# 3. Состав языка С#. Простые программы на С#

## 3.1. Элементы языка С#

В тексте на любом естественном языке можно выделить четыре основных элемента: символы, слова, словосочетания, предложения. Алгоритмический язык также содержит такие элементы, только слова называют лексемами (элементарными конструкциями), словосочетания – выражениями, предложения – операторами. Лексемы образуются из символов, выражения – из символов и лексем, операторы из символов, лексем и выражений.

### 3.1.1. Алфавит языка (символы)

* прописные и строчные латинские буквы и знак подчеркивания;
* арабские цифры от 0 до 9;
* специальные знаки “{},| []()+-/ % \* . \ ’ : ; & ? < > = ! # ^
* пробельные символы (пробел, символ табуляции)
* символы перехода на новую строку.

### 3.1.2. Лексемы

* **Идентификаторы** – имена объектов программ.
* Могут быть использованы латинские буквы, цифры и знак подчеркивания. Разрешается также использовать буквы национальных алфавитов.
* Прописные и строчные буквы различаются, например, PROG1, prog1 и Prog1 – три различных идентификатора.
* Первым символом должна быть буква или знак подчеркивания (но не цифра).
* Пробелы в идентификаторах не допускаются.

**Нотация** –соглашение о правилах создания имен.

* **Нотация Паскаля:** каждое слово, составляющее идентификатор, начинается с прописной буквы: MaxLength, TekSum.
* **Венгерская нотация:** наличие префикса, соответствующего типу величины, например, iMaxLength, fTekSum.
* **Нотация Camel**: с прописной буквы начинается каждое слово, составляющее идентификатор, кроме первого, например, maxLength, tekSum.
* Можно разделять слова, составляющие имя, знаками подчеркивания: max\_length, tek\_sum, при этом все составные части начинаются со строчной буквы.

В С# для именования различных видов программных объектов чаще всего используются две нотации: Паскаля и Camel. Приняты следующие соглашения: имена классов пишутся с прописной буквы (нотация Паскаля), имена переменных – с маленькой буквы (нотация Camel). Имена должны имеет смысл, т.к. программа должна быть понятна тому, кто ее читает.

* **Ключевые (зарезервированные) слова** – это слова, которые имеют специальное значение для компилятора. Их нельзя использовать в качестве идентификаторов. Можно использовать ключевые слова в качестве идентификаторов, если поставить перед ними @ (@if – разрешенный идентификатор).
* **Знаки операций** – это один или несколько символов, определяющих действие над операндами. Операции делятся на унарные, бинарные и тернарную по количеству участвующих в этой операции операндов. Символы, составляющие знак операций, могут быть как специальными(&&, | , <) так и буквенными (as , new).
* **Константы** – это неизменяемые величины. Существуют логические, целые, вещественные, символьные и строковые константы, а также константа null. Компилятор выделяет константу в качестве лексемы (элементарной конструкции) и относит ее к одному из типов по ее внешнему виду.
* **Разделители** – скобки, точка, запятая пробельные символы.
* **Комментарии** - предназначены для записи пояснений к программе и формирования документации.

Однострочный комментарий

// пример комментария

Многострочный комментарий

/\* пример многострочного

комментария\*/

## 3.2.Типы данных в С#

Тип данных однозначно определяет:

* внутреннее представление данных, и, следовательно, и множество их возможных значений;
* допустимые действия над данными (операции и функции).

**Классификация типов:**

* по строению элемента
  + простые – нельзя разбить на более мелкие элементы (int, double, char, bool);
  + структурированные – можно представить в виде набора элементов (структуры, классы, массивы, строки);
* по способу создания
  + встроенные – не требуют специального описания, каждому типу соответствует свое ключевое слово (int, diouble);
  + определяемые программистом – задается внутреннее представление, операции и имя типа (классы, структуры);
* по способу хранения в памяти
  + значимые – хранятся в стеке (память, выделяемая для переменных функции);
  + динамические – хранятся в динамической памяти (куче / heap).

значение

адрес

значение

Рисунок 1 Значимые и динамические переменные

Все стандартные простые типы данных определены в MS.Net являются значимыми типами и определены на основе структур. Поскольку эти типы определены в MS.Net (система типов CTS), ими могут пользоваться программы на любых языках, определенных на этой платформе. Язык С# имеет собственные синонимы этих типов.

