**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

Лабораторная работа № 8

по ОБЪЕКТНО-ОРИЕНТИРОВАННОМУ ПРОГРАММИРОВАНИЮ

«Обработка исключений»

Вариант 11

Студент \_\_\_\_\_\_\_\_\_\_\_ Целищев А.Е.

Группа ПМ-21-2

Руководитель

Доцент \_\_\_\_\_\_\_\_\_\_\_ Кургасов В.В.

Липецк 2022г.

**Оглавление:**

[Цель работы: 2](#_Toc114052946)

[Задание кафедры: 3](#_Toc114052947)

[Код программы: 4](#_Toc114052948)

[Пример выполнения: 10](#_Toc114052949)

[Вывод: 12](#_Toc114052950)

[Ответы на контрольные вопросы: 13](#_Toc114052951)

# **Цель работы:**

Научиться использовать встроенное в язык средство обработки исключительных ситуаций для предотвращения аварийного завершения работы программы в случае ошибок в данных.

# ***Задание кафедры:***

Для класса, полученного в результате лабораторной работы №2, перегрузить еще как минимум три операции. Создать собственную иерархию класса обработки исключений. В ситуациях, при которых может возникать ошибка, генерировать собственное исключение и его обрабатывать. В программе должны быть использованы блоки обработки исключений: try, catch. В лабораторной работе описать все обрабатываемые в программе исключительные ситуации.

# 

# ***Код программы:***

#include <iostream>

#include <typeinfo>

#include <limits>

#include <vector>

**using** **namespace** std**;**

// класс BaseException унаследован от класса exception

class base\_exception**:** public exception **{**

protected**:**

string text**;** // объяснение к исключению - общее для всех производных классов

public**:**

virtual string what**()** **=** 0**;** // чистая виртуальная функция, которая выводит текст исключения

**};**

//класс, обрабатывающий исключение при вводе ошибочной размерности матрицы

class wrong\_dimension final **:**public base\_exception **{**

public**:**

//конструктор по умолчанию

wrong\_dimension**()** **{**

text **=** "Ошибка. Неправильно введена размерность!"**;**

**}**

//конструктор с заданным текстом

wrong\_dimension**(**string \_text**)** **{**

text **=** \_text**;**

**}**

//переопределяем виртуальную функцию what()

string what**()** **{**

**return** text**;**

**}**

**};**

class int\_exception final **:**public base\_exception **{**

public**:**

//конструктор по умолчанию

int\_exception**()** **{**

text **=** "Ошибка. Тип данных не целочисленный!"**;**

**}**

//конструктор с заданным текстом

int\_exception**(**string \_text**)** **{**

text **=** \_text**;**

**}**

//переопределяем виртуальную функцию what()

string what**()** **{**

**return** text**;**

**}**

**};**

//начало объявления класса

class Matrix **{**

//объявление свойств и методов класса в общедоступной области видимости

public**:**

//конструктор по умолчанию

Matrix**()** **{**

rows **=** 0**;**

cols **=** 0**;**

nums **=** **NULL;**

**}**

//конструктор копирования

Matrix**(**const Matrix**&** A**)** **{**

rows **=** A**.**rows**;**

cols **=** A**.**cols**;**

nums **=** **new** int**[**A**.**rows **\*** A**.**cols**];**

**for** **(**int i **=** 0**;** i **<** A**.**rows **\*** A**.**cols**;** i**++)** **{**

nums**[**i**]** **=** A**.**nums**[**i**];**

**}**

**}**

//конструктор c параметрами

Matrix**(**int **\***values**,** int m**,** int n**)** **{**

rows **=** m**;**

cols **=** n**;**

nums **=** **new** int**[**rows **\*** cols**];**

**for** **(**int i **=** 0**;** i **<** rows **\*** cols**;** i**++)** **{**

nums**[**i**]** **=** values**[**i**];**

**}**

**}**

//деструктор

**~**Matrix**()** **{**

**if** **(**nums **!=** **NULL)** **{**

**delete** nums**;**

**}**

**}**

//перегрузка оператора присваивания

const Matrix**&** **operator** **=** **(**const Matrix**&** A**)** **{**

rows **=** A**.**rows**;**

cols **=** A**.**cols**;**

**if** **(**nums **!=** **NULL)** **{**

**delete** nums**;**

**}**

nums **=** **new** int**[**rows **\*** cols**];**

**for** **(**int i **=** 0**;** i **<** rows **\*** cols**;** i**++)** **{**

nums**[**i**]** **=** A**.**nums**[**i**];**

**}**

**return** **\*this;**

**}**

//перегрузка оператора умножения на скаляр через дружественную функцию

friend Matrix **operator** **\*** **(**const Matrix**&** A**,** const int b**)** **{**

