Лекция 2
Основные понятия языка

1. Состав языка. Алфавит, лексемы, константы
2. Типы данных

1. Состав языка. Алфавит, лексемы, константы

Язык программирования можно уподобить очень примитивному иностранному языку с жесткими правилами без исключений. Изучение иностранного языка обычно 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.

Примерно так же поступим и мы при изучении языка C#.

Алфавит и лексемы. Все тексты на языке пишутся с помощью его алфавита. Например, в русском языке один алфавит (набор символов), а в албанском – другой.

В C# используется **кодировка символов `Unicode`**.

Давайте разберемся, что это такое.

Компьютер умеет работать только с числами, и для того чтобы можно было хранить в его памяти текст, требуется определить, каким числом будет представляться (кодироваться) каждый символ.

Соответствие между символами и кодирующими их числами называется ***кодировкой***, или ***кодовой таблицей*** (character set).

Существует множество ***различных кодировок символов***.

Например, в Windows часто используется ***кодировка ANSI***, а конкретно – ***CP1251***. Каждый символ представляется в ней одним байтом (8 бит), поэтому в этой кодировке можно одновременно задать только 256 символов.

В *первой половине* кодовой таблицы находятся латинские буквы, цифры, знаки арифметических операций и другие распространенные символы.

Вторую половину занимают символы русского алфавита. Если требуется представлять символы другого национального алфавита (например, албанского), необходимо использовать другую кодовую таблицу.

Кодировка ***Unicode*** позволяет представить символы всех существующих алфавитов одновременно, что коренным образом улучшает переносимость текстов.

Каждому символу соответствует свой уникальный код. Естественно, что при этом для хранения каждого символа требуется больше памяти. Первые 128 Unicode-символов соответствуют первой части кодовой таблицы ANSI.

Алфавит C# включает:

- буквы (латинские и национальных алфавитов) и символ подчеркивания (_), который употребляется наряду с буквами;
- цифры,
- специальные символы, например, **+**, *****, **{** и **&**;
- пробельные символы (пробел и символы табуляции);
- символы перевода строки.

Из символов составляются более крупные строительные блоки: лексемы, директивы препроцессора и комментарии.

Лексема (token – часто это слово ленятся переводить и пишут просто «токен») – это минимальная единица языка, имеющая самостоятельный смысл.

Существуют следующие виды лексем:

- имена (идентификаторы);
- ключевые слова;
- знаки операций;
- разделители,
- литералы (константы).

Лексемы языка программирования аналогичны словам естественного языка.

Пример. Лексемами являются:

-число 128 (но не его часть 12),

- имя Vasia,

- ключевое слово goto и

- знак операции сложения +.

Далее мы рассмотрим лексемы подробнее.

Директивы препроцессора пришли в C# из его предшественника – языка C++.

Препроцессором называется предварительная стадия компиляции, на которой формируется окончательный вид исходного текста программы.

Например, с помощью ***директив*** (инструкций, команд) ***препроцессора*** можно включить или выключить из процесса компиляции фрагменты кода. Директивы препроцессора не играют в C# такой важной роли, как в C++. Мы рассмотрим их в свое время.

Комментарии предназначены для записи пояснений к программе и формирования документации. Правила записи комментариев описаны далее.

Из лексем составляются выражения и операторы. Выражение задает правило вычисления некоторого значения. Например, выражение a + b задает правило вычисления суммы двух величин.

Примечание – Компилятор выполняет перевод программы в *исполняемый файл на языке* ***IL*** за две последовательные фазы: лексического и синтаксического анализа. При лексическом анализе из потока символов, составляющих исходный текст программы, выделяются лексемы (по другим лексемам, разделителям, пробельным символам и переводам строки). На этапе синтаксического анализа программа переводится в исполняемый код. Кроме того, компилятор формирует сообщения о синтаксических ошибках.

Оператор задает законченное описание некоторого действия, данных или элемента программы. Например: `int a;`

Это – оператор описания целочисленной переменной **a**.