Для обработки данных простых типов используются предопределенные операции, обозначаемые специальными знаками. Кроме того, можно использовать поля и методы, определенные в типе.

У всех простых типов есть статические (общие для всего класса) поля, хранящие максимальное (MaxValue) и минимальное (MinValue) значения для данного типа данных.

* Статический метод преобразования строки символов в значение заданного типа:

Тип.Parse (строка символов) – результатом является значение заданного типа.

* Метод объекта для преобразования значения в строку символов:

Объект.ToString() – результатом является строка символов, в которую преобразовано значение объекта.

Данные в программе могут присутствовать либо в виде переменных, либо в виде констант. Форма записи константы однозначно определяет ее значение и тип. Тип переменной должен быть определен путем явного объявления. Переменные при объявлении могут быть инициализированы любым значением, определенным на момент объявления.

* **Целочисленные типы**

Целые числа со знаком

|  |  |  |
| --- | --- | --- |
| Тип С# | Тип СTS | Размер в байтах |
| sbyte | Sbyte | 1 |
| short | Int16 | 2 |
| int | Int32 | 4 |
| long | Int64 | 8 |

Целые числа без знака

|  |  |  |
| --- | --- | --- |
| Тип С# | Тип СTS | Размер в байтах |
| byte | Byte | 1 |
| ushort | UInt16 | 2 |
| uint | UInt32 | 4 |
| ulong | UInt64 | 8 |

Примеры:

int a, b=10, c=int.Parse(“355”), d=b+c, e=short.MaxValue;

Константы

|  |  |  |
| --- | --- | --- |
| Десятичная константа | Последовательность десятичных цифр, за которой могут следовать символы U/u (unsigned) и/или L/l (long) | 8, 0, 192345  8u, 1045l, 34lu, 123UL |
| Шестнадцатеричная константа | Последовательность шестнадцатеричных цифр, которым предшествуют символы 0х или 0Х, за цифрами могут следовать символы U/u (unsigned) и/или L/l (long) | 0хА, 0Х00F, 0х123  0x1AFLU, 0XFFu |

* **Вещественные типы**

|  |  |  |  |
| --- | --- | --- | --- |
| Тип С# | Тип СTS | Размер в байтах | Точность в цифрах |
| float | Single | 4 | 7 |
| double | Double | 8 | 15 |

Примеры.

double a, d=.5, c=double.Parse(“5,5”);

Константы

|  |  |  |
| --- | --- | --- |
| Константа с фиксированной точкой | [цифры].[цифры][суффикс]  Суффикс – это символы F/f (float) или D/d (double) | 5.7, .0001, 41.  5.7d, .0001f |
| Константа с плавающей точкой | [цифры][.][цифры]E|e[+|-] [цифры] [суффикс] | 0.5е5, .11е-5, 5Е3  0.5e5d, .11e-5f, 5E3d |

* **Финансовый тип**

|  |  |  |  |
| --- | --- | --- | --- |
| Тип С# | Тип СTS | Размер в байтах | Точность в цифрах |
| Decimal | Decimal | 16 | 28 |

Представляет собой вещественное значение, причем способ кодирования этого значения не совпадает ни с одним способом кодирования вещественных чисел. Поэтому денежный тип несовместим ни с одним вещественным типом.

Константы

Целая или вещественная константа с суффиксом М: 25.5М

* **Логический тип**

|  |  |
| --- | --- |
| Тип С# | Тип СTS |
| bool | Bool |

Константы

true, false

Логический тип не совместим ни с каким другим типом.

* **Символьный тип**

|  |  |  |
| --- | --- | --- |
| Тип С# | Тип СTS | Размер в байтах |
| char | Char | 2 (код символа в таблице кодировки Unicode) |

Константы

|  |  |  |
| --- | --- | --- |
| Символьная константа | Символ, заключенный в апострофы. | * символ, имеющий графическое представление : 'А', '5', '\*', 'ю' ; * управляющая последовательность : '\0', '\n'; * символ в виде шестнадцатеричного кода : '\xF', '\х74'; * символ в виде escape-последовательности Unicode: '\u0041' – код латинской буквы А. |

**Символьный тип имеет статические методы:**

Char.IsDigit(char c) //проверка c - цифра (возвращает true) или нет (false).

Char.IsLetter(char c) //проверка c - буква (возвращает true) или нет (false).

Char.IsLetterOrDigit(char c) //проверка c – цифра или буква (возвращает true) или нет (false).

