

Состав языка C#

Тема 2





Вопросы

1. Алфавит
2. Лексемы
 - Идентификаторы
 - Ключевые слова
 - Константы
 - Разделители
 - Комментарии
- Типы данных
 - Классификация
 - Типы значения и типы ссылки
 - Встроенные типы данных
 - Переменные
 - Выражения
 - Операции
 - Явное и неявное преобразование типов
- Консольный ввод и вывод

Элементы

Язык

Естественный язык:

- символы 
- слова 
- словосочетания 
- предложения 

программирования:

- символы
- лексемы
- выражения
- операторы

1. Алфавит языка (символы)

- прописные и строчные латинские буквы и знак подчеркивания;
- арабские цифры от 0 до 9;
- специальные знаки “{ }, | [] () + - / % * . \ ' : ; & ? < > = ! # ^
- пробельные символы (пробел ' ', символ табуляции '\t')
- символы перехода на новую строку '\n'.

2. Лексемы языка

- **Идентификаторы** – имена объектов C# программ.
 - 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.
 - Разрешается использовать буквы национальных алфавитов.

Нотации

- Нотация — соглашение о правилах создания имен.
 - **Нотация Паскаля**: каждое слово, составляющее идентификатор, начинается с прописной буквы: `MaxLength`, `PosCount`.
 - **Венгерская нотация**: наличие префикса, соответствующего типу величины, например, `iMaxLength`, `iPosCount`.
 - **Нотация Camel**: с прописной буквы начинается каждое слово, составляющее идентификатор, кроме первого, например, `maxLength`, `posCount`.
 - Можно разделять слова, составляющие имя, знаками подчеркивания: `max_length`, `pos_count`, при этом все составные части начинаются со строчной буквы

Лексемы языка

- **Ключевые (зарезервированные) слова** – это слова, которые имеют специальное значение для компилятора.
- Их нельзя использовать в качестве идентификаторов.
- **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.
- Символы, составляющие знак операций, могут быть как специальными (&&, | , <) так и буквенными (as , new).

Лексемы языка

- **Константы** – это неизменяемые величины.
- Существуют логические, целые, вещественные, символьные и строковые константы, а также константа null.
- 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