Идентификаторы

Имена в программах служат той же цели, что и имена в мире людей, – чтобы обращаться к программным объектам и различать их, то есть идентифицировать.

Поэтому имена также называют ***идентификаторами***.

В идентификаторе могут использоваться ***буквы, цифры и символ подчеркивания***. Прописные и строчные буквы различаются, например, sysop, SySoP и SYSOP – три разных имени.

Первым символом идентификатора может быть буква или знак подчеркивания, но не цифра. Длина идентификатора не ограничена. Пробелы внутри имен не допускаются.

В идентификаторах C# разрешается использовать помимо латинских букв буквы национальных алфавитов.

Пример. **Пёсик** или **зз** являются правильными идентификаторами.

Более того, в идентификаторах можно применять даже так называемые ***escape-последовательности Unicode***, то есть представлять символ с помощью его кода в шестнадцатеричном виде с префиксом **\u**, например, \u00F2.

Примечание – Примеры неправильных имен:

21ate, Big gig, Б#г;

первое начинается с цифры, второе и третье содержат недопустимые символы (пробел и #).

Имена даются элементам программы, к которым требуется обращаться: ***переменным***, ***типам***, ***константам***, ***методам***, ***меткам*** и т. д.

Идентификатор создается на этапе объявления переменной (метода, типа и т. п.), после этого его можно использовать в последующих операторах программы. При выборе идентификатора необходимо иметь в виду следующее:

□ *идентификатор не должен совпадать с ключевыми словами*;

Примечание – Впрочем, для использования ключевого слова в качестве идентификатора его достаточно предварить символом. Например, правильным будет идентификатор @if.

□ *не рекомендуется начинать идентификаторы с двух символов подчеркивания, поскольку такие имена зарезервированы для служебного использования*.

Для улучшения читабельности программы следует давать объектам осмысленные имена, составленные в соответствии с определенными правилами. Понятные и согласованные между собой имена – основа хорошего стиля программирования.

Существует несколько видов так называемых ***нотаций – соглашений о правилах создания имен***.

1. В *нотации* ***Паскаля*** каждое слово, составляющее идентификатор, начинается с прописной буквы, например,

`MaxLength`, **`MyFuzzyShooshpanchik`**.

2. Венгерская *нотация* (ее предложил венгр по национальности, сотрудник компании Microsoft) отличается от предыдущей наличием префикса, соответствующего типу величины, например,

`iMaxLength`, **`lpfnMyFuzzyShooshpanchik`**.

3. Согласно *нотации* ***Camel***, с прописной буквы начинается каждое слово, составляющее идентификатор, кроме первого, например, **`maxLength`**, **`myFuzzyShooshpanchik`**. Человеку с богатой фантазией абрис имени может напоминать верблюда, откуда и произошло название этой нотации.

4. Еще одна традиция – разделять слова, составляющие имя, знаками подчеркивания: **`max_length`**, **`my_fuzzy_shooshpanchik`**, при этом все составные части начинаются со строчной буквы.

В C# для именования различных видов программных объектов чаще всего используются две нотации: ***Паскаля*** и ***Camel***. Многобуквенные идентификаторы в примерах соответствуют рекомендациям, приведенным в спецификации языка. Кроме того, в примерах для краткости часто используются однобуквенные имена. В реальных программах такие имена можно применять только в ограниченном наборе случаев.

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 C# приведен в таблице на следующем слайде.

Знаки операций и разделители

Знак операции – это один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются.

Пример. В выражении `a += b`

знак `+=` является знаком операции, а `a` и `b` – операндами.

Символы, составляющие знак операций, могут быть:

- специальными, например, `&&`, `|` и `<`,

- буквенными, такими как `as` или `new`.

Операции делятся на **унарные**, **бинарные** и **тернарную** по количеству участвующих в них операндов. Один и тот же знак может интерпретироваться по-разному в зависимости от контекста.

Все знаки операций, за исключением `[]`, `()` и `?:`, представляют собой отдельные лексемы.