int **\***tmp **=** **new** int**[**A**.**rows **\*** A**.**cols**];**

**for** **(**int i **=** 0**;** i **<** A**.**rows **\*** A**.**cols**;** i**++)** **{**

tmp**[**i**]** **=** A**.**nums**[**i**]** **\*** b**;**

**}**

Matrix **\***C **=** **new** Matrix**(**tmp**,** A**.**rows**,** A**.**cols**);**

**return** **\***C**;**

**}**

//перегрузка оператора сложения через дружественную функцию

friend Matrix **operator** **+** **(**const Matrix**&** A**,** const Matrix**&** B**)** **{**

int **\***tmp **=** **new** int**[**A**.**rows **\*** A**.**cols**];**

**for** **(**int i **=** 0**;** i **<** A**.**rows **\*** A**.**cols**;** i**++)** **{**

tmp**[**i**]** **=** A**.**nums**[**i**]** **+** B**.**nums**[**i**];**

**}**

Matrix **\***C **=** **new** Matrix**(**tmp**,** A**.**rows**,** A**.**cols**);**

**return** **\***C**;**

**}**

//перегрузка оператора вычитания через дружественную функцию

friend Matrix **operator** **-** **(**const Matrix**&** A**,** const Matrix**&** B**)** **{**

int **\***tmp **=** **new** int**[**A**.**rows **\*** A**.**cols**];**

**for** **(**int i **=** 0**;** i **<** A**.**rows **\*** A**.**cols**;** i**++)** **{**

tmp**[**i**]** **=** A**.**nums**[**i**]** **-** B**.**nums**[**i**];**

**}**

Matrix **\***C **=** **new** Matrix**(**tmp**,** A**.**rows**,** A**.**cols**);**

**return** **\***C**;**

**}**

//перегрузка оператора умножения на матрицу через дружественную функцию

friend Matrix **operator** **\*(**const Matrix**&** A**,** const Matrix**&** B**)** **{**

int **\***tmp **=** **new** int**[**A**.**rows **\*** B**.**cols**];**

**for** **(**int i **=** 0**;** i **<** A**.**rows **\*** B**.**cols**;** i**++)** **{**

tmp**[**i**]** **=** 0**;**

**}**

**for** **(**int i **=** 0**;** i **<** A**.**rows**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** B**.**cols**;** j**++)** **{**

**for** **(**int c **=** 0**;** c **<** A**.**cols**;** c**++)** **{**

tmp**[**i **\*** B**.**cols **+** j**]** **+=** A**.**nums**[**i **\*** A**.**cols **+** c**]** **\*** B**.**nums**[**c **\*** B**.**cols **+** j**];**

**}**

**}**

**}**

Matrix **\***C **=** **new** Matrix**(**tmp**,** A**.**rows**,** B**.**cols**);**

