# Лабораторная работа №1

# Выполнение простой программы. Типы данных. Организация ввода и вывода данных

## 1. Цель задания:

1. Определение типов переменных.
2. Организация ввода и вывода данных.
3. Вычисление выражений простых типов с использованием стандартной библиотеки Math для вычисления математических функций.
4. Организация тестирования программы с проверкой достаточности тестов.

## 2. Теоретические сведения

### 2.1. Консольные приложения

Среда Visual Studio.NET работает на платформе Windows и ориентирована на создание Windows- и веб-приложений, однако разработчики предусмотрели работу с консольными приложениями. При запуске консольного приложения операционная система создает так называемое консольное окно, через которое идет весь ввод-вывод программы. Внешне это напоминает работу в операционной системе в режиме командной строки, когда ввод-вывод представляет собой поток символов.

Консольные приложения наилучшим образом подходят для изучения языка, так как в них не используется множество стандартных объектов, необходимых для создания графического интерфейса.

### 2.2.Структура программы на С#

С# — объектно-ориентированный язык, поэтому написанная на нем программа представляет собой совокупность взаимодействующих между собой классов. Описание класса начинается с ключевого слова class, за которым следуют его имя и далее в фигурных скобках — список элементов класса (его данных и функций, называемых также методами).

Каждое приложение должно содержать метод Main — с него начинается выполнение программы. Все методы описываются по единым правилам.

Упрощенный синтаксис метода:

[ спецификаторы ] тип имя\_метода ( [ параметры ] )

{

тело метода: действия, выполняемые методом

}

Директива using System разрешает использовать имена стандартных классов из пространства имен System непосредственно (без указания имени пространства). Ключевое слово namespace создает для проекта собственное пространство имен.

//подключаем пространство имен System

using System;

//пространство имен

namespace App

{

//класс

class Program

{

//метод

static void Main(string[] args)

{

// текст программы

}

}

}

### 2.3. Элементы языка C#

1. Алфавит языка который включает

* 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).

1. Из символов формируются лексемы языка:

* *Идентификаторы* – имена объектов. В идентификаторе могут быть использованы латинские буквы, цифры и знак подчеркивания. 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. В идентификаторах С# разрешается использовать помимо латинских букв буквы национальных алфавитов, например, Массив – разрешенный идентификатор.
* *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 @, например, @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.
* *Константы* – это неизменяемые величины. Существуют логические, целые, вещественные, символьные и строковые константы. 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.

### 2.4. Константы в C#

Константа – это лексема, представляющая изображение фиксированного числового, строкового или символьного значения. Константы делятся на 5 групп:

* логические;
* целые;
* вещественные (с плавающей точкой);
* символьные;
* строковые.

Компилятор выделяет лексему и относит ее к той или другой группе, а затем внутри группы к определенному типу по ее форме записи в тексте программы и по числовому значению.

*Целые константы* могут быть десятичными и шестнадцатеричными.

Таблица 1. Целые константы

|  |  |  |
| --- | --- | --- |
| Название | Определение | Примеры |
| Логическая константа | Истина (true) или ложь (false) | true  false |
| 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 константы* могут иметь две формы представления: с фиксированной точкой и с плавающей точкой.

Таблица 2. Вещественные константы

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

*Символьная константа* – представляет собой любой символ в кодировке Unicode. Символьные константы записываются в одной из четырех форм:

* «обычный» символ, имеющий графическое представление (кроме апострофа и символа перевода строки), — 'А', '5', '\*', 'ю' ;
* управляющая последовательность — '\0', '\n';
* символ в виде шестнадцатеричного кода — '\xF', '\х74';
* символ в виде escape-последовательности Unicode — '\uA81B'.

Последовательности, начинающиеся со знака \, называются управляющими, они используются:

* для представления символов, не имеющих графического отображения, например:

\a – звуковой сигнал,

\b – возврат на один шаг,

\n – перевод строки,

\t – горизонтальная табуляция;

\0 – нуль-символ.

