Программирование на Microsoft Visual C# 2013
I семестр

Лабораторная работа №1. Первая программа на C#

Запускаем Microsoft Visual C# 2013 Express -> Меню -> Файл -> Новый проект -> В списке выбираем Console Application (Консольное приложение) -> называем проект и жмем ОК. В итоге будет создан проект с одним файлом с расширением *.cs в котором мы и будем писать нашу программу. В этом файле уже есть несколько шаблонных строк кода.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HelloWorld
{
 class Program
 {
  static void Main(string[] args)
  {
  }
 }
}
```

И это уже готовая программа, только она не делает ничего.
Добавим несколько строк кода между фигурными скобками функции Main, чтобы вывести «Привет» на экран.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HelloWorld
{
 class Program
 {
  static void Main(string[] args)
  {
    Console.WriteLine("Hello, World!");
    Console.ReadKey();
  }
 }
}
```

Чтобы скомпилировать и запустить программу жмем F5. В итоге, видим на экране наше приветствие. Детально код разбирать пока не будем, скажем только, что мы имеем класс Program, внутри которого объявлена статическая функция (метод) Main, что делает его главным классом приложения, и такой класс может быть только один. Функция Main является точкой входа программы, и она должна обязательно быть в любой консольной программе.

Строка Console.WriteLine("Hello, World!"); выводит сообщение на экран. После этого, чтобы программа не закрылась, и мы смогли увидеть результат, командой Console.ReadKey(); мы ожидаем нажатия клавиши пользователем. После нажатия клавиши приложение соответственно закрывается.

Лабораторная работа №2 Переменные, типы данных, константы.

Переменная – это именованная область памяти. В переменную можно записывать данные и считывать. Данные, записанные в переменной, называются значением переменной.

Си-шарп – язык жесткой типизации. Каждая переменная должна быть определенного типа данных. Ниже, в таблице наведены встроенные типы данных языка Си-шарп:

Тип	Область значений	Размер
sbyte	-128 до 127	Знаковое 8-бит целое
byte	0 до 255	Беззнаковое 8-бит целое
char	U+0000 до U+ffff	16-битовый символ Unicode
bool	true или false	1 байт*
short	-32768 до 32767	Знаковое 16-бит целое
ushort	0 до 65535	Беззнаковое 16-бит целое
int	-2147483648 до 2147483647	Знаковое 32-бит целое
uint	0 до 4294967295	Беззнаковое 32-бит целое
long	-9223372036854775808 до 9223372036854775807	Знаковое 64-бит целое
ulong	0 до 18446744073709551615	Беззнаковое 64-бит целое
float	±1,5*10-45 до ±3,4*1033	4 байта, точность — 7 разрядов
double	±5*10-324 до ±1,7*10306	8 байтов, точность — 16 разрядов
decimal	(-7,9 * 1028 до 7,9 * 1028) / (100–28)	16 байт, точность — 28 разрядов

*Здесь нет ошибки. Оперативная память - массив байтов, где каждый байт имеет уникальный адрес. Для bool достаточно одного бита: 0 - false, 1 - true, но минимальная адресуемая сущность - байт, поэтому ненулевой байт считается за истину, нулевой - ложью.

Для того, чтобы использовать переменную, ее сначала нужно объявить:

```
static void Main(string[] args)
{
  int a; // объявляем переменную a типа int
  a = 5; // записываем в переменную a число 5
  int b, c; // объявить можно сразу несколько переменных через запятую
  bool d; // объявляем переменную d типа bool
  d = true; // записываем в переменную d значение true (истина)
  long e = 10; // при объявлении переменной можно сразу же задавать ей значение, это называется инициализацией
  float f = 5.5f; // чтобы записать число с плавающей точкой типа float, нужно после значения добавлять суффикс f.
  char g = 'g'; // объявление символьной переменой g с ее инициализацией значением символа 'g'
}
```

При использовании переменной, в которую не было записано значение, компилятор выдаст ошибку "Use of unassigned local variable [variableName]".

```
static void Main(string[] args)
{
  int a;
  Console.WriteLine(a); //ошибка
}
```

Язык Си-шарп чувствительный к регистру символов. Переменные max и Max это не одно и то же. Не забывайте этого, чтобы не иметь лишних проблем. Имя переменной должно отображать суть данных, которые она отображает. Не стоит называть переменные ни о чем не говорящими именами типа a, b, c. Используйте английские слова. Высота – height, возраст – age и т. д. НИКОГДА не используйте кириллические символы в именах переменных.

Преобразование встроенных типов данных

Переменные одного типа можно преобразовывать в переменные другого типа. Преобразование бывает явным и неявным. Неявное преобразование выполняет компилятор.

Пример неявного преобразования:

```
static void Main(string[] args)
{
  int a = 35;
  short b = 10;
  a = b; // неявное преобразование. Так как int большего размера, чем short – утери данных не будет
  b = a; // ошибка компиляции, нельзя тип большего размера неявно преобразовать в тип меньшего размера
}
```

При явном преобразовании необходимо непосредственно перед переменной, которую вы хотите преобразовать, указать в скобках тип, к которому приводится переменная.

Пример явного преобразования:

```
static void Main(string[] args)
{
  int a = 35000;
  short b = 10;
  b = (short) a; // в этом случае уже ошибки не будет. Так как максимальное значение типа short 32767, здесь будет утеря данных.
}
```

Константы

Константа – это переменная, значение которой нельзя изменить. Константы используются для гарантирования того, что данные в этой переменной не изменятся. Для того, чтобы объявить константу, перед обычным объявлением переменной нужно добавить ключевое слово const:

```
static void Main(string[] args)
{
  const int months = 12; // объявление константы
  months = 13; // ошибка компиляции
}
```

При объявлении константы она должна обязательно быть проинициализирована значением. Константы также делают ваш код более красивым, читаемым.

```
static void Main(string[] args)
{
  const int months = 12;
  const int monthSalary = 1024;
  int yearSalary = monthSalary * months;
}
```

Гораздо понятнее чем:

```
static void Main(string[] args)
{
  int yearSalary = 12 * 1024;
}
```

Константы могут быть двух типов: простые литералы и строчные:

```
static void Main(string[] args)
{
  Console.WriteLine(100); // 100 есть 100 и этого не изменить, это константа, а точнее числовой литерал
  Console.WriteLine("Hello!"); // строка “Hello!” является строчным литералом
}
```

Здесь стоит отличать константы от переменных-констант, последние имеют имя, как в примере с месяцами и зарплатой.

Ключевое слово var

Начиная с версии C# 3.0 в язык было добавлено ключевое слово var, которое позволяет создавать переменные без явного указания типа данных. Тип данных такой переменной определяет компилятор по контексту инициализации.

```
static void Main(string[] args)
{
  var number = 5; // number будет типа int
  var text ="some text"; // text будет типа string
  var number2 = 0.5; // number2 будет типа double
}
```

var сохраняет принцип строгой типизации в Си-шарп. Это означает, что после того, как для переменной уже был определен тип, в нее нельзя записать данные другого типа:

```
static void Main(string[] args)
{
  var number = 5;
  number = "some text"; // ошибка, number определен как int
}
```

Ключевое слово var следует использовать в первую очередь с LINQ выражениями (при работе с базами данных)

```
static void Main(string[] args)
{
 var query = from s in bdContext.Students selects;
}
```

Ключевое слово var имеет ограничения по его использованию - var не может быть в качестве:

- поля класса
- аргумента функции
- возвращаемого типа функции
- переменной, которой присваивается null

Нововведение var является достаточно противоречивым среди разработчиков на C#, некоторые используют его где только возможно, другие его избегают (код становится плохо читаемым).