Таблица 1 – Ключевые слова C#

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

Разделители используются для разделения или, наоборот, группирования элементов.

Примеры разделителей: скобки, точка, запятая.

Перечислим все знаки операций и разделители, использующиеся в C#:

```
{}[]().,:; + -*/ ‘ * & | A ! ~ =
<>?-++ __ && || « » == {=<=>=+= -= *= /= %=
&= |= A= «= »= ->
```

Литералы

Литералами, или константами, называют ***неизменяемые величины***.

В C# есть **логические**, **целые**, **вещественные**, **символьные** и **строковые** константы, а также константа `null`.

Компилятор, выделив константу в качестве лексемы, относит ее к одному из типов данных по ее внешнему виду. Программист может задать тип константы и самостоятельно.

Описание и примеры констант каждого типа приведены в таблице 2. Примеры, иллюстрирующие наиболее часто употребляемые формы констант, выделены полужирным шрифтом (вначале можно обратить внимание только на них).

Таблица 2 – Константы в C#

Константа	Описание	Примеры
Логическая	`true` (истина) или `false` (ложь)	`true, false`

Целая	*Десятичная*: последовательность десятичных цифр (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), за которой может следовать суффикс (`U, u, L, 1, UL, Ul, uL, ul, LU, Lu, 1U, lu`)
	Шестнадцатеричная: символы `0x` или `0X`, за которыми следуют шестнадцатеричные цифры (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F), а за цифрами, в свою очередь, может следовать суффикс (`U, u, L, 1, UL, Ul, uL, ul, LU, Lu, Ш, lu`)
		`8 0 199226`
		`8u 0Lu 199226L`
		`0xA 0x1B8`
		`0X00FF`
		`0xAU 0x1B8LU`
		`0X00FF1`

Вещественная	*С фиксированной точкой*:
	`[цифры] [.] [цифры] [суффикс]`
	Суффикс – один из символов `F, f, D, d, M, m`
	С порядком:
	`[цифры][.][цифры]{E|e}[+|-]`
	`[цифры][суффикс]`
	Суффикс – один из символов
	`F, f, D, d, M, m`
		`5.7 .001 35`
		`5.7F .00ld 35`
		`5F .00lf 35m`
		`0.2E6 .lle+3`
		`5E-10`
		`0.2E6D .lle-3`
		`5E10`

Константа `null`	Ссылка, которая не указывает ни на какой объект	`null`

Константа	Описание	Примеры
Символьная	Символ, заключенный в апострофы	'A' 'ю' '\0* An' ' \xF' '4x74' '\uA81B'
Строковая	Последовательность символов, заключенная в кавычки	"Здесь был Vasia" "иЗначение г = \xF5 \n" "Здесь был \u0056\u0061" "C:\\temp\\file1.txt" @"C:\temp\file1.txt"

Примечание – квадратные скобки при описании означают необязательность заключенной в них конструкции.

Рассмотрим константы табл. 2 более подробно.

***Логических* литералов** всего два. Они широко используются в качестве признаков наличия или отсутствия чего-либо.

Целые* литералы** могут быть представлены либо в десятичной, либо в шестнадцатеричной системе счисления, а ***вещественные – только в десятичной системе, но в двух формах: с фиксированной точкой и с порядком. Вещественная константа с порядком представляется в виде мантиссы и порядка. Мантисса записывается слева от знака экспоненты (E или e), порядок – справа от знака. Значение константы определяется как произведение мантиссы и возведенного в указанную в порядке степень числа 10 (например, **1.3e2 = 1,3 • 10^2 = 130**). При записи вещественного числа целая часть может быть опущена, например, **`.1`**.

Примечание – Пробелы внутри числа не допускаются. Для отделения целой части от дробной используется не запятая, а точка. Символ E не представляет собой знакомое всем из математики число e, а указывает, что далее располагается степень, в которую нужно возвести число 10.

