Практическая работа 12
Обработка и генерация исключений

Цель занятия

Получить практические навыки обработки и генерации исключений в модулях

Перечень оборудования и программного обеспечения

Персональный компьютер
Microsoft Office (Word, Visio)
Microsoft Visual Studio 2010

Краткие теоретические сведения

В .NET Framework предусмотрена развитая система обработки ошибок. Механизм обработки ошибок C# позволяет закодировать пользовательскую обработку для каждого типа ошибочных условий, а также отделить код, потенциально порождающий ошибки, от кода, обрабатывающего их.

Пользовательские ошибки (user errors)

В отличие от программных ошибок, пользовательские ошибки обычно возникают из-за тех, кто запускает приложение, а не тех, кто его создает. Например, ввод конечным пользователем в текстовом поле неправильно оформленной строки может привести к генерации ошибки подобного рода, если в коде не была предусмотрена возможность обработки некорректного ввода.

Исключения (exceptions)

Исключениями, или исключительными ситуациями, обычно называются аномалии, которые могут возникать во время выполнения и которые трудно, а порой и вообще невозможно, предусмотреть во время программирования приложения. К числу таких возможных исключений относятся попытки подключения к базе данных, которой больше не существует, попытки открытия поврежденного файла или попытки установки связи с машиной, которая в текущий момент находится в автономном режиме. В каждом из этих случаев программист (и конечный пользователь) мало что может сделать с подобными "исключительными" обстоятельствами.

Структурированная обработка исключений в .NET представляет собой методику, предназначенную для работы с исключениями, которые могут возникать на этапе выполнения. Даже в случае программных и пользовательских ошибок, которые ускользнули от глаз программиста,

однако, CLR будет часто автоматически генерировать соответствующее исключение с описанием текущей проблемы. В библиотеках базовых классов .NET определено множество различных исключений, таких как FormatException, IndexOutOfRangeException, FileNotFoundException, ArgumentOutOfRangeException и т.д.

В терминологии .NET под "исключением" подразумеваются программные ошибки, пользовательские ошибки и ошибки времени выполнения. Прежде чем погружаться в детали, давайте посмотрим, какую роль играет структурированная обработка исключений, и чем она отличается от традиционных методик обработки ошибок.

Составляющие процесса обработки исключений в .NET

Программирование со структурированной обработкой исключений подразумевает использование четырех следующих связанных между собой сущностей:

тип класса, который представляет детали исключения;

член, способный генерировать (throw) в вызывающем коде экземпляр класса исключения при соответствующих обстоятельствах;

блок кода на вызывающей стороне, ответственный за обращение к члену, в котором может произойти исключение;

блок кода на вызывающей стороне, который будет обрабатывать (или перехватывать (catch)) исключение в случае его возникновения.

Для того чтобы справиться с возможными ошибочными ситуациями в коде C#, программа обычно делится на блоки трех разных типов:

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.

Блоки finally инкапсулируют код, очищающий любые ресурсы или выполняющий другие действия, которые обычно нужно выполнить в конце блоков try или catch. Важно понимать, что этот блок выполняется независимо от того, сгенерировано исключение или нет.

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. Ниже приведена общая форма определения блоков try/catch для обработки исключительных ситуаций:

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

```
}
catch (ExcepType2 exOb) {
// Обработчик исключения типа ExcepType2.
}
...
```

где ExcepType — это тип возникающей исключительной ситуации. Когда исключение генерируется оператором try, оно перехватывается составляющим ему пару оператором catch, который затем обрабатывает это исключение. В зависимости от типа исключения выполняется и соответствующий оператор catch. Так, если типы генерируемого исключения и того, что указывается в операторе catch, совпадают, то выполняется именно этот оператор, а все остальные пропускаются. Когда исключение перехватывается, переменная исключения exOb получает свое значение. На самом деле указывать переменную exOb необязательно. Так, ее необязательно указывать, если обработчику исключений не требуется доступ к объекту исключения, что бывает довольно часто. Для обработки исключения достаточно и его типа.

