Лабораторная работа 2

2. Операторы и выражения

Варианты заданий

Выполнить задания 1-4 по предложенным вариантам:

	№ Задания			
	1	2	3	4
Вариант 1	1.I	2.I	3.I	4.I
Вариант 2	1.II	2.II	3.II	4.II
Вариант 3	1.III	2.III	3.III	4.III
Вариант 4	1.I	2.II	3.II	4.I
Вариант 5	1.II	2.I	3.III	4.II
Вариант 6	1.III	2.II	3.I	4.I
Вариант 7	1.I	2.III	3.I	4.II
Вариант 8	1.I	2.I	3.II	4.III
Вариант 9	1.II	2.III	3.I	4.II
Вариант 10	1.III	2.I	3.III	4.I

Задание 1.

При вычислениях не использовать класс Math. Только операции умножения.

I. Дано число x. Вычислите число x^6 при помощи трех операций умножения.

II. Дано число x. Вычислите число x^8 при помощи трех операций умножения.

III. Дано число x. Вычислите число x^7 при помощи четырех операций умножения.

Задание 2.

При вычислениях не использовать класс Math. Только арифметические операции.

I. Дано натуральное число. Найдите число десятков в его десятичной записи (то есть вторую справа цифру его десятичной записи)

II. Дано трехзначное число. Найдите сумму его цифр.

III. Присвоить целой переменной d первую цифру из дробной части положительного вещественного числа (так, если x=32.975, то d=9).

Задание 3.

Приложение вычисляет значение x и выводит его на консоль,

где m, n, p, a, b, c, d, e – это имена изменяемых параметров (вводятся пользователем с

консоли). Вычисление выражения построить так, чтобы минимизировать время его вычисления. Использовать класс Math.

Оцените время вычисления выражения в условных единицах (уе), исходя из следующих предположений: присваивание - 1 уе, операции сдвига -2 уе, сложение, вычитание - 3 уе, умножение - 5 уе, деление - 7 уе, вызов стандартной функции - 13 уе.

I. $$x = \frac{(123 + 527 * a / b^{2p}) * (123 - 527 * a / b^{2p})}{\sqrt[3]{856 + c^n * d^m} * (e - \lg 6)}$$

II. $$x = \frac{2^{1/n} * (12,3 + 52,7 * a^n / b^{2,5}) * (12,3 - 52,7 * a^n / b^{2,5})}{\ln(\sqrt{132,5 + \sin e^m}) - \cos(d^p * c)}$$

III. $$x = \frac{2^n (\sin(a) + \cos(b * c / d^m) * (\sin(a) - \cos(b * c / d^m)}{\lg(\sqrt[3]{a + b * c / d^m}) - e / 2^p}$$

Задание 4. (Необходимые для решения задачи значения введите самостоятельно).

I. Российские ученые из Сколково разработали ПО для марсохода. Чтобы доставить марсоход с наименьшими затратами, необходимо дождаться 2018 года, когда Марс приблизится к Земле на минимальное расстояние. Т.к. это отечественная разработка, используются исторические, проверенные временем единицы измерения длины. Для правильной доставки необходимо рассчитать минимальное расстояние между Землей и Марсом в аршинах.

II. Капитан Джек Воробей всегда точно рассчитывает расстояние до горизонта, пользуясь формулой $S = [(R+h)^2 - R^2]^{1/2}$ где:

- R - радиус Земли;
- h - высота глаз наблюдателя над поверхностью в метрах.

Оказавшись на необитаемом острове Кокос, сможет ли он увидеть сушу с самой высокой точки острова.

III. Филиальные сети крупных компаний охватывают все бОльшие территории. Однако при открытии офисов в отдаленных районах развивающихся стран многие сталкиваются со слаборазвитой ИТ-инфраструктурой. В этом случае для передачи медиаконтента придумали использовать специально обученных собак, переносящих террабайтные жесткие диски со средней скоростью 12 км/ч. Между каждыми 2-мя офисами курсирует одна собака. К прибытию собаки в офисе успевают накопиться данные, чтобы заполнить диск полностью. На каком расстоянии должны находиться офисы, чтобы сравнительно выгоднее (с точки зрения времени передачи) было передавать данные по оптоволоконной сети со скоростью 1 Гигабит/с.

Методические указания

Оглавление

Код приложений в C# состоит из операторов, состоящих из ключевых слов, выражений и операторов.

Для ввода числовых значений используются преобразования класса Convert, либо метода Parse() для каждого из числовых типов.

