АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

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

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
```

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)**

Условные операторы

Условные конструкции - одни из базовых компонентов многих языков программирования, которые направляют работу программы по одному из путей в зависимости от определенных условий.

Одной из таких конструкций в языке программирования C# является конструкция

if..else

Конструкция проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет определенный код.

Простая форма:

```
if (условие)
{
    выполняемые инструкции
}
```

После ключевого слова if ставится условие. Условие должно представлять значение типа bool. Это может быть непосредственно значение типа bool или результат условного выражения или другого выражения, которое возвращает значение тиа bool. И если это условие истинно (равно true), то срабатывает код, который помещен после условия внутри фигурных скобок.

Условные операторы

Полная форма:

if (условие)

{

выполняемые инструкции

}

else

{

выполняемые инструкции

}

Блок else выполняется, если условие после if ложно, то есть равно false.

Условные операторы

Конструкция ***if … else if*** :

```
if (условие)
{
    выполняемые инструкции
}
else if (условие)
{
    выполняемые инструкции
}
else
{
    выполняемые инструкции
}
```

Оператор ветвления

Оператор ветвления *switch:*

Оператор ветвления *switch* является альтернативой оператору *if* … *else*. Его обычно использую в случаях, когда имеется более сложный набор условий, состоящий из нескольких вариантов.

Суть работы: программа ищет значение, которое соответствует переменной для сравнения, и далее выполняет указанные инструкции.

Структура оператора:

switch (выражение)
{
case *возможный вариант выражения1*:
  код, выполняемый если выражение имеет значение1;
  break; //место выхода из case
case *возможный вариант выражения*2:
  код, выполняемый если выражение имеет значение2;
  break;//место выхода из case
…
case *возможный вариант выражения*N:
код, выполняемый если выражение имеет значениеN
break;
default:
код, выполняемый если выражение не имеет ни одно из выше указанных значений
break;
}

Оператор ветвления

Если необходимо, чтобы после выполнения текущего блока **case** выполнялся другой блок **case**, можно использовать вместо **break** оператор **goto case**:

```
int number = 1;
switch (number)
{
    case 1:
        Console.WriteLine("case 1");
        goto case 5; // переход к case 5
    case 3:
        Console.WriteLine("case 3");
        break;
    case 5:
        Console.WriteLine("case 5");
        break;
    default:
        Console.WriteLine("default");
        break;
}
```

Оператор ветвления

Возвращение значения из switch

Конструкция **switch** позволяет возвращать некоторое значение. Для возвращения значения в блоках **case** используется оператор **return**.

Пример:

```
int D1(int o1, int a, int b)
{
    switch (o1)
    {
        case 1: return a + b;
        case 2: return a - b;
        case 3: return a * b;
        default: return 0;
    }
}

int result1 = D1(1, 10, 5);          // 15
Console.WriteLine(result1);          // 15
int result2 = D1(3, 10, 5);          // 50
Console.WriteLine(result2);          // 50
```

Оператор ветвления

Получение результата непосредственно из конструкции **switch**:

Пример_1:

```
int D1(int o1, int a, int b)
{
    int result = o1 switch {
        1 => a + b,
        2 => a - b,
        3 => a * b,
        _ => 0
    };
    return result;
}
```

Теперь не требуется оператор **case**, а после сравниваемого значения ставится оператор стрелка **=>**. Значение справа от стрелки выступает в качестве возвращаемоего значения. Вместо оператора **default** используется почерк _. В итоге результат конструкции switch будет присвиваиваться переменной result.

Оператор ветвления

Получение результата непосредственно из конструкции **switch**:

Пример_2:

```
int D1(int o1, int a, int b)
{
    return o1 switch
    {
        1 => a + b,
        2 => a - b,
        3 => a * b,
        _ => 0
    };
}
```

Пример_3:

```
int D1(int o1, int a, int b) => o1 switch
{
    1 => a + b,
    2 => a - b,
    3 => a * b,
    _ => 0
};
```

Циклы

Циклы являются управляющими конструкциями, позволяя в зависимости от определенных условий выполнять некоторое действие множество раз. В C# имеются следующие виды циклов:

- for
- foreach
- while
- do...while

Цикл for

Цикл **for** имеет следующее формальное описание:

```
for ([действия_до_выполнения_цикла]; [условие]; [действия_после_выполнения])
{
    // тело цикла
}
```

Объявление цикла for состоит из трех частей.

Первая часть объявления цикла - действия, которые выполняются один раз до выполнения цикла (определение переменные, которые будут использоваться в цикле).

Вторая часть - условие, при котором будет выполняться цикл. Пока условие равно true, будет выполняться цикл.

Третья часть - действия, которые выполняются после завершения блока цикла. Эти действия выполняются каждый раз при завершении блока цикла.

Циклы

Пример:

