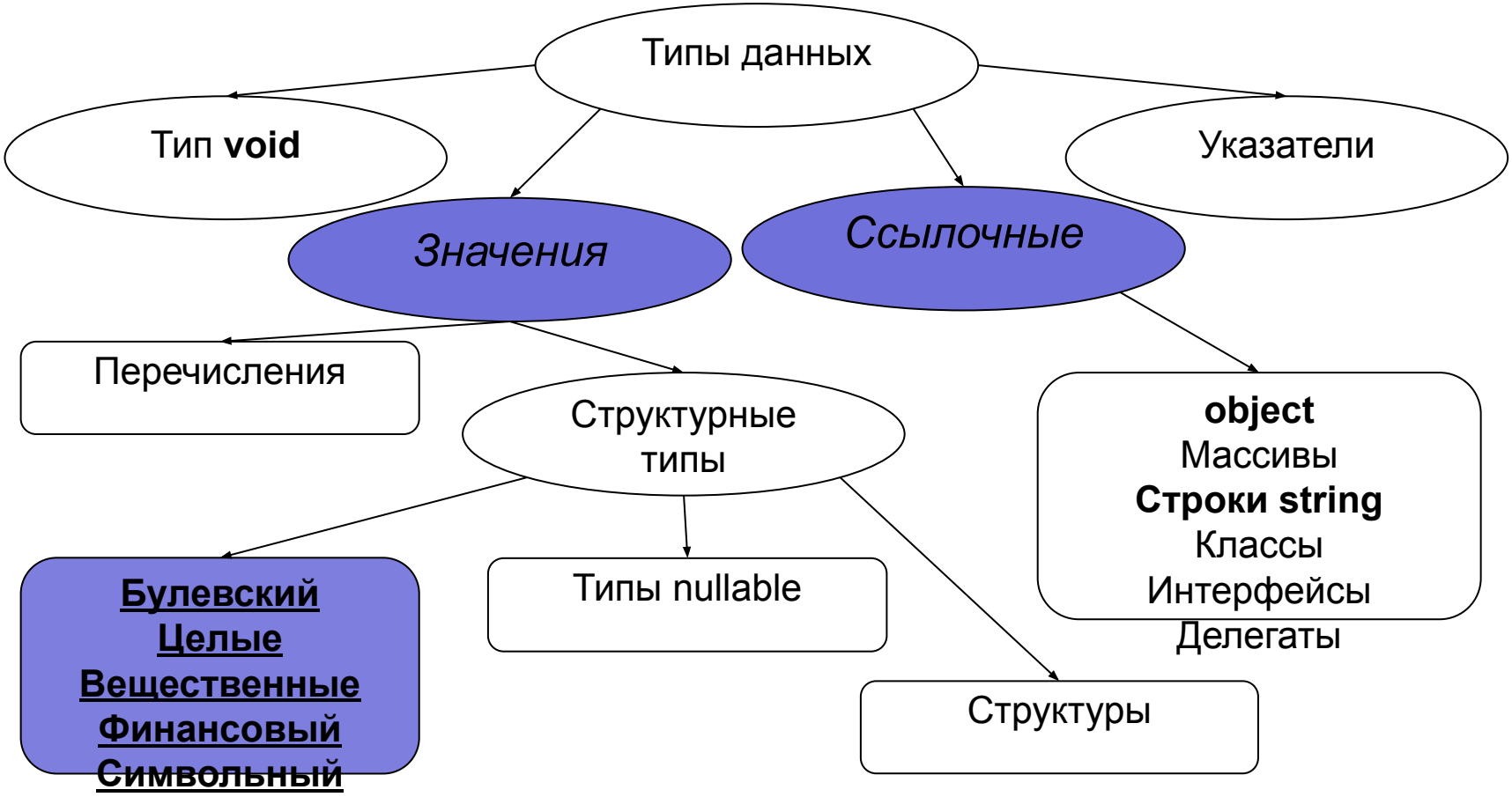
АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Концепция типа данных

Тип данных определяет:

- внутреннее представление данных => множество их возможных значений
- допустимые действия над данными => операции и функции



Типы данных C#

Типы данных бывают разные: одни заданы раз и навсегда в самом языке (базовые типы), другие программист сам задает (например, структуры, классы, объекты).