Если требуется сформировать отрицательную целую или вещественную константу, то перед ней ставится знак унарной операции изменения знака (-), например:

`-218, -022, -0x3C,  -4.8,-0.1e4.`

Когда компилятор распознает константу, он отводит ей место в памяти в соответствии с ее видом и значением. Если по каким-либо причинам требуется явным образом задать, сколько памяти следует отвести под константу, используются суффиксы, описания которых приведены в табл. 3..

Таблица 3 – Суффиксы целых и вещественных констант

Суффикс	Значение
L, l	Длинное целое (`long`)
U, u	Беззнаковое целое (`unsigned`)
F, f	Вещественное с одинарной точностью (`float`)
D, d	Вещественное с двойной точностью (`double`)
M, m	Финансовое десятичного типа (`decimal`)

Символьная константа – любой символ в кодировке Unicode.

Символьные константы записываются *в одной из четырех форм*, представленных в табл. 2:

- «обычный» символ, имеющий графическое представление (кроме апострофа и символа перевода строки), – **`'AVio', '*';`**
- управляющая последовательность – **`'\0'`**, **`' \n'`**;
- символ в виде шестнадцатеричного кода – **`'\xF'`**, **`'\x74'`**;
- символ в виде escape-последовательности Unicode – **`'\uA81B'`**.

Управляющей последовательностью, или ***простой escape-последовательностью***, называют определенный символ, предваряемый обратной косой чертой.

Управляющая последовательность интерпретируется как одиночный символ и используется для представления:

- кодов, не имеющих графического изображения (например, **\n** – переход в начало следующей строки);
- символов, имеющих специальное значение в строковых и символьных литералах, например, апострофа **'**.

В табл. 4 приведены допустимые значения последовательностей. Если непосредственно за обратной косой чертой следует символ, не предусмотренный таблицей, возникает ошибка компиляции.

Таблица 4 – Управляющие последовательности в C#

Вид	Наименование
\a	Звуковой сигнал
\b, \f	Возврат на шаг, перевод страницы (формата)
\n, \r	Перевод строки, возврат каретки
\t, \v	Горизонтальная, вертикальная табуляция
\\	Обратная косая черта
\', \"	Апостроф, кавычка
\0	Нуль-символ

Символ, представленный в виде шестнадцатеричного кода, начинается с префикса **\0x**, за которым следует код символа. Числовое значение должно находиться в диапазоне от 0 до 2^{16} - 1, иначе возникает ошибка компиляции.

Escape-последовательности Unicode служат для представления символа в кодировке Unicode с помощью его кода в шестнадцатеричном виде с префиксом **\u** или **\U**, например, **\u00F2**, **\U00010011**.

Коды в диапазоне от **\U10000** до \U**10FFFF** представляются в виде двух последовательных символов (коды, превышающие **\U10FFFF**, не поддерживаются).

I. Управляющие последовательности обоих видов могут использоваться и в ***строковых константах***, называемых иначе ***строковыми литералами***.

Пример 1. Если требуется вывести несколько строк, можно объединить их в один литерал, отделив одну строку от другой символами **\n**:

```
"Никто не доволен своей\nвнешностью, но каждый доволен\nсвоим умом"
```

Этот литерал при выводе будет выглядеть так:

```
Никто не доволен своей
внешностью, но каждый доволен
своим умом
```

Пример 2. Если внутри строки требуется использовать кавычку, ее предваряют косой чертой, по которой компилятор отличает ее от кавычки, ограничивающей строку:

```
"Издательский дом \"Питер\""
```

Как видите, строковые литералы с управляющими символами несколько теряют в читабельности, поэтому в C# введен второй вид литералов – дословные литералы (verbatim strings).

II. ***Дословные литералы*** предваряются символом **@**, который отключает обработку управляющих последовательностей и позволяет получать строки в том виде, в котором они записаны. Например, два приведенных выше литерала в дословном виде выглядят так:

```
@"Никто не доволен своей
внешностью, но каждый доволен
своим умом"
@"Издательский дом "Питер"".
```

Чаще всего дословные литералы применяются в регулярных выражениях и при задании полного пути файла, поскольку в нем присутствуют символы обратной косой черты, которые в обычном литерале пришлось бы представлять с помощью управляющей последовательности.

