Дисциплины:
«Основы автоматизированной обработки данных» ***(«двойка»)***,
«Технологии программирования» ***(«тройка»)***

Интегрированный экзамен (7 семестр)

Цель текущего семестра: введение в современные технологии ***разработки*** прикладного программного обеспечения с использованием объектно-ориентированного языка ***C#*** ***в среде программирования MS Visual Studio***

Состав учебных мероприятий

- **Первый полусеместр** :
  - лекции/практики *(они же лекции)* - *4 часа в неделю*;
  - **контрольная работа 1** ***(на восьмой неделе)***

    Темы КР: *Числовые константы. Скалярные объекты встроенных типов данных. Выражения, операции и преобразование типов. Блочный оператор. Операторы циклов, Вложенные циклы. Одномерные и двумерные массивы*;
- **Второй полусеместр** :
  - практики *(они же лекции)* - *2 часа в неделю;*
  - лабораторные работы ***(№№1-4)***

    - 4 часа в неделю в компьютерных классах (Э311/Э316)*;*

    По каждой лабораторной работе оценивается:
    - *выполнение базовой части* - ***до 5 баллов***,
    - *выполнение индивидуального задания, оформления и защита отчёта*

      - остальные баллы;
  - **контрольная работа 2** ***(на последней неделе)***

    Темы КР: *Объявление и элементы класса. Конструкторы класса. Свойство. Статические элементы класса. Наследование. Скрытие наследуемых элементов;*
- **Сессия**: **зачёт** в зачётную неделю в письменной форме.

Курсовая работа:

- задание выдаётся после выполнения обязательной части ***ЛР№3***,
- выполняется и демонстрируется на стационарных компьютерах в компьютерном классе, оформляется отчёт
- отчет защищается - ***в зачётную неделю***.

Технологическая карта

1. Лекции: коэффициент значимости совокупных результатов лекционных занятий – 0.3		
Текущая аттестация на лекциях	**Сроки(дата начала - дата окончания)**	**Максимальная оценка в баллах**
контрольная работа 1	01/09/2019 - 26/10/2019	**100**
Весовой коэффициент значимости результатов текущей аттестации по лекциям – 0.5		
Промежуточная аттестация по лекциям – зачет Весовой коэффициент значимости результатов промежуточной аттестации по лекциям – 0.5		
2. Практические занятия: коэффициент значимости совокупных результатов практических занятий – 0.2		
Текущая аттестация на практических/семинарских занятиях	**Сроки(дата начала - дата окончания)**	**Максимальная оценка в баллах**
контрольная работа 2	28/10/2019 - 21/12/2019	**100**
Весовой коэффициент значимости результатов текущей аттестации по практическим/семинарским занятиям– 1.0		
Промежуточная аттестация по практическим занятиям – (не предусмотрено) Весовой коэффициент значимости результатов промежуточной аттестации по практическим/семинарским занятиям – 0.0		
3. Лабораторные занятия: коэффициент значимости совокупных результатов лабораторных занятий – 0.5		
Текущая аттестация на лабораторных занятиях	**Сроки(дата начала - дата окончания)**	**Максимальная оценка в баллах**
Выполнение лабораторных работ (индивидуальных заданий) №1=20, №2=30, №3=10, №4=40	28/10/2019 - 21/12/2019	**100**
Весовой коэффициент значимости результатов текущей аттестации по лабораторным занятиям– 1.0		
Промежуточная аттестация по лабораторным занятиям – (не предусмотрено) Весовой коэффициент значимости результатов промежуточной аттестации по лабораторным занятиям – 0.0		
4. Курсовая работа: коэффициент значимости совокупных результатов курсовой работы		
Весовой коэффициент текущей аттестации выполнения курсовой работы/проекта – 0.0 (не предусмотрено)		
Весовой коэффициент промежуточной аттестации выполнения курсовой работы/проекта – защиты – 1.0		

Рекомендуемая литература :

1. **Шилдт, Герберт** C# 4.0: полное руководство.: Пер. с англ.-М.:ООО «И.Д.Вильямс», 2013.-1056с.:ил.
2. **Ватсон Б.** C# 4.0 на примерах. - СПб.:БХВ-Петербург, 2011.-608с.:ил.
3. **Эндрю Троелсен**. Язык программирования C# 2008 и платформа .NET 3.5 Framework. 4-е изд.: Пер с англ. М.: ВИЛЬЯМС, 2009. 1168с. ил.
4. **Джейсон Прайс, Майк Гандерлой**. Visual C# /NET. Полное руководство: Пер. с англ. К.: ВЕК+, СПб.: КОРОНА принт, К.: НТИ, М.: Энтроп, 2008. 960 с.
5. **Павловская Т.А.** C#. Программирование на языке высокого уровня. Учебник для вузов. СПб.: Питер, 2009. 432с.: ил.
6. **C# на примерах** – Спб.:Наука, 2016 . – 304с., ил.
7. **Язык C#: краткое описание и введение в технологии программирования**: учебное пособие / О. М. Котов. - Екатеринбург: Изд-во Урал. ун-та, 2014. 208 с.

Введение

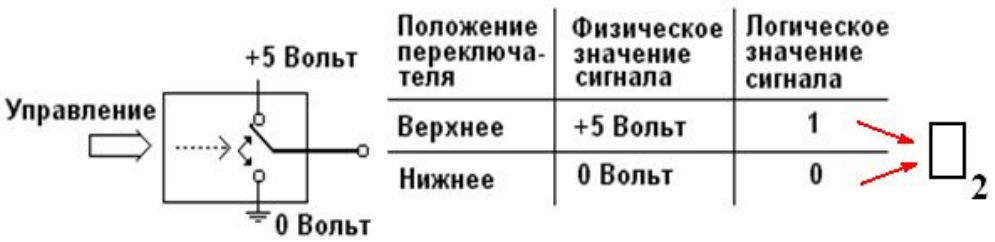
Не секрет, что ***информационные технологии*** – это наиболее динамично и масштабно развивающаяся сфера современного производства.

Основу любой цифровой информационной системы, как известно, составляют ***электронные устройства***, выполняющие ***программируемую обработку данных***. Среди этих устройств ***компьютер*** является основным и в наибольшей степени универсальным техническим средством оперирования с ***информацией*** *(дальше речь пойдёт только о компьютерах)*.

Информация существует в виде ***сигналов***.

Сигнал – это технически различимый ***параметр*** некоторого ***физического процесса***. В электронных устройствах такими процессами являются ***электрические процессы***, а сигналом является уровень ***напряжения***.

Цифровой сигнал характеризуется ***конечным*** набором ***разрешённых уровней***. Количество уровней, в свою очередь, определяет используемую систему счисления *(позиционную)*. На современном уровне развития электроники оптимальным являются ***два уровня напряжения*** и, соответственно, ***двоичная*** *(каноническая)* система счисления:



Положение переключа-теля	Физическое значение сигнала	Логическое значение сигнала
Верхнее	+5 Вольт	1
Нижнее	0 Вольт	0

$\square_2$

Системы счисления

Система счисления	Основание	Алфавит системы счисления
Двоичная	2	0,1
Десятичная	10	0,1,2,3,4,5,6,7,8,9
Шестнадцатеричная	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Формула значения числа

$$(a_{n-1}a_{n-2}\dots a_1a_{0,}a_{-1}a_{-2}\dots a_{-m})_b = \sum_i a_i b^i$$

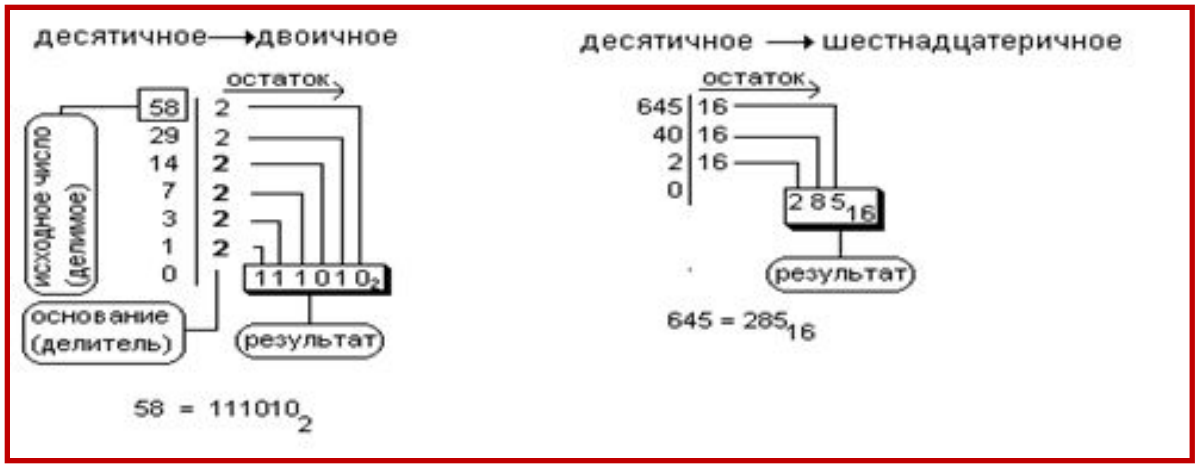
Целая часть **Дробная часть**

$$(a_{n-1}a_{n-2}\dots a_1a_{0,}a_{-1}a_{-2}\dots a_{-m})_b =$$

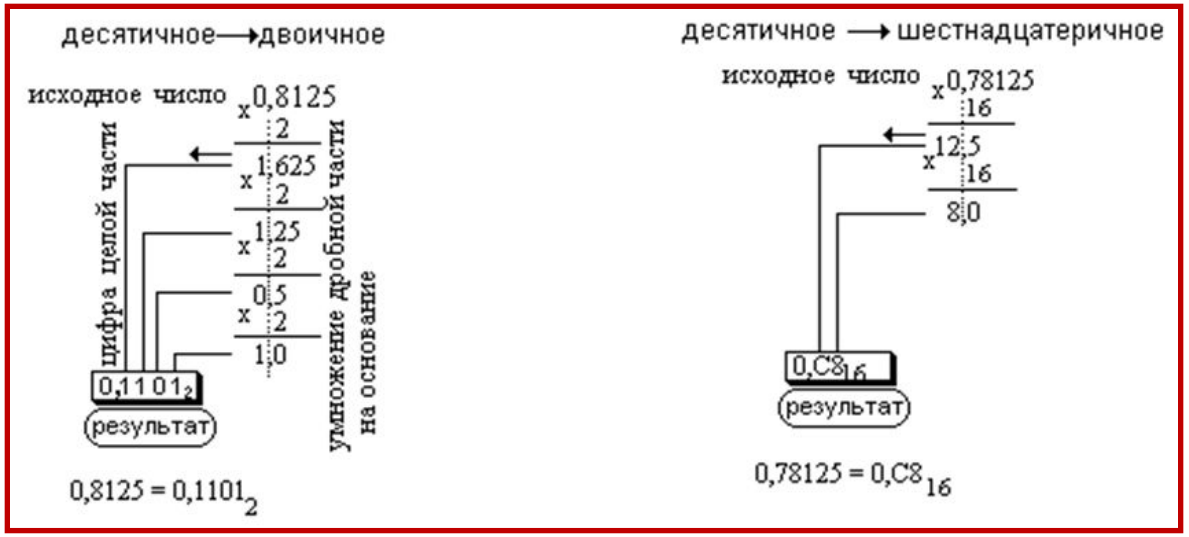
$$= a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \cdots + a_1b^1 + a_0b^0 + a_{-1}b^{-1} + a_{-2}b^{-2} + \cdots + a_{-m}b^{-m}$$