Ссылочные типы

Все типы данных, о которых мы говорили выше, являются структурными. Также существуют ссылочные типы. Из базовых типов к ссылочным относятся object и string. Тип object является базовым для всех остальных типов данных. Типу string соответствует строка символов Unicode.

Пример использования типа string.

```
static void Main(string[] args)
{
  string hello = "Hello!";
  Console.WriteLine(hello);
}
```

Структурные типы данных в Си-шарп хранятся в стеке. Для этих данных зарезервирована область в стеке.

Стек — это структура данных, которая сохраняет элементы по принципу «последним пришёл — первым вышел». Примером из жизни служит стопка тарелок. Скорость работы со стеком гораздо выше, чем с оперативной памятью, следовательно, использование стека повышает скорость работы программы.

Ссылочные типы хранятся в куче.

Куча — это область динамической памяти, которая выделяется приложению для хранения данных (например объектов). Доступ к данным в куче осуществляется медленнее, чем к стеку. Переменные ссылочных типов хранят ссылки на данные. К ссылочным типам относятся классы, интерфейсы, делегаты и массивы.

Задание. Создайте новый проект или откройте предыдущий, объявите несколько переменных различных типов, примените явное и неявное преобразование. Создайте константную переменную, попробуйте изменить ее значение.

Лабораторная работа №3. Арифметические и логические операции.

Все операции делятся на два типа: унарные и бинарные. К унарным относятся операции, в которых участвует один операнд. В бинарных операциях – два операнда. Операнд – это данные, которые принимают участие в операции. Например, оператор сложения «+» – бинарный 2+3, здесь операндами являются числа 2 и 3. Список бинарных арифметических операций приведен в таблице:

Операция	Запись
Сложение	a + b
Вычитание	a - b
Деление	a / b
Умножение	a * b
Нахождение остатка от деления	a % b

При делении двух целых чисел результатом также будет целое число. Например, при делении 9/5 результатом будет число 1. Чтобы получить точный результат с десятичной точкой, нужно чтобы делимое и/или делитель были типа float или double. Например, при делении 9 / 5f (суффикс f указывает, что данная константа типа float) результатом будет 1.8. Оператор «%» возвращает остаток от деления. Результатом операции 9 % 5 будет 4. Примером применения оператора «%» может быть процесс проверки числа на четность. Для этого мы ищем остаток от деления числа на 2. Если число четное, результатом будет 0, если нечетное – 1.

Чтобы повысить приоритет операции, используются скобки, как и в обычной арифметике.

2+2*2=6

(2+2)*2=8

При использовании оператора «+» для строк, он выполняет операцию конкатенации. Конкатенация – объединение нескольких объектов (например, строк) в один.

```
static void Main(string[] args)
{
  string str1 = "Hello", str2 = "World";
  Console.WriteLine(str1 + ", " + str2); //выводит на экран "Hello, World"
}
```

Унарные операторы в Си-шарп

Унарных арифметических операторов в Си-шарп есть всего два: инкрементация «++» и декрементация «--»; Инкрементация увеличивает операнд на единицу, а декрементация - уменьшает на единицу.

```
static void Main(string[] args)
{
  int a = 0, b = 5;
  a++; // a=1;
  b--; // b=4
}
```

Инкрементация и декрементация может быть префиксной и постфиксной. При префиксной форме оператор стоит перед операндом, а при постфиксной - после. Префиксная форма сначала увеличивает (уменьшает) значение, и после этого выполняются остальные действия, а при постфиксной форме наоборот - сначала выполнятся все действия, а после увеличится (уменьшится) значение:

```
static void Main(string[] args)
{
  int a = 2, b = 3, c, d = 3;
  c = a + ++b; // c = 6, сначала инкремент, потом сложение
  c = a + d++; // c = 5, сначала сложение, потом инкремент*
}
```

Разница между префиксной и постфиксной формах в том, когда выполнится инкремент - в начале или после вычисления всего выражения. То есть, в строке "c = a + d++;" сначала суммируются a и d, потом результат (5) присваивается c (вычисление выражения заканчивается здесь), и уже дальше увеличивается d;

Везде где можно использовать инкрементацию/декрементацию стоит это делать, так как она работает быстрее оператора сложения/вычитания.

В Си-шарп также есть возможность использования краткой формы выражения:

```
static void Main(string[] args)
{
  int a = 2, b = 3;
a += b; // равноценно выражению a = a + b;
  a -= b; // равноценно выражению a = a - b;
  a *= b; // равноценно выражению a = a * b;
  a /= b; // равноценно выражению a = a / b;
  a %= b; // равноценно выражению a = a % b;
}
```

Класс Math

В классе Math собраны все основные тригонометрические функции, функция возведение числа в степень, нахождение квадратного корня и другие.

Для возведения числа в степень, используется функция Pow([число], [степень]);

```
static void Main(string[] args)
{
  float a, b = 9;
```

```
  a = (float) Math.Pow(b, 2); // возводим переменную b в степень 2. Pow() возвращает результат в типе данных double, поэтому мы тут применили явное преобразование. Конечно, можно было обойтись без преобразования, объявив переменную a типа double
  Console.WriteLine(a); // выводит на экран число 81
  Console.ReadKey();
}
```

Для нахождения квадратного корня служит функция Sqrt([число]); возвращаемый результат также в типе данных double

```
static void Main(string[] args)
{
  double a, b = 9;
  a = Math.Sqrt(b);
  Console.WriteLine(a); // выводит на экран число 3
  Console.ReadKey();
}
```

Для нахождения косинуса и синуса используются cos([угол в радианах]) и sin([угол в радианах]) соответственно.

180 [градусов] = пи [радиан].

Чтобы перевести градусы в радианы, необходимо значение в градусах умножить на Пи и разделить на 180. Число Пи объявлено константой в классе Math.

```
static void Main(string[] args)
{
  double a;
  a = Math.Cos(60 * Math.PI / 180); // переводим 60 градусов в радианы
  Console.WriteLine(a); // выводит на экран 0.5
  a = Math.Sin(60 * Math.PI / 180);
  Console.WriteLine(a); // выводит на экран 0.866…
  Console.ReadKey();
}
```

Логические операторы в Си-шарп

Логические операторы в Си-шарп служат для работы с логическим типом данных (bool), который может принимать только два значения – true или false. Их можно разделить на две категории: простые логические операторы и операторы сравнения.