Базовым типам сопоставлены фиксированные ключевые слова, не базовые типы программист сам именует.

Вид	Примеры
Булевские	*true* *false*
Целые дес. шестн.	8 199226 0xA 0x1B8 0X00FFL
Веществ. с тчк. с порядком	5.7 .001f 0.2E6 .11e–3 5E10
Символьные	'A' '\x74' '\0'
Строковые	“Привет”
Константа	*null*

Логический и целые

Название	Ключевое слово	Тип .NET	Диапазон значений	Описание	Размер, бит
Булевский	bool	**Boolean**	**true, false**		
Целые	sbyte	**SByte**	–128 — 127	знаковое	8
	byte	**Byte**	0 — 255	беззнаковое	8
	short	**Int16**	–32768 —32767	знаковое	16
	ushort	**UInt16**	0 — 65535	беззнаковое	16
	int	**Int32**	$\approx(-210^9 — 210^9)$	знаковое	32
	uint	**UInt32**	$\approx(0 — 410^9)$	беззнаковое	32
	long	**Int64**	$\approx(-910^{18} — 910^{18})$	знаковое	64
	ulong	**UInt64**	$\approx(0— 1810^{18})$	беззнаковое	64

Остальные

Название	Ключевое слово	Тип .NET	Диапазон значений	Описание	Размер, бит
Символьный	char	**Char**	U+0000 — U+ffff	символ Unicode	16
Веществен-ные	float	**Single**	$1.5 10^{-45}$ — $3.4 10^{38}$	7 цифр	32
	double	**Double**	$5.0 10^{-324}$ — $1.7 10^{308}$	15-16 цифр	64
Финансовый	decimal	**Decimal**	$1.0 10^{-28}$ — $7.9 10^{28}$	28-29 цифр	128
Строковый	string	**String**	длина ограничена объемом доступной памяти	строка из символов Unicode	
object	object	**Object**	можно хранить все, что угодно	всеобщий предок	

Переменные

В C# принято объявлять данные:

тип_переменной имя_переменной

- *Переменная* — это величина, которая во время работы программы может изменять свое значение.
- Все переменные, используемые в программе, должны быть описаны.
- *Для* каждой переменной задается ее *имя и тип*:

```
 int     number;
float    x, y;
char     option;
```

Тип переменной выбирается исходя из диапазона и требуемой точности представления данных.

Область действия и время жизни переменных

- Переменные описываются внутри какого-либо блока (класса, метода или блока внутри метода)
  - Блок — это код, заключенный в фигурные скобки. Основное назначение блока — группировка операторов.
  - Переменные, описанные непосредственно внутри класса, называются полями класса.
  - Переменные, описанные внутри метода класса, называются локальными переменными.
- Область действия переменной - область программы, где можно использовать переменную.
- Область действия переменной начинается в точке ее описания и длится до конца блока, внутри которого она описана.
- Время жизни: переменные создаются при входе в их область действия (блок) и уничтожаются при выходе.

Инициализация переменных

• При объявлении можно присвоить переменной начальное значение (инициализировать).

```
int   number = 100;
float      x     = 0.02;
char      option = ’ю’;
```

```
int a = 4;

System.Int32 b = 4;
```

При инициализации можно использовать не только константы, но и выражения - главное, чтобы на момент описания они были вычислимыми, например:

```
int b = 1, a = 100;

int x = b * a + 25;
```

• Поля класса инициализируются «значением по умолчанию» (0 соответствующего типа).

• Рекомендуется всегда инициализировать переменные при описании.

Инициализация переменных

При присвоении значений надо учитывать: все вещественные значения (дробные числа) рассматриваются как значения типа **double**. Чтобы указать, что дробное число представляет тип **float** или тип **decimal**, необходимо к при описание добавлять суффикс: F/f - для float и M/m - для decimal.

```
float a = 3.14F;
float b = 30.6f;

decimal c = 1005.8M;
decimal d = 334.8m;
```

Все целочисленные переменные рассматриваются как значения типа **int**. Чтобы явным образом указать, что целочисленное значение представляет значение типа uint, надо использовать суффикс **U/u**, для типа **long** - суффикс **L/l**, а для типа **ulong** - суффикс **UL/ul**:

```
uint a = 10U;
long b = 20L;
ulong c = 30UL;
```

;

Неявная типизация

Можно использовать модель неявной типизации:

var a = "Hello word ";

var c = 20;

Для неявной типизации вместо названия типа данных используется ключевое слово **var**. При компиляции компилятор сам выводит тип данных исходя из присвоенного значения. Так как по умолчанию все целочисленные значения рассматриваются как значения типа **int**, переменная **c** будет иметь тип **int**.

