**Введение**

1. **Платформа Microsoft .Net**
   1. **Библиотека типов и пространства имен**
2. **Структура программы????**
3. **Типы данных**
4. **Переменные**
5. **классы, объекты**
6. **Средства ввода вывода**
7. **Операции и Выражения**
   1. **???**
   2. **???**
   3. **???**
   4. **???**
8. **Операторы**
   1. **???**
   2. **???**
   3. **???**
   4. **For each**
   5. **Try Catch**
9. **Массивы и строки**
   1. **???**
   2. **???**
   3. **???**
10. **Функции**
    1. **???**
    2. **???**
    3. **???**
11. **Структуры**
    1. **???**
    2. **???**
    3. **???**
12. **Файлы**
    1. **???**
    2. **???**
    3. **???**
13. **Приложение**

**Содержание**

**Литература**

1. **Типы данных**

В языке **C#** все типы данных делятся на две категории – ссылочные типы и типы значений (или значимые, структурные).

**Типы значений**

Типы значений хранятся в стеке. Стек - это область памяти, которая используется для передачи параметров в методы и хранения определенных в пределах методов локальных переменных. Данные переменной типа значения хранятся в самой переменной.

К типам значений относятся стандартные типы данных:

- Целочисленные типы (*byte, sbyte, char, short, ushort, int, uint, long, ulong*);   
- Типы с плавающей запятой (*float, double*);

- Тип *decimal*;

- Тип *bool*;

- Пользовательские структуры (*struct*);

- Перечисления (*enum*).

**Ссылочные типы**

Переменная ссылочного типа содержит не данные, а ссылку на них. Сами данные в этом случае уже хранятся в куче. Куча - это область памяти, в которой размещаются управляемые объекты, и работает сборщик мусора. Сборщик мусора освобождает все ресурсы и объекты, которые уже не нужны.

К ссылочным типам относятся:

- Классы (*class*);

- Интерфейсы (*interface*);

- Делегаты (*delegate*);

- Тип *object*;

- Тип *string*.

Таблица 1

**Типы данных в C#**

|  |  |  |  |
| --- | --- | --- | --- |
| **Логический тип** | | | |
| **Имя типа** | **Системный тип** | **Значения** | **Размер** |
| Bool | System.Boolean | true, false | 8 бит |
| **Арифметические целочисленные типы** | | | |
| Sbyte | System.SByte | -128 — 127 | Знаковое, 8 Бит |
| Byte | System.Byte | 0 — 255 | Беззнаковое, 8 Бит |
| Short | System.Short | -32768 —32767 | Знаковое, 16 Бит |
| Ushort | System.UShort | 0 — 65535 | Беззнаковое, 16 Бит |
| Int | System.Int32 | ≈(-2\*10^9 — 2\*10^9) | Знаковое, 32 Бит |
| Uint | System.UInt32 | ≈(0 — 4\*10^9) | Беззнаковое, 32 Бит |
| Long | System.Int64 | ≈(-9\*10^18 — 9\*10^18) | Знаковое, 64 Бит |
| Ulong | System.UInt64 | ≈(0— 18\*10^18) | Беззнаковое, 64 Бит |
| Арифметический тип с плавающей точкой | | | |
| Float | System.Single | +1.5\*10^-45 - +3.4\*10^38 | 7 цифр |
| Double | System.Double | +5.0\*10^-324 - +1.7\*10^308 | 15-16 цифр |
| **Арифметический тип с фиксированной точкой** | | | |
| Decimal | System.Decimal | +1.0\*10^-28 - +7.9\*10^28 | 28-29 значащих цифр |
| **Символьные типы** | | | |
| Char | System.Char | U+0000 - U+ffff | 16 бит Unicode символ |
| String | System.String | Строка из символов Unicode |

1. **Переменные**

***Переменная*** – это величина, значение которой может изменяться в процессе выполнения программы.

Переменная обладает именем, значением и типом. Имя и тип задаются при объявлении переменной и остаются неизменными на все время ее жизни. Этот процесс называется объявлением переменной. Например:

int I, j, k; char c; double x, y, z;

Получение начального значения переменной называется ее ***инициализацией***. Важной особенностью языка C# является требование обязательной инициализации переменной до начала ее использования. Попытка использовать неинициализированную переменную приводит к ошибкам, обнаруживаемым еще на этапе компиляции. Инициализация переменных, как правило, выполняется в момент объявления, хотя и может быть отложена. Например:

int k=5, j; float x=0.1, y;

j=0; y=sin(x);

Здесь k и x – переменные, инициализированные в момент объявления, j и y – переменные с отложенной инициализацией.