Формула значения числа позволяет выполнить
преобразование из любой ***недесятичной*** *системы в* ***десятичную***

Последовательное ***деление с остатком***:
преобразование из ***десятичной*** системы в ***недесятичную <u>целой</u>*** *части числа*



Последовательное ***умножение***:
преобразование из ***десятичной*** системы в ***недесятичную <u>дробной</u>*** *части числа*



Преобразование между шестнадцатеричной и двоичной системами

основано на «***родственности***» их оснований: каждые ***четыре разряда*** двоичной системы соответствуют ***одной цифре*** шестнадцатеричной системы:

Система счисления		
десятичная	**двоичная**	**шестнадцатеричная**
0	**0000**	**0**
1	**0001**	**1**
2	**0010**	**2**
3	**0011**	**3**
4	**0100**	**4**
5	**0101**	**5**
6	**0110**	**6**
7	**0111**	**7**
8	**1000**	**8**
9	**1001**	**9**
10	**1010**	**A**
11	**1011**	**B**
12	**1100**	**C**
13	**1101**	**D**
14	**1110**	**E**
15	**1111**	**F**

Примеры преобразований и вычислений

$1234=1\cdot10^3+2\cdot10^2+3\cdot10^1+4\cdot10^0$

$1234_{16}=1\cdot16^3+2\cdot16^2+3\cdot16^1+4\cdot16^0=4660$

$00111010,11_2=1\cdot2^5+1\cdot2^4+1\cdot2^3+1\cdot2^1+1\cdot2^{-1}+1\cdot2^{-2}=32+16+8+2+0.5+0.25=58.75$

$A7_{16}+1EF_{16}=296_{16}$

$$\begin{array}{r} A7_{16} \\ +1EF_{16} \\ \hline 296_{16} \end{array}$$

n	7	6	5	4	3	2	1	0
2^n	128	64	32	16	8	4	2	1

$000111010,001011_2 = 3A,2C_{16}$

n	-1	-2	-3	-4
2^n	0,5	0,25	0,125	0,0625

Для самостоятельного решения

$A24_{16}+2CC_{16}=?_{16}$

$CA4_{16}+FF_{16}=?_{16}$

$110,1111_2=?_{16}$

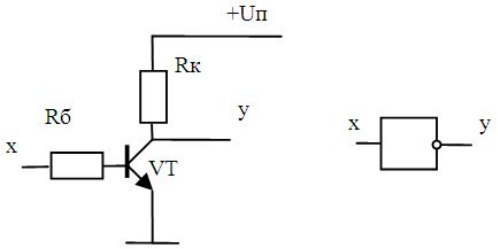
$101,0111_2=?_{10}$

$46+10100111_2+1C_{16}=?$

$1D_{16}+11010111_2=?_{10}$

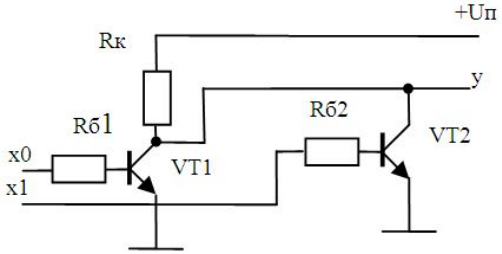
Элементы схемотехники

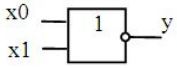
Инвертор:



x	y
0	1
1	0

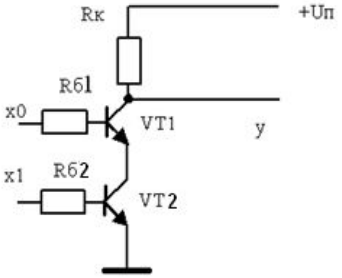
Элемент 2ИЛИ-НЕ:

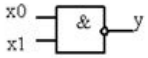




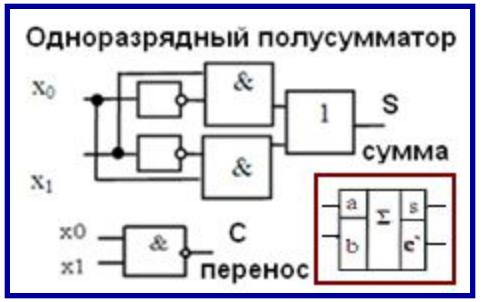
x1	x0	y
0	0	1
0	1	0
1	0	0
1	1	0

Элемент 2И-НЕ:

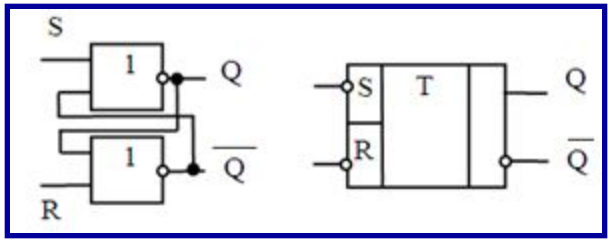




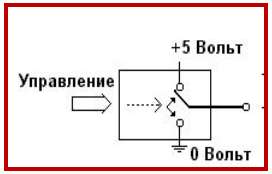
x1	x0	y
0	0	1
0	1	1
1	0	1
1	1	0



Триггер: размещение двоичного разряда



R	S	Q^{n+1}
0	0	Q^n
0	1	0
1	0	1
1	1	Неопределённое состояние



Внутреннее представление числовых данных

Беззнаковые целочисленные форматы

Байтовый *(8 разрядов)*

Формула максимального числа в ***n*** разрядах:

$$max = b^n - 1$$

7	6	5	4	3	2	1	0
0	1	1	1	1	0	0	1

Номер разряда

Содержимое разряда

→ диапазон [0, …, 255] – **256** значений

Короткий целый *(2 байта, 16 разрядов)*

Старший байт								Младший байт							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	0	0	1	1	1	1	0	0	1

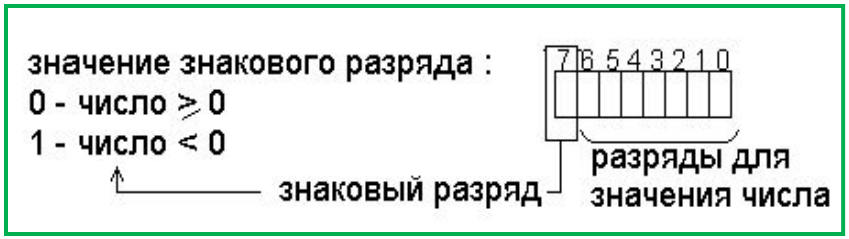
→ диапазон [0 ,..., 65 535]

Средний (*двойное слово, 32 разряда)*

→ диапазон [0 … 4 294 967 295]

Знаковые целочисленные форматы

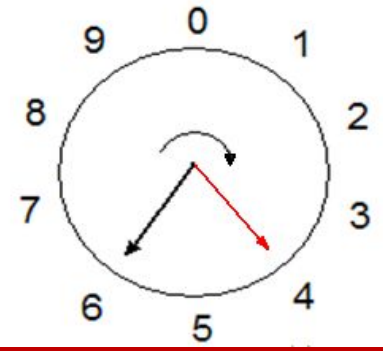
Байтовый



Целый формат	Число разрядов	Диапазон значений	Ширина диапазона
Байтовый	8	-128 ... 127	256
Короткий	16	-32 768 ... 32 767	65536
Средний	32	- 2 147 483 648... 2 147 483 647	4 294 967 296

Представление отрицательных чисел в форме ***дополнений до двух***

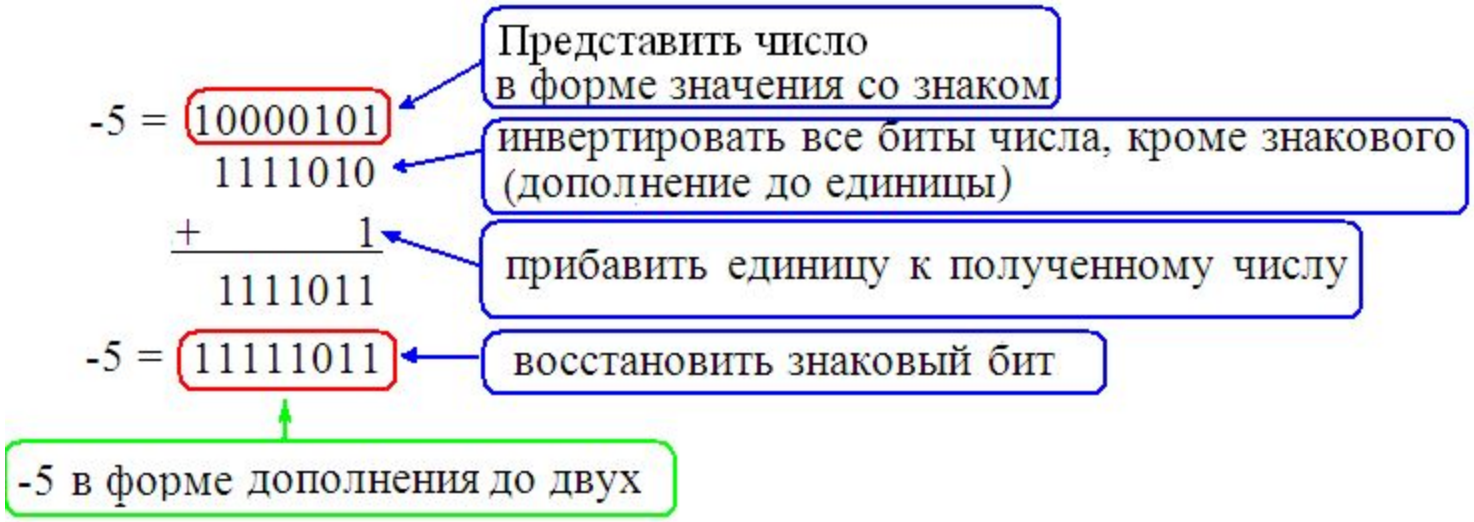
Основным устройством процессора является ***сумматор***, предназначенный прежде всего для ***сложения*** двоичных чисел. Для того, чтобы сумматор мог выполнять ещё и операцию ***вычитания***, отрицательные числа должны быть представлены в форме ***дополнений до двух***