Пример, в котором будем обрабатывать исключение, возникающее при делении числа на 0:

```
static int MyDel(int x, int y)
    {
       return x / y;
    }
    static void Main(string[] args)
     {
        {
        link1:
           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);
           }
           // Обрабатываем исключение возникающее при делении на ноль
catch (DivideByZeroException)
           {
              Console.WriteLine("Делениена 0 detected!!!\n");
goto link1;
           }
           // Обрабатываем исключение при некорректном вводе числа в
консоль
catch (FormatException)
```

```
            {
               Console.WriteLine("ЭтоНЕчисло!!!\n");
goto link1;
            }
            Console.ReadLine();
```

Данный простой пример наглядно иллюстрирует обработку исключительной ситуации при делении на 0 (DivideByZeroException), а так же пользовательскую ошибку при вводе не числа (FormatException).

Последствия неперехвата исключений

Перехват одного из стандартных исключений, как в приведенном выше примере, дает еще одно преимущество: он исключает аварийное завершение программы. Как только исключение будет сгенерировано, оно должно быть перехвачено каким-то фрагментом кода в определенном месте программы. Вообще говоря, если исключение не перехватывается в программе, то оно будет перехвачено исполняющей системой. Но дело в том, что исполняющая система выдаст сообщение об ошибке и прервет выполнение программы.

Такие сообщения об ошибках полезны для отладки программы, но, по меньше мере, нежелательны при ее использовании на практике! Именно поэтому так важно организовать обработку исключительных ситуаций в самой программе.