**return** **\***C**;**

**}**

void print\_matrix**()** **{**

//Открытый метод, печатающий матрицу в консоль

**for** **(**int i **=** 0**;** i **<** rows**;** i**++)** **{**

**for** **(**int j **=** cols **\*** i**;** j **<** cols **\*** i **+** cols**;** j**++)** **{**

cout **<<** nums**[**j**];**

**if** **(**j **!=** **(**cols **\*** i **+** cols **-** 1**))** **{**

cout **<<** " "**;**

**}**

**}**

cout **<<** "\n"**;**

**}**

**}**

//объявление свойств и методов класса в закрытой области видимости

private**:**

int rows**;** //кол-во строк в матрице

int cols**;** //кол-во столбцов в матрице

int **\***nums**;** //элементы матрицы

**};**

//функция для ввода размерности матрицы

int**\*** dimension\_reading**()** **{**

static int dim**[**2**];**

dim**[**0**]** **=** **-**1**;**

dim**[**1**]** **=** **-**1**;**

int m**,** n**;**

**try** **{**

cout **<<** "Введите размерность матрицы (m x n) через пробел: "**;**

cin **>>** m**;**

**if** **(**cin**.**fail**())** **{**

cin**.**clear**();**

cin**.**ignore**(**numeric\_limits**<**streamsize**>::**max**(),**'\n'**);**

**throw** wrong\_dimension**(**"Ошибка. Вводимые параметры должны быть целыми числами!"**);**

**}**

cin **>>** n**;**

**if** **(**cin**.**fail**())** **{**

cin**.**clear**();**

cin**.**ignore**(**numeric\_limits**<**streamsize**>::**max**(),**'\n'**);**

**throw** wrong\_dimension**(**"Ошибка. Вводимые параметры должны быть целыми числами!"**);**

**}**

**if** **(**m **\*** n **<=** 0 **||** m **<** 0 **||** n **<** 0**)** **{**

**throw** wrong\_dimension**(**"Ошибка. Нулевая или отрицательная размерность!"**);**

**}**

**}**

**catch** **(**wrong\_dimension e**)** **{**

cout **<<** e**.**what**()** **<<** '\n'**;**

**return** dim**;**

**}**

dim**[**0**]** **=** m**;**

dim**[**1**]** **=** n**;**

**return** dim**;**

**}**

//функция для ввода элементов матрицы

vector **<**int**>** nums\_reading**(**int m**,** int n**)** **{**

int tmp**;**

vector **<**int**>** nums**;**

**for** **(**int i **=** 0**;** i **<** m**;** i**++)** **{**

cout **<<** "Введите элементы " **<<** i **+** 1 **<<** " строки через пробел: "**;**