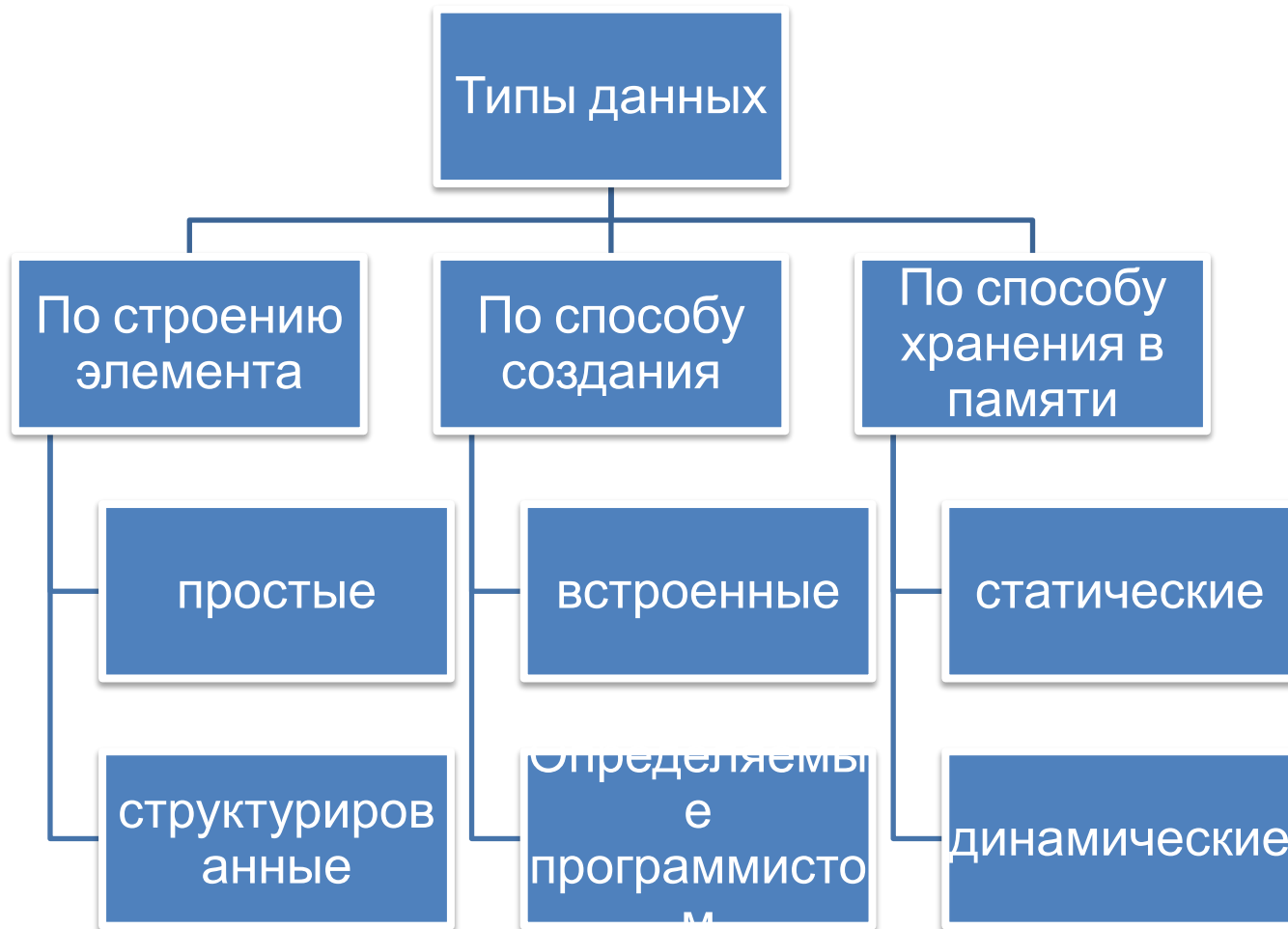
Комментарии

- Предназначены для записи пояснений к программе и формирования документации.
 - Однострочный комментарий
 - // пример комментария
 - Многострочный комментарий
 - /* пример многострочного комментария*/

