Обработка исключительных ситуаций

Виды ошибок

1. **Синтаксические ошибки.** Это самые простые ошибки, так как если в вашей программе есть такая ошибка, то программа просто не будет создана, а компилятор выдаст сообщение об ошибке с указанием строки в исходном коде, где была найдена ошибка.
2. **Семантические ошибки.** Эти ошибки не подвластны компилятору. Такие ошибки обычно очень трудно найти и поэтому их называют труднонаходимыми.
3. **Ошибки времени выполнения.** Это ошибки, которые могут произойти во время выполнения программы. Например, если пользователь введёт ноль в качестве делителя - тогда произойдёт ошибка, так как на ноль делить нельзя.

Пример

В этом примере задачей функции является возвращение среднего для двух переданных чисел.

Ошибка кроется в неучёте приоритета операторов (деление в выражении вычисляется до операции сложения) и отсутствии по этой причине скобок.

```
int average(int a, int b)
{
   return a + b / 2;}
```

правильная запись (a + b) / 2

Общее понятие исключительной ситуации

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.

- **Целочисленное деление на ноль.**
- **Ошибка при попытке считать данные с внешнего устройства.** Если данные не удаётся получить, любые дальнейшие запланированные операции с ними бессмысленны.
- **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.
- **Появление сигнала аварийного отключения электропитания системы**. Прикладную задачу, по всей видимости, решить не удастся, в лучшем случае прикладная программа может позаботиться о сохранении данных.
- **Появление на входе коммуникационного канала данных, требующих немедленного считывания**. Чем бы ни занималась в этот

Общее понятие исключительной ситуации

Обработка исключений – это описание реакции программы на подобные события (исключения) во время выполнения программы. Реакцией программы может быть корректное завершение работы программы, вывод информации об ошибке и запрос повторения действия (при вводе данных).

Примерами исключений может быть:

- деление на ноль;
- конвертация некорректных данных из одного типа в другой;
- попытка открыть файл, которого не существует;
- доступ к элементу вне рамок массива;
- исчерпывание памяти программы;
- другое.

Механизм обработки исключений

Для реализации механизма обработки исключений в язык Си++ введены следующие три ключевых (служебных) слова: **try** (контролировать), **catch** (ловить), **throw** (генерировать, порождать).

- Блоки **try** инкапсулируют код, формирующий часть нормальных действий программы, которые потенциально могут столкнуться с серьезными ошибочными ситуациями.
- Блоки **catch** инкапсулируют код, который обрабатывает ошибочные ситуации, происходящие в коде блока try. Это также удобное место для протоколирования ошибок.

```
try
{
  //блок кода, в котором возможно исключение
}
catch ([тип исключения] [имя])
{
  //блок кода – обработка исключения
}
```

Try и catch

Основу обработки исключительных ситуаций в C# составляет пара ключевых слов try и catch. Эти ключевые слова действуют совместно и не могут быть использованы порознь.

```
try {
// Блок кода, проверяемый на наличие ошибок.
}
catch (ExcepType1 exOb) {
// Обработчик исключения типа ExcepType1.
}
catch (ExcepType2 exOb) {
// Обработчик исключения типа ExcepType2.
}
...
```

Пояснения

Выполняется код в блоке **try**, и, если в нем происходит исключение типа, соответствующего типу, указанному в catch, то управление передается блоку catch. При этом, весь оставшийся код от момента выбрасывания исключения до конца блока try не будет выполнен. После выполнения блока catch, оператор try-catch завершает работу.

Указывать имя исключения не обязательно. Исключение представляет собою объект, и к нему мы имеем доступ через это имя. С этого объекта мы можем получить, например, стандартное сообщение об ошибке (Message). В этом объекте хранится детальная информации об исключении.

Если тип выброшенного исключения не будет соответствовать типу, указанному в catch – исключение не обработается, и программа завершит работу аварийно.

Несколько блоков catch

```
 try
{
   //блок1
}
catch (FormatException)
{
  //блок-обработка исключения 1
}
catch (FileNotFoundException)
{
  //блок-обработка исключения 2
}
```

В зависимости от того или другого типа исключения в блоке try, выполнение будет передано соответствующему блоку catch.

Типы исключений

Exception – базовый тип всех исключений. Блок catch, в котором указан тип Exception будет «ловить» все исключения.

FormatException – некорректный формат операнда или аргумента (при передаче в метод).

NullReferenceException - В экземпляре объекта не задана ссылка на объект, объект не создан

IndexOutOfRangeException – индекс вне рамок коллекции

FileNotFoundException – файл не найден.

