



ПОДГОТОВКА К ЭТ *2020 EDITION*

Автор презентации – Сагалов Даниил БПИ-196

Вводное Слово

От автора и об авторе:

Меня зовут Сагалов Даниил, я обучаюсь на 2 курсе ОП «Программная Инженерия». Ровно год назад я как и Вы сдавал ЭТ по дисциплине программирование! А ещё вёл подготовку к нему для своих однокурсников. В итоге, систематизация материала и попытки не просто заучить, но и понять, почему некоторые вещи работают именно так помогли мне получить оценку 10 на летнем ЭТ.

Основная цель данной консультации – разобрать типовые задания ЭТ и посмотреть, какие ловушки и хитрости встречаются в зимнем тесте, а также вспомнить некоторые теоретические тонкости.

Большинство задач взято из данного источника, спасибо автору за материал:

[Подготовка к ЭТ 2020-2021](#)

Парадигмы Программирования

- **Императивная:** исходный код программы представляет из себя набор инструкций, выполняющихся последовательно. Противоположность – декларативная.
- **Декларативная:** описывается проблема, которую необходимо решить и необходимый результат. Декларативные программы не содержат переменных и операторов присваивания.
- **Доказательная:** технология разработки программ с доказательством правильности (то есть проверкой правильности работы программы).
- **Объектно-ориентированная:** данные и методы обрабатываются в объекты, что делает код модульным (объектная часть), присутствует возможность наследования и организации системы связей между объектами (ориентированная часть).
- **Функциональная:** объекты неизменяемы, в функциях нет переменных, состояний и циклов (вместо этого есть рекурсия)



Парадигмы Программирования

- **РеФЛЕКСивная:** программа способна анализировать себя или другой код. Позволяет автоматическую генерацию кода программой.
- **Обобщённая:** создание универсального кода, позволяющего избегать дублирования при написании программ (пример – System.Collections.Generic в C#)
- **Логическая:** основана на автоматическом доказательстве теорем с использованием логических следствий
- **Распределённая:** позволяет одновременно выполнять одну и ту же задачу параллельно, дробить её на некоторые подзадачи.

Поддержка Парадигм в C#

Императивная	+
Декларативная	-/+
Объектно-Ориентированная	+
Функциональная	+/-
Доказательная	-
Рефлексивная	-/+
Обобщённая	+
Логическая	-
Распределённая	-/+

Задание 1

Верно, что C# ПОЛНОСТЬЮ поддерживает следующие парадигмы (укажите все верные ответы):

- 1) Объектно-ориентированная.
- 2) Рефлексивная.
- 3) Императивная.
- 4) Обобщённая.
- 5) Логическая.

Задание 2

Верно, что C# хотя бы частично поддерживает следующие парадигмы (укажите все верные ответы):

- 1) Доказательная.
- 2) Распределённая.
- 3) Декларативная.
- 4) Функциональная.
- 5) Рефлексивная.

Задачи

Ответ 1: 134

Ответ 2: 2345

Ответы

Служебные и Контекстно- Ключевые СЛОВА. Идентификаторы.

Ключевые (служебные) слова в C# - предварительно зарезервированные слова, которые имеют специальные значения для компилятора. **Важно:** так как @ не является частью имени идентификатора, то ключевые слова МОГУТ быть идентификаторами (пример - @if объявляет идентификатор с именем if). Этот факт можно проверить, если попытаться создать одновременно:

`int @hello; и int hello;`

Такой код не скомпилируется – формально, это один и тот же идентификатор.

Контекстно-ключевые слова имют специальное значение только при определённых сценариях, однако **не являются зарезервированными**. Могут иметь различное значение в нескольких контекстах – например, *where* или *partial*.

Идентификаторы **не могут**: содержать пробелов, начинаться с цифр, заканчиваться на @ или содержать произвольных символов юникода.