Переменной **hello** присваивается строка, поэтому эта переменная будет иметь тип **string.**

Эти переменные подобны обычным, однако они имеют некоторые ограничения.

Нельзя сначала объявить неявно типизируемую переменную, а затем ее инициализировать:

var a;

a = 20;

Нельзя указать в качестве значения неявно типизируемой переменной null:

var c=null;

Именованные константы

Вместо значений констант можно (и нужно!) использовать в программе их имена.

Это облегчает читабельность программы и внесение в нее изменений.

При использовании констант надо помнить, что объявить их можно только один раз и к моменту компиляции они должны быть определены.

```
const  float weight = 61.5;
const  int    n        = 10;
const  float  g        = 9.8;
```

Ошибки:

1. `const  float weight;`
2. `const  float weight = 61.5;`
   `weight = 67;`

Литералы

Литералы представляют неизменяемые значения (иногда их называют константами). Их можно передавать переменным в качестве значения. Бывают логическими, целочисленными, вещественными, символьными и строчными.

Отдельный литерал представляет ключевое слово `null`

Логические литералы представлены двумя логическими константами - **true** (истина) и **false** (ложь):

Console.WriteLine(true);

Console.WriteLine(**false**);

Литералы

Целочисленные литералы представляют положительные и отрицательные целые числа, например, 1, 2, 3, 4, -7, -109.

Целочисленные литералы могут быть выражены в десятичной, шестнадцатеричной и двоичной форме.

Десятичное представление:

Console.WriteLine(-11);

Двоичное представление:

Числа в двоичной форме предваряются символами 0b, после которых идет набор из нулей и единиц:

Console.WriteLine(0b11);

Шестнадцатиричное представление:

Для записи числа в шестнадцатеричной форме применяются символы 0x, после которых идет набор символов от 0 до 9 и от A до F, которые собственно представляют число:

Console.WriteLine(0x0A);

Литералы

Вещественные литералы представляют собой дробные числа. Этот тип литералов имеет две формы.

Первая форма - вещественные числа с фиксированной запятой, при которой дробную часть отделяется от целой части точкой.

Console.WriteLine(3.14);

Console.WriteLine(-0.35);

Вторая форма - вещественные литералы могут определяться в экспоненциальной форме MEp, где M — мантисса, E - экспонента, которая фактически означает "*10^" (умножить на десять в степени), а p — порядок.

Console.WriteLine(3.2e3); 3200

Console.WriteLine(3.2e-1); 0.32

Литералы

Символьные литералы представляют одиночные символы. Символы заключаются в одинарные кавычки.

Обычные:

Console.WriteLine(‘A’);

Управляющие последовательности:

Управляющая последовательность представляет символ, перед которым ставится слеш. Данная последовательность интерпретируется определенным образом.

- '\n' - перевод строки
- '\t' - табуляция - длинный отступ
- '\\' - слеш

Символы могут определяться в виде шестнадцатеричных кодов. Для этого в одинарных кавычках указываются символы '\x', после которых идет шестнадцатеричный код символа из таблицы ASCII.

Console.WriteLine(“\x38”);

Символы могут определяться с применением кодов из таблицы символов Unicode/ Для этого в одинарных кавычках указываются символы '\u', после которых идет шестнадцатеричный код Unicode. Например, код '\u0411' представляет кириллический символ 'Б‘.

Литералы

Строковые литералы представляют строки. Строки заключаются в двойные кавычки:

Console.WriteLine(“Hello word");

Для вывода внутри строки двойной кавычки, необходимо перед ней ставить слеш:

Console.WriteLine("Компания \"Кондор\"");

В строках можно использовать управляющие последовательности. Например, последовательность '\n' осуществляет перевод на новую строку:

Console.WriteLine("Hello \nword ");

null представляет ссылку, которая не указывает ни на какой объект. То есть по сути отсутствие значения.