$6 - 2 = 6 + (-2) = 6 + Доп_{10}(-2) = 6 + 8 = 4$

$Доп_{10}(-2) = 10 - 2 = 8$

Порядок получения ***дополнения до двух***



Пример

$6_{10} - 15_{10} = ?$

$-15 = 10001111_2$

1110000_2

$+ \quad 1$

11110001_2

дополнение до двух числа -15

$6 = 00000110_2$ → 00000110_2

$+ \; 11110001_2$

11110111_2

результат в ФДД, т.к. отрицательный

11110111_2

0001000_2

$+ \quad 1$

$10001001_2 = -9$

окончательный результат

Ситуация переполнения

Примеры **переполнений**

65 + 65 = ?

```
 01000001
+01000001
 10000010
 11111101
+        1
 11111110 = -126
```

результат

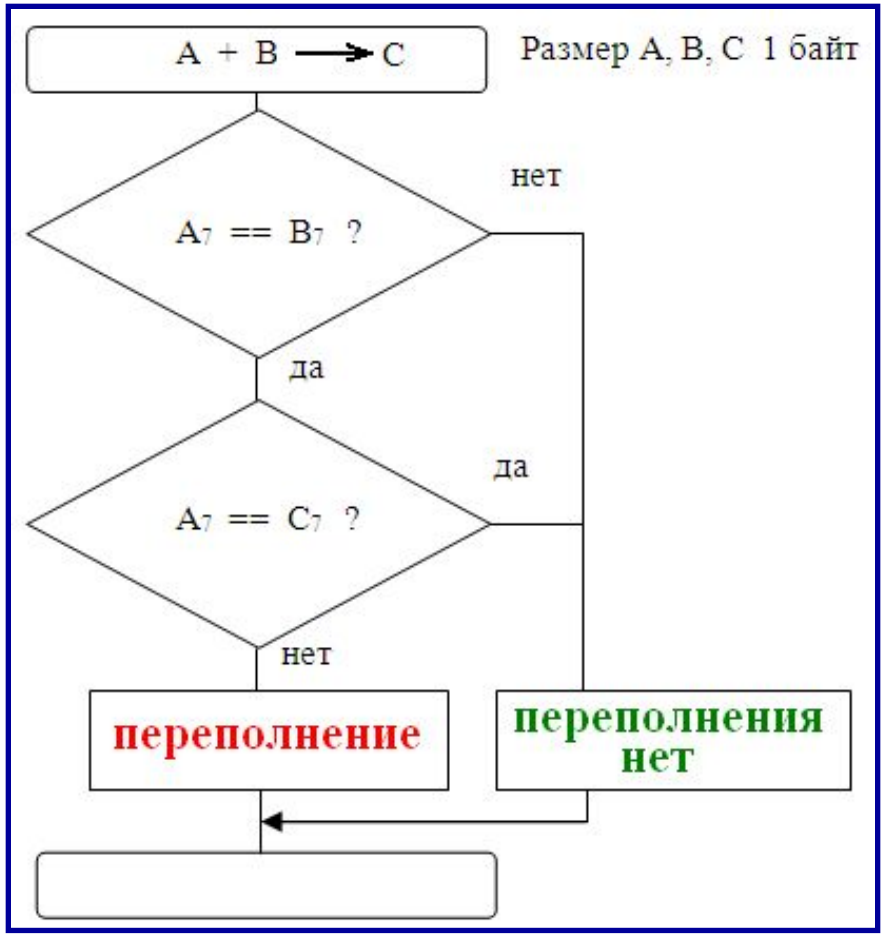
-125 - 125 = ?

$Доп_2(-125)$ = 10000011

```
 10000011
 00000110 = 6
```

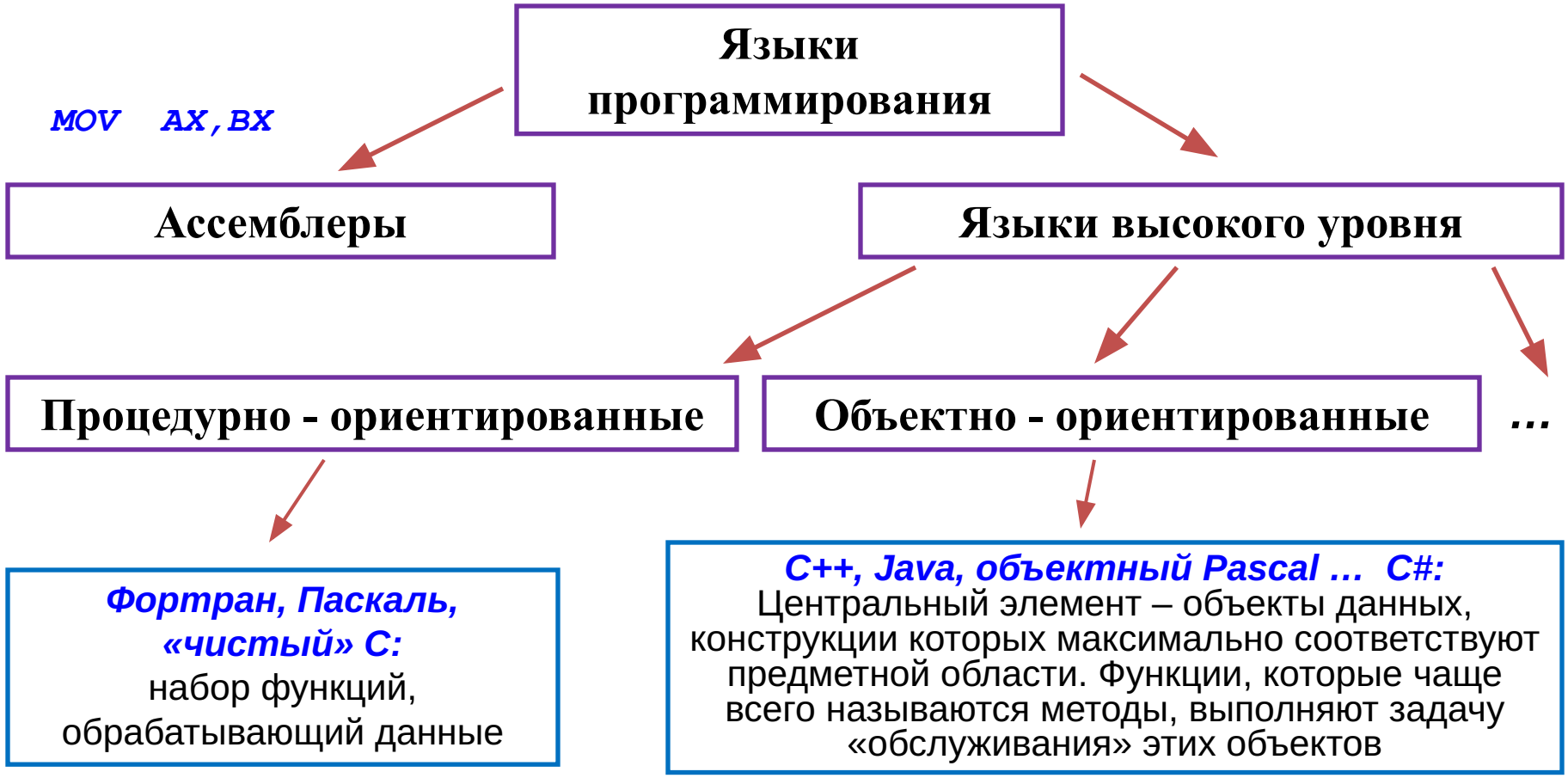
результат

Алгоритм проверки



Языки программирования

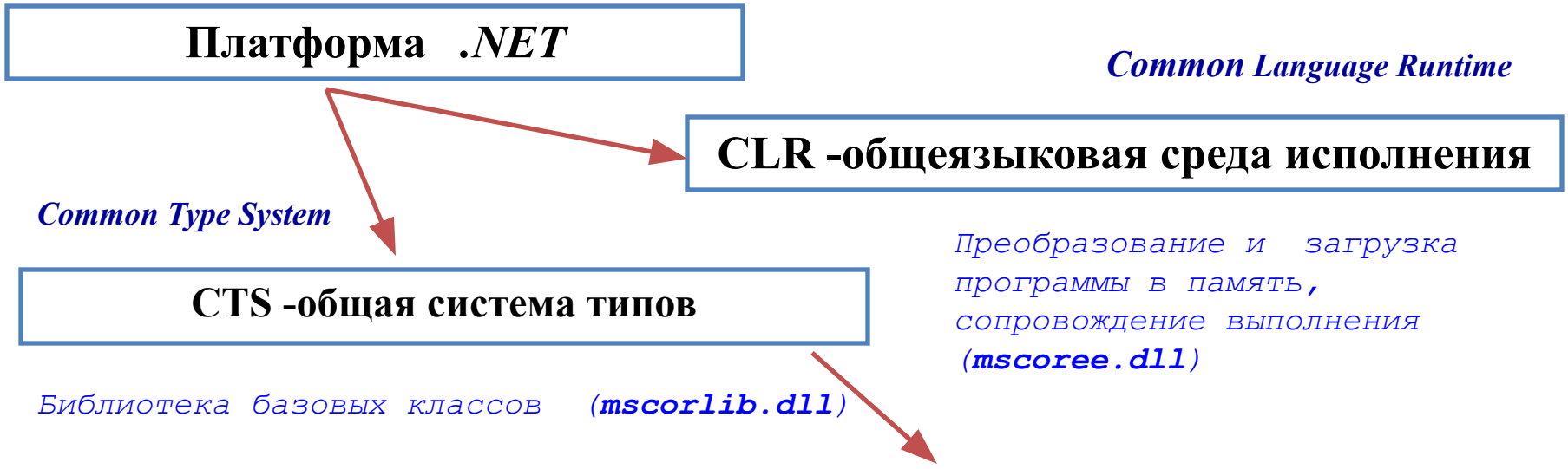
-это способ записи исходного кода программ для реализации того либо иного алгоритма обработки данных. Выполняет эту обработку, как известно, процессор.
По степени «близости» к процессору языки программирования делятся на ***низкоуровневые*** (машинно-ориентированные) и ***высокоуровневы***е.
Языки ***высокого уровня,*** *прежде всего,* различаются на ***процедурно-*** и ***объектно-*** ориентированные