Ключевые (Служебные) Слова

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	using static	virtual	void
volatile	while		

Контекстно- Ключевые Слова

async await by

descending dynamic equals

from get global

group into join

let nameof on

orderby partial (тип) partial (метод)

remove select set

unmanaged
(ограничение
универсального типа)

value (для свойств) var

when (условие
фильтра) where (ограничение
универсального
типа) where
(предложение
запроса)

yield init (C# 9.0) record (C# 9.0)

with (C# 9.0) nint (C# 9.0) nuint (C# 9.0)

Задание 1

Урок 1: идентификаторы ▲

Идентификатор в языке C# НЕ допускает:

- 1) Строчных и заглавных букв русского алфавита
- 2) Цифр в начале
- 3) Пробелов
- 4) Символов нижнего подчёркивания
- 5) Произвольных символов Юникода

Задание 2

Выберите служебные слова языка C# (укажите все верные ответы):

- 1) static
- 2) void
- 3) dynamic
- 4) int
- 5) else

Задание 3

Выберите контекстно-ключевые слова языка C# (укажите все верные ответы):

- 1) Main
- 2) null
- 3) var
- 4) object
- 5) double

ЗАДАЧИ

Ответ 1: **235**

Ответ 2: **1245**

Ответ 3: **3**

Main не является контекстно-ключевым!

Ответы

Прочее

Для использования сокращённых квалификационных имён без указания пространства имён используют ключевое слово **using**. **using static** позволяет использовать сокращённые имена для статических методов типов.

Без указания модификатора доступа методы (как и другие члены типов) по умолчанию **private**.

Метод Main() – точка входа в программу, может иметь типы возвращаемого значения **void** или **int** (*или **async Task**/**async Task<int>***, но в тесте это вряд ли встретится), а также содержать optionalный параметр – массив **string[]** (по умолчанию с именем **args**). **Main** в C# может быть определён в любом классе программы, однако должен существовать в единственном экземпляре и должен быть помечен ключевым словом **static**.

Задание 1

Урок 1: идентификаторы ▲

Переменная в языке C# может иметь имя:

- 1) 8bytes
- 2) inline_args
- 3) _8Gb_RAM_
- 4) foreach@
- 5) аннигиляторнаяПушка

Задание 2

Урок 1: метод Main ▲

Основной (главный) метод консольного приложения на языке C# может иметь заголовок:

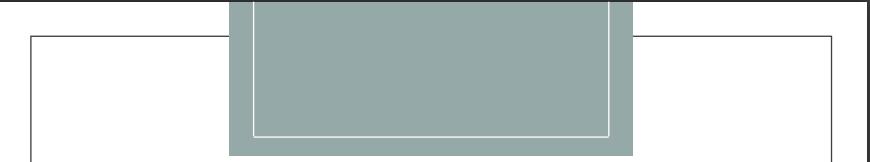
- 1) static void main(string[] args)
- 2) static public void Main()
- 3) public static int Main(string[] args)
- 4) static void Main()
- 5) public static void Main(string args)

Задание 3

Урок 1: метод Main ▲

Выполнение консольного приложения на языке C# всегда начинается с метода:

- 1) Который указан первым в определении класса Program
- 2) Main, причём он должен быть обязательно определён в классе с именем Program
- 3) Main, причём этот метод может быть указан в любом классе приложения
- 4) Main того класса, который указан в первой директиве `using`
- 5) Имя которого указывает пользователь при запуске приложения



ЗАДАЧИ

Ответ 1: 235

Имена переменных не начинаются с цифр и не могут заканчиваться символом @.

Ответ 2: 234

Имя Main начинается с заглавной буквы, порядок public static не важен, а параметрами Main не может быть одиночная строка.

Ответ 3: 3

Main может быть определён в любом классе.

Ответы

Операции C# и Их Приоритет

Все операции в C# кроме операции присваивания, операции объединения с null и тернарной условной операции выполняются слева направо (являются левоассоциативными).

Помните, что все операции присваивания ($+= * = <<=$ и т. д.) имеют наименьший приоритет.

\sim - операция побитового дополнения, инвертирует биты переданного операнда.

Приоритет постфиксных операций инкремента и декремента выше приоритета их префиксных аналогов.

Приоритет побитовых операций: AND XOR OR, затем после них условные AND OR.

Тернарная операция по приоритету предпоследняя.

Постфиксный и префиксный инкремент/декремент не могут быть использованы одновременно, такой код не скомпилируется.