* для представления символов: \ , ’ , ? , ” ( \\, \’ ,\? ,\” );
* для представления символов с помощью шестнадцатеричных кодов (\х73, \0хF5).

Escape-последовательности Unicode служат для представления символа в кодировке Unicode с помощью его кода в шестнадцатеричном виде с префиксом \u или \U.

*Строковая константа* – это последовательность символов, заключенная в кавычки. Внутри строк также могут использоваться управляющие символы. Например:

”\nНовая строка”,

”\nНовый курс\”Алгоритмические языки программирования\””.

В С# введен второй вид строковых констант – дословные литералы (verbatim strings). Эти литералы предваряются символом @, который отключает обработку управляющих последовательностей и позволяет получать строки в том виде, в котором они записаны.

@”Новый курс “Алгоритмические языки программирования””

### 2.5. Типы данных в C#

Тип данных однозначно определяет:

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

Типы данных можно *классифицировать* по различным признакам.

* По структуре типы C# можно разделить на простые и структурированные. К простым типам относят типы, которые характеризуются одним значением.
* По тому, кем создан тип данных, выделяют встроенные и пользовательские типы данных. Встроенные типы данных уже существуют в языке, пользовательские типы создаются программистом (классы).
* По тому, в какой момент выделяется память под переменные данного типа, выделяют статические и динамические данные. Под статические переменные память выделяется в процессе компиляции программы, под динамические данные память выделяется во время выполнения программы с помощью специальных операций (new).
* По способу хранения выделяют типы-значения и типы-ссылки. Переменные типов-значений представляют собой последовательность байтов в памяти, необходимый объем памяти выделяет компилятор. Переменная ссылочного типа хранит адрес, по которому расположены данные, сами данные хранятся в динамической памяти.

Таблица 3. Встроенные типы данных С#

|  |  |  |  |
| --- | --- | --- | --- |
| Тип данных | Определение | Размер | Диапазон |
| bool | Логический тип данных, данные этого типа могут принимать значения true и false. | 1 байт | true, false |
| sbyte | Знаковый целый тип | 1 байт | -128 . . 127 |
| byte | Беззнаковый целый тип | 1 байт | 0 . . 255 |
| short | Знаковый целый тип | 2 байта | -32768 . . 32767 |
| ushort | Беззнаковый целый тип | 2 байта | 0 . . 65535 |
| int | Знаковый целый тип | 4 байта | -2\*109 . . 2\*109 |
| uint | Беззнаковый целый тип | 4 байта | 0 . . 4\*109 |
| long | Знаковый целый тип | 8 байт | -9\*1018 . . 9\*1018 |
| ulong | Беззнаковый целый тип | 8 байт | 0 . . 18\*1018 |
| char | Символьный тип, Unicode-символ | 1 байт | U+0000 . . U+ffff |
| float | Вещественный тип | 4 байта | 1.5\*10-45. . 3.4\*1038 |
| double | Вещественный тип | 8 байт | 5.0\*10-324. .1.7\*10308 |
| decimal | Финансовый тип для денежных вычислений | 16 байт | 1.0\*10-28. . 7.9\*1028 |
| string | Строковый тип, строка Unicode символов | Длина ограничена объемом доступной памяти |  |
| object | Всеобщий предок, можно хранить, что угодно |  |  |

### 2.6. Переменные

Переменная в C# – именованная область памяти, в которой хранятся данные определенного типа. У переменной есть имя и значение. Имя служит для обращения к области памяти, в которой хранится значение. Перед использованием любая переменная должна быть описана. 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.

int a; float x;

### 2.7. Операции

В соответствии с количеством операндов, которые используются в операциях они делятся на унарные (один операнд), бинарные (два операнда) и тернарную (три операнда).

