Основные понятия языка C#. Переменные и выражения

- Состав языка. Типы данных. Переменные. Именованные выражения. Операции. Линейные программы. Рекомендации по программированию.

- Язык программирования можно уподобить очень примитивному иностранному языку с жесткими правилами без исключений. Изучение иностранного языка начинают с алфавита, затем переходят к словам и законам построения фраз, и только в результате длительной практики и накопления словарного запаса появляется возможность свободно выражать на этом языке свои мысли.

- Все тексты на языке пишутся с помощью его алфавита. В C# используется кодировка символов Unicode, задающая соответствие между символами и кодирующими числами. Кодировка Unicode позволяет представить символы всех существующих алфавитов одновременно, что улучшает переносимость текстов. Каждому символу соответствует свой уникальный код. Первые 128 Unicode-символов соответствуют первой части кодовой таблицы ANSI (кодировки Windows)

Алфавит C# включает:

Буквы (латинских и национальных алфавитов) и символ подчеркивания, который употребляется наряду с буквами;

Цифры;

Специальные символы, например, +,*,{, &;

Символы перевода строки.

Из символов составляются: лексемы, директивы препроцессора и комментарии

- Лексемы – это минимальные единицы языка, имеющие самостоятельный смысл. Существуют следующие виды лексем: имена (идентификаторы), ключевые слова, знаки операций, разделители, литералы (константы).

- Директивы препроцессора используются на предварительной стадии компиляции при формировании окончательного вида программы.
- Комментарии предназначены для записи пояснений к программе и формаирования текста документа.

- Из лексем составляются выражения и операторы. Выражение задает правило вычисления некоторого значения, например, a+b.
- Оператор задает законченное описание некоторого действия, описание данных или описание элемента программы, например, int a (оператор описания целочисленной переменной).

Идентификаторы

- Имена в программах служат для той же цели, что и имена людей – чтобы обращаться к программным объектам и различать их, то есть идентифицировать. Поэтому имена называются также идентификаторами. В идентификаторе могут использоваться буквы, цифры и символ подчеркивания. Прописные и строчные буквы различаются.

- Первым символом идентификатора может быть буква или знак подчеркивания, но не цифра. Длина идентификатора не ограничена. Пробелы внутри имен не допускаются.
  В идентификаторах C# допускается использовать помимо латинских букв буквы латинских алфавитов. Можно применять также escape-последовательности Unicode с префиксом \u, например, \u00F2.

При выборе идентификатора нужно иметь в виду следующее:
идентификатор не должен совпадать с ключевыми словами;
не рекомендуется начинать идентификаторы с двух символов подчеркивания, поскольку такие имена зарезервированы для служебного использования.

Ключевые слова

- Ключевые слова – это зарезервированные идентификаторы, которые имеют специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. Список ключевых слов начинается с abstract, as и заканчивается void, while.

Литералы или константы

- Литералами или константами называют неизменяемые величины. В C# есть логические, целые, вещественные константы, а также константа null (ссылка, которая не указывает ни на какой объект). Компилятор, выделив константу в качестве лексемы, относит ее к одному из типов данных по ее внешнему виду. Программист может задать тип константы самостоятельно.

- Когда компилятор распознает константу, он отводит ей место в памяти в соответствии с ее видом и значением. Если по какой-то причине требуется явным образом задать, сколько памяти следует отвести под константу, используются суффиксы.

L, l	Длинное целое
U, u	Беззнаковое целое
F, f	Вещественное с одинарной точностью
D, d	Вещественное с двойной точностью
M, m	Финансовое десятичного типа

Типы данных

- Тип данных однозначно определяет:
  внутренне представление данных, а следовательно, и множество их возможных значений;
  допустимые действия над данными (операции и функции).
  Например, целые и вещественные числа, даже если они занимают одинаковый объем памяти имеют совершенно разные диапазоны возможных значений.