Приоритет Операций в C#

Категория или Имя	Оператор
Первичный	. ? .? [] () x++ x-- new typeof checked unchecked default nameof delegate sizeof stackalloc ->
Унарный	+x -x ! ~ ++x --x (T)x await & * <i>(указатели)</i> true false
Диапазон	..
Мультипликативный	* / %
Аддитивный	x+y x-y
Сдвиг	<< >>
Тестирование типов и относительный	x < y x <= y x > y x >= y is as
Равенство	== !=
Логическое И/Побитовое И	x&y
Логическое исключающее ИЛИ/Побитовое логическое исключающее ИЛИ	x^y
Логическое ИЛИ/Побитовое логическое ИЛИ	x y
Условное И	x&&y
Условное ИЛИ	x y
Оператор объединения с NULL	x??y
Условный (Тернарный) оператор	cond ? t : f
Назначение и объявление лямбда-выражений	= += -= *= /= %= &= = ^= <<= >>= ??= => <i>(лямбда)</i>

Задание 1

Выберите операторы, которые одновременно являются и унарными, и бинарными (укажите все верные ответы):

- 1) +
- 2) %
- 3) !
- 4) -
- 5) ?:

Задание 2

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static int Meth(int x, int y) {
        return ++x + y--;
    }

    static void Main() {
        int x = 5;
        int y = 7;
        Console.WriteLine(Meth(--y + x--, x++ - ++y));
    }
}
```

на экран будет выведено:

Примечание:

*Если возникнет ошибка компиляции, введите: ****

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 3

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        byte b = 80;
        Console.WriteLine(~~~~b--);
    }
}
```

на экран будет выведено:

Примечание:

*Если возникнет ошибка компиляции, введите: ****

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 4

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int x = 10;
        Console.WriteLine(x++);
        Console.WriteLine(++x);
        Console.WriteLine(-x + ++x);
    }
}
```

на экран будет выведено:

Примечание:

*Если возникнет ошибка компиляции, введите: ****

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 5

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int i = 0;
        do {
            Console.WriteLine(i * i + i + 2);
        } while (i != 0);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибки исполнения или исключение, введите: +++

ЗАДАЧИ

Задание 6

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int x = 15;
        int y = 10;
        if ((x & y) < 10 && ++x > ~(++y))
            Console.WriteLine(x + y);
        else
            Console.WriteLine(x - y);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибки исполнения или исключение, введите: +++

Задание 7

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int x = 10;
        int y = 5;
        while (x != ++y) {
            ++x;
            y++;
            Console.WriteLine(--x + --y);
        }
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибки исполнения или исключение, введите: +++

Задание 8

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int j = 12;
        for (int i = 0; i < --j; i++) {
            Console.WriteLine(j + i);
        }
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибки исполнения или исключение, введите: ++

Задание 9

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        double a = 3.5;
        double b = 4.5;
        if (a + b == 8)
            Console.WriteLine(a - b);
        Console.WriteLine(2 * a - 3 * b);
        else
            Console.WriteLine(a + b);
        Console.WriteLine(4 * a - 10 * b);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 10

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int x = 10;
        int y = 20;
        if ((x & 9) < 8)
            Console.WriteLine(y * 3);
        else
            Console.WriteLine(x % 5);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int a = 7;
        int b = 4;
        int c;
        c = (a | b) > 6 ? 13 % 5 : 8 << 3;
        Console.WriteLine(c);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 11

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int x = 10;
        int y = 25;
        int z = x | y >> 3 << 2 & 10 % 3;
        Console.WriteLine(z);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 12

Задание 13

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        Console.WriteLine(10 | 5 >> 2);
    }
}
```

на экран будет выведено:

Ответы

Задание	Ответ	Задание	Ответ
1	14	8	111111111111 (12 единиц)
2	9 (<i>передаётся 6 + 5 и 4 - 7</i>)	9	*** (<i>Посмотрите на блок if</i>)
3	79	10	0
4	10120	11	3
5	2	12	10 (%) >> << &)
6	5	13	11
7	16171819		

Escape-последовательности в C#

Escape-последовательность	Имя символа	Escape-последовательность	Имя-символа
\'	Одинарная кавычка	\n	Новая строка
\”	Двойная кавычка	\r	Возврат каретки
\\"	Обратный слеш	\t	Горизонтальная табуляция
\0	Null	\v	Вертикальная табуляция
\a	Звуковое предупреждение	\u	Escape-последовательность UTF-16
\b	Backspace	\U	Escape-последовательность UTF-32
\f	Перевод страницы	\x	Как \u, только отличается длиной переменной

Циклы в C#

Цикл while выполняется после каждой проверки условия.

Цикл do { } while в начале выполняется, затем проверяет условие. То есть, сработает 1 раз гарантировано.