|  |  |  |
| --- | --- | --- |
| 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) целочисленных операндов или отношений, целочисленный результат ложь(0) или истина( не 0) | |
| || | дизъюнкция (ИЛИ) целочисленных операндов или отношений, целочисленный результат ложь(0) или истина(не 0) | |
| Тернарная | | |
| ?: | Условная операция  в ней используется три операнда.  Выражение1 ? Выражение2 : Выражение3;  Первым вычисляется значение выражения1. Если оно истинно, то вычисляется значение выражения2, которое становится результатом. Если при вычислении выражения1 получится 0, то в качестве результата берется значение выражения3.  Например:  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 | |

Приоритеты операций.

|  |  |
| --- | --- |
| Ранг | Операции |
| 1 | ()[ ] . |
| 2 | ! - ++ -- (тип) sizeof |
| 3 | \* / % (мультипликативные бинарные) |
| 4 | + - (аддитивные бинарные) |
| 5 | < > <= >= (отношения) |
| 6 | == != (сравнения) |
| 7 | && (конъюнкция «И») |
| 8 | || (дизъюнкция «ИЛИ») |
| 9 | ?: (условная операция) |
| 10 | = \*= /= %= -= &= ^= |= <<= >>= (операция присваивания) |

### 2.8. 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. Если выражение формирует целое или вещественное число, то оно называется арифметическим. Пара арифметических выражений, объединенная операцией сравнения, называется отношением. Если отношение имеет ненулевое значение, то оно – истинно, иначе – ложно.

### 2.9. Ввод и вывод данных

Любая программа при вводе исходных данных и выводе результатов взаимодействует с внешними устройствами. Совокупность стандартных устройств ввода и вывода, то есть клавиатуры и экрана, называется консолью. В языке С#, как и во многих других, нет операторов ввода и вывода. Вместо них для обмена с внешними устройствами применяются стандартные объекты. Для работы с консолью в С# применяется класс Console, определенный в пространстве имен System. В этом классе определены методы:

Console.Write(форматная строка, [параметры]);

Console.WriteLine(форматная строка, [параметры]);

где форматная строка – это строковая константа, в которой содержится текст для вывода, параметры – это переменные, значения которых будут при выводе подставлены в форматную строку.

Методы отличаются друг от друга тем, что метод WriteLine() выполняет переход на следующую строку после вывода форматной строки.

int x=10, y=15;

Console.WriteLine(”x={0},\ny={1}”,x,y};

int x=10, y=15;

Console.WriteLine(”x=”+x};

Console.WriteLine(”y=”+y};

int x=10, y=15;

Console.Write(”x=”+x+”\n”};

Console.Write(”y=”+y+”\n”};

При вводе данных с клавиатуры используются методы Console.Read() и Console.ReadLine().

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.

string buf;

Console.WriteLine(”Введите значение переменной”);

buf=Console.ReadLine();

int x=int.Parse(buf);//преобразует строку в целое число

### 2.10. Использование математических функций

В выражениях часто используются математические функции, например синус или возведение в степень. Они реализованы в классе Math, определенном в пространстве имен System.