Имя переменной должно отображать суть данных, для которых она используется.

***Нельзя давать имя переменной, совпадающее с ключевым словом языка, имя не должно начинаться с цифр.***

Язык **С#** чувствителен к регистру символов, например, переменные **name** и **Name** -это две совершенно разные переменные.

**Преобразование встроенных типов данных**

Переменные одного типа можно преобразовывать в переменные другого типа явно и неявно. 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 постоянная величина, значение которой нельзя изменить. Для того, чтобы объявить константу, перед обычным объявлением переменной добавляется ключевое слово const:

static void Main(string[] args)

{

   const int months = 12; // *объявление константы*

   months = 13; // *ошибка компиляции*

}

При объявлении константы она, также как и переменная, должна обязательно быть проинициализирована значением. Константы также делают 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 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 **С#**. 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** имеет ограничения по его использованию - **var** не может быть в качестве:

- *поля класса*;

- *аргумента функции*;

- *возвращаемого типа функции*;

- *переменной, которой присваивается* **null.**

**Область действия переменной**

Все переменные по области действия можно разделить на ***локальные*** и ***глобальные***.

Переменная действует только в той области, где она объявлена. В учебнике не рассматривается объектно-ориентированное программирование (в котором как правило проект содержит….!!!! См раздел 2.). Поэтому речь идет о рассмотрении проекта, включающего один программный файл.

Область действия глобальной переменной – вся программа. В нашем случае глобальная переменная должна быть задана до функции **main** (см. пример ???, схему на рис ???)

Пример:???

Рис. ???? Схема области действия переменных примера ???

1. **Классы, объекты**

**С#** язык объектно-ориентированный, вся работа идет с использованием основных составляющих ООП - классов и объектов. Даже если мы пишем обычную программу, без использования классов нам не обойтись. Например, главная функция **Main** – это всегда метод класса, типы в **С#**, даже основные, являются классами, консольный ввод и вывод осуществляется через класс **Console**, для использования математических функций существует класс **Math**.

Классом называется абстрактный тип данных. Класс состоит из определений данных класса (член-данных) и функций, их обрабатывающих (член-функций). По иной терминологии член-данные – свойства, член-функции – методы.

**Объект**–это переменная, тип которой – класс, и определяется он обычным образом. Объект также называют **экземпляром** класса.

**Класс**

**Объект**

**Объект**

**Объект**

Класс Math содержит два статических поля, задающих константы E и PI, и 23 статических метода:

* тригонометрические функции: **Sin(x)**, **Cos(x)**, **Tan(x)**.

Возвращают соответственно ***синус***, ***косинус*** или ***тангенс*** указанного угла **х**. **х** может иметь только тип **double**. Угол задается в радианах. Для преобразования градусов в радианы пользуемся формулой

**х \*=** [**Math.PI**](https://msdn.microsoft.com/ru-ru/library/system.math.pi.aspx) **/ 180**;

* обратные тригонометрические функции:

**Asin(sinx)** - возвращает угол, синус которого равен указанному числу,

**ACos(cosx)** - возвращает угол, косинус которого равен указанному числу,

**ATan(tanx)** - возвращает угол, тангенс которого равен указанному числу,

**ATan2(sinx, cosx)** - возвращает угол, тангенс которого равен отношению двух указанных чисел.

Все аргументы должны иметь тип **double**;