Цикл for позволяет выполнить несколько действий в конце каждой итерации, их можно перечислить через запятую.
Пример: `for (int i = 0; i < 5; i++, Console.WriteLine(i));`

Цикл foreach позволяет перебирать все элементы массива (или коллекции).

Задание 1

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int res = 0;
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 10; j++) {
                res += i ^ 3 + j ^ 2;
            }
        }
        Console.WriteLine(res);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 2

В результате выполнения фрагмента программы:

```
using System;

class Program {
    public int Meth(int x, int y) {
        int res = 0;
        for (int i = 0; i < 5; i++) {
            res += x % y;
        }
        return res;
    }

    static void Main() {
        int x = 30;
        int y = 50;
        Console.WriteLine(Meth(x, y));
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 3

Выберите литералы, влияющие на вывод строк в консоль (укажите все верные ответы):

- 1) \n
- 2) \r
- 3) \a
- 4) \t
- 5) \w

Задание 4

Выберите верные утверждения (укажите все верные ответы):

- 1) Цикл for можно сделать бесконечным.
- 2) Цикл while может ни разу не пройти.
- 3) Цикл do-while всегда выполняется хотя бы 1 раз.
- 4) Цикл repeat выполняется до тех пор, пока условие ложно.
- 5) Циклы могут иметь в теле более одной операции.

Задачи

Ответ 1: 375

Ответ 2: ***

Попытка вызвать нестатический метод без создания объекта == некомпиль.

Ответ 3: 124

3 – звуковой сигнал, 5 – не существует.

Ответ 4: 1235

Какой такой repeat..? until, что ли? Війді отсюда, разбійнік паскаліст.

Ответы

Пару Слов о CLS и Умалчиваемых Значениях

Спецификация CLS определяет набор вещей, необходимых многим распространённым приложениям с использованием. Она также предоставляет своего рода перечень компонентов, которые должен поддерживать любой язык, реализуемый на базе .NET.

Следует отметить, что `unsigned` типы (`ushort`, `uint`, `ulong`) и `sbyte` в CLS НЕ входят.

В C# поля имеют умалчивающие значения – 0 для всех целочисленных и вещественных типов, `false` для `bool`, `\x0` для `char` и `null` для всех ссылочных типов. Помните, что в отличие от полей локальные переменные умалчивающих значений не имеют, при попытке работать с неинициализированной локальной переменной (не по `out`-ссылке) возникнет ошибка компиляции.

Числовые Литералы

Целочисленные литералы могут быть десятичным числом (`int a = 4`), шестнадцатеричным числом (`int b = 0x2A`) или двоичным числом (`int c = 0b00101010`).

Символ “`_`” может использоваться как числовой разделитель со всеми числовыми литералами (`int c1 = 0B_0010_1010`).

Добавление суффикса `u` или `U` преобразует число к `uint` или `ulong` (если значение превышает границу `uint`).

Добавление суффикса `l` или `L` преобразует число к `long` или `ulong` (если значение превышает границу `long`).

Добавление суффикса `lu` или `ul` преобразует число к `ulong`.

Для вещественных литералов типа `float` используется суффикс `f`.

Для вещественных литералов типа `decimal` используется суффикс `m`.

Классификация Типов C#

	Типы Значений	Ссылочные Типы
Предопределённые (элементарные) типы	byte sbyte short ushort int uint long ulong float double decimal char bool	object/dynamic string
Библиотечные/определяемые программистом	структуры перечисления	массивы классы интерфейсы делегаты записи (C# 9.0)

Byte C#

- Капитан очевидность утверждает, что byte занимает в памяти 1 байт
- Принимает значения от 0 до 255 (забыли – вспоминаем двоичную систему)
- Содержит определение для MinValue и MaxValue

Sbyte C#

- Капитан очевидность снова утверждает, что sbyte занимает в памяти 1 байт
- Принимает значения от -128 до 127
- Содержит определение для MinValue и MaxValue
- Несовместим с CLS

Short C#

- short (Int16) занимает в памяти 2 байта
- Принимает значения от -32768 до 32767