В Си-шарп есть следующие логические операторы:

! – оператор «НЕ» является унарным и возвращает противоположное значение операнда.

```
static void Main(string[] args)
{
  bool a, b = true, c = false;
  a = !b; // a = false
  a = !c; // a = true
}
```

|| - оператор «ИЛИ» является бинарным и возвращает false только тогда, когда оба операнда равны false, в остальных случаях результат будет true;

```
static void Main(string[] args)
{
  bool a, bTrue = true, bFalse = false;
  a = bFalse || bFalse; // a = false
  a = bFalse || bTrue; // a = true
  a = bTrue || bFalse; // a = true
```

```
  a = bTrue || bTrue; // a = true
}
```

&& - оператор «И» является бинарным и возвращает true только тогда, когда оба операнда равны true, в остальных случаях результат будет false;

```
static void Main(string[] args)
{
  bool a, bTrue = true, bFalse = false;
  a = bFalse && bFalse; // a = false
  a = bFalse && bTrue; // a = false
  a = bTrue && bFalse; // a = false
  a = bTrue && bTrue; // a = true
}
```

К операторам сравнения относятся:

Оператор	Название
>	больше
<	меньше
>=	больше или равно
<=	меньше или равно
==	равно
!=	неравно

```
static void Main(string[] args)
{
  bool a;
  int b = 2, c = 3, d = 2;
  a = b > c; // a = false
  a = b < c; // a = true
  a = b >= c; // a = false
  a = b >= d; // a = true
  a = b == c; // a = false
  a = b == d; // a = true
  a = b != c; // a = true
}
```

Задание. Есть прямоугольный треугольник с катетами a=5, b=7. Найдите площадь треугольника, s=?. Используя теорему Пифагора, найдите длину гипотенузы, c=?. Кроме этого, найдите длину гипотенузы еще и с помощью теоремы косинусов.

Лабораторная работа №4 Условные операторы в Си-шарп. Тернарный оператор.

Условные операторы служат для ветвления программы. В зависимости от некоторого условия выполняется тот или другой набор команд. В Си-шарп есть три условных оператора: «if-else», «switch» и «?:» - тернарный оператор.

Оператор «if-else»

Данный оператор имеет следующую структуру:

```
if ([условное выражение])
      {
  Блок кода, который нужно выполнить при удовлетворении условия,   [условное выражение] = true (истина)
}
else
```

```
{
  Блок кода, который нужно выполнить при неудовлетворении условия,   [условное выражение] = false (ложь)
}
```

Часть else не является обязательной и может отсутствовать.
Пример использования оператора «if-else» в программе, которая проверяет вводимое число на чётность:

```
static void Main(string[] args)
{
  int a;
  Console.WriteLine("Введите число:");
  a = Convert.ToInt32(Console.ReadLine()); // вводим данные с клавиатуры*
  if (a % 2 == 0) //проверяем число на чётность путем нахождения остатка от деления числа на 2
  {
    Console.WriteLine("Число " + a + " - чётное");
  }
  else
  {
    Console.WriteLine("Число " + a + " - нечётное");
  }
  Console.ReadKey();
}
```

* Функция Console.ReadLine() позволяет ввести данные с клавиатуры. Данные вводятся как строка, а так как нужно число, мы преобразовываем ее в числовой тип. Для преобразования мы используем функцию Convert.ToInt32().

Если после if или else необходимо выполнить лишь одну команду, фигурные скобки можно опускать:

```
if ([условное выражение])
[команда1] // команда1 выполнится лишь если условное выражение истинно
[команда2]// команда2 выполнится в любом случае
```

Оператор if может иметь несколько условий:

```
if ([логическое выражение1])
{блок1}
else if ([логическое выражение2])
{блок2}
else
{блок3}
```

Пример программы, которая определяет, какое из двух введенных чисел больше:

```
static void Main(string[] args)
{
  int a, b;
  Console.WriteLine("Введите первое число:");
  a = Convert.ToInt32(Console.ReadLine());
  Console.WriteLine("Введите второе число:");
  b = Convert.ToInt32(Console.ReadLine());
  if (a > b)
    Console.WriteLine("Первое число больше второго");
  else if (a < b)
```

```
    Console.WriteLine("Второе число больше первого");
  else
    Console.WriteLine("Числа равны");

  Console.ReadKey();
}
```

Логическое выражение может быть сложнее. Здесь и используются логические операторы «!», «||» и «&&».

Пример программы которая дает совет, что делать, в зависимости от температуры на дворе:

```
static void Main(string[] args)
{
  int t;
  Console.WriteLine("Введите температуру во дворе");
  t = Convert.ToInt32(Console.ReadLine());
  if (t < -20 || t > 40) //если температура меньше -20 или больше 40
    Console.WriteLine("Вам лучше посидеть дома!");
  else
    Console.WriteLine("Можете идти гулять");
  Console.ReadKey();
}
```

Оператор switch

В некоторых случаях удобно использовать условный оператор «switch» вместо «if-else». Он имеет следующую структуру:

```
switch (выражение)
{
  case значение1:
    блок1;
    break;
  case значение2:
    блок2;
    break;
    ...
  case значениеN:
    блокN;
    break;
  default:
    блокN+1;
    break;
}
```

Выражение сравнивается последовательно со значениями. Если выражение равно значению – выполняется соответственный блок кода и при достижении ключевого слова break оператор switch заканчивает работу. Если выражение не будет соответствовать ни одному значению, тогда выполнится блок после default.

Пример программы с использованием switch, которая выводит на экран название дня недели соответственно вводимому порядковому номер дня:

```
static void Main(string[] args)
{
  int a;
  Console.WriteLine("Введите порядковый номер дня недели:");
  a = Convert.ToInt32(Console.ReadLine());
```

```
  switch (a)
  {
    case 1:
      Console.WriteLine("Понедельник");
      break;
    case 2:
      Console.WriteLine("Вторник");
      break;
    case 3:
      Console.WriteLine("Среда");
      break;
    case 4:
      Console.WriteLine("Четверг");
      break;
    case 5:
      Console.WriteLine("Пятница");
      break;
    case 6:
      Console.WriteLine("Суббота");
      break;
    case 7:
      Console.WriteLine("Воскресенье");
      break;
    default :
      Console.WriteLine("Ошибка");
      break;
    }
    Console.ReadKey();
}
```

Тернарный оператор «?:»

Этот оператор используется для сокращения объема кода. Им можно заменять простые по сложности операторы if-else. Тернарный оператор имеет такую структуру:

логическое выражение ? выражение1 : выражение2

Сначала вычисляется логическое выражение. Если оно истинно, то вычисляется выражение1, в противном случае - вычисляется выражение2. Пример использования тернарного оператора «?:» в той же программе для проверки числа на чётность:

```
static void Main(string[] args)
{
  int a;
  Console.WriteLine("Введите число:");
  a = Convert.ToInt32(Console.ReadLine());
  Console.WriteLine( a % 2 == 0 ? "Число чётное" : "Число нечётное" );
  Console.ReadKey();
}
```

«?:» также можно использовать для присваивания значений. Пример программы, которая находит большее число из двух вводимых:

```
static void Main(string[] args)
{
  int a, b, max;
  Console.WriteLine("Введите первое число:");
  a = Convert.ToInt32(Console.ReadLine());
```

```
  Console.WriteLine("Введите второе число:");
  b = Convert.ToInt32(Console.ReadLine());
  max = a > b ? a : b;
}
```

Задание: 1) С клавиатуры вводятся два числа – количество забитых голов хозяевами и гостями в футбольном матче. Вывести на экран результат игры – победили хозяева/гости/ничья.

2) Напишите программу, которая будет проверять число на кратность 3-м и 7-ми (это числа 21, 42, 63…). Вывести на экран соответствующее сообщение.