Char.IsUpper(char c) //проверка c – в верхнем регистре (возвращает true) или нет (false).

Char.IsLower(char c) //проверка c – в нижнем регистре (возвращает true) или нет (false).

Char.ToUpper(char c) //возвращает символ в верхнем регистре .

Char.ToLower(char c) //возвращает символ в нижнем регистре.

## 3.3. Переменные

* Переменная – именованная область памяти, в которой хранятся данные определенного типа.
* Имя переменной должно соответствовать правилам, по которым формируются идентификаторы C#, отражать смысл хранимой величины и быть легко распознаваемым.
* Тип переменной выбирается исходя из диапазона и требуемой точности представления данных.
* При объявлении переменную можно сразу инициализировать.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | int x=10; | | |
| 10 | | | |
|  |  |  |  | тип | имя | значение |

* С# программа состоит из классов. Класс содержит поля (данные) и методы (функции).
* Переменная, описанная в методе класса, называется локальной.
* Блок — это код, заключенный в фигурные скобки. Основное назначение блока — группировка операторов.
* Область действия переменной, то есть область программы, где ее можно использовать, начинается в точке описания и длится до конца блока, внутри которого она описана. Область действия распространяется на вложенные блоки.
* Имя переменной должно быть уникальным в области ее действия.

**Пример 1**

namespace pr1

{

class Program

{

//с ошибками!!! ☹

static void Main(string[] args)

{

int x, y, z;

z = x + y; //переменные x и y не проинициализированы!

}

}

}

**Пример 2**

namespace pr2

{

class Program

{

//с ошибками!!! ☹

static void Main(string[] args)

{

//калькулятор:)

int x, y, z;

Console.WriteLine("Введите число");

x = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Введите число");

y = Convert.ToInt32(Console.ReadLine());

z = x + y;

Console.WriteLine("{0}+{1}={2}",x,y,z);

for ( int x = 0; x < 10; x++) //int x уже объявлено

{

Console.WriteLine("Введите число");

y = Convert.ToInt32(Console.ReadLine());

z = x + y;

Console.WriteLine("{0}+{1}={2}",x,y,z);

}

Console.WriteLine("{0}+{1}={2}",x,y,z);

}

}

}

**Пример 3**

namespace pr3

{

class Program

{

//с ошибками!!! ☹

static void Main(string[] args)

{

//калькулятор:)

int x, y, z;

Console.WriteLine("Введите число");

x = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Введите число");

y = Convert.ToInt32(Console.ReadLine());

z = x + y;

Console.WriteLine("{0}+{1}={2}",x,y,z);

for (x = 0; x < 5; x++)

{

Console.WriteLine("Введите число");

int w = Convert.ToInt32(Console.ReadLine());

z = x + w;

Console.WriteLine("{0}+{1}={2}", x, w, z);

}

Console.WriteLine("{0}+{1}={2}", x, w, z); //w не существует в данном блоке

}

}

}

**Пример 4**

//без ошибок☺

namespace pr4

{

class Program

{

static void Main(string[] args)

{

//калькулятор:)

int x, y, z;

Console.WriteLine("Введите число");

x = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Введите число");

y = Convert.ToInt32(Console.ReadLine());

z = x + y;

Console.WriteLine("{0}+{1}={2}",x,y,z);

for (x = 0; x < 5; x++)

{

Console.WriteLine("Введите число");

int w = Convert.ToInt32(Console.ReadLine());

z = x + w;

Console.WriteLine("{0}+{1}={2}", x, w, z);

}

}

}

}

## 3.4. Именованные константы

* Можно запретить изменять значение переменной, задав при ее описании ключевое слово const.
* Именованные константы должны обязательно инициализироваться при описании.
* При инициализации можно использовать не только константу, но и выражение

const int b = 1;

// const распространяется на обе переменные

const float a = 0.1f, у = 0.1f;

const int b = 1. а = 100;

const int х = b \* а + 25;

## 3.5. Выражения

* Из констант, переменных, разделителей и знаков операций можно конструировать выражения. Каждое выражение представляет собой правило вычисления нового значения.
* Каждое выражение состоит из одного или нескольких операндов, символов операций и ограничителей.
* Операндами могут быть:
  1. Вызов метода
  2. Переменная
  3. Константа
  4. Выражение
* Если выражение формирует целое или вещественное число, то оно называется арифметическим.
* Пара арифметических выражений, объединенная операцией сравнения, называется отношением.
* Любое выражение, завершенное ”;” называется оператором. Одиночный символ ”;”, не относящийся ни к одному оператору – пустой оператор.
* В зависимости от количества операндов, различают унарные, бинарные и тернарную операцию.