Первое представление об языке C#

Язык программирования C# *(произносится «си шарп», хотя название задумывалось, как «си диез»)* вобрал в себя лучшие черты своих предшественников и в большей степени отвечает современным потребностям:

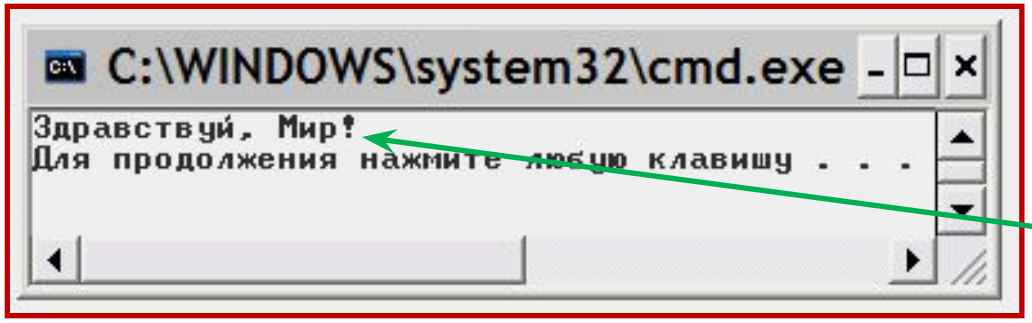
- ***Преемственность (в значительной степени) с языками C, C++, Java;***
- ***Проще, чем C++, с сопоставимой «мощностью» и эффективностью;***
- ***Полностью объектно-ориентированный (любая сущность - объект);***
- ***Компонентно – ориентированный;***
- ***Основан на платформе «.NET» (произносится «дот нет»);***



Технология создания приложения на C#

Наряду с языком **C#,** на платформе **.NET** могут быть использованы ещё несколько десятков других языков программирования *(C++, VisualBasic, JScript и другие)*

Первая программа



С лёгкой руки разработчиков ещё *языка C* **первая программа** на новом *(для разработчика)* языке должна выводить на консольное окно *(то есть на экран в текстовом режиме)* простое приветствие

void Main() - заголовок метода Main(): объявляет пустой возвращаемый тип (void) и пустой список входных параметров.

WriteLine - вызов функции (метода) с входным аргументом – строковой константой

static объявляет метод Main() статическим, что определяет возможность его запуска на выполнения без явного создания объекта класса;

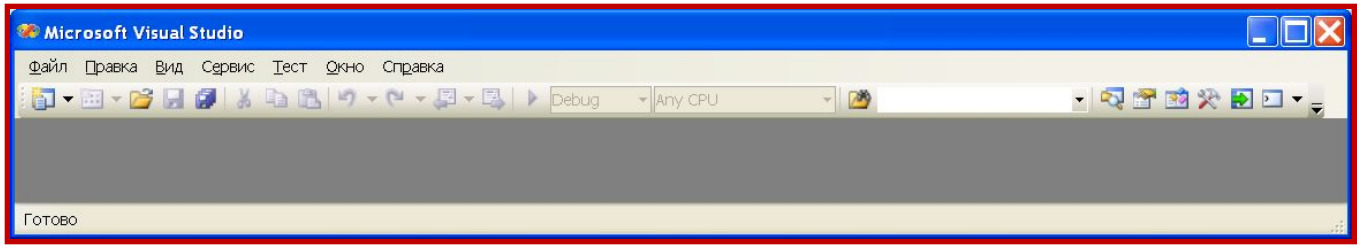
```
class FirstProgram
    {
    static void Main()
        {
            System.Console.WriteLine("Здравствуй, Мир!");
        }
    }
```

class – это ключевое слово для задания конструкции объекта, FirstProgram - имя этой конструкции, а метод Main() - часть этого объекта;

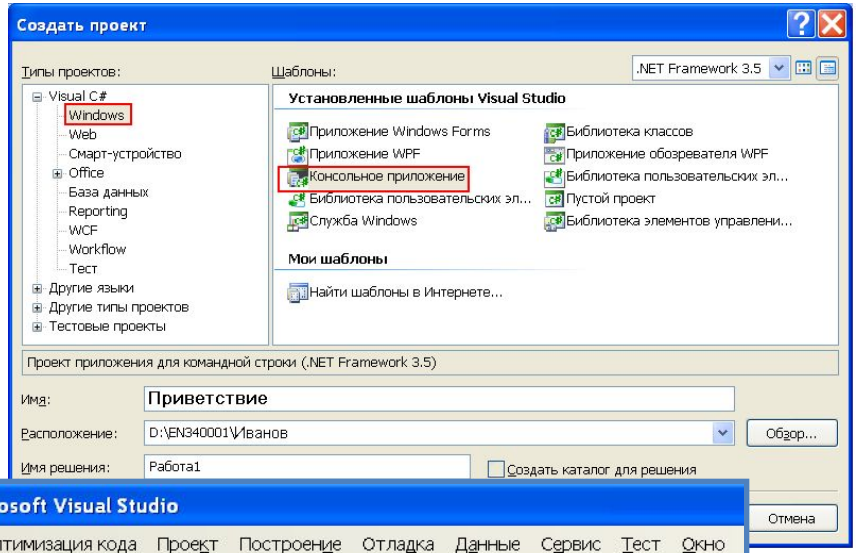
System.Console указывает, что метод WriteLine() является элементом класса Console в корневом пространство имён System

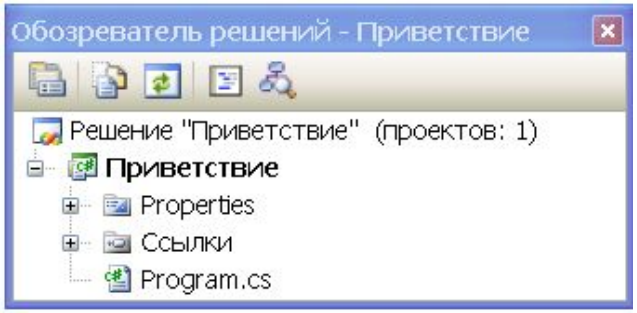
Первая программа в среде Visual Studio *(режим консольного приложения)*

