**Лабораторная работа № 4**

**Исключения**

**Цель работы:**

Приобретение навыков практического применения, закрепление знаний при создания простейших программ с использованием исключений.

**Задачи:**

1. Реализовать решение задания своего варианта, промежуточные значения при вычислениях помещать в массив фиксированной длины (длину вводит пользователь);
2. При решении задач производить обработку следующих исключительных ситуаций: ввода пользователем недопустимых значений, переполнения при вычислении математических выражений, деление на 0, индекс вне границ массива;
3. Выполнить требования к лабораторным работам (см. файл «Требования к лабораторным»);
4. Ознакомиться с литературой по теме лабораторной работы;
5. Ответить на контрольные вопросы.

*Дополнительно:*

1. создать свое исключение, для обработки ошибок

**Контрольные вопросы:**

1. Объясните назначение конструкции try-catch-finally;
2. Как генерировать исключения?
3. Как создать пользовательское исключение?
4. Как обработать несколько исключений?
5. Для чего используются операторы checked и unchecked?

**Пояснения и примеры:**

Инструкция try-catch состоит из блока try, за которым следует одно или несколько предложений catch, в которых определяются обработчики для различных исключений. При возникновении исключения среда CLR ищет оператор catch, который обрабатывает это исключение. Если выполняющийся в данный момент метод не содержит такого блока catch, то среда CLR рассматривает метод, который вызвал текущий метод, и т. д. по стеку вызовов. Если блок catch не найден, то среда CLR отображает пользователю сообщение о необработанном исключении и останавливает выполнение программы.

**Блок try** содержит защищаемый код, в котором могут происходить исключения. Этот блок выполняется до момента возникновения исключения или до своего успешного завершения. Например, следующая попытка приведения объекта null вызовет исключение [NullReferenceException](http://msdn.microsoft.com/ru-ru/library/system.nullreferenceexception(v=VS.90).aspx).

object o2 = null;

try

{

int i2 = (int)o2; // Error

}

Хотя предложение catch можно использовать без аргументов для перехвата любого типа исключения, такой подход не рекомендуется. В общем случае следует перехватывать только те исключения, устранение причин которых известно. Поэтому следует всегда задавать аргумент объекта, производного от [System.Exception](http://msdn.microsoft.com/ru-ru/library/system.exception(v=VS.90).aspx). Например:

catch (InvalidCastException e)

{

}

В одной и той же инструкции try-catch можно использовать несколько предложений catch. В этом случае будет иметь значение порядок следования предложений catch, поскольку предложения catch будут проверяться именно в этом порядке. Более общие исключения следует перехватывать после более частных. При упорядочивании блоков catch таким образом, чтобы последующий блок был недостижимым, компилятор вызовет ошибку.

В этом примере блок try содержит обращение к методу ProcessString, который может вызвать исключение. В предложении catch содержится обработчик исключения, который просто выводит на экран сообщение. Если инструкция throw вызывается из метода MyMethod, система осуществляет поиск инструкции catch и выводит на экран сообщениеException caught.

class TryFinallyTest

{

static void ProcessString(string s)

{

if (s == null)

{

throw new ArgumentNullException();

}

}

static void Main()

{

string s = null; // For demonstration purposes.

try

{

ProcessString(s);

}

catch (Exception e)

{

Console.WriteLine("{0} Exception caught.", e);

}

}

}

/\*

Output:

System.ArgumentNullException: Value cannot be null.

at TryFinallyTest.Main() Exception caught.

\* \*/

**Блок finally** позволяет освободить все ресурсы, выделенные в блоке [try](http://msdn.microsoft.com/ru-ru/library/0yd65esw(v=VS.90).aspx), а также выполнить код, который должен выполняться даже в случае возникновения исключения. Управление всегда передается блоку finally независимо от того, как была завершена обработка блока try.

Если для обработки исключений, происходящих в блоке инструкций используется блок catch, то блок finally позволяет гарантировать, что определенный набор инструкций будет выполнен независимо от того, как был осуществлен выход из блока try.

В этом примере содержится одна недопустимая инструкция преобразования, вызывающая исключение. При запуске программы появляется сообщение об ошибке во время выполнения, однако блок finally все равно будет выполнен и будет выведен результат.

public class ThrowTest

{

static void Main()

{

int i = 123;

string s = "Some string";

object o = s;

try

{

// Invalid conversion; o contains a string not an int

i = (int)o;

}

finally

{

Console.Write("i = {0}", i);

}

}

}

В приведенном выше примере произойдет исключение System.InvalidCastException.

Хотя исключение и было перехвачено, инструкция вывода в блоке finally все равно будет исполнена, т. е. на экран будет выведена строка:

i = 123

Распространенное использование catch вместе с finally — это получение и использование ресурсов в блоке try, обработка исключительных ситуаций в блоке catch и высвобождение ресурсов в блоке finally.

public class EHClass

{

void ReadFile(int index)

{

// To run this code, substitute a valid path from your local machine

string path = @"c:\users\public\test.txt";

System.IO.StreamReader file = new System.IO.StreamReader(path);

char[] buffer = new char[10];

try

{

file.ReadBlock(buffer, index, buffer.Length);

}

catch (System.IO.IOException e)

{

Console.WriteLine("Error reading from {0}. Message = {1}", path, e.Message);

}

finally

{

if (file != null)

{

file.Close();

}

}

// Do something with buffer...