- Содержит определение для MinValue и MaxValue

Ushort C#

- ushort (UInt16) занимает в памяти 2 байта
- Принимает значения от 0 до 65535
- Содержит определение для MinValue и MaxValue
- Несовместим с CLS

Int C#

- int (Int32) занимает в памяти 4 байта
- Принимает значения от -2 147 483 648 до 2 147 483 647
- Содержит определение для MinValue и MaxValue

Uint C#

- uint (UInt32) занимает в памяти 4 байта
- Принимает значения от 0 до 4 294 967 295
- Содержит определение для MinValue и MaxValue
- Несовместим с CLS

Long C#

- long (Int64) занимает в памяти 8 байт
- Принимает значения от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 (надеемся, этого в ЭТ не будет)
- Содержит определение для MinValue и MaxValue

ULong C#

- uint (UInt64) занимает в памяти 8 байт
- Принимает значения от 0 до 18446744073709551615 (многа)
- Содержит определение для MinValue и MaxValue
- Несовместим с CLS

Float C#



- float (Single) занимает в памяти 4 байта
- Принимает значения от $\pm 1,5 \times 10^{-45}$ до $\pm 3,4 \times 10^{38}$
- Содержит определение для MinValue, MaxValue, NaN, NegativeInfinity, PositiveInfinity и Epsilon
- При делении на 0.0f получаем бесконечность, такой код работает корректно
- Точность до 9 знаков после запятой
- Для значений на конце дописывается суффикс f (в противном случае – CE).
- Программисты на Unity любят этот тип))

Double C#

- double занимает в памяти 8 байт
- Принимает значения от $\pm 5,0 \times 10^{-324}$ до $\pm 1,7 \times 10^{308}$
- При делении на 0.0 получаем бесконечность, такой код работает корректно
- Содержит определение для MinValue, MaxValue, NaN, NegativeInfinity, PositiveInfinity и Epsilon
- Точность до 17 знаков после запятой
- Не требует суффиксов на конце

Decimal C#

- decimal занимает в памяти 16 байт
- Принимает значения от $\pm 1,0 \times 10^{-28}$ до $\pm 7,9228 \times 10^{28}$
- Содержит определение для MinValue, MaxValue, NaN, NegativeInfinity, PositiveInfinity и Epsilon
- При делении на 0.0m получаем бесконечность, такой код работает корректно
- Точность до 29 знаков после запятой
- Для значений на конце дописывается суффикс m (в противном случае – CE).
- Не принимается в качестве аргумента для Math.Pow()

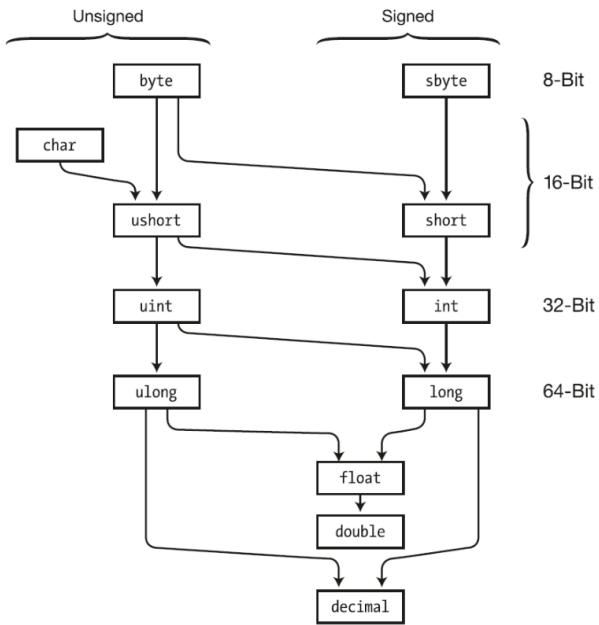
Bool C#

- bool (Boolean) занимает в памяти 1, 2 или 4 байт(а) (по различным причинам)
- Принимает значения false и true. В другие типы неявно не преобразовывается *плач C++ программиста вдалеке*
- Умалчиваемое значение – false

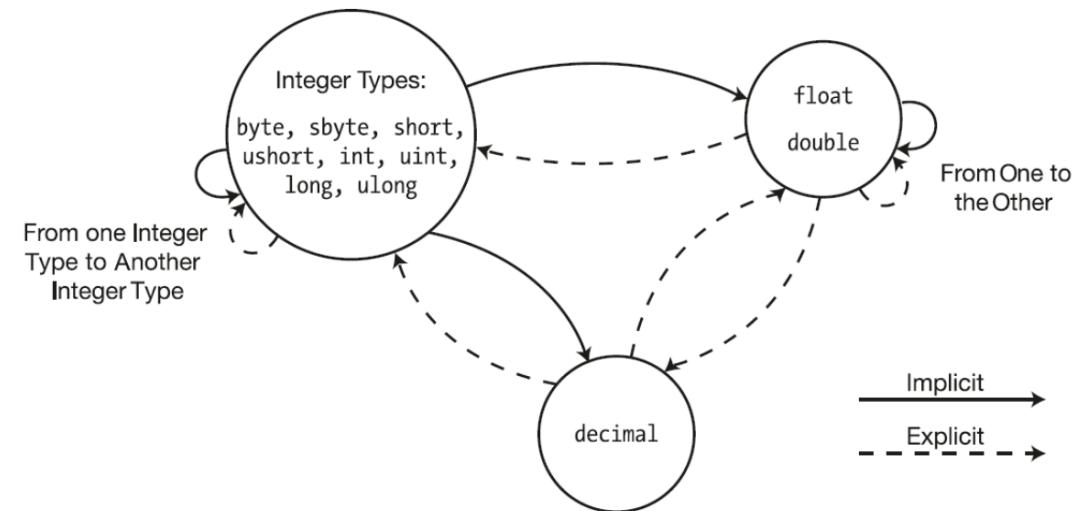
Char C#

- `char` занимает в памяти 2 байта.
- Принимает значения от U+0000 до U+FFFF в 16-разрядном Unicode
- (!) Содержит определение для `.MinValue` и `.MaxValue`
- Умалчивающее значение – \x00

Неявные преобразования числовых типов



Преобразования числовых типов (implicit – неявное, explicit – явное)



ПРЕОБРАЗОВАНИЯ ТИПОВ C#

Некоторые Особенности Вычислений

Деление числа на 0 (для вещественных чисел) приводит к результату бесконечность.

Деление 0.0 на 0.0 даёт NaN.

Остаток от деления отрицательного числа всегда отрицателен, а от положительного — положителен.

Важно — при попытке задать double b = 1/0; код не скомпилируется, так как 1 и 0 воспринимаются компилятором как целочисленные литералы.

Бесконечность можно привести к другим типам — это даст наименьшее значение указанного типа (кроме decimal, попытка привести бесконечность к данному типу приводит к OverflowException)

По умолчанию в настройках проекта вычисления выполняются в unchecked контексте.

Задание 1

Выберите свойства, которые содержатся в целочисленном типе int (укажите все верные ответы):

- 1) MaxValue
- 2) MinValue
- 3) NaN
- 4) PositiveInfinity
- 5) NegativeInfinity

Задание 2

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        byte a = 100;
        byte b = 200;
        Console.WriteLine((byte)(a + b));
    }
}
```