Например:

```
string buf = Console.ReadLine(); //считываем строку – буфер для ввода числа
int x1 = Convert.ToInt32(buf); // целое число (1 вариант)
int x2 = Int32.Parse(Console.ReadLine()); // считываем и преобразовываем целое
                                          // число сразу (2 вариант)
// для вещественных чисел - аналогично
double y1 = Convert.ToDouble(Console.ReadLine()); // (1 вариант)
double y2 = Double.Parse(Console.ReadLine()); // (2 вариант)
```

Стоит помнить, что в коде в качестве разделителя для вещественного числа выступает точка, а в консоли – запятая.

2.1. Выражения и операции

Выражения строятся из операндов - констант, переменных, функций, - объединенных знаками операций и скобками. При вычислении выражения определяется его значение и тип. Большинство операций в языке C#, их приоритет и порядок наследованы из языка C++.

Приведем таблицу приоритетов операций, в каждой строке которой собраны операции одного приоритета, а строки следуют в порядке приоритетов, от высшего к низшему.

Таблица 2.1. Приоритеты операций языка C#

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

Рассмотрим подробнее операции из таблицы 2.1, отнесенные к высшему приоритету и выполняемые в первую очередь.

2.2. Выражения в скобках

Любое выражение, взятое в скобки, получает высший приоритет и должно быть вычислено, прежде чем к нему будут применимы какие-либо операции. Скобки позволяют изменить стандартный порядок вычисления выражения и установить порядок, необходимый для вычисления в данном конкретном случае. В сложных выражениях скобки полезно расставлять даже в том случае, если стандартный порядок совпадает с требуемым, поскольку наличие "лишних" скобок зачастую увеличивает наглядность записи выражения.

Вот классический пример выражения со скобками:

```
result = (x1 + x2) * (x1 - x2);
```

Понятно, что если убрать скобки, то первой выполняемой операцией будет операция умножения и результат вычислений будет совсем другим.

Поскольку согласно стандартному порядку выполняются вначале арифметические операции, потом операции отношения, а затем логические операции, то можно было бы не ставить скобки в следующем выражении:

```
bool temp = x1 + x2 > x1 - x2 && x1 - 2 < x2 + 1;
```

Однако для наглядности иногда скобки лучше расставлять:

```
bool temp = ((x1 + x2) > (x1 - x2)) && ((x1 - 2) < (x2 + 1));
```

2.3. Инкремент и декремент

Операции увеличения и уменьшения на 1 (++ и --). Имеют две формы записи – префиксную и постфиксную. В префиксной форме сначала изменятся операнд, а затем его значение становится результирующим значением выражения, а в постфиксной форме значением выражения является исходное значения операнда, после чего он изменяется.

```
    static void Main()
    {
        int x = 3, y = 3;
        Console.WriteLine("Значение префиксного выражения: {0} \n", ++x);
        Console.WriteLine("Значение постфиксного выражения: {0} \n", y++);
        Console.WriteLine("Значение x после приращения: {0} \n", x);
        Console.WriteLine("Значение y после приращения: {0} \n", y);
    }
```

Результат работы программы:

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

В справочной системе утверждается, что к высшему приоритету относятся постфиксные операции **x++** и **x--**, это нашло отражение в таблице 2.1. Префиксные операции имеют на единицу меньший приоритет.

Следует также заметить, что запись в C# выражения " --x++ " приведет к ошибке.

Важнее помнить, что хороший стиль программирования рекомендует использовать эти операции только в выражениях, не содержащих других операндов. Еще лучше вообще не использовать их в выражениях, а применять их только как операторы:

```
x++;   y--;
```

В этом случае фактически исчезает побочный эффект изменения значения, являющийся опасным средством, и операции используются как краткая запись операторов:

```
x = x + 1;    y = y - 1;
```

2.4. Методы

Для выполнения определенных блоков кода используются методы. Методы бывают двух типов:

- функции, т.е. методы, возвращающие значения;
- процедуры – методы, не возвращающие значения.

Например, для вывода на консоль используется метод WrileLine() класса Console. Для передачи данных в метод используются параметры метода.

Для реализации вызова методов в программе C# без создания экземпляра класса используются т.н. «статические» методы с использованием ключевого слова ***static***. На данном этапе изучения языка будем использовать только статические методы. С помощью ключевого слова ***static*** обозначаются также «глобальные» переменные (в кавычках, т.к. данная переменная является глобальной только в пределах текущего класса – в языке C# глобальные переменные отсутствуют по определению).