* гиперболические функции: **Tanh(x), Sinh(x)**, **Cosh(x)** возвращает гиперболический ***тангенс***, ***синус*** или ***косинус*** указанного угла. Аргумент должен иметь тип ***double***;
* экспоненту и логарифмические функции: Exp(х) - возвращает значение **e**, возведенное в указанную степень, Log(х) - возвращает натуральный логарифм (с основанием **e**) указанного числа, Log(х,a) - возвращает логарифм указанного числа x по заданному основанию a, Log10(х) - возвращает логарифм с основанием 10 указанного числа. Все аргументы должны иметь тип double;
* модуль – Abs(х). Аргумент **х** может иметь тип Decimal, Double, Int16, Int32, Int64, Sbyte, Single. Возвращает абсолютное значение числа использованного типа;
* квадратный корень – Sqrt(х) - Возвращает квадратный корень из указанного числа типа double;
* знак числа– Sign(х). Аргумент х может иметь тип Decimal, Double, Int16, Int32, Int64, Sbyte, Single;
* функции округления – Ceiling(х) - Возвращает наименьшее целое число, которое больше или равно заданному, х может быть типа Decimal или Double, Floor(x) - возвращает наибольшее целое число, которое меньше или равно указанному, х может быть типа Decimal или Double, Round;
* минимум, максимум, степень, остаток – Min(x,y) возвращает меньшее из двух чисел, Max(х,у) – возвращает большее из двух чисел. Переменные х и у могут быть типа Byte, Decimal, Double, Int16, Int32, Int64, Sbyte, Single, UInt16, UInt32, UInt64. Pow(х,у) Возвращает указанное число х, возведенное в указанную степень у, х и у типа Double. IEEERemainder(х,у) - Возвращает остаток от деления х на у, х и у типа Double.

Пример использования**:**

double R=Double.Parse(Console.ReadLine());

double S=Math.PI\*Math.Pow(R, 2);

**Ввод данных. Класс Console**

Функции ввода и вывода данных на консоль содержит класс Console. Для ввода данных используют две операции:

Console.Read() - считывает с клавиатуры информацию, преобразуя её в тип **string**;

Console.ReadLine() - делает то же самое что и Console.Read(), но при этом переводит курсор на следующую строчку.

Переменной типа string легко присвоить считанную с клавиатуры информацию, так как она относится к такому же типу:

**string text=Console.ReadLine();**

Чтобы присвоить считанную информацию переменным других типов, нужно преобразовать полученную строку в тот же тип, что и переменная. Функция преобразования текстовой информации в другие типы вводится так: сначала пишется тип, затем ставится точка, потом слово **Parse** и после него в скобках - **Console.ReadLine()** или же взятый в кавычки готовый текст.

int i=int.Parse(Console.ReadLine());

double d=double.Parse(Console.ReadLine());

Здесь создана переменная типа **int** и ей присваивается считанный с клавиатуры при помощи функции **Console.ReadLine()** текст, который преобразуется в нужный тип при помощи функции **int.Parse()**.

*Вывод данных. Класс* ***Console***. Для вывода данных на экран в программе используются функции:

**Console.Write()** - выводит указанную в скобках информацию на экран.

**Console.WriteLine()** - делает то же самое, переводя курсор на следующую строку.

Чтобы вывести на экран число, символ или текст, его нужно просто вставить в скобки:

**Console.WriteLine(13)** - выводит число 13;

**Console.WriteLine('f')**- выводит букву **f**;

**Console.WriteLine("Привет, мир!")** - выводит текст **Привет, мир!**

Чтобы вывести одну переменную любого типа, достаточно просто указать в скобках её название:

**Console.WriteLine(v)**- выводит значение переменной v.

**Console.WriteLine(“Результат=”+v)**

Однако, если нужно вывести значения переменных в контексте какого-либо текста, используют следующий приём: в скобках пишут взятый в кавычки текст с забитыми под переменные местами, потом ставят запятую и пишут через запятые все необходимые для вывода переменные. Для того, чтобы оставить в тексте место под переменную, необходимо там, где её значение нужно вывести, указать в фигурных скобках её порядковый номер в списке переменных, что идёт после текста (нумерация начинается с 0). Допустим, переменная x имеет значение 5, a y - 9.

**Console.WriteLine("х равно {0}, а у равно {1}", х, у)**

Текст, выведенный на экран, будет иметь следующий вид:

**х** равно 5, а **у** равно 9

**Преобразование данных. Класс Convert**

Служит для преобразования одного базового типа в другой базовый тип. Класс **Convert** содержит статические методы, которые можно вызывать для преобразования целых чисел и строк, представляющих собой целое число. Каждая из этих методов преобразования включает аргумент, который позволяет указать в системе счисления; двоичный файл (по основанию 2), восьмеричным (основание 8), шестнадцатеричным (основание 16) и десятичным (основание 10). Существует набор методов для преобразования целочисленных типов-примитивов в строку и один для преобразования строки в каждой из простых целочисленных типов:

**ToString(Byte**, **Int32)** и **ToByte(String, Int32)**, чтобы преобразовать значение в байтах из строки в указанной системе счисления.