**for** **(**int j **=** n **\*** i**;** j **<** n **\*** i **+** n**;** j**++)** **{**

**while** **(true)** **{**

**try** **{**

cin **>>** tmp**;**

**if** **(**cin**.**fail**())** **{**

cin**.**clear**();**

cin**.**ignore**(**numeric\_limits**<**streamsize**>::**max**(),**'\n'**);**

**throw** int\_exception**();**

**}**

**break;**

**}**

**catch** **(**int\_exception e**)** **{**

cout **<<** e**.**what**()** **<<** '\n'**;**

**}**

**}**

nums**.**push\_back**(**tmp**);**

**}**

**}**

**return** nums**;**

**}**

int main**()** **{**

system**(**"chcp 65001 > nul"**);**

int m**,** n**;**

cout **<<** "Размерность матриц A и B\n"**;**

**while** **(true)** **{**

int**\*** ptr **=** dimension\_reading**();**

**if** **(**ptr**[**0**]** **!=** **-**1**)** **{**

m **=** ptr**[**0**];**

n **=** ptr**[**1**];**

**break;**

**}**

**}**

cout **<<** "\n"**;**

int values**[**m **\*** n**];**

cout **<<** "\nВвод элементов матрицы A:\n"**;**

vector **<**int**>** tmp **=** nums\_reading**(**m**,** n**);**

**for** **(**int i **=** 0**;** i **<** m **\*** n**;** i**++)** **{**

values**[**i**]** **=** tmp**[**i**];**

**}**

//создание экземпляра класса с использованием конструктора с параметрами

Matrix A**(**values**,** m**,** n**);**

int b**;**

**while** **(true)** **{**

**try** **{**

cout **<<** "Введите коэфф. b: "**;**

cin **>>** b**;**

**if** **(**cin**.**fail**())** **{**

cin**.**clear**();**

cin**.**ignore**(**numeric\_limits**<**streamsize**>::**max**(),**'\n'**);**

**throw** int\_exception**();**

**}**

**break;**

**}**

**catch** **(**int\_exception e**)** **{**

cout **<<** e**.**what**()** **<<** '\n'**;**

**}**

**}**

cout **<<** "\nИсходная матрица A:\n"**;**

A**.**print\_matrix**();** //вывод матрицы A на экран

cout **<<** "\nМатрица C = A \* b:\n"**;**

//использование перегруженного оператора умножения на скаляр

Matrix C **=** A **\*** b**;**

C**.**print\_matrix**();** //вывод матрицы C на экран

cout **<<** "\nВвод элементов матрицы B:\n"**;**

tmp **=** nums\_reading**(**m**,** n**);**

**for** **(**int i **=** 0**;** i **<** m **\*** n**;** i**++)** **{**

values**[**i**]** **=** tmp**[**i**];**

**}**

//создание экземпляра класса с использованием конструктора с параметрами

Matrix B**(**values**,** m**,** n**);**

cout **<<** "\nИсходная матрица B:\n"**;**

B**.**print\_matrix**();** //вывод матрицы B на экран

cout **<<** "\nМатрица C = A + B:\n"**;**

//использование перегруженного оператора сложения

C **=** A **+** B**;**

C**.**print\_matrix**();** //вывод матрицы C на экран

cout **<<** "\nМатрица C = A - B:\n"**;**

//использование перегруженного оператора вычитания

C **=** A **-** B**;**

C**.**print\_matrix**();** //вывод матрицы C на экран

cout **<<** "\nРазмерность матрицы B для умножения\n"**;**

**while** **(true)** **{**

**try** **{**

int**\*** ptr **=** dimension\_reading**();**

**if** **(**ptr**[**0**]** **!=** **-**1**)** **{**

**if** **(**n **!=** ptr**[**0**])** **{**

**throw** wrong\_dimension**(**"Ошибка. Кол-во столбцов матрицы A не равняется кол-ву строк матрицы B!"**);**

**}**

m **=** ptr**[**0**];**

n **=** ptr**[**1**];**

**break;**

**}**

**}**

**catch** **(**wrong\_dimension e**)** **{**

cout **<<** e**.**what**()** **<<** '\n'**;**

**}**

**}**

cout **<<** "\nВвод элементов матрицы B:\n"**;**

tmp **=** nums\_reading**(**m**,** n**);**

**for** **(**int i **=** 0**;** i **<** m **\*** n**;** i**++)** **{**

values**[**i**]** **=** tmp**[**i**];**

**}**

//создание экземпляра класса с использованием конструктора с параметрами

Matrix B1**(**values**,** m**,** n**);**

cout **<<** "\nМатрица C = A \* B:\n"**;**

//использование перегруженного оператора умножения

C **=** A **\*** B1**;**

C**.**print\_matrix**();** //вывод матрицы C на экран

**return** 0**;**

**}**

# 

# ***Пример выполнения:***

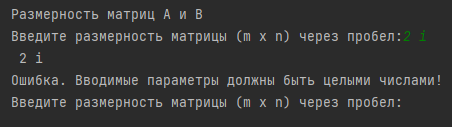


Рисунок 1. Результат отладки программы. Ошибка «Вводимые параметры должны быть целыми числами!».

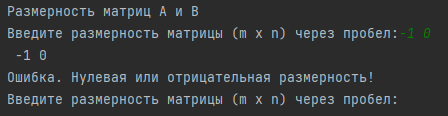


Рисунок 2. Результат отладки программы. Ошибка «Нулевая или отрицательная размерность!».

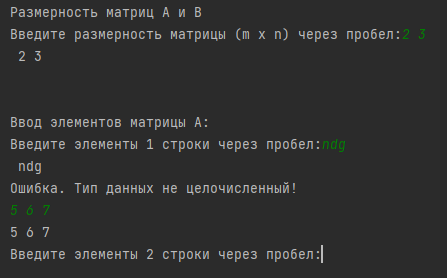


Рисунок 3. Результат отладки программы. Ошибка «Тип данных не целочисленный!».

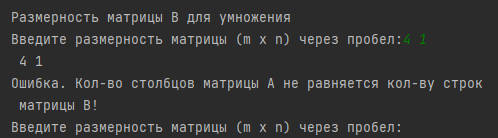


Рисунок 4. Результат отладки программы. Ошибка «Кол-во столбцов матрицы A не равняется кол-ву строк матрицы B!».

# 

Рисунок 5. Результат отладки программы. Корректный ввод данных (часть 1).

# 

Рисунок 6. Результат отладки программы. Корректный ввод данных (часть 2).

# ***Вывод:***

Я научился использовать встроенное в язык средство обработки исключительных ситуаций для предотвращения аварийного завершения работы программы в случае ошибок в данных.