Пример 2:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
namespace ConsoleApplication2
{
   class Program
   {
      static int MyDel(int x, int y)
      {
         return x / y;
      }
      static void Main()
      {
      link1:
         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);
        }
        // Обрабатываем исключение возникающее при делении на ноль
catch (DivideByZeroException ex)
        {
            Console.WriteLine("Делениена 0 detected!!!\n");
            Console.WriteLine("ОШИБКА: " + ex.Message + "\n\n");
goto link1;
        }
        // Обрабатываем исключение при некорректном вводе числа в консоль
catch (FormatException ex)
        {
            Console.WriteLine("ЭтоНЕчисло!!!\n");
Console.WriteLine("ОШИБКА: " + ex.Message + "\n\n");
goto link1;
        }
        Console.ReadLine();
    }
  }
}
```

В данном примере используется свойство Message, класса Exception, для вывода информации о возникшем исключении. Исключение, выбрасываемое, если формат аргумента не соответствует спецификациям параметра вызываемого метода.

Часто встречающиеся исключения

№	Исключение	Описание
1	FormatException	Исключение, выбрасываемое, если формат аргумента не соответствует спецификациям параметра вызываемого метода.
2	DivideByZeroException	Исключение, выбрасываемое при попытке деления целого или дробного числа на нуль.
3	NotFiniteNumberException	Исключение, которое выбрасывается, когда значение с плавающей запятой является плюс бесконечностью, минус бесконечностью или не является числовым (NaN).
4	OverflowException	Исключение, которое выбрасывается, когда при выполнении арифметических операций,

		операций приведения типов и преобразования происходит переполнение.
5	FileNotFoundException	Это исключение создается, когда попытка доступа к файлу, не существующему на диске, заканчивается неудачей.
6	ApplicationException	Это исключение выбрасывается при происхождении устранимой ошибки приложения

Задания

1 Изучить теоретические сведения и задание к работе

2 В соответствии с вариантом задания разработать и отладить программный модуль, используя исключение, выбрасываемое, если формат аргумента не соответствует спецификациям параметра вызываемого метода.

Порядок выполнения работы (Пример выполнения)

Исходные данные:

Есть массив целых чисел размером n. С клавиатуры вводится два числа - порядковые номера элементов массива, которые необходимо суммировать. Например, если ввели 3 и 5 - суммируются 3-й и 5-й элементы. Описать функцию целого типа, возвращающую сумму заданных элементов массива.

Решение

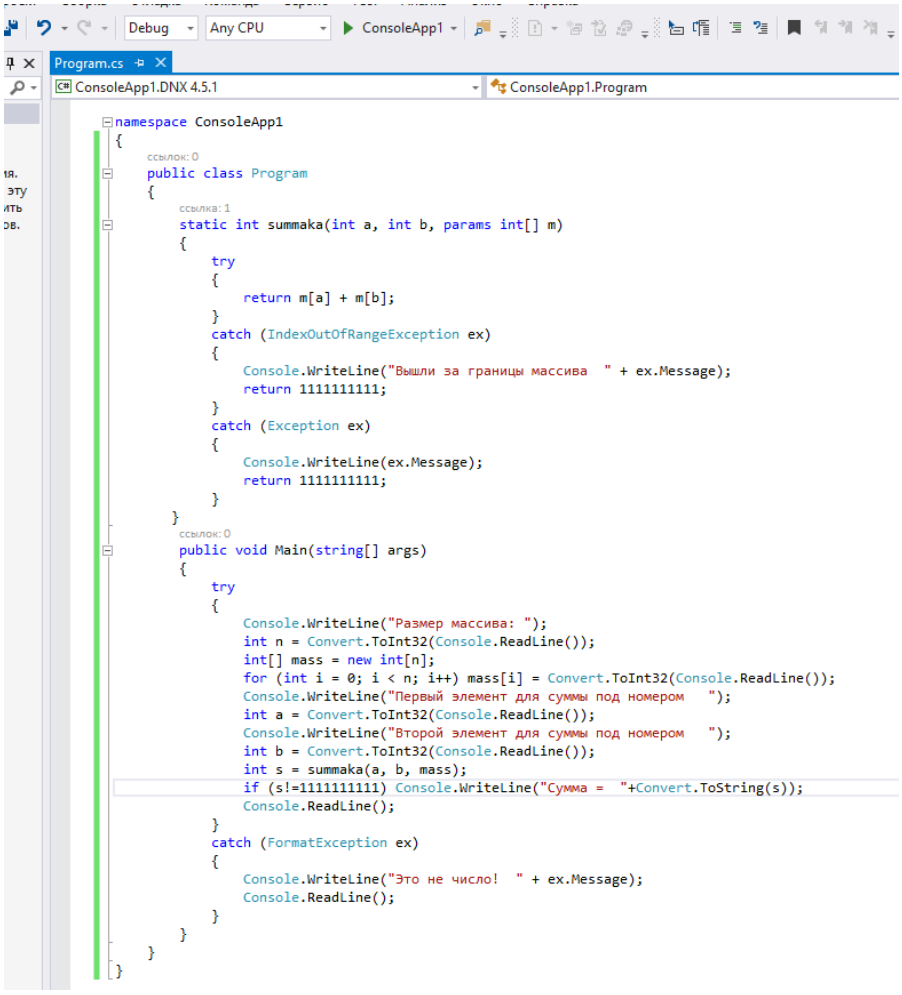
При создании программы нужно предусмотреть случаи, когда были введены не числа, и когда одно из чисел, или оба больше размера массива.

2 Текст программы.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
namespace ConsoleApp1
{
   public class Program
   {
      static int summaka(int a, int b, params int[] m)
      {
         try
```

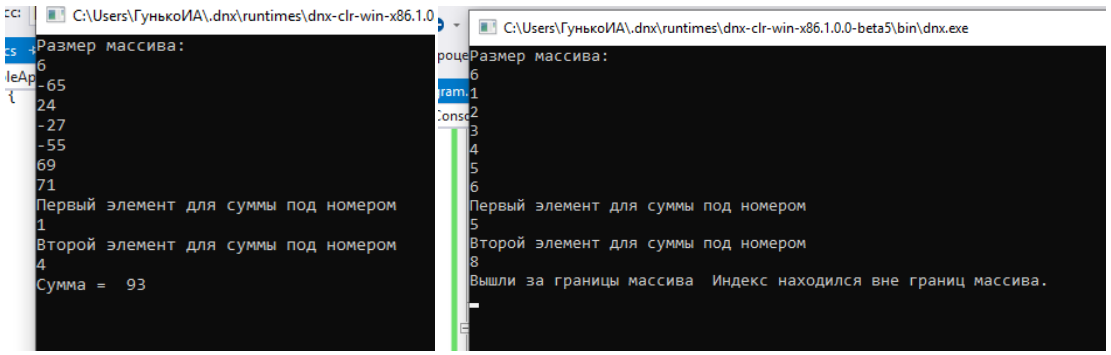
```
        {
...
        {
            Console.WriteLine("Это не число!  " + ex.Message);
            Console.ReadLine();
        }
    }
}
    }
```