1.Запустить среду

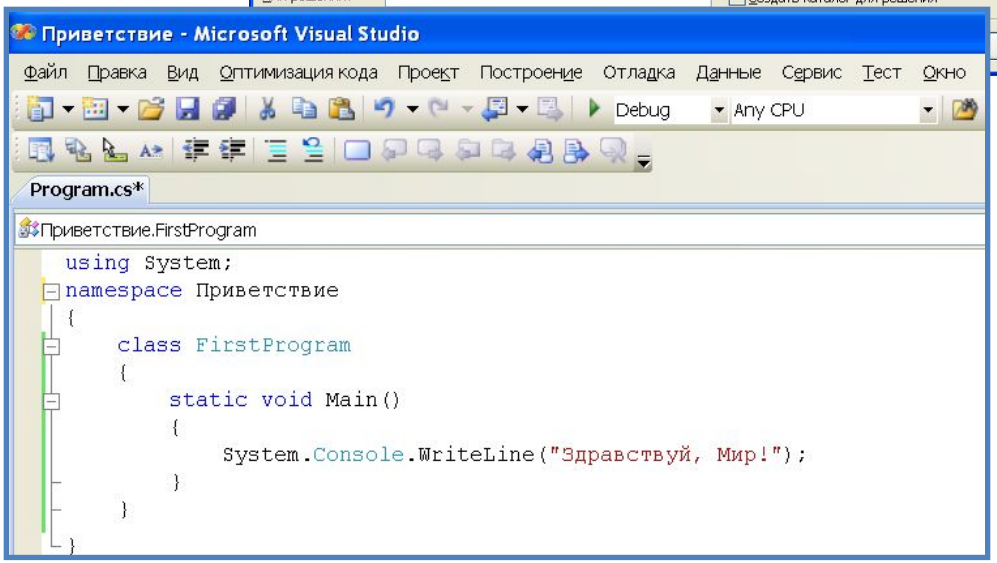


2. Создать проект ***Файл -> Создать -> Проект***





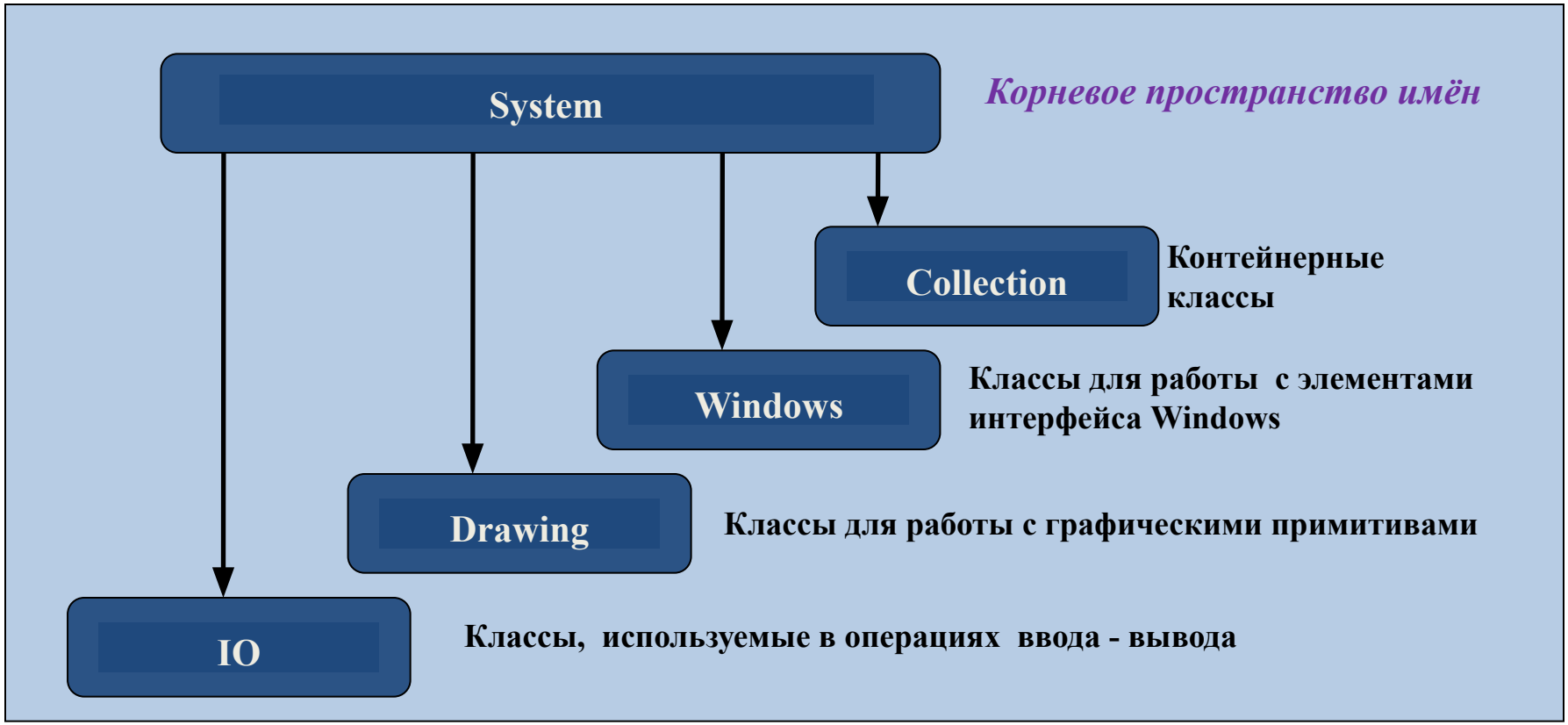
3. В файле **Program.cs**
(откроется автоматически)
рекомендуется
переименовать класс и
набрать одну строчку



4. Запустить на выполнение с помощью:
Отладка -> Запуск без отладки

Пространства имён

Система *(библиотека) типов CTS насчитывает более* ***4000*** *различных типов, которые для удобства работы объединены в группы – пространства имён. Группы, в свою очередь, представляют собой иерархическую древовидную систему:*

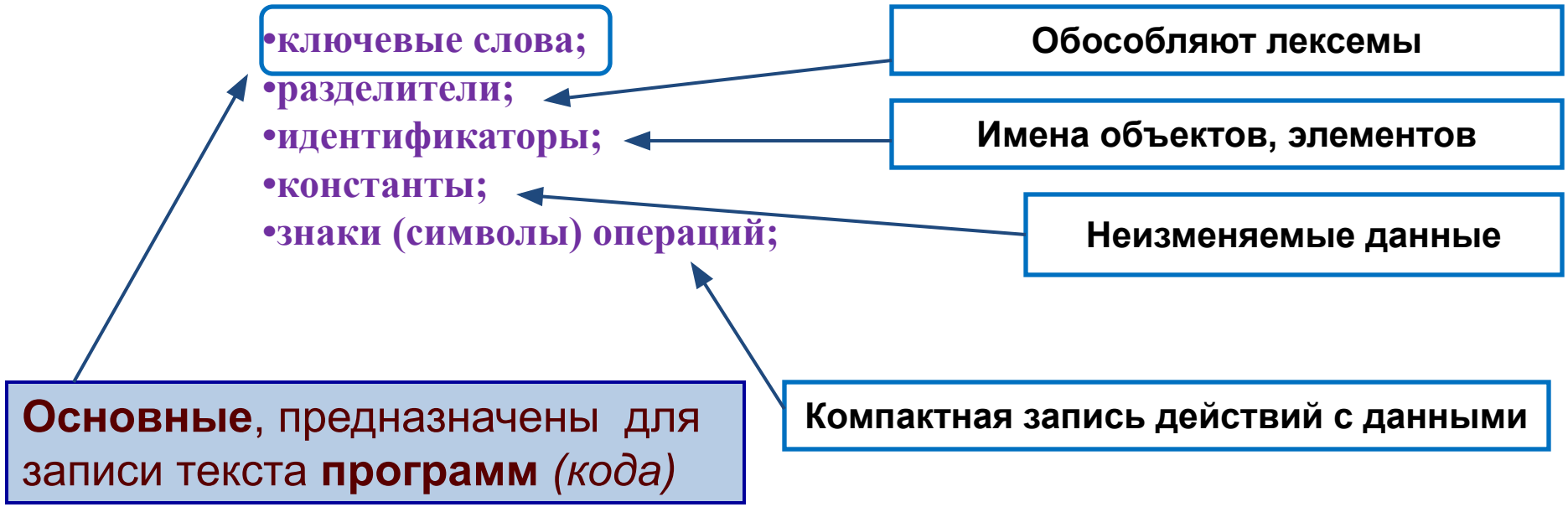


Общая характеристика языка

Алфавит *(набор литер)*
языка программирования ***C#*** составляют символы:

- строчные и прописные ***буквы*** латинского алфавита;
- ***цифры*** от 0 до 9;
- символ ***подчеркивания*** ‘_’;
- набор специальных символов: **" { } | [ ] + - % / \; ' : ? < > = ! & # ~ ***
- прочие символы.

В свою очередь, алфавит ***C#*** служит для построения слов, которые называются ***лексемами***. Различают пять типов ***лексем*:**



Ключевые слова

языка программирования ***C#*** составляют 77 конструкций, большая часть которых заимствованы из языков C, C++ и Java:

abstract	false	params	this
as	finally	private	throw
base	fixed	protected	true
bool	float	public	try
break	for	readonly	typeof
byte	foreach	ref	uint
case	goto	return	ulong
catch	if	sbyte	unchecked
char	implicit	sealed	unsafe
checked	in	short	ushort
class	int	sizeof	using
const	interface	stackalloc	virtual
continue	internal	static	void
decimal	is	string	volatile
default	lock	struct	while
delegate	long	switch	
do	namespace	operator	
double	new	out	
else	null	override	
enum	object		
event			
explicit			
extern			

Разделители

обособляют ***лексемы*** языка и могут быть следующих видов:

- **пробел,**
- **табуляция,**
- **символ новой строки,**
- **скобки,**
- **комментарии**

Комментарий – это текст, который предназначен только для читающего программу человека и игнорируется компилятором. В ***C#*** это можно сделать одним из двух способов :

парный комментарий -

/* коммен-

тарий */

строчный комментарий –

// *комментарий* - заканчивается в конце данной строки

xml – комментарий *(для создания самодокументируемых программ)*

/// комментарий

Идентификаторы

- это ***имена***, которые назначает программист. Именуются объекты, классы, структуры, методы, метки и т.п.

Правила написания идентификаторов достаточно простые:

- **состоят из одного или более символов;**
- **идентификатор не может совпадать с ключевым словом;**
- **размер идентификатора не ограничен;**
- **первый символ – обязательно буква;**
- **можно использовать русские буквы;**

Обычно даётся осмысленное, отражающее назначение элемента имя

Хорошую читабельность имени придают знаки нижнего подчёркивания:
my_first_variable

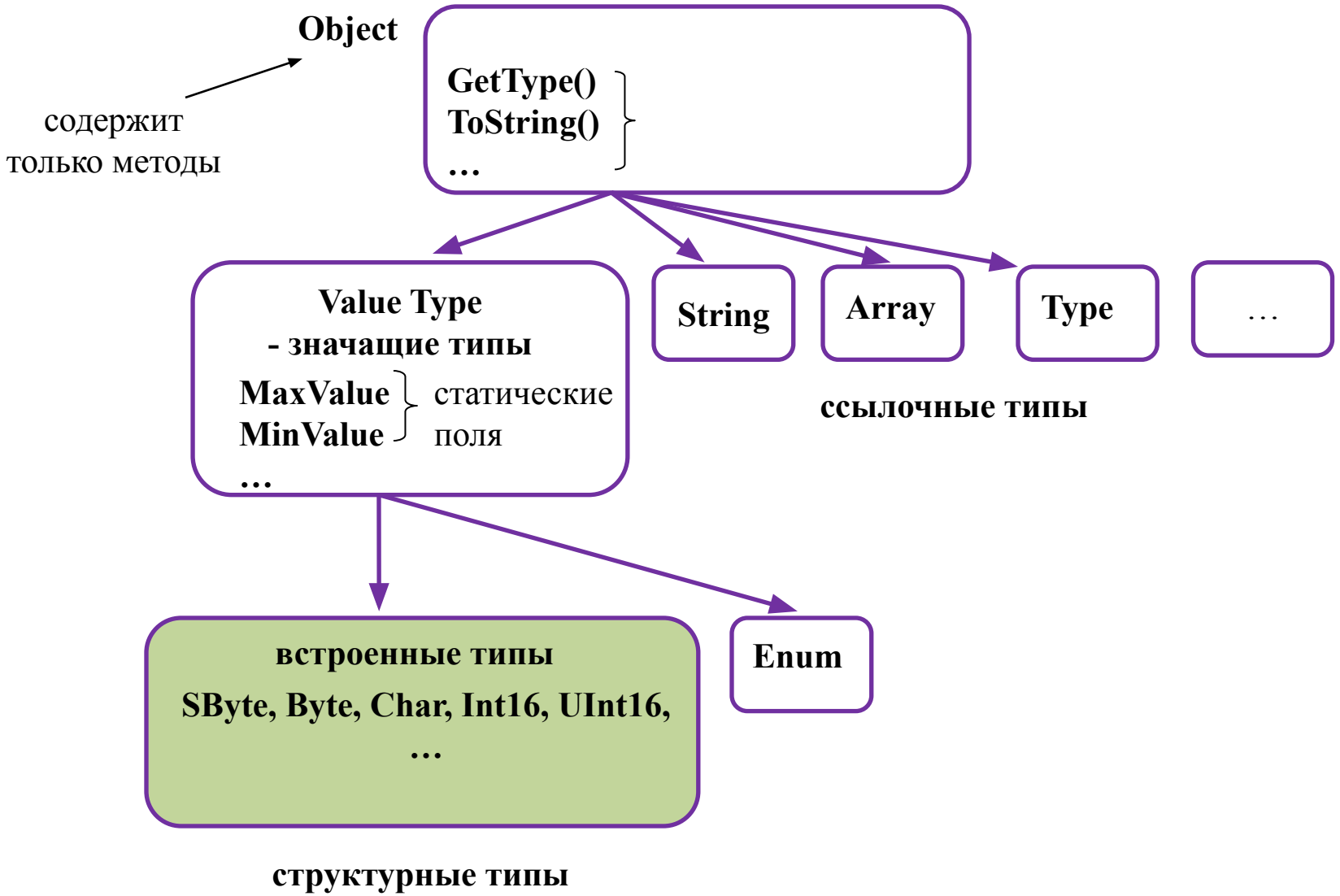
или использование прописных букв MyFirstVariable