}

}

**Оператор throw** используется для сообщения о случаях аномальных ситуаций (исключений) в ходе выполнения программы.

В этом примере демонстрируется вызов исключения с помощью оператора throw.

public class ThrowTest2

{

static int GetNumber(int index)

{

int[] nums = { 300, 600, 900 };

if (index > nums.Length)

{

throw new IndexOutOfRangeException();

}

return nums[index];

}

static void Main()

{

int result = GetNumber(3);

}

}

/\*

Output:

The System.IndexOutOfRangeException exception occurs.

\*/

**Ключевое слово checked** используется для явного включения проверки переполнения при выполнении арифметических операций и преобразований с данными целого типа.

По умолчанию выражение, содержащее только константные значения, вызывает ошибку компилятора в том случае, если результат его вычисления выходит за допустимые пределы значений конечного типа. Если выражение содержит одно или несколько неконстантных значений, компилятор не выполняет проверку переполнения. Вычисление выражения, присвоенного переменной i2 в следующем примере, не вызывает ошибку компилятора.

**C#**

// The following example causes compiler error CS0220 because 2147483647

// is the maximum value for integers.

//int i1 = 2147483647 + 10;

// The following example, which includes variable ten, does not cause

// a compiler error.

int ten = 10;

int i2 = 2147483647 + ten;

// By default, the overflow in the previous statement also does

// not cause a run-time exception. The following line displays

// -2,147,483,639 as the sum of 2,147,483,647 and 10.

Console.WriteLine(i2);

По умолчанию эти неконстантные выражения также не проверяются на переполнение во время выполнения, и они не создают исключений переполнения. В предыдущем примере в качестве суммы двух положительных целых чисел выводится значение -2 147 483 639.

Проверку переполнения можно включить посредством параметров компилятора, настройки среды или использования ключевого слова checked. В следующих примерах демонстрируется использование выражения checked или блока checked для обнаружения переполнения, возникающего в результате предыдущего сложения во время выполнения. В обоих примерах создается исключение переполнения.

**C#**

// If the previous sum is attempted in a checked environment, an

// OverflowException error is raised.

// Checked expression.

Console.WriteLine(checked(2147483647 + ten));

// Checked block.

checked

{

int i3 = 2147483647 + ten;

Console.WriteLine(i3);

}

Для запрета проверки переполнения можно использовать ключевое слово [unchecked](http://msdn.microsoft.com/ru-ru/library/a569z7k8.aspx).

**Литература:**

1. Троелсен Э. - C# и платформа .NET. Библиотека программиста. Обработка исключений 177 – 187 стр.;
2. статья «checked (Справочник по C#)»;
3. C# - лекции IntUit (Биллиг В.А.). Лекция 23, пункт Обработка исключительных ситуаций 288 стр.;
4. Рихтер C# via .Net. Глава 19 Исключения 408 – 443 стр.;

**Для справки:**

используйте следующие исключения, информацию по ним смотрите в msdn:

|  |  |
| --- | --- |
| Исключение | Описание |
| [ArithmeticException](http://msdn.microsoft.com/ru-ru/library/system.arithmeticexception.aspx) | Основной класс исключений, происходящих при выполнении арифметических операций, таких как DivideByZeroException и OverflowException. |
| [ArrayTypeMismatchException](http://msdn.microsoft.com/ru-ru/library/system.arraytypemismatchexception.aspx) | Создается, когда массив не может хранить данный элемент, поскольку фактический тип элемента несовместим с фактическим типом массива. |
| [DivideByZeroException](http://msdn.microsoft.com/ru-ru/library/system.dividebyzeroexception.aspx) | Создается при попытке разделить целое число на ноль. |
| [IndexOutOfRangeException](http://msdn.microsoft.com/ru-ru/library/system.indexoutofrangeexception.aspx) | Создается при попытке индексирования массива, если индекс меньше нуля или выходит за границы массива. |
| [InvalidCastException](http://msdn.microsoft.com/ru-ru/library/system.invalidcastexception.aspx) | Создается, когда происходит сбой явного преобразования из основного типа в интерфейс либо в производный тип во время выполнения. |
| [NullReferenceException](http://msdn.microsoft.com/ru-ru/library/system.nullreferenceexception.aspx) | [Создается при попытке ссылки на объект, значение которого равно null.](http://msdn.microsoft.com/ru-ru/library/edakx9da.aspx) |
| [OutOfMemoryException](http://msdn.microsoft.com/ru-ru/library/system.outofmemoryexception.aspx) | [Создается при неудаче попытки выделения памяти с помощью оператора new. Это означает, что память, доступная для среды выполнения, уже исчерпана.](http://msdn.microsoft.com/ru-ru/library/fa0ab757.aspx) |
| [OverflowException](http://msdn.microsoft.com/ru-ru/library/system.overflowexception.aspx) | Создается при переполнении арифметической операции в контексте checked. |
| [StackOverflowException](http://msdn.microsoft.com/ru-ru/library/system.stackoverflowexception.aspx) | Создается, когда стек выполнения переполнен за счет слишком большого количества вызовов отложенных методов; обычно является признаком очень глубокой или бесконечной рекурсии. |
| [TypeInitializationException](http://msdn.microsoft.com/ru-ru/library/system.typeinitializationexception.aspx) | Создается, когда статический конструктор создает исключение, и не существует ни одного совместимого предложения catch для его захвата. |