Таблица 4 Основные поля и методы класса Math

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Пример вызова |
| Abs() | Абсолютное значение | int a=Math.Abs(-10); |
| Acos() | Арккосинус | double x; . . . double y=Math.Acos(x); |
| Asin() | Арксинус | double x; . . . double y=Math.Asin(x); |
| Atan() | Арктангенс | double x; . . . double y=Math.Atan(x); |
| Atan2() | Арктангенс - угол, тангенс которого это результат деления второго аргумента на первый | double x,y; . . . double z=Math.Atan(x,y); |
| BigMul() | Произведение | long x,y;. . . .long z=Math.BigMul(x,y); |
| Celling() | Округление до большего целого | double a=Math.Cellin01  g(3.4567); |
| Cos() | Косинус | double x; . . . double y=Math.Cos(x); |
| Cosh() | Гиперболический косинус | double x; . . . double y=Math.Cosh(x); |
| DivRem() | Деление и остаток | int b, c; c = Math.DivRem(15, 5, out b); |
| E | База натурального логарифма | 2.71828 |
| Exp() | Экспонента | double x; … double y=Math.Exp(x); |
| Floor() | Округление до меньшего целого | double a=Math.Floor(3.4567); |
| IEEERemainder() | Остаток от деления | double a=Math. IEEERemainder (12, 5); |
| Log() | Натуральный логарифм | double x;….. double y=Math.Log(x); |
| Log10() | Десятичный логарифм | double x;….. double y=Math.Log10(x); |
| Max() | Максимум из двух чисел | int x= Math.Max(3,6); |
| Min() | Минимум из двух чисел | int x= Math.Min(3,6); |
| Pi | Значение числа пи | 3.14159 |
| Pow() | Возведение в степень | double x=Math.Pow(2,3); |
| Round() | Округление | double a=Math.Round (3.4567); |
| Sign() | Знак числа | int x=Math.Sign(-1); |
| Sin() | Синус | double x; . . . double y=Math.Sin(x); |
| Sinh() | Гиперболический синус | double x; . . . double y=Math.Sinh(x); |
| Sqrt() | Квадратный корень | double x; . . . double y=Math.Sqrt(x); |
| Tan() | Тангенс | double x; . . . double y=Math.Tan(x); |
| Tanh() | Гиперболический тангенс | double x; . . . double y=Math.Tanh(x); |

## Постановка задачи

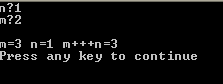
1. Найти значения выражений 1,2,3,4. Если необходимо, то определить при каких исходных данных выражение не может быть вычислено и выдать сообщение об ошибке. Составить систему тестов и проверить их полноту по критериям черного и белого ящиков.
2. Определить, принадлежит ли точка заштрихованной области. Составить систему тестов и вычислить полученное выражение.
3. Для задачи 3 вычислить значение выражения, используя различные вещественные типы данных (float и double). Результаты вычислений вывести на печать. Объяснить полученные результаты.

## Варианты

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задача 1 | Задача 2 | Задача 3 |
| 1 | 1. n++ +m 2. m-- >n 3. n-- >m |  | а=1000, b=0.0001 |
| 2 | 1. ++n\*++m 2. m++<n 3. n++>m   4) |  | а=1000, b=0.0001 |
| 3 | 1) m---n  2) m++<n  3) n++>m  4) |  | а=1000, b=0.0001 |
| 4 | 1) n++\*m  2) n++<m  3) --m>n  4) |  | а=1000, b=0.0001 |
| 5 | 1) –m - n++  2) m\*m < n++  3) n-- > ++m  4) |  | а=1000, b=0.0001 |
| 6 | 1) m-++n  2) m++>--n  3) m--<++n  4) |  | а=1000, b=0.0001 |
| 7 | 1) m + --n  2) m ++ <--n  3) –m > n--  4) |  | а=1000, b=0.0001 |
| 8 | 1) n / m++  2)m++ < --n  3) (m/n++)<n/++m  4) |  | а=1000, b=0.0001 |
| 9 | 1) m++ / n--  2) ++m < n--  3) n-- > m  4) |  | а=1000, b=0.0001 |
| 10 | 1) m++ / --n  2) m / n++ < n—  3)m + n++ > n+m  4) |  | а=1000, b=0.0001 |
| 11 | 1) n++ +m--  2) n\*m < n++  3) n-- > ++m  4) |  | а=1000, b=0.0001 |
| 12 | 1) n++ \* m  2) m-- < n  3)++m > n  4) |  | а=1000, b=0.0001 |
| 13 | 1) n++ / --m  2) ++m < n--  3)—m > ++n  4) |  | а=1000, b=0.0001 |
| 14 | 1) n++ \* --m  2) n-- < m++  3) –n > --m  4) |  | а=1000, b=0.0001 |
| 15 | 1) n++ /--m  2)n-- > n/m++  3)m < n++  4) |  | а=1000, b=0.0001 |
| 16 | 1) m++ / --n  2) m / n < n--  3)m + n++ > n+m  4) |  | а=1000, b=0.0001 |
| 17 | 1) n++ + m--  2) n\*m < n++  3) n-- > ++m  4) |  | а=1000, b=0.0001 |
| 18 | 1) n++ \* m  2) m-- < n  3)++m > n/m  4) |  | а=1000, b=0.0001 |
| 19 | 1) n++ / --m  2) ++m < n--  3)—m > ++n  4) |  | а=1000, b=0.0001 |
| 20 | 1) n++ \* --m  2) n-- < m++  3) --n/m > --m  4) |  | а=1000, b=0.0001 |
| 21 | 1) n++ / --m  2) n-- > n/m++  3) m < n++  4) |  | а=1000, b=0.0001 |
| 22 | 1) n++ \* m  2) n++ < m  3) --m > n  4) |  | а=1000, b=0.0001 |
| 23 | 1) --m - n++  2) m / --n < n++  3) n-- > ++m  4) |  | а=1000, b=0.0001 |
| 24 | 1) m - ++n  2) m++ > --n  3) m-- < ++n  4) |  | а=1000, b=0.0001 |
| 25 | 1) m + --n  2) m++ < --n  3) --m > n--  4) |  | а=1000, b=0.0001 |