**ToString(Int16, Int32)** и **ToInt16(String, Int32)**, для преобразования 16-разрядное целое число со знаком в строку в указанной системе счисления.

**ToString(Int32, Int32)** и **ToInt32(String, Int32)**, для преобразования 32-разрядное знаковое целое число в строку в указанной системе счисления.

**ToString(Int64, Int32)** и **ToInt64(String, Int32)**, для преобразования 64-разрядное знаковое целое число в строку в указанной системе счисления.

**ToSByte(String, Int32)**, чтобы преобразовать строковое представление значения байтов в формате, соответствующем байт со знаком.

**ToUInt16(String, Int32)**, чтобы преобразовать строковое представление целого числа в указанном формате в 16-разрядное целое число без знака.

**ToUInt32(String, Int32)**, чтобы преобразовать строковое представление целого числа в указанном формате в 32-разрядное целое число без знака.

**ToUInt64(String, Int32)**, чтобы преобразовать строковое представление целого числа в указанном формате в 64-разрядное целое число без знака.

1. **Операции и Выражения**

**6.1. Присваивание**

Выражение присваивания состоит из левой и правой части. Левая часть - это **список** переменных, в котором знак равенства выступает в качестве разделителя. Правая часть - это выражение.

Выражение правой части вычисляется, при необходимости приводится к типу переменных левой части, после чего все переменные левой части получают значение вычисленного выражения.

Все переменные в списке левой части должны иметь один тип или неявно приводиться к одному типу.

**Пример**

double x,y,z,w =1, u =7, v= 5;   
x = y = z = w =(u+v+w)/(u-v-w);

**6.2. Специальные случаи присваивания**

Для присваиваний вида "x=x+1" и "x=x-1", в которых переменная увеличивается или уменьшается на единицу, используются специальные префиксные и постфиксные операции инкримента "x++" и декримента "x--".

Другой важный частный случай - это присваивания вида:

X = X <operator> (expression)

Для таких присваиваний используется краткая форма записи:

X <operator>= expression

В качестве операции разрешается использовать арифметические, логические (побитовые) операции и операции сдвига языка C#.

Пример:

x += u+v; y /=(u-v); b &= (x<y);

**6.3. Выражения**

Выражения строятся из операндов - констант, переменных, функций, - объединенных знаками операций и скобками.

При вычислении выражения определяется его значение и тип. Эти характеристики однозначно задаются значениями и типами операндов, входящих в выражение, и правилами вычисления выражения.

Операции в выражениях выполняются слева направо с соблюдением приоритета. Изменить приоритет операции можно с помощью ( ).

Приоритет выполнения операций

|  |  |  |
| --- | --- | --- |
| Приоритет | Категория | Операции |
| 0 | Первичные | (expr) x.y f(x) a[x] x++ x-- new sizeof(t) typeof(t) |
| 1 | Унарные | + - ! ~ ++x --x (T)x |
| 2 | Мультипликативные (Умножение) | \* / % |
| 3 | Аддитивные (Сложение) | + - |
| 4 | Сдвиг | << >> |
| 5 | Отношения, проверка типов | < > <= >= is as |
| 6 | Эквивалентность | == != |
| 7 | Логическое И | & |
| 8 | Логическое исключающее ИЛИ (XOR) | ^ |
| 9 | Логическое ИЛИ (OR) | | |
| 10 | Условное И | && |
| 11 | Условное ИЛИ | || |
| 12 | Условное выражение | ?: |
| 13 | Присваивание | = \*= /= %= += -= <<= >>= &= ^= |= |

**Операции increment, decrement**

Операции инкремента - "увеличить на единицу" и декремента - " уменьшить на единицу " могут быть префиксными и постфиксными.

К высшему приоритету относятся ***постфиксные*** операции x++ и x--.

***Префиксные*** операции ++х и --х имеют на единицу меньший приоритет.

Для префиксных операций результатом их выполнения является измененное значение x, постфиксные операции возвращают в качестве результата значение x до изменения.

Пример.

Int k=5, t=3, g;

g=--k+5; t++;

В результате g=9, k=4, t=4.