- Каждое выражение в программе имеет определенный тип. Величин, не имеющих никакого типа не существует. Компилятор использует информацию о типе при проверке допустимости описанных в программе действий.

- Память, в которой хранятся данные во время выполнения программы, делится на две области: стек (stack) и динамическую область или хип (heap). Стек используется для хранения величин, память под которые выделяет компилятор, а в динамической области память резервируется и освобождается во время выполнения программы. Основным местом для хранения данных в C# является хип.

Название	Ключевое слово	Тип .NET	Размер битов
Логический тип	bool	Boolean	8
Целые типы	sbyte byte short ushort int uint long ulong	Sbyte Byte Int16 Uint16 Int32 UInt32 Int64 UInt64	8 8 16 16 32 64 64 64
Символьный тип	char		16

Вещественные	float double	Single Double	32 64
Финансовый тип	decimal	Decimal	128
Строковый тип	string	String	
Тип object	object	Object	Может хранить все, что угодно

Переменные, операции и выражения

- Переменная – это именованная область памяти, предназначенная для хранения данных определенного типа. Во время выполнения программы значение переменной может быть изменено. Все переменные используемые в программе должны быть описаны явным образом.

- Имя переменной служит для обращения к области памяти, предназначенной для хранения данных. Имя дает программист.
  Тип переменной выбирается, исходя из диапазона и требуемой точности представления данных.
  Возможно инициализировать переменную рот ее объявлении, но это не обязательно.

- Иногда инициализацию выполняет компилятор. Переменная, описанная внутри класса, называется полем класса, ей автоматически присваивается значение по умолчанию (как правило 0 соответствующего типа). Переменная, описанная внутри метода класса, называется локальной переменной. Ее инициализация возлагается на программиста.

- Область действия переменной начинается в точке ее описания и длится до конца блока (кода в фигурных скобках), внутри которого она описана. Имя переменной должно быть уникальным в области ее действия. Как правило, переменным с большой областью действия дается более длинное имя.

Операции и выражения

- Выражение – это правило вычисления значения. В выражении участвуют операнды, объединенные знаками операций. Операндами простейшего выражения могут быть константы, переменные, и вызовы функций. По количеству участвующих в одной операции операндов операции делятся на унарные, бинарные и тетрарную.

- Операции в выражении выполняются в определенном порядке в соответствии с ее приоритетом. Самый высокий приоритет у операций доступа к элементу и вызова элемента или делегата, самый низкий приоритет у поразрядных сдвигов вправо и влево с присваиванием.

- Если операнды, входящие в выражение, имеют разный тип или операция для этого типа не определена, то перед вычислением производится автоматическое преобразование типа по правилу, обеспечивающему преобразованию более коротких типов к более длинным для сохранения значимости и точности.

Основные операции C#

- 1. Операции инкремента и декремента
- 2. Операция new
- 3. Операции отрицания
- 4. Явное преобразование типа
- 5. Операции умножения, деления и получения остатка
- 6. Операции сложения и вычитания
- 7. Операции сдвига

- 8. Операции отношения и проверки на равенство
- 9. Поразрядные логические операции
- 10. Условные операции
- 11. Условная операция

Операции инкремента и декремента

- Операции инкремента ++ и декремента --, называемые также операциями увеличения и уменьшения на единицу, имеют две формы записи – префиксную, когда знак операции записывается перед операндом, и постфиксную – после операнда. В префиксной форме сначала изменяется операнд, а затем его значение становится результирующим значением выражения, а в постфиксной форме значением выражения является исходное значение операнда, после чего он изменяется.

```
using System;
namespace ConsoleApplication1{
    class Class1    { static void Main()
        {  int x = 3, y =  3;
Console.Write("Значение префиксного
выражения: ");
Console.WriteLine(++x);
Console.Write("Значение x после
приращения: ");
```

```
Console.WriteLine(x);
Console.Write("Значение постфиксного выражения: ");
        Console.WriteLine(y++);
Console.Write("Значение y после приращения: ");
Console.WriteLine( y );
    }   } }
```

Результат операции

- Значение префиксного выражения: 4
- Значение x после приращения: 4
- Значение постфиксного выражения: 3
- Значение y после приращения: 4
- Для продолжения нажмите любую клавишу . . .