Выражения

- *Выражение* - правило вычисления значения.
- В выражении участвуют *операнды*, объединенные знаками операций.
- Операндами выражения могут быть константы, переменные и вызовы функций.
- Операции выполняются в соответствии с *приоритетами*.
- Для изменения порядка выполнения операций используются *круглые скобки*.
- Результатом выражения всегда является значение определенного типа, который определяется типами операндов.
- Величины, участвующие в выражении, должны быть *совместимых типов*.

- t + Math.Sin(x)/2 * x

результат имеет вещественный тип

- a <= b + 2

результат имеет логический тип

- x > 0 && y < 0

результат имеет логический тип

Тип результата выражения

- *Если операнды*, *входящие в выражение*, *одного типа*, *и операция для этого типа определена*, *то результат выражения будет иметь тот же тип*.
- Если операнды разного типа и (или) операция для этого типа не определена, перед вычислениями автоматически выполняется преобразование типа по правилам, обеспечивающим приведение более коротких типов к более длинным для сохранения значимости и точности.
- Автоматическое (*неявное*) преобразование возможно не всегда, а только если при этом не может случиться потеря значимости.
- Если неявного преобразования из одного типа в другой не существует, программист может задать *явное* преобразование типа с помощью операции (тип)x.

Приоритеты операций C#

1. Первичные (), [], ++, --, new, …
2. Унарные ~, !, ++, --, -, …
3. Типа умножения (мультипликативные) *, /, %
4. Типа сложения (аддитивные) +, -
5. Сдвига <<, >>
6. Отношения и проверки типа <, >, is, …
7. Проверки на равенство ==, !=
8. Поразрядные логические &, ^, |
9. Условные логические &&, ||
10. Условная ?:
11. Присваивания =, *=, /=,…

- Инкремент, декремент
- Умножение, деление, получение остатка
- Сложение, вычитание

Инкремент

```
int x = 3, y = 3;
Console.Write( "Значение префиксного выражения: " );
Console.WriteLine( ++x );
Console.Write( "Значение x после приращения: " );
Console.WriteLine( x );
```

```
int x = 3, y = 3;
Console.Write( "Значение постфиксного выражения: " );
Console.WriteLine( y++ );
Console.Write( "Значение y после приращения: " );
Console.WriteLine( y );
```

Результат работы программы:

```
Значение префиксного выражения: 4
Значение x после приращения: 4
Значение постфиксного выражения: 3
Значение y после приращения: 4
```

Декремент

```
int x = 3, y = 3;
Console.Write( "Значение префиксного выражения: " );
Console.WriteLine( --x );
Console.Write( "Значение x после приращения: " );
Console.WriteLine( x );
```

```
int  a = 3;
int  b = 5;
int  c = 40;
int  d = c- - - b * a;    // a=3  b=5  c=39  d=?
```

```
int x = 3, y = 3;
Console.Write( "Значение постфиксного выражения: " );
Console.WriteLine( y-- );
Console.Write( "Значение y после приращения: " );
Console.WriteLine( y );
```

Результат работы программы:

```
Значение префиксного выражения: 2
Значение x после приращения: 2
Значение постфиксного выражения: 3
Значение y после приращения: 2
```

Операции отрицания

```
sbyte a = 3, b = -63, c = 126;
        bool  d = true;
        Console.WriteLine( -a );   // Результат -3
        Console.WriteLine( -c );   // Результат -126
        Console.WriteLine( !d );   // Результат false
        Console.WriteLine( ~a );   // Результат -4
        Console.WriteLine( ~b );   // Результат 62
        Console.WriteLine( ~c );   // Результат -127
```