Приоритеты операций:

* + Вызов метода
  + Унарные операции
  + Арифметические операции
  + Сдвиги
  + Отношения
  + Битовые
  + Логические
  + Тернарная
  + Присваивание

## 3.6. Операции С#

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| **Унарные операции** | |
| . | Доступ к элементу структуры |
| [] | Доступ к элементу массива |
| ++ | Увеличение на единицу:  префиксная операция - увеличивает операнд до его использования,  постфиксная операция увеличивает операнд после его использования. |
| -- | Уменьшение на единицу:  префиксная операция - уменьшает операнд до его использования,  постфиксная операция уменьшает операнд после его использования. |
| typeof | Получение типа |
| - | Унарный минус |
| + | Унарный плюс |
| ! | Логическое отрицание (НЕ).  В качестве логических значений используется 0 (false) - ложь и не 0 (true) - истина, отрицанием 0 будет 1, отрицанием любого ненулевого числа будет 0. |
| new | Выделение памяти |
| (тип) | Преобразование типа |
| **Бинарные операции** | |
| **Мультипликативные** | |
| \* | умножение операндов арифметического типа |
| / | деление операндов арифметического типа (если операнды целочисленные, то выполняется целочисленное деление) |
| % | получение остатка от деления целочисленных операндов |
| **Аддитивные** | |
| + | бинарный плюс (сложение арифметических операндов) |
| - | бинарный минус (вычитание арифметических операндов) |
| **Операции отношения и поверки типа** | |
| < | меньше, чем |
| <= | меньше или равно |
| > | больше |
| >= | больше или равно |
| is | проверка принадлежности типу |
| as | приведение типа |
| **Операции сравнения** | |
| == | равно |
| != | не равно |
| **Логические операции** | |
| & | конъюнкция (И) логических операндов или отношений, результат ложь или истина |
| | | дизъюнкция (ИЛИ) логических операндов или отношений, результат ложь или истина |
| ^ | исключающее ИЛИ (неравнозначность), результат ложь или истина |
| && | сокращенная конъюнкция (если один из операндов – ложь, то второй операнд не вычисляется), результат ложь или истина |
| || | сокращенная дизъюнкция (если один из операндов – ложь, то второй операнд не вычисляется), результат ложь или истина |
| **Тернарная** | |
| ?: | Условная операция  x<0 ? -x : x ; //вычисляется абсолютное значение x |
| **Присваивание** | |
| = | присваивание |
| \*= | умножение с присваиванием (мультипликативное присваивание) |
| /= | деление с присваиванием |
| %= | деление с остатком с присваиванием |
| += | сложение с присваиванием |
| -= | вычитание с присваиванием |

## 3.7. Преобразование типов

Переменной одного типа может быть присвоено значение переменной другого типа. Например, можно присвоить переменной типа float значение типа int .

int i;

float f;

i = 10;

f = i; // float-переменной присваивается int-значение.

При этом, если используются совместимые типы, значение справа стороны (от оператора присваивания) автоматически преобразуется в значение "левостороннего" типа, т.е. i автоматически преобразуется к типу float, а затем присваивается f.

### 3.7.1. Автоматическое преобразование типов

При присвоении значения одного типа данных переменной другого типа будет выполнено автоматическое преобразование типов, если

* эти два типа совместимы;
* тип приемника больше (т.е. имеет больший диапазон представления чисел), чем тип источника.

При соблюдении этих двух условий выполняется преобразование с расширением, или расширяющее преобразование. Например, тип int – достаточно "большой" тип, чтобы сохранить любое допустимое значение типа byte, а поскольку как int , так и byte – целочисленные типы, здесь может быть применено автоматические преобразование.

Пример.

long l;

double d;

l = 100123285L;

d = l;

Console.WriteLine("l и d: " + l + " " + d);

Несмотря на возможность автоматического преобразования типов из long в double, обратное преобразование типов (из double в long) автоматически не выполняется, поскольку это преобразование не является расширяющим.

long l;

double d;

d = 100123285L;

l= d;//ОШИБКА!!!

Также не существует автоматического преобразования между типом decimal и float (или double), а также из числовых типов в тип char (или bool). Кроме того, несовместимы и типы char и bool .