Лабораторная работа №5. Массивы в Си-шарп. Класс List.

Массив – это набор однотипных данных, которые располагаются в памяти последовательно друг за другом. Доступ к элементам массива осуществляется по индексу (номеру) элемента. Массив может содержать элементы любого типа данных, можно даже создавать массив массивов (ступенчатый массив). Количество элементов в массиве называется размером массива. Массивы относятся к ссылочным типам данных.

Массивы в Си-шарп могут быть одномерными и многомерными. Одномерные массивы.

Одномерный массив по-другому еще называется вектором, и для доступа к его элементам используется только один индекс. Выглядит вектор примерно так:

В Си-шарп объявление массива имеет такую структуру:

тип[] имя_массива = new тип[размер массива];

Пример:

```
int[] array = new int[5]; // создаем массив целых чисел
string[] seasons = new string[4] {"зима","весна","лето","осень"}; //объявление массива строк и его инициализация значениями.
```

Если происходит инициализация, оператор new можно упускать:

```
string[] seasons = {"зима","весна","лето","осень"}; //корректно
```

Доступ к элементам осуществляется по индексу. Следует помнить, что индексация начинается с нуля – первый элемент массива имеет индекс 0, а последний n-1, где n – размер массива.

```
static void Main(string[] args)
{
  int[] numbers = new int[5];
  numbers[0] = 5;
  numbers[1] = 2;
  numbers[4] = 3;
  numbers[5] = 2; // ошибка, индекс вне рамок массива
}
```

В качестве массива можно представить, например, список студентов в группе (имена), показатели температуры воздуха за последние несколько дней и так далее.

Многомерные массивы

Одним из случаев многомерного массива служит двумерный массив (матрица). В матрице для доступа к элементам необходимо использовать два индекса.

a00	a01	…	a0m-1
a10	a11	…	a1m-1
…	…	…	…
an-10	an-11	…	an-1m-1

Количеством индексов, используемых для доступа к элементам массива называется размерность массива.

int[,] numbers1 = new int[2, 2]; // объявление двумерного массива
int[,,] numbers2 = new int[2, 2 ,3]; // объявление трехмерного массива
int[,] numbers3 = new int[3, 2] { {6, 0},
{5, 7},
{8, 9} }; // инициализация двумерного массива

Элементу массива numbers1 с координатами 1,1 присвоим значение 8: numbers1[1, 1] = 8;

Приведу немного бредовый пример данных, которые можно было бы представить в качестве двумерного массива:

Есть матрица 7*4. Строки, которых семь, будут соответствовать дням недели, а 4 столбца - времени дня (00:00, 06:00, 12:00, 18:00). В качестве данных будет температура воздуха.

Значение температуры с координатами 2,3 будет соответствовать температуре в среду в 18:00. На практике, конечно, такие данные представлялись бы в другом виде, но для примера подойдет.

Ступенчатые (jagged) массивы в Си-шарп

Ступенчатый (jagged) массив – это массив массивов. В нем длина каждого массива может быть разной. Примерно это выглядит так:

a00	a01	a02	
a10	a11	a12	a13
a20	a21		
a30	a31	a32	a33

Пример объявления ступенчатого массива:

```
static void Main(string[] args)
{
  int[][] array = new int[3][]; // объявляем массив, который содержит 3 массива
  array [0] = new int[3]; //создание внутреннего массива
  array [1] = new int[2];
  array [2] = new int[5];
}
```

Доступ к элементам осуществляется по тому же принципу, как и с многомерными массивами, только тут уже участвуют две пары квадратных скобок (продолжение примера выше):

```
array [0][1] = 5;
array [1][1] = 8;
array [1][2] = 5; // ошибка, индекс «2» вне границ массива
```

Свойство Length

Все массивы являются объектами и у них есть некоторые свойства. Самым полезным для нас будет свойство Length, которое возвращает количество элементов в массиве (во всех размерностях)

```
static void Main(string[] args)
{
  int[] numbers = new int[5];
  int size = numbers.Length; // size = 5
}
```

Класс List

Класс List служит для работы со списками, о чем и говорит его название. Это такой «навороченный» массив. Главное отличие от простого массива в том, что он динамический – вы можете вставлять и удалять элементы в любое время, в то время как в простом массиве размер указывается при создании и сделать его больше или меньше нельзя.

```
static void Main(string[] args)
{
  List<string> teams = new List<string>(); // создание списка
  teams.Add("Barcelona"); // добавление элемента
  teams.Add("Chelsea");
  teams.Add("Arsenal");
  List<string> teams2 = new List<string>() {"Dynamo", "CSKA" }; // инициализация
}
```

Добавление элементов

Для добавления элементов в список в нем реализовано несколько методов:

Метод	Описание
Add([элемент])	добавляет элемент в конец списка
AddRange([список элементов])	добавляет в конец списка элементы указанного списка
Insert([индекс],[элемент])	вставляет элемент на позицию соответствующую индексу, все элементы «правее» будут сдвинуты на одну позицию
InsertRange([индекс], [список элементов])	то же самое, только вставляется множество элементов

Удаление элементов

Метод	Описание
Remove([элемент])	удаляет первое вхождение указанного элемента из списка
RemoveRange([индекс], [количество])	удаляет указанное количество элементов, начиная с указанной позиции

RemoveAt([индекс])	удаляет элемент, который находится на указанной позиции
Clear()	удаляет все элементы списка

Свойство Count соответствует свойству обычного массива – Length – количество элементов.

```
static void Main(string[] args)
{
  List<string> teams = new List<string>() { "Inter", "Milan", "Bayern", "Juventus"};
  teams.Insert(2,"Barcelona"); // вставляем в список элемент "Barcelona" на позицию 2
  teams.Remove("Milan"); // удаляем первое вхождение элемента "Milan" из списка
  List<string> newTeams = new List<string>() { "Liverpool", "Roma", "Borussia", "Valencia" };
  teams.AddRange(newTeams); // добавляем в конец списка элементы списка newTeams
}
```

Стоит помнить, что простые массивы работают быстрее, чем списки List. Если в вашей программе не особо важна производительность и вы не работаете с большими количествами данных, то удобнее использовать список, в противном случае нужно использовать простые массивы.

Задание. Работать с массивами без использования циклов в большинстве случаев нет смысла. О циклах мы поговорим в следующем уроке. А так, пока можете создать разные типы массивов, записать что-то, вывести на экран некоторые элементы массива, попробуйте получить доступ до элемента вне рамок массива. Создайте список, попробуйте добавить элемент в конец списка, вставить элемент по индексу и так далее.

Лабораторная работа №6. Циклы в Си-шарп. Операторы break и continue.

Циклы служат для многократного повторения некоторого фрагмента кода.
В Си-шарп есть четыре оператора циклов: for, while, do-while, foreach.
Цикл for

Этот цикл используется тогда, когда наперед известно, сколько повторений нужно сделать. Он имеет следующую структуру:

```
for (инициализация счетчика; условие продолжения; итерация)
{
  //блок кода, который будет повторяться
}
```

Пример программы, которая выводит на экран числа 0, 1, 2, 3, 4:

```
static void Main(string[] args)
{
  for (int i = 0; i < 5; i++) // цикл выполнится 5 раз
  {
    Console.WriteLine(i);
  }
}
```

Сначала происходит создание и инициализация счетчика, i=0. Дальше идет проверка условия (i < 5), если результат будет «истина», то дальше выполняется блок кода в теле цикла. В конце итерации происходит изменение значения счетчика (в данном примере увеличение на единицу). После этого вновь происходит проверка условия и так далее. Когда условие будет «ложь», цикл работу завершит.