Сравните два варианта записи одного и того же пути:

```
"C: \\app\\bin\\debug\\a.exe"
@"C:\app\bin\debug\a.exe"
```

Строка может быть пустой (записывается парой смежных двойных кавычек ""), пустая символьная константа недопустима.

Константа **`null`** представляет собой значение, задаваемое по умолчанию для величин так называемых *ссылочных типов*, которые мы рассмотрим далее.

Комментарии. Комментарии предназначены для записи пояснений к программе и формирования документации. Компилятор комментарии игнорирует. Внутри комментария можно использовать любые символы. В C# есть два вида комментариев: однострочные и многострочные:

- ***однострочный*** комментарий начинается с двух символов прямой косой черты (*//*) и заканчивается символом перехода на новую строку,
- ***многострочный*** заключается между символами-скобками */** и ***/** и может занимать часть строки, целую строку или несколько строк.

Примечание – Комментарии не вкладываются друг в друга: символы *//* и */** не обладают никаким специальным значением внутри комментария.

- еще одна разновидность комментариев, которые ***начинаются с трех подряд идущих символов косой черты*** (*///*). Они предназначены для формирования документации к программе в формате XML. Компилятор извлекает эти комментарии из программы, проверяет их соответствие правилам и записывает их в отдельный файл.

2. Типы данных

Данные, с которыми работает программа, хранятся в оперативной памяти.

Естественно, что компилятору необходимо точно знать:

- сколько места они занимают,
- как именно закодированы,
- какие действия с ними можно выполнять.
- Все это задается при описании данных с помощью ***типа***.

Тип данных однозначно определяет:

- внутреннее 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; целые числа можно умножать друг на друга, а, например, символы нельзя.

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);
- ***динамическая 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 с помощью специальных команд.

Основным местом для хранения данных в C# является ***хип***.

Классификация типов. Любая информация легче усваивается, если она «разложена по полочкам». Поэтому, прежде чем перейти к изучению конкретных типов языка C#, рассмотрим их классификацию. Типы можно классифицировать по разным признакам.

Если принять за основу ***строение*** элемента, все типы можно разделить на:

- простые (не имеют внутренней структуры)
- структурированные (состоят из элементов других типов).

По своему «***создателю***» типы можно разделить на:

- *встроенные* (стандартные);
- *определяемые программистом*.

Различные классификации типов данных C#

Для данных ***статического типа*** память выделяется в момент объявления, при этом ее требуемый объем известен.

Для данных ***динамического типа*** размер данных в момент объявления может быть неизвестен, и память под них выделяется по запросу в процессе выполнения программы.

Встроенные типы. Не требуют предварительного определения. Для каждого типа существует ключевое слово, которое используется при описании переменных, констант и т.д.

Если же программист определяет собственный тип данных, он описывает его характеристики и сам дает ему имя, которое затем применяется точно так же, как имена стандартных типов. Описание собственного типа данных должно включать всю информацию, необходимую для его использования: внутреннее представление и допустимые действия.

Встроенные типы C# приведены в табл. 5. Они однозначно соответствуют стандартным классам библиотеки .NET, определенным в пространстве имен System.

