# Лабораторная работа №1. Создание простого консольного приложения, реализующего заданный алгоритм.

## Типы данных в C#

Для описания простых типов данных в C# приведем сводную таблицу.

### Целые типы

|  |  |  |
| --- | --- | --- |
| Название типа | Диапазон значений | Размер |
| sbyte | -128..127 | Знаковое 8-битное целое |
| byte | 0..255 | Беззнаковое 8-битное целое |
| char | U+0000..U+FFFF | 16-битный Unicode-символ |
| short | -32.768..32.767 | Знаковое 16-битное целое |
| ushort | 0.65535 | Беззнаковое 16-битное целое |
| int | -2.147.483.648..2.147.483.647 | Знаковое 32-битное целое |
| uint | 0..4.294.967.295 | Беззнаковое 32-битное целое |
| long | -9,223,372,036,854,775,808..9,223,372,036,854,775,807 | Знаковое 64-битное целое |
| ulong | 0..18,446,744,073,709,551,615 | Беззнаковое 64-битное целое |

### Типы с плавающей точкой

|  |  |  |
| --- | --- | --- |
| Название типа | Примерный Диапазон значений | Точность |
| float | +1.5E-45..+3.4E38 | 7 знаков |
| double | +5.0E-324..+1.7E308 | 15-16 знаков |

### Тип decimal

Decimal описывает 128-битный числовой тип. Он может быть как целым, так и вещественным с большой точностью, что делает его популярным для хранения денежных сумм.

|  |  |  |
| --- | --- | --- |
| Название типа | Примерный Диапазон значений | Точность |
| decimal | 1.0E-28 to 7.9E28 | 28-39 значащих цифр |

## Переменные

Данные в C# хранятся в переменных соответствующего типа. C# является языком со строгой типизацией - значит, компилятор гарантирует, что переменная всегда имеет значение своего типа.

Общий вид объявления переменной:

type identifier1 = value1, identifier2 = value2;

Здесь type – один из типов данных C#, identifier – имя переменной. Можно инициализировать переменную, определяя знак равенства (=) и value (в виде литерала подходящего типа).

Пример объявления переменной:

int a, b, c; // три переменных типа int

int a = 10; // переменная a типа int, a //инициализируется

## Арифметические операторы

В С# определены следующие арифметические операторы.

|  |  |
| --- | --- |
| Оператор | Действие |
| + | Сложение |
| - | Вычитание |
| \* | Умножение |
| / | Деление |
| % | Деление по модулю |
| ++ | Инкремент |
| -- | декремент |

Операторы инкремента (++) и декремента (—) увеличивают и уменьшают значение операнда на единицу, соответственно. Как будет показано ниже, эти операторы обладают специальными свойствами, которые делают их весьма интересными для рассмотрения.

Итак, оператор инкремента выполняет сложение операнда с числом 1, а оператор декремента вычитает 1 из своего операнда. Это значит, что инструкция

**х = х + 1;**

аналогична такой инструкции:

**х++;**

Точно так же инструкция

**х = х - 1;**

аналогична такой инструкции:

**х--;**

Операторы инкремента и декремента могут стоять как перед своим операндом, так и после него. Например, инструкцию

**х = х + 1;**

можно переписать в виде префиксной формы:

**++х;**

или в виде постфиксной формы:

**х++;**

Если оператор инкремента или декремента используется как часть большего выражения, то форма его применения имеет важное значение. Если такой оператор применен в префиксной форме, то С# сначала выполнит эту операцию, чтобы операнд получил новое значение, которое затем будет использовано остальной частью выражения. Если же оператор применен в постфиксной форме, то С# использует в выражении его старое значение, а затем выполнит операцию, в результате которой операнд обретет новое значение. Рассмотрим следующий фрагмент кода:

**х = 10;**

**У = ++х;**

В этом случае переменная у будет установлена равной 11. Но если в этом коде префиксную форму записи заменить постфиксной, переменная у будет установлена равной 10:

**х = 10;**

**у = х++;**

В обоих случаях переменная х получит значение 11. Разница состоит лишь в том, в какой момент она станет равной 11 (до присвоения ее значения переменной у или после).