мой_первый_объект

МояПерваяПеременная

Иерархия встроенных типов в C#

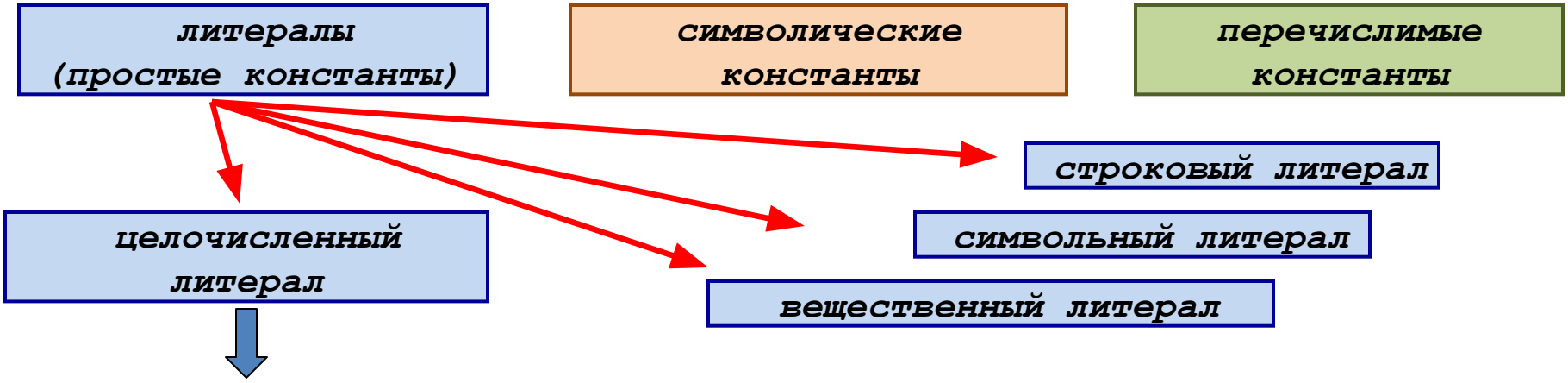
Все типы данных в ***C#*** являются производными от системного «супербазового» типа ***Object***



Константы

Константами называются ***объекты данных***, которые ***не изменяют*** своего значения на всём времени выполнения программы.

Константы в C# бывают трёх типов :



Целочисленный литерал (*или целочисленная константа*) служит для записи **целых** значений и является последовательностью **цифр**. Эта последовательность может предваряться знаком '-', который в данном случае является **операцией** смены знака.

Целочисленный литерал, начинающийся с ***0x*** или ***0X***, воспринимается транслятором, как шестнадцатеричное целое. В этом случае он может включать ещё и символы от ***A*** (или ***a)***, до ***F*** (или ***f)***

Целочисленный литерал

В зависимости от значения целочисленный литерал размещается последовательно в одном из следующих форматов (по мере увеличения значения)

целочисленный знаковый размером 4 байта	***max*** $= 2^{31} - 1$
целочисленный беззнаковый размером 4 байта	***max*** $= 2^{32} - 1$
целочисленный знаковый размером 8 байт	***max*** $= 2^{63} - 1$
целочисленный беззнаковый размером 8 байт	***max*** $= 2^{64} - 1$

Непосредственно ***за константой*** могут располагаться в произвольном сочетании один или два специальных суффикса:

U (или ***u***) и ***L*** (или ***l***)

использование беззнакового формата

использование формата удвоенного размера – 8 байт

Вещественный литерал

Запись десятичного числа в естественной форме (разделитель целой и дробной части – точка)

Запись десятичного числа в экспоненциальной форме (разделитель мантисса/порядок – символ E(e))

Размещается вещественный литерал в ***8-байтовом*** плавающем формате (соответствует плавающему типу с удвоенной точностью – ***double***)

Непосредственно за константой могут располагаться один из двух специальных суффиксов:

F *(или f)* и **M** *(или m)*

Использование формата одинарной точности - *float* размером 4 байта

Использование формата повышенной точности - *decimal* размером 16 байт

Символьный литерал

Одна или несколько литер, заключенных в **апострофы**

Значением является соответствующий **код** *(Unicode)*

Размещается в поле размером **два байта**

- символ, заключённый в апострофы, например, `D`. (печатный символ)

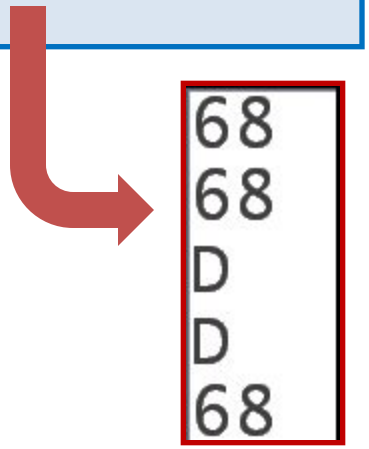
- целочисленная константа после обратного слеша. (для кодов, которые не представлены на клавиатуре, при отсутствии символического представления). Например, константа `\x44` содержит код буквы D (шестнадцатеричный)

- заключенная в апострофы литера после символа обратного слеша, т.н. esc – последовательности, команды управления некоторыми устройствами:

`\u` - задание 4-х значного шестнадцатеричного кода в системе Unicode;
`\a` - звуковой сигнал (\u0007);
`\b` - возврат на одну позицию назад (\u0008);
`\n` - переход на новую строку (\u000A);
`\r` - возврат каретки (курсора) в первую позицию строки (\u000D);
`\t` - переход к следующей метке горизонтальной табуляции (\u0009);
`\v` - переход к следующей метке вертикальной табуляции (\u000B);
`\0` - конец строки, нуль - символ (\u0000), терминатор);
`\`` - апостроф;
`\"` - кавычки;

Пример вывода на консоль целочисленных и символьных констант

```
using System;
class Primer
   {  static void Main()
       {
         Console.WriteLine ( 68 );
         Console.WriteLine ( 0x44 );
         Console.WriteLine ('D');
         Console.WriteLine ('\x44');
         Console.WriteLine ( 0x44.ToString() );    // константа – это тоже объект!
       }
    }
```



Статические методы класса Console :

WriteLine() *– вывод строки на консоль с переводом курсора на новую строку ;*

Write() *– то же самое, но без перевода курсора);*

ReadLine() *– ввод строки с консоли;*

Read() *– чтение с консоли одного символа;*

Строковый литерал

- это последовательность символов *(в одном из возможных форматов)*, заключенных в двойные кавычки, например “Строка”

Пример вывода строк

```
using System;
class Primer0
{
    static void Main()
    {
       Console.WriteLine("\t\u0041\x41\r\x42\x43\b\u0044");
       Console.WriteLine("A" + "\xA" + "A");
    }
}
```

```
BD          AA
A
A
```

Средства форматирования

Представляют собой дополнительные ***входные аргументы*** для методов ***WriteLine*** или ***Write***

В минимальном составе аргументами методов WriteLine или Write являются объекты, значения которых необходимо вывести на консоль (предыдущий пример).Вывод на консоль - текстовый, преобразование выполняется автоматически по типу и размеру объекта, методу WriteLine в третьем вызове передаются два аргумента:

Если аргументов более одного, используют строку управления выводом, которая является по сути строковой константой. В предельно простом случае она должна содержать т.н. «места заполнения» вида {номер}, где номер – очередность входного аргумента при нумерации с нуля:

```
using System;
class Primer
{
   static void Main( )
    {
     Console.WriteLine ( "Простые константы:" );
     Console.WriteLine ( "десятичная {0}",100);
     Console.WriteLine("шестн. {1} символьная {2} веществ. {0}",  3.1415, 0x64, 'd' );
    }
 }
```

```
Простые константы:
десятичная 100
шестн. 100 символьная d веществ. 3,1415
```

Средства форматирования *(продолжение)*

Полный вариант места заполнения имеет следующий вид:

{ номер[,размещение][формат] }

размещение - целое число, определяющее желаемую ширину поля(количество позиций).Положительное значение соответствует позиционированию числа по правой границе поля, отрицательное – по левой границе. Если ширина поля не достаточна для размещения числа,параметр размещение игнорируется, незанятые позиции заполняются пробелами.

формат – запись вида Axx, где A – спецификатор формата преобразования, xx – спецификатор точности(допустимые значения от 0 до 99).