Таблица 5 – Встроенные типы C#

Название	Ключевое слово	Тип .NET	Диапазон значений	Описание	Размер, бит
Логический тип	`bool`	`Boolean`	true, false		
Целые типы	`sbyte`	`SByte`	От-128 до 127	Со знаком	8
	`byte`	`Byte`	От 0 до 255	Без знака	8
	`short`	`Intl6`	От -32 768 до 32 767	Со знаком	16
	`ushort`	`UIntl6`	От 0 до 65 535	Без знака	16
	`int`	`Int32`	От $-2 \cdot 10^9$ до $2 \cdot 10^9$	Со знаком	32
	`uint`	`UInt32`	От 0 до $4 \cdot 10^9$	Без знака	32
	`long`	`Int64`	От $-9 \cdot 10^{18}$ до $9 \cdot 10^{18}$	Со знаком	64
	`ulong`	`UInt64`	От 0 до $18 \cdot 10^{18}$	Без знака	64

Название	Ключевое слово	Тип .NET	Диапазон значений	Описание	Размер, бит
Символьный тип	`char`	`Char`	От U+0000 до U+ffff	Unicode-символ	16
Вещественные[1]	`float` `double`	`Single` `Double`	От $1.5 \cdot 10^{-45}$ до $3.4 \cdot 10^{38}$ От $5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	7 цифр 15-16 цифр	32
Финансовый тип	`decimal`	`Decimal`	От $1.0 \cdot 10^{-28}$ до $7.9 \cdot 10^{28}$	28-29 цифр	128
Строковый тип	`string`	`String`	Длина ограничена объемом доступной памяти	Строка из Unicode-символов	
Тип `object`	`object`	`Object`	Можно хранить все что угодно	Всеобщий предок	

Как видно из таблицы, существуют несколько вариантов представления целых и вещественных величин. Программист выбирает тип каждой величины, используемой в программе, с учетом необходимого ему диапазона и точности представления данных.

Целые типы, а также символьный, вещественные и финансовый типы можно объединить под названием *арифметических типов*.

Внутреннее представление величины целого типа – целое число в двоичном коде. В знаковых типах старший бит числа интерпретируется как знаковый (0 – положительное число, 1 – отрицательное). Отрицательные числа чаще всего представляются в так называемом дополнительном коде. Для преобразования числа в дополнительный код все разряды числа, за исключением знакового, инвертируются, затем к числу прибавляется единица, и знаковому биту тоже присваивается единица. Беззнаковые типы позволяют представлять только положительные числа, поскольку старший разряд рассматривается как часть кода числа.

Примечание – Если под величину отведено n двоичных разрядов, то в ней можно представить 2^n различных сочетаний нулей и единиц. Если старший бит отведен под знак, то диапазон возможных значений величины – $[-2^{n-1}, 2^n-1 - 1]$, а если все разряды используются для представления значения, диапазон смещается в область положительных чисел и равен $[0, 2^n - 1]$ (см. табл. 5).

Вещественные типы, или типы данных *с плавающей точкой*, хранятся в памяти компьютера иначе, чем целочисленные. Внутреннее представление вещественного числа состоит из двух частей – мантиссы и порядка, каждая часть имеет знак. Длина мантиссы определяет точность числа, а длина порядка – его диапазон. В первом приближении это можно представить себе так: например, для числа $0{,}381 \cdot 10^4$ хранятся цифры мантиссы 381 и порядок 4, для числа $560{,}3 \cdot 10^2$ – мантисса 5603 и порядок 5 (мантисса нормализуется), а число 0,012 представлено как 12 и 1.

Конечно, в этом примере не учтены система счисления и другие особенности.

Все вещественные типы могут представлять как положительные, так и отрицательные числа. Чаще всего в программах используется тип **double**, поскольку его диапазон и точность покрывают большинство потребностей. Этот тип имеют вещественные литералы и многие стандартные математические функции.

Примечание – Обратите внимание на то, что при одинаковом количестве байтов, отводимых под величины типа fl**oat** и **int**, диапазоны их допустимых значений сильно различаются из-за внутренней формы представления. То же самое относится к **long** и **double**.

Тип **decimal** предназначен для денежных вычислений, в которых критичны ошибки округления. Как видно из табл. 2.5, тип float позволяет хранить одновременно всего 7 значащих десятичных цифр, тип double – 15-16. При вычислениях ошибки округления накапливаются, и при определенном сочетании значений это даже может привести к результату, в котором не будет ни одной верной значащей цифры! Величины типа decimal позволяют хранить 28-29 десятичных разрядов.