## 5. Методические указания

1. Ввод данных для заданий А и Б организовать с клавиатуры.
2. Вывод результатов для задания А организовать в виде:



1. Для проверки возможности вычислений использовать условный оператор if:

if(n==1) Console.WriteLine (“Нельзя вычислить”);

else {

k=m/--n;

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

}

1. При выполнении задачи 2 использовать переменную логического типа, а не условный оператор.
2. При выполнении задачи 3 использовать вспомогательные переменные для хранения промежуточных значений.
3. При работе с данными типа float использовать операцию приведения типа:

float c = (float)Math.Pow(a + b, 2);

1. Для типа float результат должен получиться отличным от 1.

## 6. Содержание отчета

Для каждой задачи привести:

1. Постановка задачи (общая и конкретного варианта).
2. Анализ классов входных и выходных данных
3. Алгоритм (блок-схема, нарисованная в Word или draw.io)
4. Код программы (листинг)
5. Тесты (в Excel)
6. Анализ достаточности тестов по критериям черного и белого ящиков (в Excel)
7. Объяснение результатов работы программы

## 7. Вопросы для защиты лабораторной работы

1. Какую структуру имеет программа на С#?

2. Что определяет тип данных?

3. Чем статические данные отличаются от динамических?

4. Чем типы-значения отличаются от типов-ссылок?

5. Какие типы данных существую в С#?

6. На что влияет размер памяти, выделяемой на переменную?

7. Какие операции можно применять к целым числам?

8. Чем отличаются префиксные и постфиксные операции ++ (--)?

9. Как выполняется тестирование по критериям черного ящика?

10. Как выполняется тестирование по критериям белого ящика?

## 8. Критерии оценивания работы

|  |  |  |
| --- | --- | --- |
| № | Критерий | Баллы |
| 1 | Решена 1-ая задача, объяснены полученные результаты | 1 |
| 2. | Решена 2-ая задача, объяснены полученные результаты | 1 |
| 3 | Решена 3-я задача, объяснены полученные результаты | 1 |
| 4 | Написаны тесты к задачам 1,2 | 2 |
| 5 | Выполнена проверка достаточности тестов по критериям черного и белого ящиков | 2 |
| 6 | Даны полные ответы на теоретические вопросы (п.5) | 1 |
| 7 | Дополнительные баллы: Написанный код соответствует стиль-гайду | 1 |