Арифметические операции

* Арифметические операции - " +, -, \*, /, % ".
* Операции " + " и " - " могут быть унарными и бинарными.
* Операция деления " / " над целыми типами осуществляет деление нацело, для типов с плавающей и фиксированной точкой - обычное деление.
* Операция " % " определена над всеми арифметическими типами и возвращает остаток от деления нацело. Тип результата зависит от типов операндов.

Операции отношения

* == равно,
* != не равно,
* < меньше,
* > больше,
* <= меньше либо равно,
* >= больше либо равно.
* При формировании сложного условия используются связки **&&** - **И** , **||** - **ИЛИ** и **! - НЕ**

**6.2. Условный оператор if**

Синтаксис в самом простом случае

if(выражение\_1) оператор\_1;  
else оператор\_2;

Если выражение\_1 верно, то выполнится оператор\_1, иначе оператор\_2. При необходимости, если нужно выполнить несколько операторов, вместо оператора может стоять блок {…}.

Пример.

If(a>b) max=a

else max=b;

Операторы if могут быть вложены друг в друга. В общем случае синтаксис имеет вид

if(выражение\_1) оператор\_1;   
else if(выражение\_2) оператор\_2;   
...   
else if(выражение\_K) оператор\_K;   
else оператор\_N;

Выражения if должны заключаться в круглые скобки и быть булевого типа.

Пример. По заданным координатам точки определить, к какой четверти плоскости она относится.

If(х>0)

if (y>0) Console.WriteLine(“I четверть”);

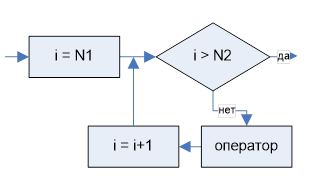
else Console.WriteLine(“IV четверть”);

else if (y>0) Console.WriteLine(“II четверть”);

else Console.WriteLine(“III четверть”);

6.3. Оператор цикла for

* for(инициализаторы; условие; список\_выражений) оператор;
* Оператор, стоящий после закрывающей скобки, задает тело цикла. В большинстве случаев телом цикла является блок.
* Сколько раз будет выполняться тело цикла, зависит от трех управляющих элементов, заданных в скобках.
* **Инициализаторы** задают начальное значение одной или нескольких переменных, часто называемых *счетчиками* или просто *переменными цикла*.
* **Условие** задает условие окончания цикла, соответствующее выражение при вычислении должно получать значение true или false.
* **Список выражений**, записанный через запятую, показывает, как меняются счетчики цикла на каждом шаге выполнения.
* Если условие цикла истинно, то выполняется тело цикла, затем изменяются значения счетчиков и снова проверяется условие. Как только условие становится ложным, цикл завершает свою работу.
* В цикле for тело цикла может ни разу не выполняться, если условие цикла ложно после инициализации, а может происходить зацикливание, если условие всегда остается истинным. В нормальной ситуации тело цикла выполняется конечное число раз.



**For (i = N1;i<=N2;i++)**

**Оператор;**

Примеры.

* for (int i =1,j =2,S=0; i<10; i++) S+=i\*j;
* for (int i =1,j =2,S=0; i<10; i++) {x\*=2; S+=i\*j;}
* for (int i =1; i<10; i++) for (int j =1; j<10; j++) S+=i\*j;

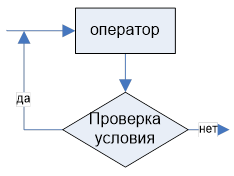
**Циклы While**

* while(выражение) оператор;
* Эта модификация соответствует стратегии: "сначала проверь, а потом делай".
* Цикл, проверяющий условие завершения в конце, соответствует стратегии: "сначала делай, а потом проверь". Тело такого цикла выполняется, по меньшей мере, один раз. Вот синтаксис этой модификации:
* do оператор while(выражение);
* Варианты



WHILE <условие>

Оператор;

**DO**

**Оператор**

**WHILE <условие>;**

**Генерация случайных чисел**

Для генерации случайных чисел в С# существует класс Random. Чтобы сгенерировать число, нужно создать объект этого класса. Затем, пользуясь методами класса Random через созданный объект, можно сгенерировать любое число – любого типа, в указанном диапазоне или без него.

Методы класса Random

[Next()](https://msdn.microsoft.com/ru-ru/library/9b3ta19y(v=vs.110).aspx) - Возвращает неотрицательное случайное целое число.