Пример программы, которая находит и выводит на экран сумму элементов массива:

```
static void Main(string[] args)
{
  int[] numbers = { 4, 7, 1, 23, 43 };
  int s = 0;
  for (int i = 0; i < numbers.Length; i++)
  {
    s += numbers[i];
  }
  Console.WriteLine(s);
  Console.ReadKey();
}
```

Пример цикла for, когда счетчик уменьшается после каждой итерации:

```
for (int i = 5; i > 0; i--) //выполнится 5 раз
{
  Console.WriteLine(i);
}
```

Счетчик можно изменять не только на единицу. Пример программы, которая выводит чётные числа (по число 50):

```
for (int i = 0; i <= 50; i+=2) //выполнится 26 раз
{
  Console.WriteLine(i);
}
```

Цикл while

Слово while переводится, как «пока», что хорошо его характеризует. Он продолжает выполнятся до тех пор, пока «истинно» некоторое условие. Он имеет такую структуру:

```
while (условие продолжения)
{
  //блок кода, который будет повторяться
}
```

Сначала проверяется условие, а дальше выполняется блок кода.

Пример той же программы, которая выводит на экран числа 0, 1, 2, 3, 4:

```
int i = 0;
while (i < 5)
{
  Console.WriteLine(i);
  i++;
}
```

Цикл может выполнятся «вечно», если задать всегда истинное условие:

```
while (true)
{
  Console.WriteLine("Вечный цикл");
}
```

Цикл do-while

Этот тот же цикл while, только здесь сначала выполняется блок кода, а уже потом идет проверка условия. Это гарантирует хотя бы один проход цикла.

```
do
{
//блок кода, который будет повторяться
}
while (условие продолжения);
```

Пример программы, которая не завершит работу, пока с клавиатуры не введут число 5:

```
static void Main(string[] args)
{
  int number;
  do
  {
    Console.WriteLine("Введите число 5");
    number = Convert.ToInt32(Console.ReadLine());
  }
  while (number != 5);
}
```

Оператор break

Из любого цикла можно досрочно выйти, используя оператор break. Использовать данный оператор есть смысл после удовлетворения некоторого условия, иначе цикл завершится на первой итерации.

Пример программы, которая проверяет, есть ли в массиве число кратное 13-ти. Найдя такое число, нет смысла дальше проверять остальные элементы массива, и здесь мы используем оператор break:

```
static void Main(string[] args)
{
  int[] numbers = { 4, 7, 13, 20, 33, 23, 54 };
  bool b = false;
  for (int i = 0; i < numbers.Length; i++)
  {
    if (numbers[i] % 13 == 0)
    {
      b = true;
      break;
    }
  }
  Console.WriteLine(b ? "В массиве есть число кратное 13" : "В массиве нет числа
кратного 13");
  Console.ReadKey();
}
```

Оператор continue

Данный оператор позволяет перейти к следующей итерации, не завершив до конца текущую.

Пример программы, которая находит сумму нечетных элементов массива:

```
static void Main(string[] args)
{
  int[] numbers = { 4, 7, 13, 20, 33, 23, 54 };
  int s = 0;
  for (int i = 0; i < numbers.Length; i++)
  {
    if (numbers[i] % 2 == 0)
      continue; //переход к следующей итерации
    s += numbers[i];
    }
  Console.WriteLine(s);
  Console.ReadKey();
}
```

Задание. 1. Вывести на экран 20 элементов последовательности 1, 4, 7, 10, 13…

2. Напишите программу, которая будет «спрашивать» правильный пароль, до тех пор, пока он не будет введен. Правильный пароль пусть будет «root».

3. Дано два массива одинаковой длины (по 10 элементов). Создайте третий массив, который будет отображать сумму первых двух массивов. Первый элемент третьего массива равен сумме первых элементов двух первых массивов и так далее.

Лабораторная работа №7. Оператор цикла foreach в Си-шарп

Оператор цикла foreach в Си-шарп служит для перебора элементов коллекции. К коллекциям относятся массивы, списки List и пользовательские классы коллекций. В данном операторе не нужно создавать переменную-счетчик для доступа к элементам коллекции, в отличии от других циклов. Оператор foreach имеет следующую структуру:

```
foreach ([тип] [переменная] in [коллекция])
{
  //тело цикла
}
```

Пример программы, в которой находится сумма элементов массива с использованием цикла foreach:

```
static void Main(string[] args)
{
  int[] numbers = { 4, 7, 13, 20, 33, 23, 54 };
  int s = 0;

  foreach (int el in numbers)
  {
    s += el;
  }
  Console.WriteLine(s);
  Console.ReadKey();
}
```

На каждой итерации в переменную el последовательно записывается элемент коллекции. На первой итерации значение переменной el равно “4” , на второй итерации - “7” и т.д. Как и в других циклах, в foreach можно использовать операторы break и continue. Данный оператор стоит использовать для получения (чтения) данных из коллекции. Не стоит использовать его для добавления или удаления элементов из коллекции, иначе вы получите исключение (ошибку) Collection was modified; enumeration operation may not execute.

Задание. Используя цикл foreach, выведите на экран все элементы массива целых чисел, которые больше 20 и меньше 50.

Лабораторная работа №8. Функции в Си-шарп. Оператор return

Функция являет собой небольшую подпрограмму. Если просто программа - это решение какой-то прикладной задачи, то функция – это тоже решение, только уже в рамках программы и, соответственно, она выполняет задачу «попроще». Функции позволяют уменьшить размер программы за счет того, что не нужно повторно писать какой-то фрагмент кода - мы просто вызываем сколько угодно и где нужно объявленную функцию. Функции в Си-шарп также называют методами. Между этими двумя понятиями разница небольшая, и тут мы будем использовать термин функция.
До этого, мы весь код писали в функции main. Функция main является главной функцией приложения и точкой входа программы. Любая функция в Си-шарп может быть объявлена только в рамках класса, так как C# - полностью объектно-ориентированный язык программирования (ООП). Объявление пользовательской функции внутри другой функции (например main) недопустимо. Объявление функции имеет следующую структуру:

```
[модификатор доступа] [тип возвращаемого значения] [имя функции] ([аргументы])
{
// тело функции
}
```

Модификатор доступа (области видимости) может быть public, private, protected, internal. Для чего это нужно будем говорить в отдельном уроке, а пока будем везде использовать public.

Между модификатором и типом может стоять ключевое слово static, что означает, что функция будет статичной. Подробно говорить о статичных функциях и переменных мы будем в отдельном уроке. Скажу только, что из статичной функции можно вызывать другие функции, если они тоже статичные. Главная функция main – всегда static. Функция может возвращать значение или не возвращать. Если функция, например, возвращает целое число, то нужно указать тип int. Если функция не возвращает никакого значения, то для этого используется ключевое слово void. Функции, которые не возвращают значение, еще называют процедурами.

