

try

{

int denomitator = 0;

if (denomitator == 0)

throw exception("Деление на ноль.");

int result = 42 / denomitator;

}

catch (const exception& e)

{

cerr << "Деление целого числа на ноль: " << e.what() << endl;

}

}

void integer\_arithmentic\_SEH()

{

\_\_try

{

short result = 32767;

if (result == numeric\_limits<short>::max())

throw exception("Сложение с максимальным числом.");

result += 1;

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER)

{

cerr << "Переполнение целого числа." << endl;

}

\_\_try

{

int denomitator = 0;

int result = 42 / denomitator;

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER)

{

cerr << "Деление целого числа на ноль." << endl;

}

}

// работа с плавающей точкой

void double\_arithmetic()

{

try

{

int a = -1;

if (a < 0)

throw exception("Вычисление корня из отрицательного числа");

double result = sqrt(-1);

}

catch (const exception& e)

{

cerr << "Ошибка при вычислении корня: " << e.what() << endl;

}

try

{

int denomitator = 0.0;

if (denomitator == 0.0)

throw exception("Деление на ноль.");

int result = 42.0 / denomitator;

}

catch (const exception& e)

{

cerr << "Деление вещественного числа на ноль: " << e.what() << endl;

}

}

void double\_arithmetic\_SEH()

{

\_\_try

{

int a = -1;

if (a < 0)

throw exception("Вычисление корня из отрицательного числа");

double result = sqrt(-1);

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER)

{

cerr << "Ошибка при вычислении корня." << endl;

}

\_\_try

{

int denomitator = 0.0;

if (denomitator == 0.0)

throw exception("Деление на ноль.");

int result = 42.0 / denomitator;

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER)

{

cerr << "Деление вещественного числа на ноль."<< endl;

}

}

// работа с динамикой

void work\_with\_dinamic()

{

try

{

int\* ptr = nullptr;

if (ptr == nullptr)

throw exception("Пустая ссылка.");

int value = \*ptr;

}

catch (const exception& e)

{

cerr << "Ошибка при обращении к пустому указателю: " << e.what() << endl;

}

try

{

vector<int> v;

int value2 = v.at(15);

}

catch (const out\_of\_range& e)

{

cerr << "Обращение за пределы массива: " << e.what() << endl;

}

vector<int> vec;

try {

while (true) {

vec.push\_back(1);

}

}

catch (const std::bad\_alloc& e) {

std::cout << "Место закончилось после вставки " << vec.size() << " элементов" << std::endl;

}

}

void work\_with\_dinamic\_SEH()

{

\_\_try

{

int\* ptr = nullptr;

int value = \*ptr;

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER)

{

cerr << "Ошибка при обращении к пустому указателю." << endl;

}

/\*\_\_try

{

vector<int> v;

int value2 = v[15];

}

\_\_except (EXCEPTION\_EXECUTE\_HANDLER)

{

cerr << "Обращение за пределы динамического массива." << endl;

}\*/

}

int main()

{

setlocale(LC\_ALL, "");

integer\_arithmentic();

double\_arithmetic();

work\_with\_dinamic();

/\*integer\_arithmentic\_SEH();

double\_arithmetic\_SEH();

work\_with\_dinamic\_SEH();\*/

}

## Код на С#

using System;

using System.Collections.Generic;

class Program

{

// Целочисленная арифметика

static void IntegerArithmetic()

{

try

{

short result = 32767;

if (result == short.MaxValue)

throw new Exception("сложение с максимальным числом.");

result += 1;

}

catch (Exception e)

{

Console.WriteLine($"Переполнение целого числа: {e.Message}");

}

try

{

int denominator = 0;

//if (denominator == 0)

// throw new Exception("Деление на ноль.");

int result = 42 / denominator;

}

catch (Exception e)

{

Console.WriteLine($"Деление целого числа на ноль: {e.Message}");

}

}

// Работа с плавающей точкой

static void DoubleArithmetic()

{

try

{

int a = -1;

if (a < 0)

throw new Exception("Вычисление корня из отрицательного числа");

double result = Math.Sqrt(-1);

}

catch (Exception e)

{

Console.WriteLine($"Ошибка при вычислении корня: {e.Message}");

}

try

{

double denominator = 0.0;

if (denominator == 0.0)

throw new Exception("Деление на ноль.");

double result = 42.0 / denominator;

}

catch (Exception e)

{

Console.WriteLine($"Деление вещественного числа на ноль: {e.Message}");

}

}

// Работа с динамикой

static void WorkWithDynamic()

{

try

{

int[] array = null;

//if (array == null)

// throw new Exception("Пустая ссылка.");

int value = array[0];

}

catch (Exception e)

{

Console.WriteLine($"Ошибка при обращении к пустому массиву: {e.Message}");