Операция new

- Операция new служит для создания нового объекта. Формат операции:
  new тип ([аргументы])
  С помощью этой операции можно создавать объекты как ссылочного так и значимого типов:
  object z = new object();
  int i = new int(); // то же что int i = 0;

- При выполнении операции new обычно сначала выделяется необходимый объем памяти (для ссылочных типов в хипе, для значимых – в стеке), а затем вызывается так называемый конструктор по умолчанию, то есть метод, с помощью которого инициализируется объект. Переменной значимого типа присваивается значение по умолчанию, которое равно нулю соответствующего типа.

Операции отрицания

- Арифметическое отрицание (унарный минус -) меняет знак операнда на противоположный (для типов int, long, double, float, decimal). Логическое отрицание (!) определено для типа bool. Меняет значение true на false и обратно. Поразрядное отрицание (~) часто называемое побитовым, инвертирует каждый разряд в двоичном представлении операнда типа int, uint, long, ulong.

- using System;
  namespace ConsoleApplication1{
  class Class1
  { static void Main()
  { 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
  } } }

Явное преобразование типа

- Формат операции:
  (тип) выражение
  Здесь тип – это имя того типа, в который осуществляется преобразование, например,
  long b =300;
  int a = (int) b; // данные не теряются
  byte d = (byte) a; // данные теряются

Операции умножения, деления и получения остатка

- using System;
  namespace ConsoleApplication1{
  class Class1 { static void Main() {
  int x = 11, y = 4;
  float z = 4;
  Console.WriteLine (z*y); // 16
  Console.WriteLine(z * 1e308); // Результат бесконечность

- Несколько операций одного приоритета выполняются слева направо. Например, в выражении 2/x*y, сначала 2 делится на x, а затем результат вычислений умножается на y.

Операции сложения и вычитания

- Тип результата операции равен наибольшему из типов операндов, но не менее int.

- using System;
  namespace ConsoleApplication1{
  class Class1 {
  static void Main() {
  int x = 11, y = 4;
  float z = 4;
  Console.WriteLine (z*y); // 16
  Console.WriteLine(z * 1e308); // Результат бесконечность

- Console.WriteLine(x / y); // Результат 2
  Console.WriteLine(x / z); // Результат 3.75
  Console.WriteLine(x % y); // Результат 3
  Console.WriteLine (1e-324 / 1e-324); //
  Результат NaN
  }
  }
  }

Операции сдвига

- Операции сдвига (<<, >>) 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 беззанакового типа), и знаковым разрядом в противоположном случае.

- ```
  using System;
  namespace ConsoleApplication1{
      class Class1    {
  static void Main()   {
      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
  }
  }
  }

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.

Поразрядные логические операции

- Поразрядные логические операции (&, |, ^) применяются к целочисленным операндам и работают с их двоичными представлениями. При выполнении операций операнды операнды сопоставляются побитно. При поразрядном И (&) бит результата равен 1 тогда, когда соответствующие биты обеих операндов равны 1. При поразрядном исключающем ИЛИ (^) бит результата равен 1 только тогда, когда соответствующий бит одного из операндов равен 1.

- ```
  using System;
  namespace ConsoleApplication1{
      class Class1    {
          static void Main()   {
  Console.WriteLine(6 & 5);  //Результат 4
  Console.WriteLine(6 | 5);  //Результат 7
  Console.WriteLine(6 ^ 5);  //Результат 3 }
  }
  }
  ```
  ```

Условные логические операции

- Условные логические операции И(&&) и ИЛИ(||) чаще всего используются с операндами логического типа. Результатом логической операции является true или false.