### 3.7.2. Приведение несовместимых типов

Приведение к типу — это явно заданная инструкция компилятору преобразовать один тип в другой. Инструкция приведения записывается в следующей общей форме:

(тип\_приемника) выражение

double х=10, у=3;

int z= (int) (х / у) ;

Если приведение приводит к сужающему преобразованию, возможна потеря информации. Например, в случае приведения типа long к типу i n t информация будет утеряна, если значение типа long больше максимально возможного числа, которое способен представить тип int , поскольку будут "усечены" старшие разряды long-значения. При выполнении операции приведения типа с плавающей точкой к целочисленному будет утеряна дробная часть простым ее отбрасыванием. Например, при присвоении переменной целочисленного типа числа 1,23 в действительности будет присвоено число 1. Дробная часть (0,23) будет утеряна.

### 3.7.3. Преобразование типов в выражениях

Преобразование типов встречается не только в инструкциях присваивания, но и в выражениях. В выражениях можно смешивать различные типы данных, если они совместимы. Например, можно смешивать типы short и long, поскольку это числовые типы. При смешении различных типов в одном выражении все его составляющие преобразуются к одному типу, причем это происходит по мере перехода от одной операции к другой.

Рисунок 2. Неявное преобразование типов в выражениях

ЕСЛИ один операнд имеет тип decimal, TO и второй "возводится в ранг", т.е. "в тип" decimal (но если второй операнд имеет тип float или double, результат будет ошибочным).

ЕСЛИ один операнд имеет тип double, TO и второй преобразуется в значение типа double.

ЕСЛИ один операнд имеет тип float, TO и второй преобразуется в значение типа

float.

ЕСЛИ один операнд имеет тип ulong, TO и второй преобразуется в значение типа ulong (но если второй операнд имеет тип sbyte, short, int или long, результат будет ошибочным).

ЕСЛИ один операнд имеет тип long, TO и второй преобразуется в значение типа long.

ЕСЛИ один операнд имеет тип uint, а второй имеет тип sbyte, short или int, ТО оба операнда преобразуются в значения типа long.

ЕСЛИ один операнд имеет тип uint, TO и второй преобразуется в значение типа uint.

ИНАЧЕ оба операнда преобразуются в значения типа int.

Согласно последнему правилу, все операнды будут преобразованы в значения типа int, если не было применено ни одно их предыдущих правил. Следовательно, в выражении все char-, sbyte-, byte-, ushort- и short-значения будут в процессе вычислений преобразованы в значения типа int. Такое "поголовное" int-преобразование называют целочисленным продвижением типа (integer promotion). Следствием этого алгоритма является то, что результат всех арифметических операций будет иметь тип по "званию" не ниже int.

**Пример 5.**

byte b,c ;

b = 10;

c =(byte)(b \* b); //будет ошибка, если не использовать приведение типов

Console.WriteLine("b\*b=" + c);

Пример 2.

char a1 = 'a', a2 = 'b';

char a3 = (char)(a1 + a2);//а1+а2 приводится к типу int!!!!

Console.WriteLine("a1+a2=" + a3);

## 3.8. Выполнение операций в выражениях

### 3.8.1. Унарные операции ++ и –

Операции инкремента (++) и декремента (--) увеличивают и уменьшают значение операнда на единицу, соответственно.

++х; // Префиксная форма оператора инкремента.

х++; // Постфиксная форма оператора инкремента.

Пример.

int m;

int n;

Console.WriteLine("Введите целое число");

string buf = Console.ReadLine();

m = Convert.ToInt32(buf);

Console.WriteLine("Введите целое число");

buf = Console.ReadLine();

n = Convert.ToInt32(buf);

int k = m++ +n;

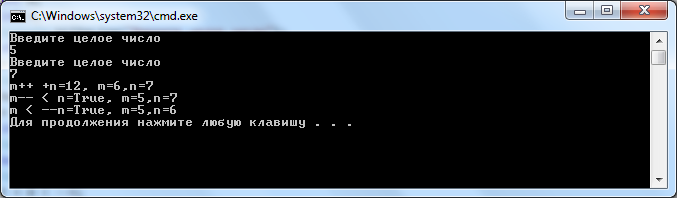
Console.WriteLine("m++ +n={0}, m={1},n={2}", k, m, n);

bool t = m-- < n;

Console.WriteLine("m-- < n={0}, m={1},n={2}", t, m, n);

t = m < --n;

Console.WriteLine("m < --n={0}, m={1},n={2}", t, m, n);



### 3.8.1. Бинарные операции + , - и \*

* Операции **вычитания** **(-) и сложения (+)** определены для типов int, uint, long, ulong, float, double и decimal.
* К величинам других типов их можно применять, если для них возможно неявное преобразование к этим типам.
* Если оба операнда целочисленные или типа decimal и результат операции слишком большой для представления с помощью заданного типа, генерируется ошибка (исключительная ситуация).