## Операторы отношений и логические операторы

Операторы отношений оценивают по "двухбалльной системе" (ИСТИНА/ЛОЖЬ) отношения между двумя значениями, а логические определяют различные способы сочетания истинных и ложных значений. Поскольку операторы отношений генерируют ИСТИНА/ЛОЖЬ-результаты, то они часто выполняются с логическими операторами.

|  |  |
| --- | --- |
| Оператор | Значение |
| == | Равно |
| != | Не равно |
| > | Больше |
| < | Меньше |
| >= | Больше или равно |
| <= | Меньше или равно |

## Управляющие операторы условного и безусловного переходов.

### Оператор условного перехода if и его конструкция

Блок if..else, как очевидствует из названия, позволяет выполнять код при выполнении определенного условия.

if (a > b)

{

Console.WriteLine("a > b");

}

else

{

Console.WriteLine("a <= b");

}

Логическое выражение в скобках должно иметь тип bool. Если оно истинно, выполнится первый блок команд, если ложно - второй. При этом если какой-либо блок команд состоит лишь из одной команды, обрамляющие фигурные скобки можно опустить:

if (a < b)

Console.WriteLine("a < b");

### Оператор switch

Оператор switch является расширенным оператором ветвления, который позволяет в зависимости от значения выражения перейти к выполнению определенного кода. По сути он эквивалентен набору блоков if, но гораздо более оптимален.

Наиболее распространенным синтаксисом switch является следующий:

int a = 1;

switch (a)

{

case 0:

//Операторы, выполняющиеся если a = 0

Console.WriteLine("A = 0");

break;

case 1:

//Операторы, выполняющиеся если a = 1

Console.WriteLine("A = 1");

break;

default:

//Операторы, выполняющиеся, если ни один из явных случаев не подошел

Console.WriteLine("A <> 0 и A <> 1");

break;

}

Блок case x выполняется, если выражение, указанное в скобках после switch равно x. Блок default - если ни одно из case-выражений не выполнилось. Блок default – необязателен. Удобно switch использовать для обработки выражений перечислимых типов.

Оператор break используется внутри switch, чтобы закончить последовательность операторов. Когда встречается оператор break , выполнение передается к первой строке кода, которая следует за полным оператором switch.

Для переходов между блоками можно использовать команды goto case и goto default.

### Цикл for

Цикл for представляет цикл с инициализирующей командой, условием окончания и командой перехода.

for (int i = 0; i < 10; i++)

{

Console.WriteLine("i=", i);

}

Логика выполнения цикла следующая: сначала выполняется инициализирующая команда, далее идет шаг цикла: проверяется условие окончания, если оно ложно, то цикл прекращается и управление переходит на следующую за циклом команду, иначе выполняется тело цикла, а после него - команда перехода.

### Цикл while

Цикл while является циклом с предусловием: сначала проверяется определенное условие перехода, а затем выполняется итерация цикла. Синтаксис цикла while в общем случае следующий:

// Задаем число

int num = 678;

// Число разрядов - 0

int len = 0;

// считаем сколько разрядов в числе

while (num > 0)

{

len++;

num /= 10;

}

Console.WriteLine(len);

Семантика очень проста: проверяем условие перехода. Если оно истинно, то переходим к телу цикла и затем снова к проверке условия, иначе - прекращаем выполнение цикла.