}

try

{

List<int> list = new List<int>();

int value2 = list[15];

}

catch (ArgumentOutOfRangeException e)

{

Console.WriteLine($"Обращение за пределы списка: {e.Message}");

}

var list = new System.Collections.Generic.List<int>();

try

{

while (true)

{

list.Add(1);

}

}

catch (OutOfMemoryException ex)

{

Console.WriteLine($"Memory ran out after inserting {list.Count} elements");

}

}

static void Main()

{

// Установка культуры

System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("ru-RU");

IntegerArithmetic();

DoubleArithmetic();

WorkWithDynamic();

}

}

Обработка стандартных исключений на С++  
#include <iostream>

#include <stdexcept>

#include <vector>

#include <limits>

// runtime\_error: общий тип исключений, которые возникают во время выполнения

// В этом примере divideNumbers проверяет, равен ли делитель нулю, и, если да, бросает исключение std::runtime\_error.

double divideNumbers(double numerator, double denominator) {

    if (denominator == 0) {

        throw std::runtime\_error("Division by zero is undefined");

    }

    return numerator / denominator;

}

int main() {

    try {

        double result = divideNumbers(10.0, 2.0);

        std::cout << "Result: " << result << std::endl;

        // Попытаемся делить на ноль

        result = divideNumbers(5.0, 0.0);

        std::cout << "Result: " << result << std::endl;  // Эта строка не выполнится

    }

    catch (const std::runtime\_error& e) {

        std::cerr << "Exception caught: " << e.what() << std::endl;

    }

    return 0;

}

// range\_error: исключение, которое возникает, когда полученный результат превосходит допустимый диапазон

// В этом примере функция getElement получает элемент из массива по заданному индексу.

// Если индекс находится вне допустимого диапазона (отрицательный или больше или равен размеру массива), функция бросает исключение std::range\_error.

int getElement(const int array[], int size, int index) {

    if (index < 0 || index >= size) {

        throw std::range\_error("Index out of range");

    }

    return array[index];

}

int main() {

    int myArray[] = {1, 2, 3, 4, 5};

    try {

        int element = getElement(myArray, 5, 2);

        std::cout << "Element at index 2: " << element << std::endl;

        // Попытаемся получить элемент с индексом за пределами массива

        element = getElement(myArray, 5, 10);

        std::cout << "This line won't be executed" << std::endl;

    } catch (const std::range\_error &e) {

        std::cerr << "Exception caught: " << e.what() << std::endl;

    }

    return 0;

}

// overflow\_error: исключение, которое возникает, если полученный результат превышает допустимый диапазон

// В этом примере функция multiply умножает два числа и проверяет на переполнение.

// Если умножение приводит к переполнению, бросается исключение std::overflow\_error.

int multiply(int a, int b) {

    // Проверка на переполнение

    if ((b > 0 && a > INT\_MAX / b) || (b < 0 && a < INT\_MIN / b)) {

        throw std::overflow\_error("Multiplication overflow");

    }

    return a \* b;

}

int main() {

    try {

        int result = multiply(5, 2);

        std::cout << "Result of multiplication: " << result << std::endl;

        // Попытаемся вызвать переполнение при умножении

        result = multiply(INT\_MAX, 2);

        std::cout << "This line won't be executed" << std::endl;

    } catch (const std::overflow\_error &e) {

        std::cerr << "Exception caught: " << e.what() << std::endl;

    }

    return 0;

}

// underflow\_error: исключение, которое возникает, если полученный в вычислениях результат имеет недопустимое отрицательное значение (выход за нижнюю допустимую границу значений)

double divideSafely(double numerator, double denominator) {

    // Проверка на переполнение

    if (denominator == 0.0) {

        throw std::runtime\_error("Division by zero");

    }

    if ((numerator > 0 && denominator < std::numeric\_limits<double>::min() / numerator) ||

        (numerator < 0 && denominator > std::numeric\_limits<double>::min() / numerator)) {

        throw std::underflow\_error("Division underflow");

    }

    return numerator / denominator;

}

int main() {

    try {

        double result = divideSafely(10.0, 2.0);

        std::cout << "Result of division: " << result << std::endl;

        // Попытаемся вызвать underflow при делении

        result = divideSafely(1.0, std::numeric\_limits<double>::min());

        std::cout << "This line won't be executed" << std::endl;

    } catch (const std::underflow\_error &e) {

        std::cerr << "Exception caught: " << e.what() << std::endl;

    } catch (const std::runtime\_error &e) {

        std::cerr << "Exception caught: " << e.what() << std::endl;

    }

    return 0;

}

// logic\_error: исключение, которое возникает при наличии логических ошибок к коде программы