### 3.8.2. Бинарная операция/

* Операция **деления (/)** определена для типов int, uint, long, ulong, float, double и decimal.
* К величинам других типов ее можно применять, если для них возможно неявное преобразование к этим типам.
* Если оба операнда целочисленные, результат операции округляется вниз до ближайшего целого числа.
* Если хотя бы один из операндов вещественный, дробная часть результата деления не отбрасывается.
* Если делитель равен нулю, генерируется ошибка (исключительная ситуация).

### 3.8.2. Бинарная операция%

* Операция вычисления **остатка от деления (%)** определена для типов int, uint, long, ulong, float, double и decimal.
* Если оба операнда целочисленные, результат операции вычисляется по формуле

*х - (х / у) \* у.*

* Если хотя бы один из операндов вещественный, результат операции вычисляется по формуле

*х - n \* у*, где *n* — наибольшее целое, меньшее или равное результату деления х на у.

* Если делитель равен нулю, генерируется ошибка (исключительная ситуация).

int x, y, z;

Console.WriteLine("Введите целое число");

x = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Введите число");

y = Convert.ToInt32(Console.ReadLine());

z = x / y; //деление целых

Console.WriteLine("{0}/{1}={2}", x, y, z);

z = x % y;//деление целых

Console.WriteLine("{0}%{1}={2}", x, y, z);

double xd, yd, zd;

Console.WriteLine("Введите вещественное число");

xd = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Введите число");

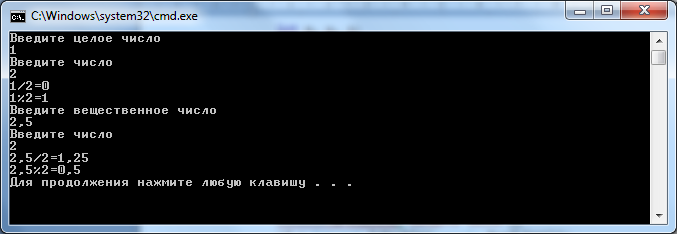
yd = Convert.ToDouble(Console.ReadLine());

zd = xd / yd;//деление вещественных

Console.WriteLine("{0}/{1}={2}", xd, yd, zd);

zd = xd % yd;// остаток от деления

Console.WriteLine("{0}%{1}={2}", xd, yd, zd);



### 3.8.3. Условная операция

Выражение1 ? Выражение2 : Выражение3;

* Первым вычисляется значение Выражения1.
* Если оно истинно, то вычисляется значение Выражения2, которое становится результатом.
* Если при вычислении Выражения1 получится false, то в качестве результата берется значение Выражения3.

Пример:

x<0 ? -x : x ; //абсолютное значение числа

## 3.9. Введение в исключения

* При вычислении выражений могут возникнуть ошибки, например, переполнение, исчезновение порядка или деление на ноль.
* В С# есть механизм, который позволяет обрабатывать подобные ошибки и таким образом избегать аварийного завершения программы: механизм обработки исключительных ситуаций (исключений).
* Если в процессе вычислений возникла ошибка, система сигнализирует об этом с помощью специального действия, называемого выбрасыванием (генерированием) исключения.
* Каждому типу ошибки соответствует свое исключение.
* Исключения позволяют разделить вычислительный процесс на две части:
  + обнаружение аварийной ситуации:   
    try{} // генерируется исключение
  + обработка аварийной ситуации:   
    catch{}//обрабатывается исключение

|  |  |
| --- | --- |
| **Имя стандартного исключения** | **Описание** |
| ArithmeticException | Ошибка в арифметических операциях |
| ArrayTypeMismatchException | 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 при создании объекта |
| OverFlowException | Переполнение при выполнении арифметических операций |
| StackOverFlowException | Переполнение стека |

Рисунок 3. Стандартные исключения

**Пример 6.**

try

{

Console.Write("Введите количество");

kolichestvo = Convert.ToInt32(Console.ReadLine());

. . . .