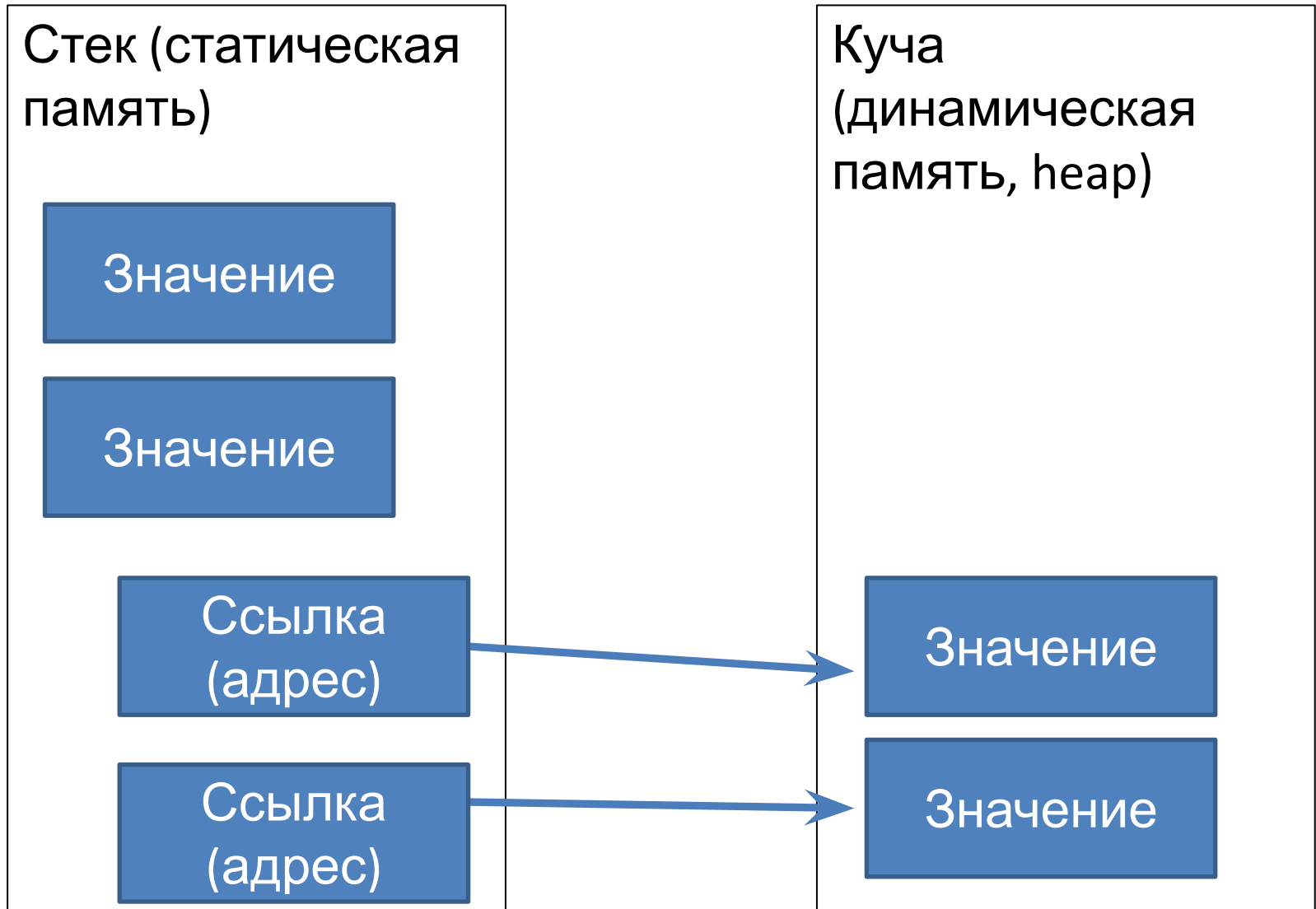
Типы данных в C#

- Тип данных однозначно определяет:
 - внутреннее представление данных, а следовательно, и множество их возможных значений;
 - допустимые действия над данными (операции и функции).

Классификация типов



Типы-значения и типы-ссылки



Встроенные типы

Название	Ключевое слово	Тип .NET (CTS)	Диапазон	Размер (биты)	Описание
Логический	bool	Boolean	true, false		
Целый	sbyte	SByte	-128..127	8	со знаком
	byte	Byte	0..255	8	без знака
	short	Int16	-32768..32767	16	со знаком
	ushort	UInt16	0..65535	16	без знака

Встроенные типы

Название	Ключевое слово	Тип .NET (CTS)	Диапазон	Размер (биты)	Описание
Целый	int	Int32	$-2 \cdot 10^9 \dots 2 \cdot 10^9$	32	со знаком
	uint	UInt32	$0 \dots 4 \cdot 10^9$	32	без знака
	long	Int64	$-9 \cdot 10^{18} \dots 9 \cdot 10^{18}$	64	со знаком
	ulong	UInt64	$0 \dots 18 \cdot 10^{18}$	64	без знака

Встроенные типы

Название	Ключевое слово	Тип .NET (CTS)	Диапазон	Размер (биты)	Описание
Символьный	char	Char	U+0000 .. U+ffff	16	Unicode - СИМВОЛ
Вещественные	float	Single	$1.5 \cdot 10^{-45}$.. $3.4 \cdot 10^{38}$	32	
	double	Double	$5.0 \cdot 10^{-324}$.. $1.7 \cdot 10^{308}$	64	

Встроенные типы

Название	Ключевое слово	Тип .NET (CTS)	Диапазон	Размер (биты)	Описание
Финансовый	decimal	Decimal	$1.0 \cdot 10^{-28} \dots 7.9 \cdot 10^{28}$	128	для денежных вычислений
Строковый	string	String	длина зависит от объема памяти		строка из Unicode - символов
тип object	object	Object	может хранить все что		всеобщий предок

Константы

Название	Определение	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 плавающей точкой	[цифры][.][цифры]E e[+ -][цифры] [суффикс]	0.5e5, .11e-5, 5E3 0.5e5d, .11e-5f, 5E3d

Константы

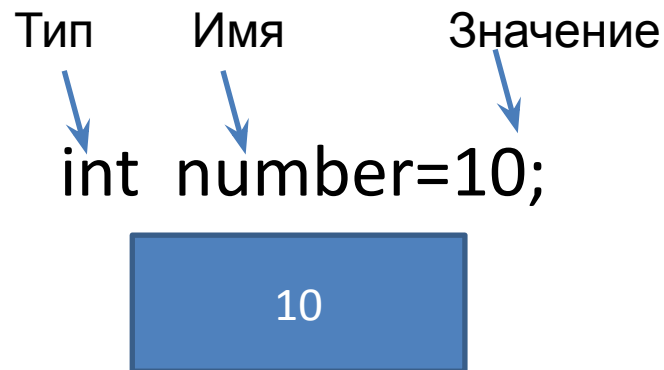
Название	Определение	Примеры
Логическая константа	Истина (true) или ложь (false)	true false
Десятичная константа	Последовательность десятичных цифр, за которой могут следовать символы U/u (unsigned) и/или L/l (long)	8, 0, 192345 8u, 1045l, 34lu, 123UL
Шестнадцатеричная константа	Последовательность шестнадцатеричных цифр, которым предшествуют символы 0x или 0X, за цифрами могут следовать символы U/u (unsigned) и/или L/l (long)	0xA, 0X00F, 0x123 0x1AFLU, 0XFFu