Называть функции стоит так, чтобы имя отображало суть функции. Используйте глаголы или словосочетания с глаголами. Примеры: GetAge(), Sort(), SetVisibility().

Аргументы – это те данные, которые необходимы для выполнения функции. Аргументы записываются в формате [тип] [идентификатор]. Если аргументов несколько, они отделяются запятой. Аргументы могут отсутствовать.

Первая строка функции, где указываются тип, имя, аргументы и т.д. называется заголовком функции.

Пример функции, которая не возвращает значение. Напишем простую функцию, которая будет заменять в массиве строк указанное имя на другое. Данная функция будет принимать три аргумента: массив строк, имя, которое необходимо заменить и новое имя. Так как функция не будет возвращать значение, указываем тип void перед именем функции.

```
public static void ReplaceName(string[] names, string name, string newName)
{
  for (int i=0; i < names.Length; i++)
  {
    if (names[i] == name)
    names[i] = newName;
  }
}
```

Сама функция очень простая. Проходим в цикле по элементам и смотрим, равен ли элемент указанному имени. Если да, то заменяем его на новое имя. Функция написана, и теперь используем ее:

```
class Program
{
  public static void ReplaceName(string[] names, string name, string newName)
  {
    for (int i=0; i < names.Length; i++)
    {
      if (names[i] == name)
        names[i] = newName;
      }
    }
  static void Main(string[] args)
  {
    string[] names = { "Sergey", "Maxim", "Andrey", "Oleg", "Andrey", "Ivan",
"Sergey" };
```

```
    ReplaceName(names, "Andrey", "Nikolay"); // вызов функции. Все строки "Andrey" в массиве будут заменены на "Nikolay"
    ReplaceName(names, "Ivan", "Vladimir");
  }
}
```

После вызова функции два раза в этой программе, массив будет выглядеть так: "Sergey", "Maxim", " Nikolay ", "Oleg", " Nikolay ", " Vladimir ", "Sergey".

Пример функции, которая возвращает значения

Напишем функцию, которая будет находить максимальное число в массиве. Аргумент у этой функции будет один – массив целых чисел. Тип возвращаемого значения – целое число int.

```
public static int GetMax(int[] array)
{
  int max = array[0];
  for (int i = 1; i < array.Length; i++)
  {
    if (array[i] > max)
      max = array[i];
  }
  return max;
}
```

Логика функции проста. Создаем переменную max, в которую записываем первый элемент массива. Дальше в цикле сравниваем каждый элемент со значением в max, если элемент больше, чем max, то записываем в max этот элемент. В конце функции используем оператор return, чтобы вернуть наш результат.

Оператор return должен быть обязательно в функции, которая возвращает значение. Используем нашу функцию:

```
class Program
{
  public static int GetMax(int[] array)
  {
    int max = array[0];
    for (int i = 1; i < array.Length; i++)
    {
      if (array[i] > max)
        max = array[i];
    }
    return max;
  }
  static void Main(string[] args)
  {
    int[] numbers = { 3, 32, 16, 27, 55, 43, 2, 34 };
    int max;
    max = GetMax(numbers); //вызов такой функции можно использовать при присваивании значения
    Console.WriteLine(GetMax(numbers)); // вызов функции также можно использовать как аргумент при вызове другой функции. WriteLine() – тоже функция.
    Console.ReadKey();
  }
}
```

Оператор return

Когда встречается этот оператор, происходит выход из функции и код ниже (если он есть) выполняться не будет (например, в функцию передан такой аргумент, при котором нет смысла выполнять функцию). Он похож на оператор break, который используется для выхода из циклов. Этот оператор также можно использовать и в функциях, которые не возвращают значение. Оператор return допустимо использовать несколько раз в функции, но этого делать не рекомендуется.

Задание.1. Напишите функцию, которая будет менять в массиве целых чисел все элементы, которые равны указанному значению (аргумент) на противоположное значение по знаку. Например, все элементы массива которые равны 5, будут меняться на -5.

2. Напишите функцию, которая будет находить минимальное число из трех.

3. Напишите функцию, которая будет возвращать указанный элемент ряда Фибоначчи. Ряд Фибоначчи – это ряд, в котором каждый следующий элемент равен сумме двух предыдущих. 1 1 2 3 5 8 13 21… Функция принимает порядковый номер элемента, и возвращает соответствующий элемент.

Лабораторная работа №9 Работа со строками в Си-шарп. Класс String

Строки в Си-шарп - это объекты класса String, значением которых является текст. Для работы со строками в этом классе определено множество методов (функций) и в этом уроке мы рассмотрим некоторые из них.

Чтобы использовать строку, ее нужно сначала создать – присвоить какое-либо значение, иначе мы получим ошибку: "Использование локальной переменной "[имя переменной]", которой не присвоено значение". Объявим простую строку и выведем ее на экран:

```
static void Main(string[] args)
{
  string s = "Hello, World!";
  Console.WriteLine(s);
}
```

```
      static void Main(string[] args)
{
  string s;
  Console.WriteLine(s); // ошибка, строка не создана
}
```

Для объединения (конкатенации) строк используется оператор "+".

```
string s = "Hello," + " World!";
```

Оператор "[]" используется для доступа (только чтение) к символу строки по индексу:

```
string s = "Hello, World!";
char c = s[1]; // 'e'
```

Свойство Length возвращает длину строки.
Спецсимволы:
Символ "\" является служебным, поэтому, чтобы использовать символ обратного слэша необходимо указывать его дважды "\\".
Символ табуляции – "\t"
Символ перевода строки – "\r\n"
Двойные кавычки – "\""

Методы (функции) класса String для работы со строками в Си-шарп
Как проверить, пуста ли строка?
Метод IsNullOrEmpty() возвращает True, если значение строки равно null, либо когда она пуста (значение равно ""):

```
static void Main(string[] args)
{
  string s1 = null, s2 = "", s3 = "Hello";
  String.IsNullOrEmpty(s1); // True
  String.IsNullOrEmpty(s2); // True
  String.IsNullOrEmpty(s3); // False
}
```

Метод IsNullOrWhiteSpace() работает как и метод IsNullOrEmpty(), только возвращает True еще и тогда, когда строка представляет собой набор символов пробела и/или табуляции ("\t"):

```
static void Main(string[] args)
{
  string s1 = null, s2 = "\t", s3 = " ", s4 = "Hello";
  String.IsNullOrWhiteSpace(s1); // True
  String.IsNullOrWhiteSpace(s2); // True
  String.IsNullOrWhiteSpace(s3); // True
  String.IsNullOrWhiteSpace(s4); // False
}
```

Как проверить, является ли одна строка "больше" другой?