// В этом примере функция performOperation проверяет, является ли переданное значение отрицательным, и если это так, бросает исключение std::logic\_error

void performOperation(int value) {

    if (value < 0) {

        throw std::logic\_error("Negative value is not allowed");

    }

    // выполнение операции, если значение положительное

}

int main() {

    try {

        performOperation(10); // Операция будет выполнена

        performOperation(-5); // Бросит исключение std::logic\_error

    } catch (const std::logic\_error &e) {

        std::cerr << "Logic error: " << e.what() << std::endl;

    }

    return 0;

}

// domain\_error: исключение, которое возникает, если для некоторого значения, передаваемого в функцию, не определен результат

// В этом примере функция calculateSquareRoot вычисляет квадратный корень из переданного значения. Если значение отрицательное, она бросает исключение std::domain\_error.

double calculateSquareRoot(double value) {

    if (value < 0) {

        throw std::domain\_error("Cannot calculate square root of a negative number");

    }

    return sqrt(value);

}

int main() {

    try {

        double result = calculateSquareRoot(25.0); // Вычислит корень из 25

        std::cout << "Result: " << result << std::endl;

        result = calculateSquareRoot(-10.0); // Бросит исключение std::domain\_error

    } catch (const std::domain\_error &e) {

        std::cerr << "Domain error: " << e.what() << std::endl;

    }

    return 0;

}

// invalid\_argument: исключение, которое возникает при передаче в функцию некорректного аргумента

// В этом примере функция processString обрабатывает строку и бросает исключение std::invalid\_argument, если строка пуста.

void processString(const std::string& str) {

    if (str.empty()) {

        throw std::invalid\_argument("Input string cannot be empty");

    }

    // Обработка строки, если она не пуста

    std::cout << str << std::endl;

}

int main() {

    try {

        processString("Hello"); // Обработает непустую строку

        processString("");      // Бросит исключение std::invalid\_argument

    } catch (const std::invalid\_argument& e) {

        std::cerr << "Invalid argument: " << e.what() << std::endl;

    }

    return 0;

}

// length\_error: исключение, которое возникает при попытке создать объект большего размера, чем допустим для данного типа

// В этом примере функция processVector обрабатывает вектор и бросает исключение std::length\_error, если размер вектора превышает максимально допустимый.

void processVector(const std::vector<int>& vec) {

    if (vec.size() > std::numeric\_limits<std::size\_t>::max()) {

        throw std::length\_error("Vector size exceeds maximum allowed length");

    }

    // Обработка вектора, если его размер допустим

    std::cout << vec.size() << std::endl;

}

int main() {

    try {

        std::vector<int> myVector(1000000); // Создаст вектор размером 1000000

        processVector(myVector);

        std::vector<int> largeVector(std::numeric\_limits<std::size\_t>::max() - 1);

        processVector(largeVector); // Бросит исключение std::length\_error

    } catch (const std::length\_error& e) {

        std::cerr << "Length error: " << e.what() << std::endl;

    }

    return 0;

}

// out\_of\_range: исключение, которое возникает при попытке доступа к элементам вне допустимого диапазона

// В этом примере функция getElement получает элемент из вектора по заданному индексу.

// Если индекс находится вне допустимого диапазона (равен или больше размера вектора), функция бросает исключение std::out\_of\_range.

int getElement(const std::vector<int>& vec, std::size\_t index) {

    if (index >= vec.size()) {

        throw std::out\_of\_range("Index is out of range");

    }

    return vec[index];

}

int main() {

    try {

        std::vector<int> myVector = {1, 2, 3, 4, 5};

        int element = getElement(myVector, 2); // Получит элемент с индексом 2

        std::cout << "Element at index 2: " << element << std::endl;

        element = getElement(myVector, 10); // Бросит исключение std::out\_of\_range

    } catch (const std::out\_of\_range& e) {

        std::cerr << "Out of range error: " << e.what() << std::endl;

    }

    return 0;

}

// bad\_alloc - исключение выбрасываемое при ошибках с динамическими структурами данных

// Когда в нашем массиве закончится место для вставки вызовется исключение bad\_alloc.

int main() {

    std::vector<int> vec;

    try {

        while (true) {

            vec.push\_back(1);

        }

    }

    catch (const std::bad\_alloc& e) {

        std::cout << "Место закончилось после вставки " << vec.size() << " элементов" << std::endl;

    }

}

# Приложение А. Библиографический список

1. Сайт с информацией по исключениям в С++ [Электронный ресурс] // <https://metanit.com/cpp/tutorial/6.3.php>
2. Официальная документация Microsoft по SHE [Электронный ресурс] // <https://learn.microsoft.com/en-us/cpp/cpp/structured-exception-handling-c-cpp?view=msvc-170>

# Приложение Б. Скриншоты работы программ

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок - Обработка исключений в С#

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок - Обработка исключений С++