Рассмотрим простой пример программы, вычисляющей квадрат числа, с использованием функции Square() и выводящей значение на консоль.

```
class Program
{
        /// <summary>
        /// Статический метод возведения числа в квадрат
        /// </summary>
        /// <param name="x">параметр метода</param>
        /// <returns>возвращает целое значение, равное квадрату
параметра</returns>
        static int Square(int x)
        {
            int res = x * x;
            return res;//возврат значения
        }

        static void Main()
        {
            int a=3;
            int a2 = Square(a);//вызов метода, результат выполнения
записывается в переменную a2
            Console.WriteLine(a2);
        }
}
```

Значение, возвращаемое методом, записывается после ключевого слова ***return***.

Обратите внимание, что в отдельные методы желательно выносить только вычислительную логику программы (формулы циклы, обработку данных). Вывод на консоль реализовывать в методе main.

Процедуры описываются с помощью ключевого слова ***void***. Т.е. методы типа ***void*** значений не возвращают. Использование статических переменных является признаком плохого тона, поэтому лучше реализовывать передачу значений через методы.

2.5. Арифметические операции

В языке C# имеются обычные для всех языков арифметические операции - "+, -, *, /, %". Все они перегружены. Операции "+" и "-" могут быть унарными и бинарными.

Операция деления "/" над целыми типами осуществляет деление нацело, а для типов с плавающей точкой - обычное деление. Операция "%" возвращает остаток от деления нацело и определена не только над целыми типами, но и над типами с плавающей точкой. Тип результата зависит от типов операндов. Приведем пример вычислений с различными арифметическими типами:

```
        /// <summary>
        /// Арифметические операции
        /// </summary>
        static void Ariphmetic()
        {
            byte b1 = 7, b2 = 3, b3;
            b3 = (byte)(b1 / b2);
            int n = -7, m = 3, p, q, r;
            p = n / m; q = n % m; r = p * m + q;
            Console.WriteLine("Операции над типом int");
            Console.WriteLine(
                "n = {0}, m = {1}, p = n/m = {2}, " +
                "q = n % m = {3}, r = p*m + q = {4}",
                n, m, p, q, r);

            Console.WriteLine("Операции над типом double");
            double x = 7.5, y = 3.5, u, v, w;
            u = x / y; v = u * y;
            w = x % y;
            Console.WriteLine(
                "x = {0}, y = {1}, u = x/y = {2}, " +
                "v = u*y = {3}, w = x % y = {4}",
                x, y, u, v, w);

            Console.WriteLine("Операции над типом decimal");
            decimal d1 = 7.5M, d2 = 3.5M, d3, d4, d5;
            d3 = d1 / d2; d4 = d3 * d2;
            d5 = d1 % d2;
            Console.WriteLine(
                "d1 = {0}, d2 = {1}, d3 = d1/d2 = {2}, " +
                "d4 = d3*d2 = {3}, d5 = d1 % d2 = {4}",
                d1, d2, d3, d4, d5);
        }//Ariphmetica
```

Результаты вычислений показаны на рис. 2.1.

Рис. 2.1. Результаты работы вызова метода Ariphmetic

2.6. Именованные константы

Введение констант уменьшает время вычислений, поскольку константы, заданные выражениями, вычисляются еще на этапе компиляции.

Рассмотрим в качестве примера вычисление значений переменных ***x*** и ***y***, заданных следующими выражениями:

$$x = \frac{(a + 53.5 * 33/37^2) * (a - 53.5 * 33/37^2)}{\sqrt[3]{(133 + 53.5 * 33/37^2)}}$$

$$y = \frac{(a + 53.5 * 33/37^2)}{(a - 53.5 * 33/37^2)}$$

Вычислять эти выражения, точно следуя приведенной записи, не следует. Вот как можно организовать эти вычисления:

```
static void EvalXY(double a, out double x, out double y)
   {
     const double C1 = 53.5 * 33 / (37 * 37);
     const double C2 = 133 + C1, C3 = 1.0 / 3;
     double t1 = a + C1, t2 = a - C1;
     x = t1 * t2 / Math.Pow(C2, C3);
     y = t1 / t2;
   }
```

Заметьте, константы будут вычислены еще на этапе компиляции, так что для вычисления выражений потребуется 5 арифметических операций и один вызов стандартной функции. Выигрыш кажется незначительным при тех скоростях, которыми обладают компьютеры. Но стоит учесть, что метод EvalXY() может вызываться многократно. И главное - даже не выигрыш во времени вычислений. Более важно, что запись выражения становится простой и позволяет легко обнаруживать ее ошибки.