Константы

Название	Определение	Примеры
Символьная константа	Символ, заключенный в апострофы.	<ul style="list-style-type: none">•символ, имеющий графическое представление : 'A', '5', '*', 'ю' ;•управляющая последовательность : '\0', '\n';•символ в виде шестнадцатеричного кода : '\xF', '\x74';•символ в виде escape-последовательности Unicode : '\uA81B'.
Строковая константа	<ol style="list-style-type: none">1. Последовательность символов, заключенная в кавычки.2. Дословные литералы.	<pre>"\nНовая строка", "\nНовый курс\ Программирование\ @" "Новый курс" "Программирование"</pre>

Переменные

- **Переменная** – именованная область памяти, в которой хранятся данные определенного типа.
- 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.



Переменные

- С# программа состоит из классов.
- Класс содержит поля (данные) и методы (функции).
- Переменная, описанная в методе класса, называется **локальной**.

```
class Program
{
    static void Main()
    {
        int x=5;
        ...
    }
    static void Func()
    {
        int x=10;
        ...
    }
}
```

Переменные

- Блок — это код, заключенный в фигурные скобки. Основное назначение блока — группировка операторов.
- **Область действия переменной**, то есть область программы, где ее можно использовать, начинается в точке описания и длится до конца блока, внутри которого она описана.
- Область действия распространяется на вложенные блоки.
- Имя переменной должно быть **уникальным** в области ее действия.

```
int number;  
bool ok;  
do  
{  
    string buf= Console.ReadLine();  
    ok= Int32.TryParse (buf, out number);  
}  
while(!ok);
```

Пример 1

```
namespace pr1
{
    class Program
    {
        //с ошибками!!!
        static void Main(string[] args)
        {
            int x, y, z;
            z = x + y;
        }
    }
}
```

Пример 2

```
namespace pr2
{
    class Program
    {
        //с ошибками!!!
        static void Main(string[] args)
        {
            int x, y, z;
            Console.WriteLine("Введите
                число");
            x = int.Parse(Console.ReadLine());
            Console.WriteLine("Введите
                число");
            y = int.Parse(Console.ReadLine());
```

```
z = x + y;
Console.WriteLine($"{x}+{y}={z}");
for ( int x = 0; x < 10; x++)
{
    Console.WriteLine("Введите
        число");
    y = int.Parse(Console.ReadLine());
    z = x + y;
    Console.WriteLine($"{x}+{y}={z}");
}
Console.WriteLine($"{x}+{y}={z}");
} //Main
} //Program
} //namespace
```