Тип **decimal** не относится к вещественным типам, у них различное внутреннее представление. Величины денежного типа даже нельзя использовать в одном выражении с вещественными без явного преобразования типа. Использование величин финансового типа в одном выражении с целыми допускается.

Любой встроенный тип C# соответствует стандартному классу библиотеки .NET, определенному в пространстве имен **System**.

Везде, где используется имя встроенного типа, его можно заменить именем класса библиотеки. Это значит, что у встроенных типов данных C# есть методы и поля. С их помощью можно, например, получить минимальные и максимальные значения для целых, символьных, финансовых и вещественных чисел:

- **double.MaxValue** (или **System.Double.MaxValue**) – максимальное число типа **double**;
- **uint.MinValue** (или **System.UInt32.MinValue**) минимальное число типа **uint**.

Примечание – Интересно, что в вещественных классах есть элементы, представляющие положительную и отрицательную бесконечности, а также значение «не число» – это

PositiveInfinity, **NegativeInfinity** и **NaN** соответственно.

При выводе на экран, например, первого из них получится слово «бесконечность». Все доступные элементы класса можно посмотреть в окне редактора кода, введя символ точки сразу после имени типа.

Типы литералов. Как уже говорилось, величин, не имеющих типа, не существует. Поэтому литералы (константы) тоже *имеют тип*. Если значение целого литерала находится внутри диапазона допустимых значений типа **`int`**, литерал рассматривается как **`int`**, иначе он относится к наименьшему из типов **`uint`**, **`long`** или **`along`**, в диапазон значений которого он входит. Вещественные литералы по умолчанию относятся к типу **`double`**.

Например, константа 10 относится к типу **`int`** (хотя для ее хранения достаточно и байта), а константа 2147483648 будет определена как **`uint`**. Для явного задания типа литерала служит суффикс, например, **`1.lf`**, **`1UL`**, **`1000m`** (все суффиксы перечислены в табл. 3). Явное задание применяется в основном для уменьшения количества неявных преобразований типа, выполняемых компилятором.

Типы-значения и ссылочные типы. Чаще всего типы C# разделяют по способу хранения элементов на *типы-значения* и *ссылочные типы*

Элементы типов-значений, или значимых типов (value types), представляют собой просто последовательность бит в памяти, необходимый объем которой выделяет компилятор. Иными словами,

- величины *значимых типов* хранят свои значения непосредственно.

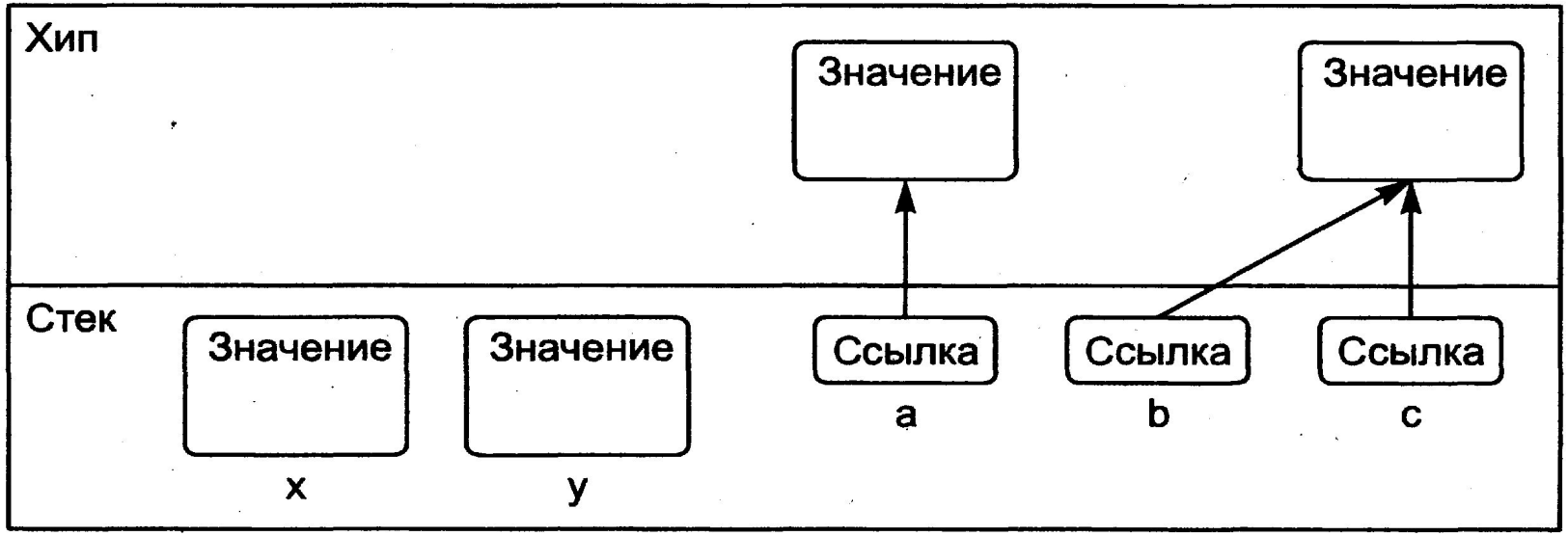
- величина *ссылочного типа* хранит не сами данные, а ссылку на них (адрес, по которому расположены данные). Сами данные хранятся в хипе.