### Цикл do

Оператор do реализует цикл с постусловием. Суть этого цикла аналогична while, но условие выхода проверяется не до итерации, а после. Это иногда бывает удобным.

int n = 10;

do

{

Console.WriteLine(n);

} while (--n > 0);

### Оператор безусловного перехода goto

goto - это просто оператор безусловного перехода. Он применяется в двух ситуациях: когда нужно перейти на обработку определенного случая в блоке switch (он будет рассмотрен позднее) или выполнить переход в определенное место программы. Во втором случае нужно указать метку в этом месте. Метка указывается просто - в нужном месте программы пишется имя метки с двоеточием на конце:

Exit: return 0;

Чтобы перейти на эту метку, используется следующий синтаксис оператора goto:

goto Exit;

Еще goto применяется если нужно перейти к определенной ветви в блоке switch:

int n = 2;

switch (level)

{

case 0:

Console.WriteLine("n < 1");

break;

case 1:

goto case 2;

case 2:

Console.WriteLine("n > 1");

goto default;

default:

Console.WriteLine("Вне диапазона");

break;

}

## Математические функции

C# содержит большое количество встроенных математических функций, которые реализованы в классе Math пространства имен System.

Некторые математические функции C#:

|  |  |
| --- | --- |
| Abs(x) | Вычисляет модуль (абсолютное значение) числа *x*. Перегружен для всех числовых типов (int, double и т.д.) |
| Acos(x) | Функция арккосинуса. Значение аргумента должно находиться в диапазоне от -1 до +1 |
| Asin(x) | Функция арксинуса. Значение аргумента должно находиться в диапазоне от -1 до +1 |
| Atan(x) | Функция арктангенса |
| Cos(x) | Функция косинуса. Аргумент задается в радианах |
| Exp(x) | Вычисляет значение (экспоненциальная функция) |
| Log(x) | Возвращает значение натурального логарифма (ln *x*) |
| Log10(x) | Возвращает значение десятичного логарифма () |
| Max(a, b) | Возвращает максимум из двух чисел *a* и *b* |
| Min(a, b) | Возвращает минимум из двух чисел *a* и *b* |
| Pow(x, a) | Возвращает значение , то есть возводит число *x* в степень *a* |
| Sin(x) | Функция синуса. Угол задается в радианах |
| Sqrt(x) | Возвращает положительное значение квадратного корня |
| Tan(x) | Функция тангенса. Угол задается в радианах |

Пример использования математической функции:

double y = Math.Cos(3.14\*2.5);

## Ввод данных

Для ввода данных обычно используется метод ReadLine, реализованный в классе Console. Особенностью данного метода является то, что в качестве результата он возвращает строку (String). Для того, чтобы получить числовое значение необходимо воспользоваться преобразованием данных:

string s = Console.ReadLine();

int x = int.Parse(s);

//Или сокращенный вариант

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

Таким образом, если нам потребуется преобразовать строковое представление в вещественное, мы можем воспользоваться методом float.Parse() или double.Parse().

## Задания.

Задание 1.

Найдите количество единиц в двоичной записи заданного числа.

Задание 2.

Требуется написать программу, определяющую, является ли четырехзначное натуральное число N палиндромом, т.е. числом, которое одинаково читается слева направо и справа налево. Число задается с типом данных int;

Задание 3.

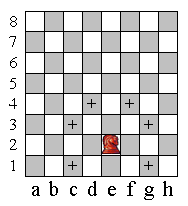
Счастливым билетом называют такой билет с шестизначным номером, где сумма первых трех цифр равна сумме последних трех. Т.е. билет с номером 385916 – счастливый, т.к. 3+8+5=9+1+6. Необходимо вывести номера всех счастливых билетиков

Задание 4\*.

Требуется сложить два целых числа А и В. Вводимые числа не превышают 10100. Примечание. Необходимо хранить все цифры в числе. Подумайте, каким образом это сделать, поскольку такие большие целые числа невозможно хранить с использованием стандартных типов данных.

Задание 5.

*«Шахматный конь»*

Конь в шахматах всегда перемещается либо на две клетки по горизонтали и на одну по вертикали, либо на одну по горизонтали и на две по вертикали. Вертикали обозначаются маленькими латинскими буквами от a до h, а горизонтали - цифрами от 1 до 8. Любая клетка на шахматной доске обозначается буквой соответствующей вертикали и цифрой соответствующей горизонтали, например, c6 или e2.

Требуется по введенным координатам поля, где стоит конь, определить все координаты клеток, на которые за один ход он может попасть .