Многие из студентов совершают типичную ошибку, записывая, например, выражение для вычисления x следующим образом:

```
x = t1 * t2 / Math.Pow(133 + C1, 1 / 3);
```

Ошибка связана с вычислением второго аргумента функции возведения в степень Pow. Здесь применяется операция деления, операнды которой - целые числа, потому результат деления нацело будет равен нулю. Обнаружить ошибку студенты могут далеко не сразу. В процедуре EvalXY ошибка становится видна мгновенно, стоит только взглянуть на значения констант, вычисленных еще на этапе компиляции.

2.7. Операции отношения

Операции отношения стоит просто перечислить, в объяснениях они не нуждаются. Всего операций 6 (==, !=, <, >, <=, >=), все они возвращают результат логического типа bool. Операции перегружены, так что их операнды могут быть разных типов.

Понятно, что перед вычислением отношения может потребоваться преобразование типа одного из операндов.

Следует обратить внимание на запись отношения эквивалентности, задаваемое двумя знаками равенства. Типичной ошибкой является привычная для математики запись:

```
if(a = b) //неправильно!
```

Выражение в скобках синтаксически корректно и воспринимается, как запись операции присваивания, допустимой в выражениях. К счастью, в большинстве случаев возникнет ошибка на этапе компиляции при попытке преобразования значения операнда b к типу bool. Но, если a и b - переменные логического типа, то никаких сообщений об ошибке выдаваться не будет, хотя результат выполнения может быть неправильным.

2.8. Логические операции

Логические операции в языке C# делятся на две категории: одни выполняются только над операндами типа bool, другие - как над булевскими, так и над целочисленными операндами.

Операций, которые выполняются только над операндами булевского типа, три (!, &&, ||). Высший приоритет среди этих операций имеет унарная операция отрицания ! x, которая возвращает в качестве результата значение, противоположное значению выражения x. Поскольку неявных преобразований типа к типу bool не существует, то выражение x задается либо переменной булевского типа, либо, как чаще бывает, выражением отношения. Возможна ситуация, когда некоторое выражение явным образом преобразуется к булевскому типу.

Следующая по приоритету бинарная операция (x && y) называется конъюнкцией, операцией "И" или логическим умножением. Она возвращает значение true в том и только в том случае, когда оба операнда имеют значение true. В остальных случаях возвращается значение false.

Следующая по приоритету бинарная операция (x || y) называется дизъюнкцией, операцией "ИЛИ" или логическим сложением. Она возвращает значение false в том и только в том случае, когда оба операнда имеют значение false. В остальных случаях возвращается значение true.

2.9. Класс Math и его функции

Кроме переменных и констант, первичным материалом для построения выражений являются функции. Большинство их в проекте будут созданы самим программистом, но не обойтись и без *встроенных функций*. Умение работать в среде Visual Studio .Net предполагает знание встроенных возможностей этой среды, знание возможностей каркаса Framework .Net, пространств имен, доступных при программировании на языке C#, а также соответствующих *встроенных классов* и функций этих классов. Продолжим знакомство с возможностями, предоставляемыми пространством имен System. В первой лабораторной мы уже познакомились с классом Convert этого пространства и частично с классом Console. Давайте рассмотрим еще один класс – *класс Math*, содержащий стандартные математические функции, без которых трудно обойтись при построении

многих выражений. Этот класс содержит два статических поля, задающих константы E и PI, а также 23 статических метода. Методы задают:

- тригонометрические функции - Sin, Cos, Tan;
- обратные тригонометрические функции - ASin, ACos, ATan, ATan2 (sinx, cosx);
- гиперболические функции - Tanh, Sinh, Cosh;
- экспоненту и логарифмические функции - Exp, Log, Log10;
- модуль, корень, знак - Abs, Sqrt, Sign;
- функции округления - Ceiling, Floor, Round;
- минимум, максимум, степень, остаток - Min, Max, Pow, IEEERemainder.

Описание методов и полей класса приведено в таблице 2.2.

Таблица 2.2. Основные поля и статические методы класса 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`	`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)`

Далее приведен пример применения методов класса Math.

```
using System;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            Console.Write("Введите x: ");
            double x = double.Parse(Console.ReadLine());
            Console.Write("Введите y: ");
            double y = double.Parse(Console.ReadLine());
            Console.WriteLine("Максимум из x и y : " + Math.Max(x, y));
            double z = Math.Pow(Math.Sin(x), 2) + Math.Pow(Math.Sin(y), 2);
            Console.WriteLine("Сумма квадратов синусов x и y : " + z);
        }
    }
}
```

Источники:

1. Биллиг В. Основы программирования на C#. Режим доступа: http://www.intuit.ru/studies/courses/2247/18/info
2. Павловская Т.А. Программирование на C#. Учебный курс. Режим доступа: http://ips.ifmo.ru/courses/csharp/index.html