}

catch (FormatException)

{

Console.WriteLine("Неправильный формат ввода");

}

## 3.9. Организация ввода-вывода

* Совокупность стандартных устройств ввода и вывода (клавиатуры + экран) называется *консолью*.
* В классе Console существует несколько вариантов методов с именами Write() и WriteLine(), предназначенных для вывода значений различных типов.
* **Один параметр** может быть любого встроенного типа (число, символ, строка).
* **Два параметра** (текстовое пояснение и значение переменной): их требуется сцепить в одну строку с помощью операции +. При этом значение переменной неявно переводится в текстовую форму с помощью метода ToString(), который определен для всех встроенных типов.
* **Несколько переменных:** первым параметром методу передается строка, содержащая кроме обычных символов, предназначенных для вывода на консоль, параметры в фигурных скобках, а также управляющие последовательности (\n, \t). Параметры нумеруются с нуля, перед выводом они заменяются значениями соответствующих переменных в списке вывода.
* Формат параметров: {N[,W]:S[R]}, где
  + N – номер аргумента,
  + W - ширина поля в поле подстановки определяет количество позиций, выделяемых для изображения подставляемого значения. Если ширина поля не указана, то она определяется автоматически - минималь­но достаточной для изображения значения. Если ширина поля указана и превышает длину помещаемого в поле значения, то при положительной длине поля W значение выравнивается по правой границе. Если перед шириной поля W стоит минус, то выравнивание выполняется по левой границе поля.
  + S – спецификатор формата (задает вид изображае­мого значения), R – спецификатор точности:
    - С,с – валютный, R – количество десятичных разрядов.
    - D,d – целочисленный, R – минимальное количество цифр.
    - E,e – экспоненциальный, R – число разрядов после точки.
    - F,f – с фиксированной точкой, R – число разрядов после точки.
    - G,g – короткий из E или F.
    - Х,х – шестнадцатеричный, R – минимальное число цифр.

**Пример 7.**

int x = 5;

Console.WriteLine(x);

Console.Write("x=" + x + "\n");

int y = 10, z = 15;

Console.WriteLine("x={0}, y={1}, z={2}\n", x, y, z);

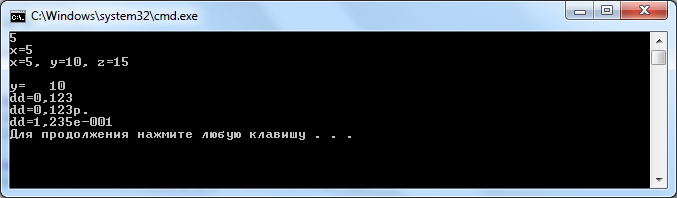
Console.WriteLine("y={0,5}",y);

double dd = 0.1234555678899;

Console.WriteLine("dd={0,5:f3}", dd);

Console.WriteLine("dd={0,5:c3}", dd);

Console.WriteLine("dd={0,5:e3}", dd);



* В классе Console определены методы ввода строки и отдельного символа, но нет методов, которые позволяют непосредственно считывать с клавиатуры числа.
* Ввод числовых данных выполняется в два этапа:
  1. Символы, представляющие собой число, вводятся с клавиатуры в строковую переменную.
  2. Выполняется преобразование из строки в переменную соответствующего типа.
* Преобразование можно выполнить:
  1. с помощью класса Convert, определенного в пространстве имен System,
  2. с помощью метода Parse, имеющегося в каждом стандартном арифметическом классе.

**Пример 8.**

Console.WriteLine("Введите строку");

string s = Console.ReadLine();

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

Console.WriteLine("Введите символ");

char с = (char)Console.Read();

Console.ReadLine();

Console.WriteLine("c = " + с);

string buf; // строка - буфер

Console.WriteLine("Введите целое число");

buf = Console.ReadLine();

int i = Convert.ToInt32(buf);

Console.WriteLine(i);

Console.WriteLine("Введите вещественное число");

buf = Console.ReadLine();

double x = Convert.ToDouble(buf);

Console.WriteLine(x);

Console.WriteLine("Введите вещественное число");

buf = Console.ReadLine();

double у = double.Parse(buf);

Console.WriteLine(у);