DivideByZeroException – деление на ноль

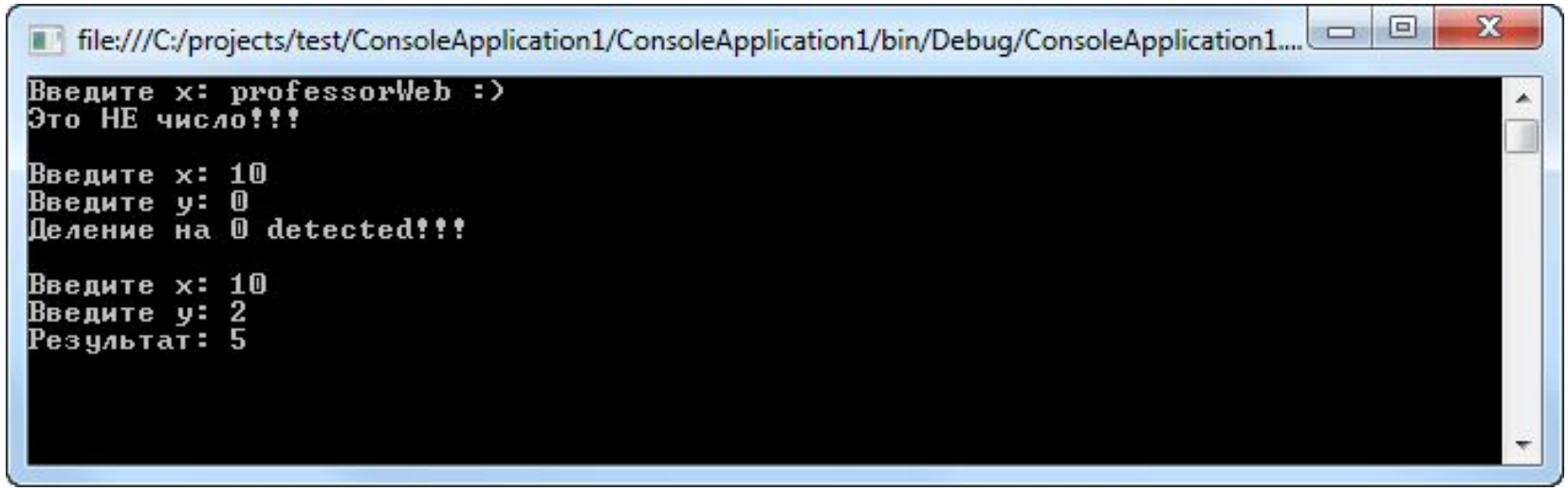
Пример 1 – деление на 0

```
namespace ConsoleApplication1
{
   class Program
   {
      static int MyDel(int x, int y)
      {          return x / y;
      }

      static void Main()
      {
         try
         {
            Console.Write("Введите x: ");
            int x = int.Parse(Console.ReadLine());
            Console.Write("Введите y: ");
            int y = int.Parse(Console.ReadLine());
            int result = MyDel(x, y);
            Console.WriteLine("Результат: " + result);
         }
```

Пример 1 – деление на 0

```
// Обрабатываем исключение возникающее при делении на ноль
            catch (DivideByZeroException)
            {
               Console.WriteLine("Деление на 0 detected!!!\n");
                Main();
            }
            // Обрабатываем исключение при неккоректном вводе числа в
консоль
            catch (FormatException)
            {
               Console.WriteLine("Это НЕ число!!!\n");
               Main();
            }

            Console.ReadLine();
        }
    }
}
```

Пример 1 -деление на 0



Пример 2 –длина строки

Но язык C# также позволяет генерировать исключения вручную с помощью оператора throw. То есть с помощью этого оператора мы сами можем создать исключение и вызвать его в процессе выполнения.

В программе происходит ввод строки, и мы хотим, чтобы, если длина строки будет больше 6 символов, возникало исключение:

Пример 2 длина строки

```
static void Main(string[] args)
{
    try
    {
        Console.Write("Введите строку: ");
        string message = Console.ReadLine();
        if (message.Length > 6)
        {           throw new Exception("Длина строки больше 6 символов");
        }
    }
    catch (Exception e)
    {       Console.WriteLine($"Ошибка: {e.Message}");
    }
    Console.Read();
}
```

Пример 3 формат ввода

```
static void Main(string[] args)
    {
      string result = "";
      Console.WriteLine("Введите число:");
      try
      {
        int a = Convert.ToInt32(Console.ReadLine()); //вводим данные, и
    конвертируем в целое число
        result = "Вы ввели число " + a;
      }
      catch (FormatException)
      {
        result = "Ошибка. Вы ввели не число";
      }
      Console.WriteLine(result);
      Console.ReadLine();
    }
```

Блок finally

Оператор try-catch также может содержать блок finally. Особенность блока finally в том, **что код внутри этого блока выполнится в любом случае**, в независимости от того, было ли исключение или нет.

```
try
{
  //блок1
}
catch (Exception)
{
  //обработка исключения
}
finally
{
 //блок кода, который выполнится обязательно
}
```

Блок finally

Выполнение кода программы в блоке finally происходит в последнюю очередь. Сначала try затем finally или catch-finally (если было исключение).

Обычно, он используется для освобождения ресурсов. Классическим примером использования блока finally является закрытие файла.

Finally **гарантирует** выполнение кода, несмотря ни на что. Даже если в блоках try или catch будет происходить выход из метода с помощью оператора return – finally выполнится.

Пример 4 – открытие файла

```
 static void ReadFile()
{
   StreamReader reader = null;
   try
   {
      reader = File.OpenText ("file.txt");
      if (reader.EndOfStream) return;
      Console.WriteLine (reader.ReadToEnd());
   }
   finally
   {
      if (reader != null) reader.Dispose();
   }
}
```