Пример 3

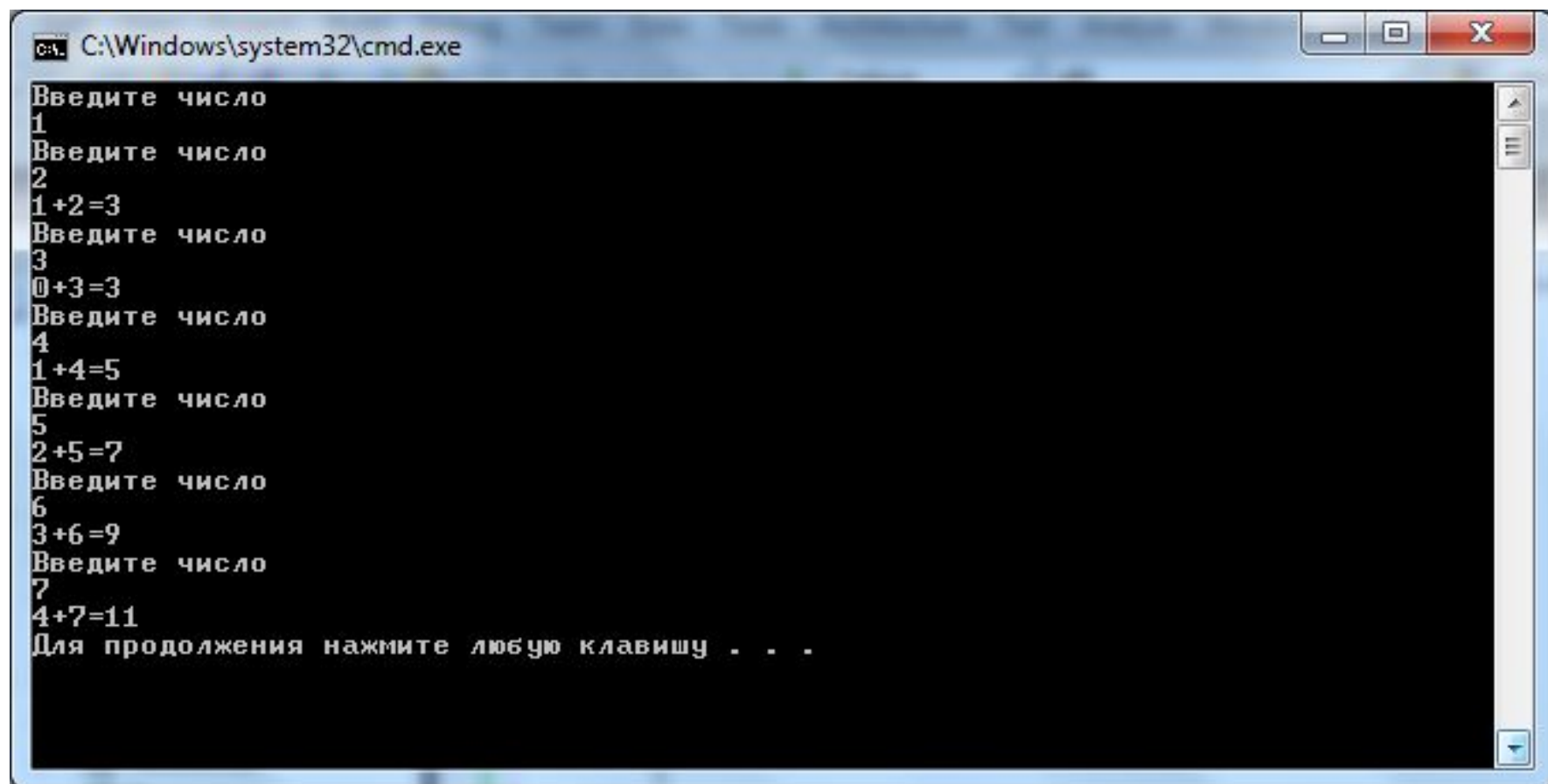
```
namespace pr3
{
    class Program
    {
        //с ошибками!!!
        static void Main(string[] args)
        {
            int x, y, z;
            Console.WriteLine("Введите
число");
            x = int.Parse(Console.ReadLine());
            Console.WriteLine("Введите
число");
            y = int.Parse(Console.ReadLine());
```

```
z = x + y;
Console.WriteLine($"{x}+{y}={z}");
for (x = 0; x < 5; x++)
    {
        Console.WriteLine("Введите
число");
        int w = int.Parse(Console.ReadLine());
        z = x + w;
        Console.WriteLine($"{x}+{w}={z}");
    }
    Console.WriteLine($"{x}+{w}={z}");
}
}
```

Пример 4

```
namespace pr4
{
    class Program
    {
static void Main(string[] args)
{
int x, y, z;
    Console.WriteLine("Введите
        число");
    x = int.Parse(Console.ReadLine());
    Console.WriteLine("Введите
        число");
    y = int.Parse(Console.ReadLine());
```

```
z = x + y;
Console.WriteLine($"x+y={z}");
for (x = 0; x < 5; x++)
    {
        Console.WriteLine("Введите число");
        int w =int.Parse(Console.ReadLine());
        z = x + w;
        Console.WriteLine($"x+w={z}");
    }
}
```



C:\Windows\system32\cmd.exe

```
Введите число  
1  
Введите число  
2  
1+2=3  
Введите число  
3  
0+3=3  
Введите число  
4  
1+4=5  
Введите число  
5  
2+5=7  
Введите число  
6  
3+6=9  
Введите число  
7  
4+7=11  
Для продолжения нажмите любую клавишу . . .
```

Именованные константы

- Можно запретить изменять значение переменной, задав при ее описании ключевое слово **const**.
- Именованные константы должны обязательно инициализироваться при описании.
- При инициализации можно использовать не только константу, но и выражение

```
const int b = 1;
```

```
// const распространяется на обе переменные
```

```
const float a = 0.1f, y = 0.1f;
```

```
const int b = 1, a = 100;
```

```
const int x = b * a + 25;
```

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**.

Основные операции

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.