# 

# ***Контрольные вопросы:***

**1) Зачем нужен блок «try»?**

Ключевое слово try служит для определения блока стейтментов (так называемого «блока try»). Блок try действует как наблюдатель в поисках исключений, которые были выброшены каким-либо из операторов в этом же блоке try. Блок try не определяет как мы будем обрабатывать исключение. Блок try обнаруживает любые исключения, которые были выброшены в нем, и направляет их в соответствующий блок catch для обработки. Блок try должен иметь, по крайней мере, один блок catch, который находится сразу же за ним, но также можетиметь и несколько блоков catch, размещенных последовательно (друг за другом).

**2) Зачем нужен блок «catch»?**

Фактически, обработка исключений — это работа блока(ов) catch. Ключевое слово catch используется для определения блока кода (так называемого «блока catch»), который обрабатывает исключения определенного типа данных.

**3) Какие исключения может обрабатывать каждый из блоков «catch»?**

Как только исключение было поймано блоком try и направлено в блок catch для обработки, оно считается обработанным (после выполнения кода блока catch), и выполнение программы возобновляется.Параметры catch работают также, как и параметры функции, причем параметры одного блока catch могут быть доступны и в другом блоке catch (который находится за ним). Исключения фундаментальных типов данных могут быть пойманы по значению (параметром блока catch является значение), но исключения не фундаментальных типов данных должны быть пойманы по константной ссылке (параметром блока catch является константная ссылка), чтобы избежать ненужного копирования. Как и в случае с функциями, если параметр не используется в блоке catch, то имя переменной можно не указывать.Это предотвратит вывод предупреждений компилятора о неиспользуемых переменных.

**4) Как задать блок «catch», чтобы он мог обрабатывать любые исключительные ситуации?**

Чтобы "catch" мог обрабатывать исключение любого типа, можно в записи реализации указать в скобках многоточие.

catch(…)

**5) Для чего используется блок «finally»?**

Блок "finally" выполняется всегда, независимо от того, произошло исключение в процессе работы блока try или нет.Наилучшее применение для блока finally - освобождение ресурсов, заказанных программой перед возникновением исключений.Несмотря на то, что система сборки мусора автоматически освобождает ненужную более оперативную память, открытые потоки, например, следует закрывать явным образом при помощи метода close.

**6) Что будет если в программе не задан блок «catch» для возникшей исключительной ситуации?**

Когда возбуждается исключительная ситуация, программа просматривает стек функций до тех пор, пока не находит соответствующий catch. Если оператор catch не найден, STL будет обрабатывать исключение в стандартном обработчике, который делает все менее изящно, чем могли бы сделать вы, показывая какие-то непонятные (для конечного пользователя) сообщения и обычно аварийно завершая программу.

Однако более важным моментом является то, что пока просматривается стек функций, вызываются деструкторы всех локальных классов, так что вам не нужно заботиться об освобождении памяти и т.п.

**7) Как работает обработка исключительных ситуаций, если заданы несколько вложенных блоков «try»?**

Блок try-catch может содержать вложенные блоки try-catch и если не будет определено соответствующего оператора catch на текущем уровен вложения, исключение будет поймано на более высоком уровне. Единственная вещь, о которой вы должны помнить, - это то, что операторы, следующие за throw, никогда не выполнятся.

**8) Когда необходима обработка исключительных ситуаций? Приведите примеры.**

Обработка исключительных ситуаций необходима тогда, когда во время выполнения программы могут возникать ситуации, когда состояние внешних данных, устройств ввода-вывода или компьютерной системы в целом делает дальнейшие вычисления в соответствии с базовым алгоритмом невозможными или бессмысленными. Примеры:

1. Целочисленное деление на ноль. Конечного результата у данной операции быть не может, поэтому ни дальнейшие вычисления, ни попытка использования результата деления не приведут к решению задачи.

2. Ошибка при попытке считать данные с внешнего устройства. Если данные не удаётся получить, любые дальнейшие запланированные операции с ними бессмысленны.

3. Исчерпание доступной памяти. Если в какой-то момент система оказывается не в состоянии выделить достаточный для прикладной программы объём оперативной памяти, программа не сможет работать нормально.