Операции сдвига

- *Операции сдвига* (<< и >>) применяются к целочисленным операндам. Они сдвигают двоичное представление первого операнда влево или вправо на количество двоичных разрядов, заданное вторым операндом.
- При *сдвиге влево* (<<) освободившиеся разряды обнуляются. При *сдвиге вправо* (>>) освободившиеся биты заполняются нулями, если первый операнд беззнакового типа, и знаковым разрядом в противном случае.
- Стандартные операции сдвига определены для типов int, uint, long и ulong.

```
byte  a = 3, b = 9;
sbyte c = 9, d = -9;
Console.WriteLine( a << 1 );      // Результат 6
Console.WriteLine( a << 2 );      // Результат 12
Console.WriteLine( b >> 1 );      // Результат 4
Console.WriteLine( c >> 1 );      // Результат 4
Console.WriteLine( d >> 1 );      // Результат -5
```

Умножение

- *Операция умножения* (*) возвращает результат перемножения двух операндов.
- Стандартная операция умножения 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. Тип результата операции равен «наибольшему» из типов операндов, но не менее int.
- Если оба операнда целочисленные или типа decimal и результат операции слишком велик для представления с помощью заданного типа, генерируется исключение System.OverflowException

Пример

```
int x = 11, y = 4;
float z = 4;
Console.WriteLine( z * y );            // Результат 16
Console.WriteLine( z * 1e308 );     // Рез. "бесконечность"
Console.WriteLine( x / y );            // Результат 2
Console.WriteLine( x / z );            // Результат 2,75
Console.WriteLine( x % y );            // Результат 3
Console.WriteLine( 1e-324 / 1e-324 );   // Результат NaN
```

Операции отношения и проверки на равенство

- *Операции отношения* (<, <=, >, >=, ==, !=) сравнивают первый операнд со вторым.
- Операнды должны быть арифметического типа.
- Результат операции — логического типа, равен true или false.

x == y -- true, если x равно y, иначе false
x != y -- true, если x не равно y, иначе false
x < y -- true, если x меньше y, иначе false
x > y -- true, если x больше y, иначе false
x <= y -- true, если x меньше или равно y, иначе false
x >= y -- true, если x больше или равно y, иначе false

Условные логические операции

```
Console.WriteLine( true && true  );      // Результат true
Console.WriteLine( true && false );      // Результат false
Console.WriteLine( true || true  );      // Результат true
Console.WriteLine( true || false );      // Результат true
```

Условная операция

- **операнд_1 ? операнд_2 : операнд_3**

Первый операнд — выражение, для которого существует неявное преобразование к логическому типу.

Если результат вычисления первого операнда равен true, то результатом будет значение второго операнда, иначе — третьего операнда.

```
int a = 11, b = 4;
        int max = b > a ? b : a;
        Console.WriteLine( max );      // Результат 11
```

Сложное присваивание в C#

x += 0.5;	соответствует	x = x + 0.5;
x *= 0.5;	соответствует	x = x * 0.5;
a %= 3;	соответствует	a = a % 3;
a <<= 2;	соответствует	a = a << 2;

Явное преобразование типа

- long b = 300;
- int a = (int) b; // данные не теряются
- byte d = (byte) a; // данные теряются

Преобразование типа

Тип	В какие типы безопасно преобразуется
byte	short, ushort, int, uint, long, ulong, float, double, decimal
sbyte	short, int, long, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
float	double
char	ushort, int, uint, long, ulong, float, double, decimal

Введение в исключения

- При вычислении выражений могут возникнуть ошибки (переполнение, деление на ноль).
- В C# есть механизм *обработки исключительных ситуаций* (*исключений*), который позволяет избегать аварийного завершения программы.
- Если в процессе вычислений возникла ошибка, система сигнализирует об этом с помощью *выбрасывания (генерирования) исключения*.
- Каждому типу ошибки соответствует свое исключение. Исключения являются классами, которые имеют общего предка — класс Exception, определенный в пространстве имен System.
- Например, при делении на ноль будет выброшено исключение DivideByZeroException, при переполнении — исключение OverflowException.

Консольный ввод

Для этого предназначен метод **Console.ReadLine()**. Он позволяет получить введенную строку.

```
name = Console.ReadLine();
```

Метод **Console.ReadLine()** считывает информацию с консоли только в виде строки. Если не окажется доступных для считывания строк, метод возвращаает значение **null**.

По умолчанию платформа .NET предоставляет ряд методов, которые позволяют преобразовать различные значения к типам **int**, **double** и **т.д**.