ВНИМАНИЕ – Несмотря на различия в способе хранения, и типы-значения, и ссылочные типы являются потомками общего базового класса **`object`**.

Классификация типов данных C# по способу хранения

Примечание – Встроенные типы выделены полужирным шрифтом, простые типы подчеркнуты

Типы `nullable` введены в версию C# 2.0



Хранение в памяти величин значимого и ссылочного типов

Рисунок иллюстрирует *разницу* между величинами значимого и ссылочного типов. Одни и те же действия над ними выполняются по-разному.

Пример. Проверка на равенство.

Величины значимого типа равны, если равны их значения.

Величины ссылочного типа равны, если они ссылаются на одни и те же данные: на рисунке *b* и *c* равны, но *a* не равно *b* даже при одинаковых значениях.

Из этого следует, что если изменить значение одной величины ссылочного типа, это может отразиться на другой.

Не все значимые типы являются простыми.

По другой классификации структуры и перечисления относятся к структурированным типам, определяемым программистом.

Упаковка и распаковка

Для того чтобы величины ссылочного и значимого типов могли использоваться совместно, необходимо иметь возможность преобразования из одного типа в другой. Язык C# обеспечивает такую возможность.

Преобразование из типа-значения в ссылочный тип называется ***упаковкой – boxing***,
обратное преобразование – ***распаковкой – unboxing***.

Если величина значимого типа используется в том месте, где требуется ссылочный тип, автоматически выполняется *создание промежуточной величины ссылочного типа*:

- создается ссылка;
- в хипе выделяется соответствующий объем памяти;
- туда копируется значение величины, то есть значение как бы упаковывается в объект.

При необходимости обратного преобразования с величины ссылочного типа «снимается упаковка», и в дальнейших действиях участвует только ее значение.

Выводы

1. Для того чтобы читать программы, необходимо понимать, из каких элементов языка они состоят. Это помогает и при поиске ошибок, и при обращении к справочной системе, и при изучении новых версий языка. Более того, изучение любого нового языка рекомендуется начинать именно с ***лексем***, которые в нем поддерживаются.

2. Понятие ***типа данных*** лежит в основе большинства языковых средств.

При изучении любого типа необходимо рассмотреть две вещи:

- его внутреннее представление, (а, следовательно, множество возможных значений величин этого типа),
- что можно делать с этими величинами.

Множество типов данных, реализуемых в языке, является одной из его важнейших характеристик. *Выбор наиболее подходящего типа* для представления данных – одно из необходимых условий создания эффективных программ.