```
for (int i = 1; i < 4; i++)
{
    Console.WriteLine(i);
}
```

Первая часть объявления цикла - int i = 1 - создает и инициализирует переменную i.

Вторая часть - условие i < 4. Пока переменная i меньше 4, будет выполняться цикл.

Третья часть - действия, выполняемые после завершения тела цикла - увеличение переменной i на единицу.

Весь процесс цикла можно представить следующим образом:

- Определяется переменная int i = 1
- Проверяется условие i < 4. Оно истинно (так как 1 меньше 4), поэтому выполняется блок цикла, а именно инструкция Console.WriteLine(i), которая выводит на консоль значение переменной i
- Блок цикла закончил выполнение, поэтому выполняется третья часть объявления цикла - i++. После этого переменная i будет равна 2.

Циклы

- Проверяется условие i < 4. Оно истинно (так как 2 меньше 4), выполняется блок цикла - Console.WriteLine(i)
- Блок цикла закончил выполнение, выполняется выражение i++. После этого переменная i будет равна 3.
- Проверяется условие i < 4. Оно истинно (так как 3 меньше 4), выполняется блок цикла - Console.WriteLine(i)
- Блок цикла закончил выполнение, выполняется выражение i++. После этого переменная i будет равна 4.
- Проверяется условие i < 4. Теперь оно возвражает false, так как значение переменной i НЕ меньше 4, цикл завершает выполнение. Далее выполняется остальная часть программы, которая идет после цикла

В итоге блок цикла сработает 3 раза, пока значение i не станет равным 4. И каждый раз это значение будет увеличиваться на 1. Однократное выполнение блока цикла называется итерацией. Таким образом, здесь цикл выполнит три итерации.

Циклы

Пример:

```
int i= 1;
for (Console.WriteLine("Начало выполнения цикла"); i < 4; Console.WriteLine($"i = {i}"))
{
    i++;
}
```

Пример:

```
int i = 1;
for (; ;)
{
    Console.WriteLine($"i = {i}");
    i++;
}
```

Пример:

```
int i = 1;
for (; i<4;)
{
    Console.WriteLine($"i = {i}");
    i++;
}
```

Циклы

Пример:

```
for (int i = 1, j = 1; i < 10; i++, j++)
    Console.WriteLine($"{i * j}");
```

Пример:

```
int i = 1;for (; i<4 && i>0;)
{
      Console.WriteLine($"i = {i}");
       i++;
}
```

Циклы

Цикл **do..while**

В цикле do сначала выполняется код цикла, а потом происходит проверка условия в инструкции while. И пока это условие истинно, цикл повторяется.

Цикл имеет следующее формальное описание:

```
do
{
    //тело цикла
}
while (условие)
```

Пример:

```
int i = 6;
do
{
    Console.WriteLine(i);
    i--;
}
while (i > 0);
```

Цикл do гарантирует хотя бы единократное выполнение действий, даже если условие в инструкции while не будет истинно.

Циклы

Цикл **while**

Цикл **while** сразу проверяет истинность некоторого условия, и если условие истинно, то код цикла выполняется.

Цикл имеет следующее формальное описание:

```
while (условие)
{
    //тело цикла
}
```

Пример:

```
int i = 6;
while (i > 0)
{
    Console.WriteLine(i);
    i--;
}
```

Циклы

Цикл **foreach**

Цикл **foreach** предназначен для перебора набора или коллекции элементов.

Цикл имеет следующее формальное описание:

```
foreach(тип_данных переменная in коллекция)
{
    // тело цикла
}
```

После оператора **foreach** в скобках сначала идет определение переменной. Затем ключевое слово **in** и далее коллекция, элементы которой надо перебрать.

При выполнении цикл последовательно перебирает элементы коллекции и помещает их в переменную, и таким образом в блоке цикла можно выполнить с ними некоторые действия.

Пример:

```
foreach(char c in "Tom")
{
    Console.WriteLine(c);
}
```

Циклы

Цикл **foreach**

Определяемая в объявлении цикла переменная должна по типу соответствовать типу элементов перебираемой коллекции. Так, элементы строки - значения типа char - символы. Поэтому переменная c имеет тип char.

При неопределенности типа элементов коллекции, можно определить переменную с помощью оператора **var**:

```
foreach (var c in "Tom")
{
    Console.WriteLine(c);
}
```

Вложенные циклы

```
for (int i = 1; i < 10; i++)
{
    for (int j = 1; j < 10; j++)
    {
        Console.Write($"{i * j} \t");
    }
    Console.WriteLine();
}
```

Операторы continue и break

Оператор **break**

Оператор позволяет выйти из цикла, не дожидаясь его завершения.

Пример:

```
for (int i = 0; i < 9; i++)
{
    if (i == 5)
        break;
    Console.WriteLine(i);
}
```

Оператор **break**

Оператор позволяет не завершать цикл, а просто пропускал текущую итерацию.

Пример:

```
for (int i = 0; i < 9; i++)
{
    if (i == 5)
        continue;
    Console.WriteLine(i);
}
```

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**.

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.