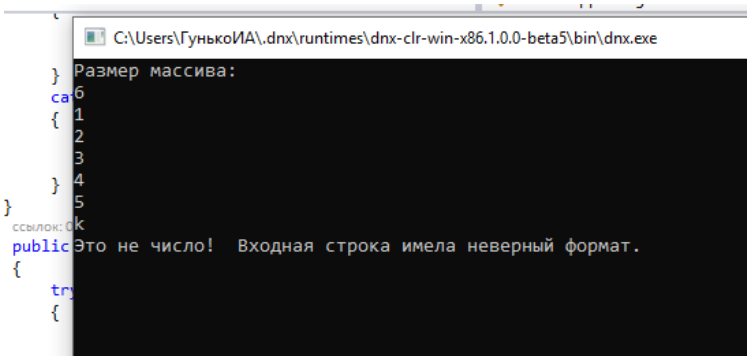
3 VisualStudio.



```
namespace ConsoleApp1
{
    public class Program
    {
        static int summaka(int a, int b, params int[] m)
        {
            try
            {
                return m[a] + m[b];
            }
            catch (IndexOutOfRangeException ex)
            {
                Console.WriteLine("Вышли за границы массива  " + ex.Message);
                return 1111111111;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                return 1111111111;
            }
        }
        public void Main(string[] args)
        {
            try
            {
                Console.WriteLine("Размер массива: ");
                int n = Convert.ToInt32(Console.ReadLine());
                int[] mass = new int[n];
                for (int i = 0; i < n; i++) mass[i] = Convert.ToInt32(Console.ReadLine());
                Console.WriteLine("Первый элемент для суммы под номером   ");
                int a = Convert.ToInt32(Console.ReadLine());
                Console.WriteLine("Второй элемент для суммы под номером   ");
                int b = Convert.ToInt32(Console.ReadLine());
                int s = summaka(a, b, mass);
                if (s!=1111111111) Console.WriteLine("Сумма =  "+Convert.ToString(s));
                Console.ReadLine();
            }
            catch (FormatException ex)
            {
                Console.WriteLine("Это не число!  " + ex.Message);
                Console.ReadLine();
            }
        }
    }
}
```

4 Результаты работы программы:





Содержание отчета

1 Название работы
2 Цель работы
3 Технические средства обучения
4 Задания (условия задач)
5 Порядок выполнения работы
6 Ответы на контрольные вопросы
7 Вывод

Варианты к практической работе:

1 Описать функцию IsIdent(S) целого типа, проверяющую, является ли строка S допустимым идентификатором, то есть непустой строкой, которая содержит только латинские буквы, цифры и символ подчеркивания «_» и не начинается с цифры. Если S является допустимым идентификатором, то функция возвращает 0. Если S является пустой строкой, то возвращается –1, если S начинается с цифры, то возвращается –2. Если S содержит недопустимые символы, то возвращается номер первого недопустимого символа. Проверить с помощью функции IsIdent две данные строки.

2 Описать функцию FillStr(S, N) строкового типа, возвращающую строку длины N, заполненную повторяющимися копиями строки-шаблона S (последняя копия строки-шаблона может входить в результирующую строку частично). Используя эту функцию, сформировать по данному числу N и двум строкам-шаблонам две результирующие строки длины N.

3. Описать процедуру UpCaseRus(S), преобразующую все строчные русские буквы строки S в прописные (остальные символы строки S не изменяются). Строка S является входным и выходным параметром. Используя процедуру UpCaseRus, преобразовать две данные строки.

4. Описать процедуру LowCaseRus(S), преобразующую все прописные русские буквы строки S в строчные (остальные символы строки S не изменяются). Строка S является входным и выходным параметром. Используя процедуру LowCaseRus, преобразовать две данные строки.