Основные операции

Операция	Описание
Унарные операции	
<code>typeof</code>	Получение типа
<code>-</code>	Унарный минус
<code>+</code>	Унарный плюс
<code>!</code>	Логическое отрицание (НЕ).
<code>new</code>	Выделение памяти
<code>(тип)</code>	Преобразование типа

Основные операции

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)

Основные операции

Бинарные операции

Операции отношения и проверки типа

<	меньше, чем
<=	меньше или равно
>	больше
>=	больше или равно
is	проверка принадлежности типу
as	приведение типа

Основные операции

Операции сравнения

==

равно

!=

не равно

Логические операции

&&

конъюнкция (И) целочисленных операндов или отношений, целочисленный результат ложь или истина

||

дизъюнкция (ИЛИ) целочисленных операндов или отношений, целочисленный результат ложь или истина

Тернарная

?:

Условная операция
 $x < 0 ? -x : x ;$

Основные операции

Операции сравнения

Присваивание

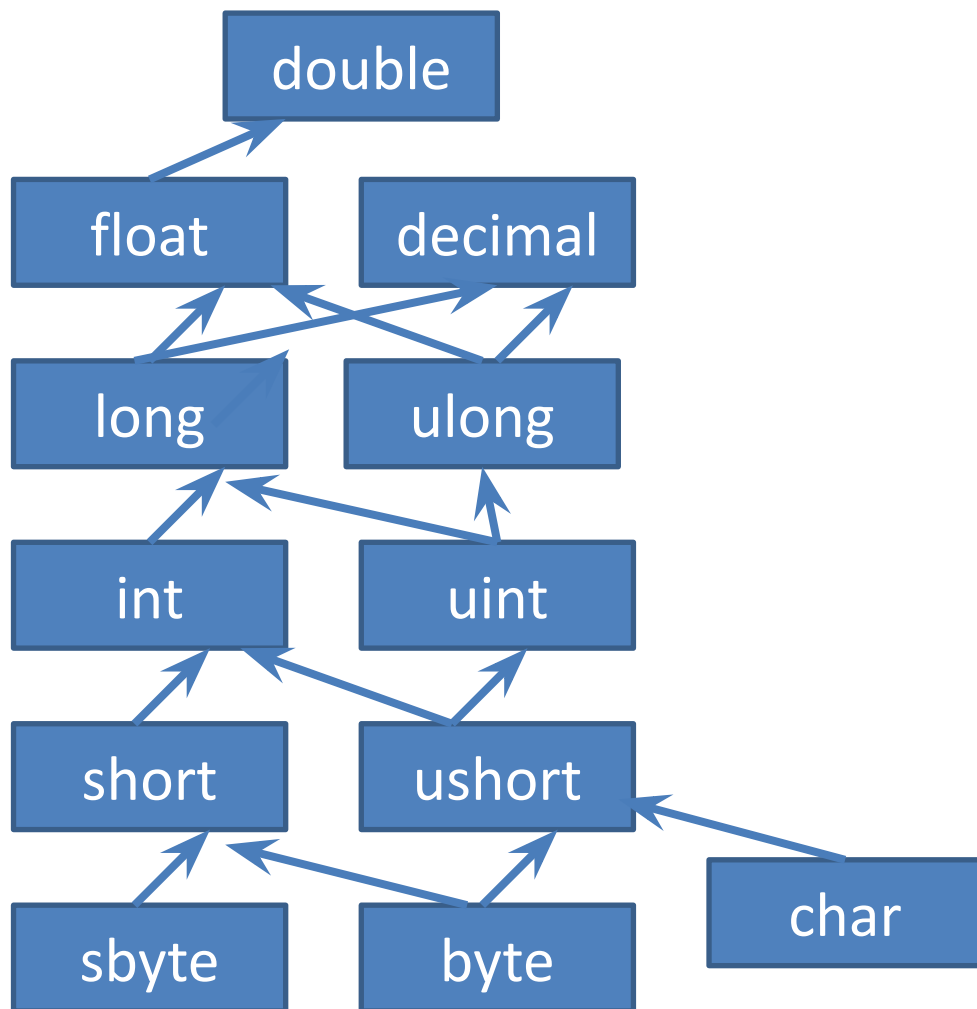
=	присваивание
*=	умножение с присваиванием (мультипликативное присваивание)
/=	деление с присваиванием
%=	деление с остатком с присваиванием
+=	сложение с присваиванием
-=	вычитание с присваиванием

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)

Неявное преобразование ТИПОВ

- Неявное преобразование типов возможно только, если не происходит потеря значимости