- using System;
  namespace ConsoleApplication1{
  class Class1 {
  static void Main() {
  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

- Условная операция (? :) – тернарная, то есть имеет три операнда. Ее формат:
  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 условной операции будет значение второго операнда, иначе – значение третьего операнда.

- Тип результата зависит от типа третьего и второго операндов.
  Условную операцию используют вместо условного оператора if для сокращения текста программы.

- using System;
  namespace ConsoleApplication1{
  class Class1 {
  static void Main() {
  int a = 11, b = 4;
  int max = b > a ? b : a;
  Console.WriteLine(max); //Результат
  11 } } }

- Другой пример применения условной операции: требуется, чтобы некоторая целая величина увеличивалась на 1, если ее значение не превышает n, иначе принимала значение 1. Это удобно реализовывать следующим образом: i = (i < n) ? i+1:1.

Операции присваивания

- Операции присваивания (=, +=, -=, *=, и т.д.) задают новое значение переменной.
  Формат простого присваивания (=)
  переменная = выражение
  Значение выражения записывается в память по адресу, определяемому именем переменной.
  В сложных операциях присваивания (+=, -=, *=, и т.д.) при вычислении выражения, стоящего в правой части, используется значение из левой части.

Линейные программы

- Линейной называется программа, все операторы которой выполняются последовательно в том порядке, в котором они записаны. Простейшим примером линейной программы является программа расчета по заданной формуле. Она состоит из трех этапов: ввод исходных данных, вычисление по формуле и вывод результатов.

- В языке C# как и многих других языках, нет операторов ввода и вывода. Вместо них применяются стандартные объекты для обмена с внешними устройствами. Для обмена с консолью (клавиатура и экран) в C# применяется класс Console, определенный в пространстве имен System. Методы этого класса Write, WriteLine применяются в следующей программе:

```
using System;
namespace ConsoleApplication1 {
    class Class1   {
        static void Main()   {
        int i = 3;
        double y = 4.12;
        decimal d = 660m;
       string s = "Вася";
        Console.WriteLine(" i= " + i);                  //1
```

```
• Console.WriteLine(" y= {0} \nd = {1}",y,d);  //2
  Console.WriteLine(" s= " + s);
        }   }  }
```

Результат работы программы

- i= 3
- y= 4,12
- d = 660
- s= Вася
- Для продолжения нажмите любую клавишу . . .

- В классе Console существуют несколько вариантов методов с именами Write, WriteLine, предназначенных для вывода значений различных типов (перегруженных методов).
  Если метод WriteLine (случай 1)вызван с одним параметром, он может быть любого встроенного типа, например, числом, символом или строкой. В одну строку можно объединить знаком +.

- Оператор 2 иллюстрирует форматный вывод. В этом случае используется другой вариант метода WriteLine, который содержит более одного параметра. Первым параметром методу передается строковый литерал, содержащий помимо обычных символов, предназначенных для вывода на консоль, параметры в фигурных скобках, а также управляющие последовательности

- Из управляющих последовательностей чаще всего используются символы перевода строки (\n) и горизонтальной табуляции (\t). Параметры нумеруются с нуля, перед выводом они заменяются значениями соответствующих переменных в списке вывода: нулевой параметр заменяется значением первой переменной, первый параметр – значением второй переменной и т. д.

- Рассмотрим простейшие способы ввода с клавиатуры. В классе Consol 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 числа, вводятся с клавиатуры в строковую переменную.
  2. Выполняется преобразование из строки в переменную соответствующего типа.

- Преобразование можно выполнить либо с помощью с помощью специализированного класса Convert, определенного в пространстве имен System, либо с помощью метода Parse, имеющегося в каждом стандартном арифметическом классе.