[Next(Int32)](https://msdn.microsoft.com/ru-ru/library/zd1bc8e5(v=vs.110).aspx) - Возвращает неотрицательное случайное целое число, которое меньше максимально допустимого значения.

[Next(Int32, Int32)](https://msdn.microsoft.com/ru-ru/library/2dx6wyd4(v=vs.110).aspx) - Возвращает случайное целое число в указанном диапазоне.

[NextBytes(Byte[])](https://msdn.microsoft.com/ru-ru/library/system.random.nextbytes(v=vs.110).aspx) - Заполняет элементы указанного массива байтов случайными числами.

[NextDouble()](https://msdn.microsoft.com/ru-ru/library/system.random.nextdouble(v=vs.110).aspx) - Возвращает случайное число с плавающей запятой, которое больше или равно 0,0 и меньше 1,0.

[Sample()](https://msdn.microsoft.com/ru-ru/library/system.random.sample(v=vs.110).aspx) - Возвращает случайное число с плавающей запятой в диапазоне от 0,0 до 1,0.

[ToString()](https://msdn.microsoft.com/ru-ru/library/system.object.tostring(v=vs.110).aspx) - Возвращает строковое представление текущего объекта.

**Пример.**

* // создаем объект класса Random
* Random realRnd = new Random();
* // случайные числа в диапазоне [0,1)
* Console.WriteLine("случайные числа в диапазоне[0,1)");
* for(int i =1; i <= 5; i++)
* { Console.WriteLine("Число " + i + "= " + realRnd.NextDouble() ); }
* // случайные числа в диапазоне[min,max]
* int min = -100, max=-10;
* Console.WriteLine("случайные числа в диапазоне ["+min +"," + max + "]");
* for(int i =1; i <= 5; i++)
* { Console.WriteLine("Число " + i + "= " + realRnd.Next(min,max) ); }

**Оператор foreach**

Применяется для перебора элементов массива. Синтаксис:

**foreach ( тип** имя **in** имя\_массива **) тело\_цикла**

*Имя* задает локальную по отношению к циклу переменную, которая будет по очереди принимать все значения из массива, например:

int[] massiv = { 24, 50, 18, 3, 16, -7, 9, -1 };

foreach ( int **x** in massiv ) Console.WriteLine( **x** );

**Программа с использованием foreach**

int[] a = { 3, 12, 5, -9, 8, -4 };

Console.WriteLine( "Исходный массив:" );

foreach ( int elem in a )

Console.Write( "\t" + elem );

Console.WriteLine();

long sum = 0; // cумма отрицательных элементов

int num = 0; // количество отрицательных элементов

foreach ( int elem in a )

if ( elem < 0 ) {

sum += elem; num++;

}

Console.WriteLine( "sum = " + sum );

Console.WriteLine( "num = " + num );

int max = a[0]; // максимальный элемент

foreach ( int elem in a )

if ( elem > max ) max = elem;

Console.WriteLine( "max = " + max );

for ( int i = 0; i < n; ++i )

if ( a[i] < 0 ) {

sum += a[i]; num++;

}

Обработка исключительных ситуаций. Оператор try – catch

*Исключительная ситуация* или *исключение* – это возникновение непредвиденного или аварийного события, которое может порождаться некорректным использованием аппаратуры или неправильной работой программы.

При возникновении ошибки необходимо:

1. Прервать выполнение программы;

2. Получить значение ошибки;

3. Вывести сообщение об ошибке и вернуть вызывающей программе некоторое приемлемое значение, которое позволит ей продолжить работу.

Исключения С# не поддерживают обработку асинхронных событий, таких как ошибки оборудования или прерывания (например, нажатие клавиш Ctrl+C). Исключения возникают тогда, когда некоторая часть программы не смогла сделать то, что от нее требовалось. При этом программа может попытаться продолжить работу, получив какое-либо приемлемое значение.

Исключения позволяют логически разделить процесс обработки ошибки на две части: обнаружение аварийной ситуации и обработка аварийной ситуации. Функция, обнаружившая ошибку, может не знать, что предпринимать для ее исправления. Использующий эту функцию код может знать, что делать, но не уметь определить место возникновения.

Исключения генерирует либо среда выполнения, либо программист с помощью оператора throw. Исключения обнаруживаются и обрабатываются в операторе try.