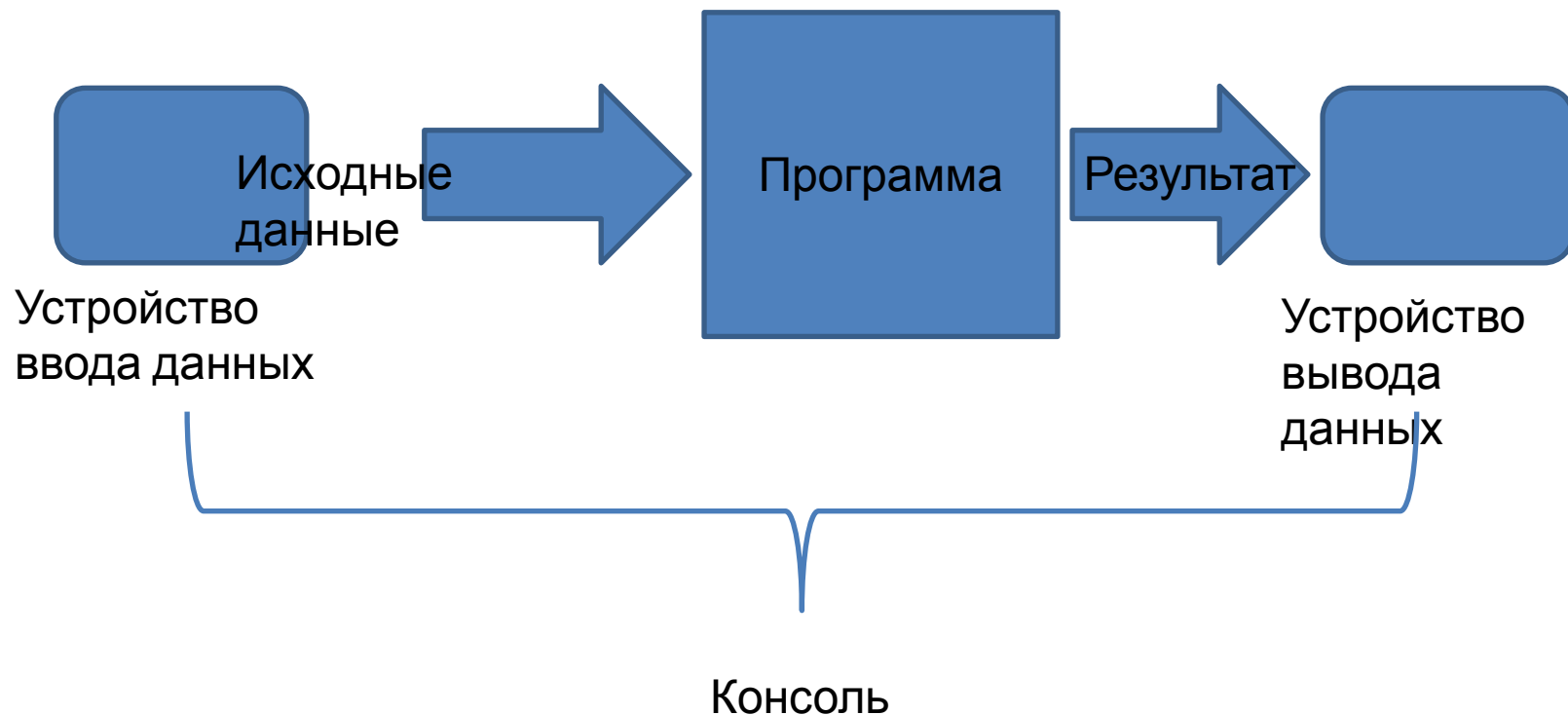


Операция явного преобразования типа

- Явное преобразование типа:
(тип) выражение

```
long b=300; //8 байт  
int a=(int)b; //4 байта  
byte c=(byte)a; //1 байт
```

Консольный ввод и вывод



Вывод

`Console.Write(string s)`

`Console.WriteLine(string s)`

```
Console.WriteLine(x);
```

```
Console.WriteLine(x.ToString());
```

```
Console.WriteLine("x="+x);
```

```
Console.WriteLine("x={0},  
y={1}", x,y);
```

```
Console.WriteLine($"x={x},  
y={y}");
```