5. Описать процедуру TrimLeftC(S, C), удаляющую в строке S начальные символы, совпадающие с символом C. Строка S является входным и выходным параметром. Дан символ C и две строки. Используя процедуру TrimLeftC, преобразовать данные строки.

6. Описать процедуру TrimRightC(S, C), удаляющую в строке S конечные символы, совпадающие с символом C. Строка S является входным и выходным параметром. Дан символ C и две строки. Используя процедуру TrimRightC, преобразовать данные строки.

7. Описать функцию InvertStr(S, K, N) строкового типа, возвращающую инвертированную подстроку строки S, содержащую в обратном порядке N символов строки S, начиная с ее K-го символа. Если K превосходит длину строки S, то возвращается пустая строка; если длина строки меньше K + N, то инвертируются все символы строки, начиная с ее K-го символа. Вывести значения функции InvertStr для данной строки S и каждой из трех пар положительных целых чисел: (K1, N1), (K2, N2), (K3, N3).

8. Описать функцию PosSub(S0, S, K, N) целого типа, возвращающую номер позиции, начиная с которой в строке S содержится первое вхождение строки S0, причем анализируются только N символов строки S, начиная с ее K-го символа (таким образом, PosSub обеспечивает поиск в подстроке). Если K превосходит длину строки S, то возвращается 0, если длина строки меньше K + N, то анализируются все символы строки, начиная с ее K-го символа. Если в требуемой подстроке строки S вхождения S0 отсутствуют, то функция возвращает 0. Вывести значения функции PosSub для данных строк S0, S и каждой из трех пар положительных целых чисел: (K1, N1), (K2, N2), (K3, N3).

9. Описать функцию PosLast(S0, S) целого типа, возвращающую номер позиции, начиная с которой в строке S содержится последнее вхождение подстроки S0. Считать, что перекрывающихся вхождений подстрок S0 строка S не содержит. Если в строке S отсутствуют подстроки S0, то функция возвращает 0. Вывести значения этой функции для двух данных пар строк S0 и S.

10. Описать функцию PosK(S0, S, K) целого типа, возвращающую номер позиции, начиная с которой в строке S содержится K-е вхождение подстроки S0 (K > 0). Если количество вхождений S0 в строке S меньше K, то функция возвращает 0. Считать, что перекрывающихся вхождений под-строк S0 строка S не содержит. Вывести значения этой функции для двух данных троек: S0, S и K.

11. Описать функцию WordK(S, K) строкового типа, возвращающую K-е слово строки S (словом считается набор символов, не содержащий пробелов и ограниченный пробелами или началом/концом строки). Если количество слов в строке меньше K, то функция возвращает пустую строку. Используя эту функцию, выделить из данной строки S слова с данными номерами K1, K2, K3.

12. Описать процедуру SplitStr(S, W, N), которая формирует по данной строке S массив W слов, входящих в S (массив W и его размер N являются выходными параметрами). Словом считается набор символов, не содержащий пробелов и ограниченный пробелами или началом/концом строки; предполагается, что строка S содержит не более 10 слов. Используя функцию SplitStr, найти количество слов N, содержащихся в данной строке S, и сами эти слова.

13. Описать функцию CompressStr(S) строкового типа, выполняющую сжатие строки S по следующему правилу: каждая подстрока строки S, состоящая из более чем четырех одинаковых символов C, заменяется текстом вида «C{K}», где K — количество символов C (предполагается, что строка S не содержит фигурных скобок «{» и «}»). Например, для строки S = «bbbccccce» функция вернет строку «bbbc{5}e». С помощью функции CompressStr сжать пять данных строк.

14. Описать функцию DecompressStr(S) строкового типа, восстанавливающую строку, сжатую процедурой CompressStr (см. пред. задание). Параметр S содержит сжатую строку; восстановленная строка является возвращаемым значением функции. С помощью функции DecompressStr восстановить две данные сжатые строки.