- **Convert.ToInt32()** (преобразует к типу int)
- **Convert.ToDouble()** (преобразует к типу double)
- **Convert.ToDecimal()** (преобразует к типу decimal)

```
age = Convert.ToInt32(Console.ReadLine());
```

Консольный вывод

Для вывода информации на консоль используется встроенный метод **Console.WriteLine**. Для вывода информации на консоль, необходимо передать ее в метод Console.WriteLine.

```
string  h = "Привет";
Console.WriteLine(h);
```

1. Для вывода на консоль в одной строке значений сразу нескольких переменных используется интерполяция:

```
string name = "Муся";
int age = 2;
double height = 0,5;
Console.WriteLine($"Имя: {name}  Возраст: {age}  Рост: {height}м");
```

2.

```
string name = "Муся";
int age = 2;
double height = 0,5;
Console.WriteLine("Имя: {0}  Возраст: {2}  Рост: {1}м", name, height, age);
```

Можно также использовать метод Console.Write(), он не добавляет переход на следующую строку, последующий консольный вывод будет выводиться на той же строке.

Консольный вывод

3.

```
int       i = 3;
double  y = 4.12;
decimal d = 600m;
string    s = "Вася";
Console.Write( i );
Console.Write( " y = {0} \nd = {1}", y, d );
Console.WriteLine( " s = " + s );
```

```
Console.WriteLine(i + " y = "  + y);
Console.WriteLine("d = " + d + " s = " + s );
```

Результат работы программы:

```
3 y = 4,12
d = 600 s = Вася
```

Поля и методы встроенных типов

Любой встроенный тип C# построен на основе стандартного класса библиотеки .NET. Это значит, что у встроенных типов данных C# есть *методы и поля*. С помощью них можно, например, получить:

- **double.MaxValue** (или System.Double.MaxValue) - максимальное число типа double;
- **uint.MinValue** (или System.UInt32.MinValue) - минимальное число типа uint.

В вещественных классах есть элементы:

- положительная бесконечность **PositiveInfinity**;
- отрицательная бесконечность **NegativeInfinity**;
- «не является числом»: **NaN**.

Математические функции: класс Math

Имя	Описание	Результат	Пояснения
`Abs`	Модуль	перегружен	|x| записывается как `Abs(x)`
`Acos`	Арккосинус	`double`	`Acos(double x)`
`Asin`	Арксинус	`double`	`Asin(double x)`
`Atan`	Арктангенс	`double`	`Atan(double x)`
`Atan2`	Арктангенс	`double`	`Atan2(double x, double y)` — угол, тангенс которого есть результат деления `y` на `x`
`BigMul`	Произведение	`long`	`BigMul(int x, int y)`
`Ceiling`	Округление до большего целого	`double`	`Ceiling(double x)`
`Cos`	Косинус	`double`	`Cos(double x)`
`Cosh`	Гиперболический косинус	`double`	`Cosh(double x)`
`DivRem`	Деление и остаток	перегружен	`DivRem(x, y, rem)`
`E`	База натурального логарифма (число *e*)	`double`	2,71828182845905
`Exp`	Экспонента	`double`	e^x записывается как `Exp(x)`

Floor	Округление до меньшего целого	double	Floor(double x)
IEEERemainder	Остаток от деления	double	IEEERemainder(double x, double y)
Log	Натуральный логарифм	double	$\log_e x$ записывается как Log(x)
Log10	Десятичный логарифм	double	$\log_{10} x$ записывается как Log10(x)
Max	Максимум из двух чисел	перегружен	Max(x, y)
Min	Минимум из двух чисел	перегружен	Min(x, y)
PI	Значение числа π	double	3,14159265358979
Pow	Возведение в степень	double	x^y записывается как Pow(x, y)
Round	Округление	перегружен	Round(3.1) даст в результате 3 Round (3.8) даст в результате 4
Sign	Знак числа	int	аргументы перегружены
Sin	Синус	double	Sin(double x)
Sinh	гиперболический синус	double	Sinh(double x)
Sqrt	Квадратный корень	double	$\sqrt{x}$ записывается как Sqrt(x)
Tan	Тангенс	double	Tan(double x)
Tanh	Гиперболический тангенс	double	Tanh(double x)