Спецификатор точности обычно указывается для значений с дробной частью. В случае целочисленных значений, по сути, он также определяет ширину поля, но при этом в качестве заполнителя используются нули

Основные спецификаторы формата:

D или d – вывод целого десятичного значения;

X или x – вывод целого шестнадцатеричного значения;

E или e – вывод плавающего десятичного значения в экспоненциальной форме;

F или f – вывод плавающего десятичного значения в форме с фиксированной (заданной) точностью;

G или g – вывод плавающего десятичного значения в универсальной форме;

N или n – вывод плавающего десятичного значения с разделением разрядов (всегда группами по три позиции);

Средства форматирования *(пример)*

```
using System;
class Пример
 {
   static void Main()
   {
      Console.WriteLine("\tФорматирование с параметрами:");
      Console.WriteLine(" десятичное{0,8} шестнадцатеричное{1,-8:X}", 0x64, 100);
      Console.WriteLine(" десятичное{0:D8} шестнадцатеричное{1:X6}", 0x64, 100);
      Console.WriteLine(" стандартная запись={0:f3}", 3.141592);
      Console.WriteLine(" экспоненциальная запись ={0:e4}", 3.141592);
      Console.WriteLine(" универсальная запись ={0:G5}", 123.45679);
      Console.WriteLine(" универсальная запись ={0:G3}", 123.45679);
      Console.WriteLine(" универсальная запись ={0:G2}", 123.45679);
      Console.WriteLine(" универсальная запись ={0:G5}", 12345.6543);
      Console.WriteLine(" универсальная запись ={0:G3}", 12345.6543);
      Console.WriteLine(" универсальная запись ={0:G2}", 12345.6543);
      Console.WriteLine(" запись с разделителями ={0:n2}", 1234567.6543);
   }
}
```

```
        Форматирование с параметрами:
десятичное     100 шестнадцатеричное64
десятичное00000100 шестнадцатеричное000064
стандартная запись=3,142
экспоненциальная запись =3,1416e+000
универсальная запись =123,46
универсальная запись =123
универсальная запись =1,2E+02
универсальная запись =12346
универсальная запись =1,23E+04
универсальная запись =1,2E+04
запись с разделителями =1 234 567,65
```

Объекты данных

Требование строгой типизации

тип идентификатор = значение ;

инициализация может быть позднее, но всегда до первого использования объекта

встроенные

пользовательские

Размерные (или значащие) – непосредственно содержат данные

Всегда размещаются в стековой памяти;

переменные, структуры и перечисления

Требуют обязательную инициализацию

Ссылочные – содержат адрес (ссылку) на объект (объекты), размещённые в памяти

Выделение памяти выполняется с помощью операции **new**

строки, массивы, объекты классов

Встроенные (примитивные) типы

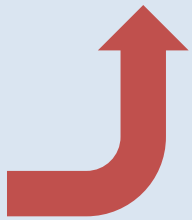
Старшинство типа

Тип	Размер, байт (состав и размеры полей, бит)	Соответствующий системный тип
`bool`	1(8)	*Boolean*
`sbyte`	1 (1_7)	*Sbyte*
`byte`	1 (8)	*Byte*
`short`	2 (1_15)	*Int16*
`ushort`	2 (16)	*UInt16*
`char`	2(16)	*Char*
`int`	4(1_31)	*Int32*
`uint`	4 (32)	*UInt32*
`long`	8(1_63)	*Int64*
`ulong`	8 (64)	*UInt64*
`float`	4 (1_8_23)	*Single*
`double`	8(1_11_52)	*Double*
`decimal`	16(1_7_8_16_96)	*Decimal*

Пример

```
using System;
class Primer
{   static void Main()
   {
byte  a = 255;//неявное преобразование  - выполняется только для целочисленных типов
sbyte b = 127;
char  c=  'd';
bool   d;
short  e;
ushort  f;
int  g =  0x100;
float   h = 5.2F;
double  i  = 123e-2;
decimal  j = 0.00000000001M;
d = false;
e = 32767;
f = 65535;
Console.WriteLine ( "{0}={1}\t\tразмер={2}", a.GetType( ),  ++a,  sizeof ( byte ) );
Console.WriteLine ( "{0}={1}\tразмер={2}", b.GetType( ),    ++b,  sizeof ( sbyte ) );
Console.WriteLine ( "{0}={1}\t\tразмер={2}", c.GetType( ),  ++c,  sizeof ( char ) );
Console.WriteLine ( "{0}={1}\tразмер={2}", d.GetType( ),       d,  sizeof ( bool ) );
Console.WriteLine ( "{0}={1}\tразмер={2}", e.GetType( ),    ++e,  sizeof ( short ) );
Console.WriteLine ( "{0}={1}\t\tразмер={2}", f.GetType( ),   ++f,   sizeof ( ushort ) );
Console.WriteLine ( "{0}={1}\tразмер={2}", g.GetType( ),    ++g,  sizeof ( int ) );
Console.WriteLine ( "{0}={1}\tразмер={2}", h.GetType( ),    ++h,  sizeof ( float ) );
Console.WriteLine ( "{0}={1}\tразмер={2}", i.GetType( ),    ++i,   sizeof ( double ) );
Console.WriteLine ( "{0}={1}\tразмер={2}", j.GetType( ),    ++j,   sizeof ( decimal ) );
   }
}
```

```
System.Byte=0                 размер=1
System.SByte=-128             размер=1
System.Char=e                 размер=2
System.Boolean=False          размер=1
System.Int16=-32768           размер=2
System.UInt16=0               размер=2
System.Int32=257              размер=4
System.Single=6,2             размер=4
System.Double=2,23            размер=8
System.Decimal=1,00000000001     размер=16
```



Выражения и операции

Выражение – это объединение операндов (объектов данных) с помощью операций (действий с данными).
Синтаксис выражений максимально приближен к алгебраической форме

```
операнд1 @ операнд2 @ операнд3;
```

Операции делятся по типу (арности) и назначению (основные группы)

- *унарные (одноместные),*
- *бинарные (двуместные),*
- *тернарные (трехместные, единственная)*

Арифметические	`+ - * / % ++ --`
Логические (булевые)	`! && || true false`
Битовые	`~ & | ^`
Сдвиговые	`<< >>`
Отношения	`== != < > <= >=`
Замещения	`= += -= *= /= %= &= |= ^= <<= >>= ??`
Доступа к элементу	`.`
Индексации	`[]`
Приведения типа	`(тип)`
Выбор по условию	`?:`
Создания объекта	`new`
Типа информации	`sizeof as is`

Старшинство операций

Знак операции	Приоритет	Арность	Действие	Примечание
-, +	0	1	Смена знака (унарные ***минус, плюс***)	
!	0	1	Логическая ***НЕ***	Только для булевого операнда
~	0	1	Инверсия	Только для целочисленных операндов
++	0	1	Инкремент	Префиксная операция выполняется **до** использования переменной, постфиксная – **после**
--	0	1	Декремент	
*****	1	2	Умножение	
/	1	2	Деление	
%	1	2	Остаток от деления	Первый операнд может быть вещественным
+	2	2	Сложение	
-	2	2	Вычитание	
<<	3	2	Сдвиг влево	Для целочисленных операндов
>>	3	2	Сдвиг вправо	
<	4	2	Отношение меньше	Результат – значение булевого типа (***true*** или ***false***)
>	4	2	Отношение больше	
<=	4	2	Отношение меньше или равно	
>=	4	2	Отношение больше или равно	
==	5	2	Отношение равенства	
!=	5	2	Отношение неравенства	

Старшинство операций *(продолжение)*

Знак операции	Приоритет	Арность	Действие	Примечание
&	6	2	Битовая ***И***	Для целочисленных операндов
^	7	2	Битовая ***исключающая ИЛИ***	
|	8	2	Битовая ***ИЛИ***	
&&	9	2	Логическая ***И***	Для булевого операнда
||	10	2	Логическая ***ИЛИ***	Для булевого операнда
??	11	2	Логического **замещения**	Проверяется ***первый*** операнд на **null** и если не равен, его значение возвращается, в противном случае, возвращается значение ***второго*** операнда
?:	12	3	Проверка или выбор по условию	

Старшинство операций *(окончание)*

Знак операции	Приоритет	Арность	Действие
=	**13**	2	Присвоение
***=**		2	Умножение с замещением
/=		2	Деление с замещением
+=		2	Сложение с замещением
-=		2	Вычитание с замещением
<<=		2	Сдвиг влево с замещением
>>=		2	Сдвиг вправо с замещением
&=		2	Битовая ***И*** с замещением
^=		2	Битовая ***исключающая ИЛИ*** с замещением
|=		2	Битовая ***ИЛИ*** с замещением

Изменить очередность операций в выражении можно как и в арифметике – парой круглых скобок

Битовые операции

Применяются только для объектов целочисленных типов.
Соответствуют одноимённым командам процессора

- ~ NOT
- | OR
- ^ XOR
- & AND

```
using System;
class Primer
{
   static void Main()
   {
      byte a = 25;
      sbyte b = 30;
      Console.WriteLine(~b);
      Console.WriteLine(a << 3);
      Console.WriteLine(a & b);
      Console.WriteLine(a | b);
      Console.WriteLine(a ^ b);
      Console.WriteLine(a);
      Console.WriteLine(b);
   }
}
```

```
-31
200
24
31
7
25
30
```

	1				E				h	= 30
	0	0	0	1	1	1	1	0	b	
~	1	1	1	0	0	0	0	1	b	ФДД
шаг1		0	0	1	1	1	1	0	b	
шаг2	+							1	b	
шаг3	1	0	0	1	1	1	1	1	b	ФЗСЗ
	-	1			F				h	= -31

Арифметические операции

**Выглядят, как привычные алгебраические выражения.
Как и в арифметике, операции умножения и деления,
старше, чем сложения и вычитания**

```
using   System;
class  Primer
{
   static    void    Main()
      {
         short      e = 25;
         int          g = 10;
         float       h= 10F;
         double    i= 25e-1;
         Console.WriteLine ( e/g );
         Console.WriteLine ( e/h );
         Console.WriteLine ( i*h );
      }
}
```

```
2
2,5
25
```

Отличие результата во *второй* строке по сравнению с результатом в *первой* строке объясняется работой механизма автоприведения типа в арифметических операциях

Преобразование типов

Любая команда процессора обрабатывает данные только одинакового типа. При этом в выражение могут быть объединены любое число операндов любого типа. Одновременное существование этих двух «взаимоисключающих» правил обеспечивается механизмом приведения типов

автоприведение типа в операции **присваивания**

автоприведение типа в **двуместной** арифметической операции

принудительное преобразование типа с помощью операции ***(тип)***

Расширяющее преобразование – в сторону повышения типа Выполняется неявно (implicit)