Пример

```
int x = 5;
```

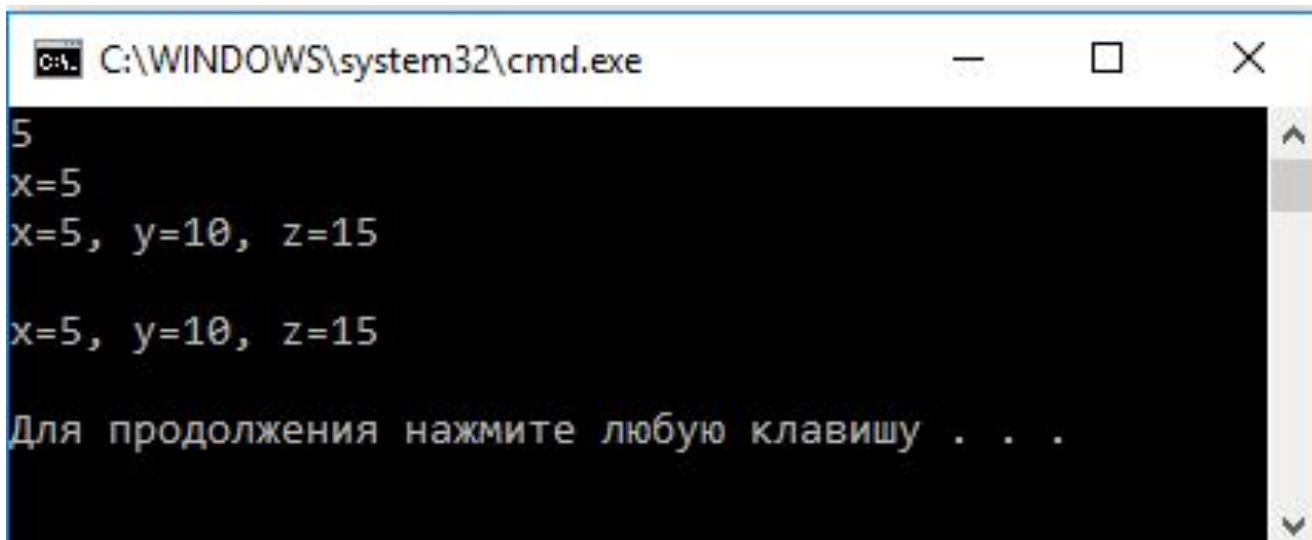
```
Console.WriteLine(x);
```

```
Console.Write("x=" + x + "\n");
```

```
int y = 10, z = 15;
```

```
Console.WriteLine("x={0}, y={1}, z={2}\n", x, y, z);
```

```
Console.WriteLine($"x={x}, y={y}, z={z}\n");
```



```
C:\WINDOWS\system32\cmd.exe
5
x=5
x=5, y=10, z=15

x=5, y=10, z=15

Для продолжения нажмите любую клавишу . . .
```

Ввод

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

Пример

```
Console.WriteLine("Введите строку");  
string s = Console.ReadLine();  
Console.WriteLine("s = " + s );
```

```
Console.WriteLine("Введите СИМВОЛ");  
char c = (char)Console.Read();  
Console.ReadLine();  
Console.WriteLine("c = " + c );
```

```
string buf; // строка - буфер  
Console.WriteLine("Введите целое число" );  
buf = Console.ReadLine();  
int i = Convert.ToInt32( buf );
```

Пример

```
Console.WriteLine("Введите  
вещественное число" );  
buf = Console.ReadLine();  
double x =  
    Convert.ToDouble(buf);
```

```
Console.WriteLine("Введите  
вещественное число" );  
buf = Console.ReadLine();  
double y = double.Parse( buf );  
  
double z;  
bool ok=double.TryParse(  
    Console.ReadLine(), out z));
```

Примеры операций: инкремент и декремент

```
int m;  
int n;  
Console.WriteLine("Введите целое число");  
string buf = Console.ReadLine();  
m = Convert.ToInt32(buf);
```

```
Console.WriteLine("Введите целое число");  
buf = Console.ReadLine();  
n = Convert.ToInt32(buf);
```

```
int k = m++ + n;  
Console.WriteLine($"m++ + n={k}, m={m},n={n}");  
bool t = m-- < n;  
Console.WriteLine($"m-- < n={t}, m={m},n={n}");  
t = m < --n;  
Console.WriteLine($"m < --n={t}, m={m},n={n}");
```

Управление консолью

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Нажмите любую клавишу. Для
выхода из программы нажмите Escape");
        ConsoleKeyInfo key;
        do
        {
            key = Console.ReadKey(true);
            Console.WriteLine($"Нажата клавиша
{key.Key.ToString()}");
        } while (key.Key != ConsoleKey.Escape);
    }
}
```

Управление консолью

```
static void Main(string[] args)
{
    Console.BufferHeight = 400;// 300 строк по
    умолчанию
    ConsoleColor old = Console.ForegroundColor;
    for (int i = 1; i < 350; i++)
    {
        if (i % 5 == 0) Console.ForegroundColor =
        ConsoleColor.Red;
        Console.WriteLine(i);
        Console.ForegroundColor = old;
    }
}
```