- using System;
  namespace ConsoleApplication1{
  class Class1 {
  static void Main() {
  Console.WriteLine("Введите строку
  ");
  string s = Console.ReadLine();
  Console.WriteLine("s= " + s);
  Console.WriteLine("Введите символ ");
  char c = (char)Console.Read(); //2

- Console.ReadLine(); //3
  Console.WriteLine("c = " + c);
  string buf; //строка буфер для ввода чисел Console.WriteLine("Введите целое число "); buf = Console.ReadLine();
  int i = Convert.ToInt32(buf); //4
  Console.WriteLine(i);
  Console.WriteLine("Введите вещественное число ");

- buf = Console.ReadLine();
  double x = Convert.ToDouble(buf); //5
  Console.WriteLine(x);
  Console.WriteLine("Введите вещественное число ");
  buf = Console.ReadLine();
  double y = double.Parse(buf); //6
  Console.WriteLine(y);
  Console.WriteLine("Введите вещественное число ");

- buf = Console.ReadLine();
  decimal z = decimal.Parse(buf); //7
  Console.WriteLine(z);
  } } }

- Введите строку
  qwertyuiop
  s= qwertyuiop
  Введите символ
  z
  c = z
  Введите целое число
  123
  123
  Введите вещественное число
  89,345678
  89,345678
  Введите вещественное число
  -345,1234
  -345,1234
  Введите вещественное число
  0,0012345
  0,0012345
  Для продолжения нажмите любую клавишу . . .

- К этому примеру необходимо сделать несколько пояснений. Ввод строки выполняется в операторе 1. Длина строки не ограничена, ввод выполняется до символа перевода строки.
  Ввод символа выполняется с помощью метода Read, который считывает один символ из входного потока (оператор 2). Метод возвращает значение типа int, представляющее собой код символа, или -1, если символов во входном потоке нет (например, пользователь нажал клавишу

- За оператором 2 записан оператор 3, который считывает остаток строки и никуда его не передает. Это необходимо потому, что ввод данных выполняется через буфер специальную область оперативной памяти. Фактически, данные сначала заносятся в буфер, а затем считываются оттуда процедурами ввода. Занесение в буфер выполняется по нажатию клавиши Enter вместе с ее кодом.

- Метод Read в отличие от RaedLine не очищает буфер, поэтому следующий после него ввод будет выполняться с того места, на котором закончился предыдущий.
  В операторах 4 и 5 используются методы класса Convert, в операторах 6 и 7 – меоды Parse классов Double, Decimal бибилиотеки .NET, которые используются через имена типов C# double, decimal.

- При вводе вещественных чисел дробная часть отделяется от целой с помощью запятой, а не точки. Допускается задавать числа с порядком, например, 1,95e-8.

Ввод-вывод в файлы

- Удобно заранее подготовить исходные данные в текстовом файле и считать их в программе. Кроме того, это дает возможность не торопясь продумать, какие исходные данные требуются, какие исходные данные требуется ввести для полной проверки программы. Вывод из программы также бывает полезно выполнить не на экран, а в текстовый файл для последующего неспешного анализа и распечатки.

- ```
  using System;
  using System.IO;                    //1
  namespace ConsoleApplication1 {
    class Class1   {
          static void Main()  {    try  {
  StreamWriter f = new
  StreamWriter("text.txt");          //2
  f.WriteLine("Вывод в текстовый файл: ");
  double a = 12.234;
  ```
  ```

- int b = 29;
  f.WriteLine("a={0,6:C} b={1,2:X}", a, b);
  int i = 3;
  double y = 4.12;
  decimal d = 600m;
  string s = "Вася«;
  f.WriteLine("i = " + i); //3
  f.WriteLine(" y={0} \n d = {1}", y, d); //4
  f.WriteLine(" s = " + s); //5

- f.Close(); //6
  }
  catch (Exception e)
  { Console.WriteLine("Error: " + e.Message);
  return; }
  }
  }
  }

- Для того, чтобы использовать в программе файлы, необходимо:
  1. Подключить пространство имен, в котором описываются стандартные классы для работы с файлами (оператор 1).
  2. Объявить файловую переменную и связать ее с файлом на диске (оператор 2).
  3. Выполнить операции ввода-вывода (операторы 3-5).
  4. Закрыть файл (оператор 6).

- Ввод из текстового файла выполняется аналогично. Текстовый файл можно создать с помощью любого текстового редактора, но удобнее использовать Visual Studio.NET. Для этого следует выбрать в меню команду File>New>File…. и в появившемся диалоговом окне выбрать тип файла text File.