# Практическая работа №1

## Постановка задачи

1. Для задачи 1 определить тип заданных выражений и найти их значения.
2. Составить систему тестов и вычислить полученное выражение для нескольких значений Х, определить при каких Х выражение не может быть вычислено.
3. Для задачи 2 записать выражение, зависящее от координат точки X1 и Y1 и принимающее значение TRUE, если точка принадлежит заштрихованной области, и FALSE, если не принадлежит.
4. Составить систему тестов и вычислить полученное выражение для нескольких точек, принадлежащих и не принадлежащих заштрихованной области.
5. Для задачи 3 вычислить значение выражения, используя различные вещественные типы данных (float и double).
6. Результаты всех вычислений вывести на печать.
7. Объяснить полученные результаты.

|  |  |  |  |
| --- | --- | --- | --- |
| 7 | 1) m+--n  2) m++<--n  3) --m>n—  4) |  | а=100, b=0.001 |

**Задача 1**

namespace z1

{

class Program

{

static void Main(string[] args)

{

int m=0, n=0, res1=0;

bool res2, res3;

double x=0, y=0, res=0;

bool ok=true;

do

{

try

{

Console.WriteLine("Введите целое число m");

m = Convert.ToInt32(Console.ReadLine());

ok = true;

}

catch (FormatException)

{

Console.WriteLine("Не правильно введено число!");

ok = false;

}

}

while (!ok);

ok=true;

do

{

try

{

Console.WriteLine("Введите целое число n");

n=Convert.ToInt32(Console.ReadLine());

ok = true;

}

catch(FormatException)

{

Console.WriteLine("Не правильно введено число!");

ok=false;

}

}

while(!ok);

res1=m+--n;

Console.WriteLine("m={0}, n={1}, m++-n={2}",m,n,res1);

res2=m++<--n;

Console.WriteLine("m={0}, n={1}, m++<--n={2}", m, n, res2);

res3 = --m > n--;

Console.WriteLine("m={0}, n={1}, --m > n--={2}", m, n, res3);

ok = true;

do

{

try

{

Console.WriteLine("Введите вещественное число х");

x = Convert.ToInt32(Console.ReadLine());

ok = true;

}

catch (FormatException)

{

Console.WriteLine("Не правильно введено число!");

ok = false;

}

}

while (!ok);

ok = true;

do

{

try

{

Console.WriteLine("ВВведите вещественное число y");

y = Convert.ToInt32(Console.ReadLine());

ok = true;

}

catch (FormatException)

{

Console.WriteLine("Не правильно введено число!");

ok = false;

}

}

while (!ok);

res = Math.Pow(Math.Pow(x, 3) + Math.Pow(x, 4), 0.2) + 1 / Math.Tan(Math.Atan(x \* x));

Console.WriteLine("x={0}, y={1}, res={2,5:f3}", x, y, res);

}

}

}

**Задача 2**

class Program

{

static void Main(string[] args)

{

double x1, y1; bool f;

try

{

Console.WriteLine("Введите координаты");

x1 = Convert.ToDouble(Console.ReadLine());

y1 = Convert.ToDouble(Console.ReadLine());

f = (x1 \* x1 + y1 \* y1 < 4) && ((y1 < Math.Abs(x1) - 2) || (y1 > -Math.Abs(x1) + 2));

Console.WriteLine("x1={0}, y1={1}, (x1 \* x1 + y1 \* y1 < 4) и ((y1 <|x1| - 2) || (y1 > -|x1| + 2))={2}", x1, y1, f);

}

catch (FormatException)

{

Console.WriteLine("Неправильно введено число!");

}

}

}

**Задача 3**

class Program

{

static void Main(string[] args)

{

float a = 100f, b = 0.001f;

float c, d, e, f, h;

c = (float)Math.Pow(a - b, 3);//(a-b)^3

d = (float)Math.Pow(a, 3);//a^3

e = (float)Math.Pow(b, 3);//b^3

f = (float)3\*a \* b \* b;//ab^2

h = (float)3\*a \* a \* b;//a^2b

float j = (c - d) / (-e + f - h);

Console.WriteLine("float res=" + j);

double A = 100, B = 0.001;

double C, D, E, F, H;

C = Math.Pow(A - B, 3);//(a-b)^3

D = Math.Pow(A, 3);//a^3

E = Math.Pow(B, 3);//b^3

F = 3\*A\* B \* B;//ab^2

H = 3\*A \* A \* B;//a^2b

double J = (C - D) / (-E +F - H);

Console.WriteLine("double res=" + J);

}

}