Ниже приведены наиболее часто используемые стандартные исключения, генерируемые средой. Они определены в пространстве имен System. Все они являются потомками класса Exception, а точнее, потомками его потомкаSystemException.

Часто используемые стандартные исключения

ArithmeticException - Ошибка в арифметических операциях или преобразованиях.

ArrayTypeMismatchExceptionм - Попытка сохранения в массиве элемента несовместимого типа.

DivideByZeroException - Попытка деления на ноль.

FormatException - Попытка передать в метод аргумент неверного формата.

IndexOutOfRangeException - Индекс массива выходит за границы диапазона.

InvalidCastException - Ошибка преобразования типа.

OutOfMemoryException - Недостаточно памяти для создания нового объекта.

OverFlowException - Переполнение при выполнении арифметических операции.

StackOverFlowException - Переполнение стека.

**Оператор try включает:**

1. **Контролируемый блок** — составной оператор, предваряемый ключевым словом **try**. В контролируемый блок включаются потенциально опасные операторы программы. Все функции, прямо или косвенно вызываемые из блока, также считаются ему принадлежащими;
2. Один или несколько **обработчиков исключений** — блоков **catch**, в которых описывается, как обрабатываются ошибки различных типов;
3. **Блок завершения finally** выполняется независимо от того, возникла ошибка в контролируемом блоке или нет.

Синтаксис оператора try:

try **блок [ блоки** catch ] **[ блок** finally ]

Отсутствовать могут либо блоки **catch**, либо блок **finally**, но не оба одновременно.

**Общий механизм обработки исключений**

1. Обработка исключения начинается с появления ошибки. Функция, в которой она возникла, генерирует исключение.
2. Запускается соответствующий обработчик исключения и управление передается ему.
3. Выполняется блок finally**,** если он есть (этот блок выполняется и в том случае, если ошибка не возникла).
4. Если обработчик не найден, вызывается стандартный̆ обработчик исключения. Его действия зависят от конфигурации среды. Обычно он выводит на экран окно с информацией̆ об исключении и завершает текущий процесс.

Обработчики исключений должны располагаться непосредственно за блоком try. Они начинаются с ключевого слова catch, за которым в скобках следует тип обрабатываемого исключения. Можно записать один или несколько обработчиков в соответствии с типами обрабатываемых исключений. Блоки catch просматриваются в том порядке, в котором они записаны, пока не будет найден соответствующий типу исключения.

Синтаксис обработчиков напоминает определение функции с одним параметром — типом исключения. Существуют три формы записи:

catch( тип имя ) { ... /\* тело обработчика \*/ }

catch( тип ) {.../\* тело обработчика \*/ }

catch { ... /\* тело обработчика \*/ }

*Первая форма* применяется, когда имя параметра используется в теле обработчика для выполнения каких-либо действий, например вывода информации об исключении. *Вторая форма* не предполагает использования информации об исключении, играет роль только его тип. *Третья форма* применяется для перехвата всех исключений. Так как обработчики просматриваются в том порядке, в котором они записаны, обработчик третьего типа (он может быть только один) следует помещать после всех остальных.

*Третья форма* применяется для перехвата всех исключений. Так как обработчики просматриваются в том порядке, в котором они записаны, обработчик третьего типа (он может быть только один) следует помещать после всех остальных.

*try{*

. . . // Контролируемый блок

}

catch ( OverflowException е ) {…} // Обработка исключений класса OverflowException (переполнение)

catch ( DivideByZeroException) {…} // Обработка исключений класса DivideByZeroException (деление на 0)

catch {…} // Обработка всех остальных исключении

Если исключение в контролируемом блоке не возникло, все обработчики пропускаются.

В любом случае, произошло исключение или нет, управление передается в блок завершения finally (если он существует), а затем — первому оператору, находящемуся непосредственно за оператором try.

В завершающем блоке обычно записываются операторы, которые необходимо выполнить независимо от того, возникло исключение или нет, например, закрытие файлов, с которыми выполнялась работа в контролируемом блоке, или вывод информации.

**int x,y;**

**try {x = Int32.Parse(Console.ReadLine());**

**y= Int32.Parse(Console.ReadLine()); }**

**catch (FormatException)**

**{ Console.WriteLine("Вы ввели не число"); x=0; y=0;}**