Сужающее преобразование – в сторону понижения типа Выполняется явно (explicit)

Гарантировано от потери данных

Возможна частичная или полная потеря данных

Автоприведение типа в операции присваивания

ОперандПриёмник = ОперандИсточник

Присваивание (=) сводится к копированию данных в место расположения ОперандаПриёмника

В случае, если типы не совпадают,выполняется неявное приведение, всегда в сторону повышения типа

Или значение выражения (соответствующего типа)

Для преобразования в сторону понижения типа используется операция явного приведения типа:

ОперандПриёмник = *(тип)* ОперандИсточник

Обычно операция явного приведения используется для преобразования в сторону понижения типа

Операция явного приведения типа

(тип) ОперандИсточник

Результатом операции является значение ОперандаИсточника, преобразованное к данному типу

Любой встроенный числовой тип может быть преобразован также к любому встроенному числовому типу, но результат может получиться с потерей данных

```
using System;
class Primer
{
    static void Main()
    {
        byte e;
        short g = 256;
        float h;
        h = g;
        e = (byte) g;
        Console.WriteLine ( g );
        Console.WriteLine ( h );
        Console.WriteLine ( e );
    }
}
```

автоприведение типа

явное приведение типа

```
256
256
0
```

Приведение типа в двуместных арифметических операциях

Операнд1 @ Операнд2

Типы операндов совпадают

Результатом операции является значение данного типа

Типы операндов не совпадают

1) Выполняется преобразование значения операнда младшего типа к старшему типу

2) Выполняется операция @ в старшем типе

3) Результатом операции формируется, как значение в старшем типе

```
using System;

    class Program
    {
        static void Main()
        {
            Console.WriteLine( 7 / 5 );
            Console.WriteLine( 7 / 5.0 );
            Console.WriteLine( 7 / (float)5 );
        }
    }
```

```
1
1,4
1,4
```

Операции замещения

Операнд1 @= Операнд2

Операнд1 =Операнд1 @Операнд2

operand1 @1= operand2 @2 operand3

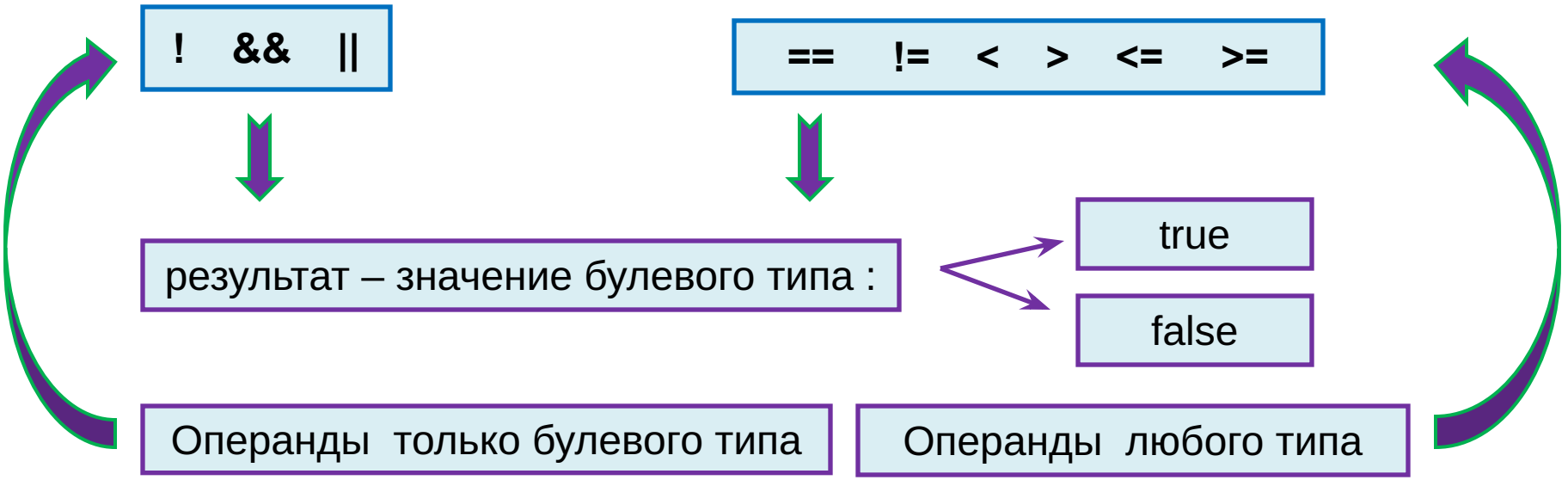
operand1 = operand1 @1 (operand2 @2 operand3)

```
using System;
class Primer9
{
    static void Main()
        {
short e = 10;
int g = 1;
float h = 27f;
e *= 25 + 14 ;
g <<= 4 + e / (e + 1);
h %= g;
Console.WriteLine ( e );
Console.WriteLine ( g );
Console.WriteLine ( h );
        }
}
```

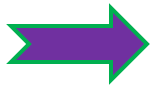
операция замещения, вне зависимости от старшинства остальных операций выражения, выполняется **последней**

```
390
16
11
```

Логические операции и операции отношения




```
static void Main()
  {
   bool  e,g ;
   Console.WriteLine( 10>>2 <= 5);
   Console.WriteLine( 10%3 > 10%2 || 10%4 > 10%5 );
   e = 10 >= 10/3*3;
   g = !e;
   Console.WriteLine ( e );
   Console.WriteLine ( g );
   Console.WriteLine (e && g );
   }
```



```
True
True
True
False
False
```

Операции отношения и логические операции используются в операторах управления

Операции инкремента и декремента

Момент выполнения данных операций зависит от формы их реализации

@operand *(префиксная)*
- значение объекта изменяется **до** его использования

operand@ *(постфиксная)*
- значение объекта изменяется **после** его использования

В качестве операнда выражения

использование

В качестве аргумента метода

```
static void Main()
    {
       int i = 1, j = 2;
       Console.WriteLine ( i++ );
       Console.WriteLine ( ++i );
       Console.WriteLine ( ++i );
       j = ++j * ( i++ - 1 );
       Console.WriteLine ( i );
       Console.WriteLine ( j );
     }
```

```
1
3
4
5
9
```

Символические константы

- предназначены для размещения значений, которые остаются неизменными на всём времени выполнения программы

- имеют имена, как объекты (переменные) встроенных типов

Возможность ассоциирования имени с содержимым константы

Возможность оперативного изменения значения константы (достаточно изменить инициализирующую константу в операторе объявления, вместо того, чтобы искать по тексту программы операторы, в которых использовалось изменяемое число

Объявление символических констант

начинается оператор объявления модификатором const

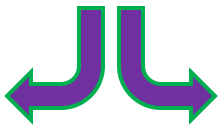
обязательной является начальная инициализация

`const тип имя = значение;` ⟷ `const имя = значение;`

по умолчанию `int`

Перечислимые *(или перечисляемые)* константы

это объекты, полученные в результате объединения некоторого количества символических констант



наиболее простой по конструкции, но уже агрегированный объект

В следующем примере для перечисления используется тип по умолчанию **int**

```
using System;
class Primer
 {              // объявление перечислений д.б. вне тела метода
                // начальное значение по  умолчанию  0
    enum Друзья { Александр, Игорь, Константин, Ярослав};//
    enum Клавиши { F1=59,  F2, F5 = 63, F6, F9 = 67};
    static void Main()
       {
         Console.WriteLine("{0}={1:d}", Друзья.Константин, Друзья.Константин);
         int i = (int) Друзья.Игорь + (int) Друзья.Ярослав + (int) Клавиши.F6;
         Console.WriteLine(i);
       }
  }
```

```
Константин = 2
68
```

Операция выбора

```
using System;
class Primer
{    static void Main()
      {
         int  j ;
         bool b = true || false && true;
         j = b ? 10 : -10;
         Console.WriteLine(j);
      }
}
```

operand1 ? operand2 : operand3;

10

```
using System;
class Primer
{  static void Main()
  {    int  i, j, k;
       string s;
       Console.WriteLine("Задайте первое число!");
       s = Console.ReadLine();
       i = Convert.ToInt32(s);
       Console.WriteLine("Задайте второе число!");
       s = Console.ReadLine();
       j = Convert.ToInt32(s);
       k = i > j ? i : j;
       Console.WriteLine("Результат =  {0}",k);
  }
}
```

Обычно **operand1** представляет собой логическое выражение.
Вопрос: что является результатом этого примера ?

Введение в тип *string*

string – это встроенный системный тип для работы с текстовой информацией.

Объект этого типа является ***ссылочным***, но объявляется, как простой ***размерный*** объект, и использован может быть также, как простой объект.

Содержимым строки является последовательность юникодов.

При объявлении объект типа string может быть инициализирован строковой константой.

Вариантом строковой константы может быть т.н. «строковый литерал»

Пример «замороженная строка»

```
using System;
class Primer
{
    static void Main()
    {
        string str = @"// копирующий строковый литерал
Строка1
  Строка2
    Строка3";
        Console.WriteLine(str);
    }
 }
```

```
Строка1
  Строка2
    Строка3
```

Введение в тип *string (продолжение)*

С объектами типа **string** часто используют такие операции «**+**» и «**[ ]**».

Операция «**[ ]**» выполняет индексацию *(доступ к элементу строки по номеру)*.

Операция «+» выполняет **конкатенацию** *(т.е. слияние)* двух или более строк:

```
using System;
class Primer
{
        static void Main()
        {
            string s = "Simple", t = "String", u="";
            u += s[1];
            u += s[5];
            u += t[3];
            Console.WriteLine(u);
            s += " ";
            s += t;
            Console.WriteLine(s);
        }
}
```

```
iei
Simple String
```

Операция присваивания

По приоритету эта операция самая младшая и может сопровождаться расширяющим автоприведением типов. **Результат** операции присваивания и по типу, и по значению соответствует ***операнду-приёмнику***.
Если операция присвоения в выражении не одна, в ней должны участвовать либо **операнды** одного типа, либо такие, чтобы последовательные присвоения выполнялись только с повышением типа, либо тип приводится явно:

```
using System;
class Primer
{  static void Main()
   {
      float a;
      int b;
      short c;
      Console.WriteLine(a = b = c = (short) 12.5 );
   }
}
```

a = 12

Конец первой части !