Для сравнивания строк используется метод Compare(). Суть сравнения строк состоит в том, что проверяется их отношение относительно алфавита. Строка "a" "меньше" строки "b", "bb" "больше" строки "ba". Если обе строки равны - метод возвращает "0", если первая строка меньше второй – "-1", если первая больше второй – "1":

```
static void Main(string[] args)
{
  String.Compare("a", "b"); // возвращает -1
  String.Compare("a", "a"); // возвращает 0
  String.Compare("b", "a"); // возвращает 1
  String.Compare("ab", "abc"); // возвращает -1
  String.Compare("Romania", "Russia"); // возвращает -1
  String.Compare("Rwanda", "Russia"); // возвращает 1
  String.Compare("Rwanda", "Romania"); // возвращает 1
}
```

Чтобы игнорировать регистр букв, в метод нужно передать, как третий аргумент true.

```
String.Compare("ab", "Ab"); // возвращает -1
String.Compare("ab", "Ab", true); // возвращает 0
```

Как перевести всю строку в верхний/нижний регистр?
Для этого используются методы ToUpper() и ToLower():

```
static void Main(string[] args)
{
  string s = "Hello, World";
  Console.WriteLine(s.ToUpper()); // выводит "HELLO, WORLD"
  Console.WriteLine(s.ToLower()); // выводит "hello, world"
  Console.ReadLine();
}
```

Как проверить, содержит ли строка подстроку?

Для проверки содержания подстроки строкой используется метод Contains(). Данный метод принимает один аргумент – подстроку. Возвращает True, если строка содержит подстроку, в противном случае – False. Пример:

```
static void Main(string[] args)
{
  string s = "Hello, World";
  if (s.Contains("Hello"))
    Console.WriteLine("Содержит");
  Console.ReadLine();
}
```

Данная программа выводит слово "Содержит", так как "Hello, World" содержит подстроку "Hello".

Как найти индекс первого символа подстроки, которую содержит строка?

Метод IndexOf() возвращает индекс первого символа подстроки, которую содержит строка. Данный метод принимает один аргумент – подстроку. Если строка не содержит подстроки, метод возвращает "-1". Пример:

```
static void Main(string[] args)
{
  string s = "Hello, World";
  Console.WriteLine(s.IndexOf("H")); // 0
  Console.WriteLine(s.IndexOf("World")); // 7
  Console.WriteLine(s.IndexOf("Zoo")); // -1
  Console.ReadLine();
}
```

Как узнать, начинается/заканчивается ли строка указанной подстрокой?

Для этого используются соответственно методы StartsWith() и EndsWith(), которые возвращают логическое значение. Пример:

```
static void Main(string[] args)
{
  string s = "Hello, World";
  Console.WriteLine(s.StartsWith("Hello")); // True
  Console.WriteLine(s.StartsWith("World")); // False
  Console.WriteLine(s.EndsWith("World")); // True
  Console.ReadLine();
}
```

Как вставить подстроку в строку, начиная с указанной позиции?

Метод Insert() используется для вставки подстроки в строку, начиная с указанной позиции. Данный метод принимает два аргумента – позиция и подстрока. Пример:

```
static void Main(string[] args)
{
  string s = "Hello World";
  Console.WriteLine(s.Insert(5,",")); // вставляет запятую на 5 позицию
  Console.ReadLine();
}
```

Как обрезать строку, начиная с указанной позиции?

Метод Remove() принимает один аргумент – позиция, начиная с которой обрезается строка:

```
static void Main(string[] args)
{
  string s = "Hello, World";
  Console.WriteLine(s.Remove(5)); // удаляем все символы, начиная с 5 позиции, на экран выведется "Hello"
```

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

В метод Remove() можно передать и второй аргумент – количество обрезаемых символов. Remove(3, 5) – удалит из строки пять символов начиная с 3-го.

Как получить подстроку из строки, начиная с указанной позиции?

Для этого используется метод Substring(). Он принимает один аргумент – позиция, с которой будет начинаться новая подстрока:

```
static void Main(string[] args)
{
  string s = "Hello, World";
  Console.WriteLine(s.Substring(7)); // получаем строку начиная с 7 позиции, выведет "World"
  Console.ReadLine();
}
```

В метод Substring(), как в метод Remove() можно передать и второй аргумент – длина подстроки. Substring (3, 5) – возвратит подстроку длиной в 5 символов начиная с 3-й позиции строки.

Как заменить в строке все подстроки указанной новой подстрокой?

Метод Replace() принимает два аргумента – подстрока, которую нужно заменить и новая подстрока, на которую будет заменена первая:

```
static void Main(string[] args)
{
  string s = "Hello, World, Hello";
  Console.WriteLine(s.Replace("Hello", "World")); //выведет "World, World, World"
  Console.ReadLine();
}
```

Как преобразовать строку в массив символов?

Метод ToCharArray() возвращает массив символов указанной строки:

```
static void Main(string[] args)
{
  string s = "Hello, World";
  char[] array = s.ToCharArray(); // элементы массива – 'H', 'e', 'l', 'l'…
}
```

Как разбить строку по указанному символу на массив подстрок?

Метод Split() принимает один аргумент - символ, по которому будет разбита строка. Возвращает массив строк. Пример:

```
static void Main(string[] args)
{
  string s = "Arsenal,Milan,Real Madrid,Barcelona";
  string[] array = s.Split(','); // элементы массива – "Arsenal", "Milan", "Real Madrid", "Barcelona"
}
```

Неизменяемые строки

Стоит знать, что объекты класса String представляют собой неизменяемые (Immutable) последовательности символов Unicode. Когда вы используете любой метод по изменению строки (например Replace()), он возвращает новую измененную копию строки, исходные же строки остаются неизменными. Так сделано потому, что операция создания новой строки гораздо менее затратна, чем операции копирования и сравнения, что повышает скорость работы программы. В Си-шарп также есть класс StringBuilder, который позволяет изменять строки. Об особенностях работы с ним мы поговорим в одном из дальнейших уроков.

Задание 1. Есть некий текст. Необходимо заменить в этом тексте все слова "Nikolay" на "Oleg".

2. Дан текст – «Сегодня мы с вами рассмотрели, как работать со строками в Си-шарп. Были описаны основные операторы и методы, которые используются для работы со строками». Обрежьте этот текст так, чтобы осталась только часть «Были описаны основные операторы и методы».

3. Дана строка, которая содержит имена пользователей, разделенные запятой – "Login1,LOgin2,login3,loGin4". Необходимо разбить эту строку на массив строк (чтобы отдельно были логины), и перевести их все в нижний регистр.

Лабораторная работа №10. Обработка исключений в Си-шарп. Оператор try-catch

В предыдущих работах, в некоторых программах мы не учитывали непредвиденные ситуации, которые могут приводить к ошибкам. Например, когда нам необходимо было ввести число. Если вместо числа мы ввели бы строку, то при конвертации этой строки в численный тип программа бы аварийно завершила работу, и мы получили бы ошибку. Такие ошибки и другие непредвиденные ситуации в Си-шарп называются исключениями.

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.

Для обработки исключений в Си-шарп используется оператор try-catch. Он имеет следующую структуру:

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

Работает это все очень просто. 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), или трассировку стека (StackTrace), которая поможет узнать место возникновения ошибки. 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. Ниже приведен пример программы, в которой используется обработка исключения некорректного формата данных:

```
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();
}
```

Типы исключений

Ниже приведены некоторые из часто встречаемых типов 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
Несколько блоков catch
Одному блоку try может соответствовать 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.