15. Описать функцию DecToBin(N) строкового типа, возвращающую строковое представление целого неотрицательного числа N в двоичной системе счисления. Результирующая строка состоит из символов «0»–«1» и не содержит ведущих нулей (за исключением представления числа 0). Используя эту функцию, получить двоичные представления двух данных чисел.

16. Описать функцию DecToHex(N) строкового типа, возвращающую строковое представление целого неотрицательного числа N в 16-ричной системе счисления. Результирующая строка состоит из символов «0»–«9», «A»–«F» и не содержит ведущих нулей (за исключением представления числа 0). Используя эту функцию, получить 16-ричные представления пяти данных чисел.

17. Описать функцию BinToDec(S) целого типа, определяющую целое неотрицательное число по его строковому представлению S в двоичной системе счисления. Параметр S имеет строковый тип, состоит из символов «0»–«1» и не содержит ведущих нулей (за исключением значения «0»). Используя эту функцию, вывести два числа, для которых даны их двоичные представления.

18. Описать функцию LPos(S0, S, L) целого типа, возвращающую номер позиции, начиная с которой в строке S содержится L-е вхождение

подстроки S0 (L > 0). Если количество вхождений S0 в строке S меньше L, то функция возвращает 0. Считать, что перекрывающихся вхождений подстрок S0 строка S не содержит. Вывести значения этой функции для двух данных троек: S0, S и L.

19. Описать функцию DecToOct(N) строкового типа, возвращающую строковое представление целого неотрицательного числа N в восьмеричной системе счисления. Результирующая строка состоит из символов «0»–«7» и не содержит ведущих нулей (за исключением представления числа 0). Используя эту функцию, получить восьмеричные представления двух данных чисел.

20. Описать функцию OctToDec (N) целого типа, определяющую целое неотрицательное число по его строковому представлению S в восьмеричной системе счисления. Параметр S имеет строковый тип, состоит из символов «0»–«7» и не содержит ведущих нулей (за исключением значения «0»). Используя эту функцию, вывести два числа, для которых даны их восьмеричные представления.

21. Описать процедуру UpCaseRus(S), преобразующую все строчные русские буквы строки S в прописные (остальные символы строки S не изменяются). Строка S является входным и выходным параметром. Используя процедуру UpCaseRus, преобразовать две данные строки.

22. Описать функцию DecToHex(N) строкового типа, возвращающую строковое представление целого неотрицательного числа N в 16-ричной системе счисления. Результирующая строка состоит из символов «0»–«9», «A»–«F» и не содержит ведущих нулей (за исключением представления числа 0). Используя эту функцию, получить 16-ричные представления пяти данных чисел.

23. Описать функцию WordK(S, K) строкового типа, возвращающую K-е слово строки S (словом считается набор символов, не содержащий пробелов и ограниченный пробелами или началом/концом строки). Если количество слов в строке меньше K, то функция возвращает пустую строку. Используя эту функцию, выделить из данной строки S слова с данными номерами K1, K2, K3.

Контрольные вопросы

1 Что такое обработка исключений?
2 Для чего в программе предусмотрена обработка исключений?
3 Перечислите несколько часто встречаемых типов исключений.
4 Какая конструкция используется для обработки исключений?
5 Структура оператора try-catch-finally.
6 Порядок выполнения блока try...catch..finally.
7 Сколько операторов catch может быть в программе и в каком порядке их надо ставить?
8 Для чего используется блок finally и является ли он обязательным?

9 Какие еще есть способы обработки исключений?
10 Когда необходимо вызывать исключения в программе?
11 Как вызываются исключения, описывающие ошибку?

Используемая литература

1. Гниденко, И. Г. Технология разработки программного обеспечения: учеб. пособие для СПО / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. — М.: Издательство Юрайт, 2017.
2. Шарп Джон Ш26 Microsoft Visual C#. Подробное руководство. 8-е изд. — СПб.: Питер, 2017.
3. Васильев А.Н. Программирование на C# для начинающих. Основные сведения. – Москва: Эксмо, 2018.
4. Васильев А.Н. Программирование на C# для начинающих. Особенности языка. – Москва: Эксмо, 2019.
5. http://msdn.microsoft.com/ru-ru/library/67ef8sbd.aspx.