- ```
  using System;
  using System.IO;
  namespace ConsoleApplication1{
      class Class1
      {  static void Main()    {
  StreamReader f = new
  StreamReader("input.txt");
              string s = f.ReadLine();
              Console.WriteLine ("s = " + s);
  ```
  ```

- char c = (char) f.Read();
  f.ReadLine();

```
Console.WriteLine(" c = " +c);
string buf;
buf = f.ReadLine();
int i = Convert.ToInt32( buf );
Console.WriteLine( i );
buf = f.ReadLine();
double x = Convert.ToDouble( buf );
```

```
            Console.WriteLine( x );
            buf = f.ReadLine();
            double y = double.Parse( buf );
            Console.WriteLine( y );
            buf = f.ReadLine();
  decimal z= decimal.Parse( buf );
  Console.WriteLine( z );
            f.Close();    }    }  }
```

Математические функции – класс Math

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

- С помощью методов этого класса можно вычислить:
  тригонометрические функции: Sin, Cos, Tan;
  обратные тригонометрические функции: Asin,Acos, Atan,
  гиперболические функции: Tanh, Sinh, Cosh;
  экспоненту и логарифмические функции: Exp, Log, Log10;
  модуль, квадратный корень, знак: Abs, Sqrt, Sign;
  округление до меньшего: Floor;
  округление до большего: Ceiling;
  минимум или максимум: Min, Max;

- степень, остаток: Pow, IEEEReminder;
  полное произведение двух целых величин: BigMul;
  деление и остаток от деления: DivRem.
  Кроме того, у класса есть два полезных поля: число π и число e.
  В качестве примера рассмотрим программу расчета по формуле:

$$y = \sqrt{\pi x} - e^{0,2\sqrt{\alpha}} + 2tg(2\alpha) + 1,6 \cdot 10^3 \cdot log_{10} x^2$$

```
using System;
namespace ConsoleApplication1{
    class Class1    {
        static void Main()     {
            string buf;
        Console.WriteLine("Введите x");
        buf = Console.ReadLine();
            double x = Convert.ToDouble(buf);
Console.WriteLine("Введите alfa");
```

```
buf = Console.ReadLine();
double a = double.Parse(buf);
double y = Math.Sqrt(Math.PI * x) -
Math.Exp(0.2 * Math.Sqrt(a)) + 2 *
Math.Tan(2 * a) + 1.6e3 *
Math.Log10(Math.Pow(x, 2));
Console.WriteLine("Для x={0} и alfa = {1}", x,
a);
Console.WriteLine(" Результат = " +y);   }   } }
```

Рекомендации по программированию

- Приступая к написанию программы, четко определите, что является ее исходными данными и что требуется получить в результате. Выбирайте тип переменных с учетом диапазона и требуемой точности представления данных.
  Давайте имена переменным, отражающие их назначение. Правильно выбранные имена могут сделать программу в некоторой степени самодокументированной. В именах следует избегать сокращений.

- Общая тенденция такова: чем больше область действия переменной, тем более длинное у нее имя. Перед таким именем можно поставить префикс типа (одну или несколько букв, по которым можно определить тип переменной). Напротив, для переменных, вся жизнь которых проходит на протяжении нескольких строк кода, лучше обойтись однобуквенными именами типа i, k.

- Переменные желательно инициализировать при их объявлении, а объявлять как можно ближе к месту непосредственного использования. С другой, удобно все объявления локальных переменных располагать в начале блока так, чтобы их было просто найти.
  Избегайте использования в программе чисел в явном виде. Константны должны иметь осмысленные имена, заданные с помощью ключевjго слова const.