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
}
```

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.

Зачем блок finally?

Очень часто можно услышать вопрос, для чего нужен этот блок? Ведь, кажется, можно освободить ресурсы просто после оператора try-catch, без использования finally. А ответ очень прост. 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.

Операторы try-catch также могут быть вложенными. Внутри блока try либо catch может быть еще один try-catch.

Задание. Есть массив целых чисел размером 10. 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. Нужно предусмотреть случаи, когда были введены не числа, и когда одно из чисел, или оба больше размера массива.

Лабораторная работа №11. Работа с файлами в Си-шарп. Классы StreamReader и StreamWriter

Файл – это набор данных, который хранится на внешнем запоминающем устройстве (например на жестком диске). Файл имеет имя и расширение. Расширение позволяет идентифицировать, какие данные и в каком формате хранятся в файле.

Под работой с файлами подразумевается:

- создание файлов;
- удаление файлов;
- чтение данных;
- запись данных;
- изменение параметров файла (имя, расширение…);
- другое.

В Си-шарп есть пространство имен System.IO, в котором реализованы все необходимые нам классы для работы с файлами. Чтобы подключить это пространство имен, необходимо в самом начале программы добавить строку using System.IO. Для использования кодировок еще добавим пространство using System.Text;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
```

Как создать файл?

Для создания пустого файла, в классе File есть метод Create(). Он принимает один аргумент – путь. Ниже приведен пример создания пустого текстового файла new_file.txt на

диске D:

```
static void Main(string[] args)
{
  File.Create("D:\\new_file.txt");
}
```

Если файл с таким именем уже существует, он будет переписан на новый пустой файл.

Метод WriteAllText() создает новый файл (если такого нет), либо открывает существующий и записывает текст, заменяя всё, что было в файле:

```
static void Main(string[] args)
{
  File. WriteAllText("D:\\new_file.txt", "текст");
}
```

Метод AppendAllText() работает, как и метод WriteAllText() за исключением того, что новый текст дописывается в конец файла, а не переписывает всё что было в файле:

```
static void Main(string[] args)
{
  File.AppendAllText("D:\\new_file.txt", "текст метода AppendAllText ()"); //допишет текст в конец файла
}
```

Как удалить файл?

Метод Delete() удаляет файл по указаному пути:

```
static void Main(string[] args)
{
  File.Delete("d:\\test.txt"); //удаление файла
}
```

Кроме того, чтобы читать/записывать данные в файл с Си-шарп можно использовать потоки. Поток – это абстрактное представление данных (в байтах), которое облегчает работу с ними. В качестве источника данных может быть файл, устройство ввода-вывода, принтер.

Класс Stream является абстрактным базовым классом для всех потоковых классов в Си-шарп. Для работы с файлами нам понадобится класс FileStream (файловый поток). FileStream - представляет поток, который позволяет выполнять операции чтения/записи в файл.

```
static void Main(string[] args)
{
  FileStream file = new FileStream("d:\\test.txt", FileMode.Open
, FileAccess.Read); //открывает файл только на чтение
}
```

Режимы открытия FileMode:

- Append – открывает файл (если существует) и переводит указатель в конец файла (данные будут дописываться в конец), или создает новый файл. Данный режим возможен только при режиме доступа FileAccess.Write.
- Create - создает новый файл(если существует – заменяет)
- CreateNew – создает новый файл (если существует – генерируется исключение)
- Open - открывает файл (если не существует – генерируется исключение)
- OpenOrCreate – открывает файл, либо создает новый, если его не существует
- Truncate – открывает файл, но все данные внутри файла затирает (если файла не существует – генерируется исключение)

```
static void Main(string[] args)
{
  FileStream file1 = new FileStream("d:\\file1.txt", FileMode.CreateNew); //создание
```

```
нового файла
  FileStream file2 = new FileStream("d:\\file2.txt", FileMode.Open); //открытие существующего файла
  FileStream file3 = new FileStream("d:\\file3.txt", FileMode.Append); //открытие файла на дозапись в конец файла
}
```

Режим доступа FileAccess:
- Read – открытие файла только на чтение. При попытке записи генерируется исключение
- Write - открытие файла только на запись. При попытке чтения генерируется исключение
- ReadWrite - открытие файла на чтение и запись.

Чтение из файла

Для чтения данных из потока нам понадобится класс StreamReader. В нем реализовано множество методов для удобного считывания данных. Ниже приведена программа, которая выводит содержимое файла на экран:

```
static void Main(string[] args)
{
  FileStream file1 = new FileStream("d:\\test.txt", FileMode.Open); //создаем файловый поток
  StreamReader reader = new StreamReader(file1); // создаем «потоковый читатель» и связываем его с файловым потоком
  Console.WriteLine(reader.ReadToEnd()); //считываем все данные с потока и выводим на экран
  reader.Close(); //закрываем поток
  Console.ReadLine();
}
```

Метод ReadToEnd() считывает все данные из файла. ReadLine() – считывает одну строку (указатель потока при этом переходит на новую строку, и при следующем вызове метода будет считана следующая строка).

Свойство EndOfStream указывает, находится ли текущая позиция в потоке в конце потока (достигнут ли конец файла). Возвращает true или false.

Запись в файл

Для записи данных в поток используется класс StreamWriter. Пример записи в файл:

```
static void Main(string[] args)
{
  FileStream file1 = new FileStream("d:\\test.txt", FileMode.Create); //создаем файловый поток
 StreamWriter writer = new StreamWriter(file1); //создаем «потоковый писатель» и связываем его с файловым потоком
  writer.Write("текст"); //записываем в файл
  writer.Close(); //закрываем поток. Не закрыв поток, в файл ничего не запишется
}
```

Метод WriteLine() записывает в файл построчно (то же самое, что и простая запись с помощью Write(), только в конце добавляется новая строка).

Нужно всегда помнить, что после работы с потоком, его нужно закрыть (освободить ресурсы), использовав метод Close().

Кодировка, в которой будут считываться/записываться данные указывается при создании StreamReader/StreamWriter:

```
static void Main(string[] args)
{
  FileStream file1 = new FileStream("d:\\test.txt", FileMode.Open);
  StreamReader reader = new StreamReader(file1, Encoding.Unicode);
```

```
  StreamWriter writer = new StreamWriter(file1, Encoding.UTF8);
}
```

Кроме того, при использовании StreamReader и StreamWriter можно не создавать отдельно файловый поток FileStream, а сделать это сразу при создании StreamReader/StreamWriter:

```
static void Main(string[] args)
{
  StreamWriter writer = new StreamWriter("d:\\test.txt"); //указываем путь к файлу, а не поток
  writer.WriteLine("текст");
  writer.Close();
}
```

Как создать папку?

С помощью статического метода CreateDirectory() класса Directory:

```
static void Main(string[] args)
{
 Directory.CreateDirectory("d:\\new_folder");
}
```

Как удалить папку?

Для удаления папок используется метод Delete():

```
static void Main(string[] args)
{
  Directory.Delete("d:\\new_folder"); //удаление пустой папки
}
```

Если папка не пустая, необходимо указать параметр рекурсивного удаления - true:

```
static void Main(string[] args)
{
  Directory.Delete("d:\\new_folder", true); //удаление папки, и всего, что внутри
}
```

Задание.1. Создайте файл numbers.txt и запишите в него натуральные числа от 1 до 500 через запятую.

2. Дан массив строк: "red", "green", "black", "white", "blue". Запишите в файл элементы массива построчно (каждый элемент в новой строке).

3. Возьмите любой текстовый файл, и найдите в нем размер самой длинной строки.