на экран будет выведено:

Задание 3

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        Console.WriteLine(15L - 12u);
    }
}
```

на экран будет выведено:

Примечание:

*Если возникнет ошибка компиляции, введите: ****

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 4

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        short s = 250;
        uint u = 13;
        Console.WriteLine(s * u);
    }
}
```

на экран будет выведено:

Примечание:

*Если возникнет ошибка компиляции, введите: ****

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 5

Укажите номера строк кода, в которых нет ошибок компиляций и исключений:

```
class Program {
    static void Main() {
        int a = 1 / 0;           //1
        double b = 50 / 0.0;     //2
        byte c = (byte)b;       //3
        float d = 11.4;         //4
        long e = (int)10;        //5
    }
}
```

- 1) 1
- 2) 2
- 3) 3
- 4) 4
- 5) 5

Задачи

Задание 6

Урок 2: типы данных

Подряд без пробелов перечислите номера строк, содержащих синтаксические ошибки:

```
char a = 98; // 1
int b = 98; // 2
char c = 'a'; // 3
int d = 'a'; // 4
int e = "a"; // 5
long = 16; // 6
short f = 1; // 7
```

Задание 7

Укажите номера строк кода, в которых НЕ содержатся ошибки компиляции:

```
class Program {
    static void Main() {
        int a = 1000000000000000000; //1
        long b = (int)-1000000000000000000; //2
        double c = (float)11.5f; //3
        byte d = (byte)c; //4
        sbyte e = -128; //5
    }
}
```

- 1) 1
- 2) 2
- 3) 3
- 4) 4
- 5) 5

Задание 8

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static int x = 20;
    static void Main() {
        {
            int x = 14;
        }
        {
            int x = 7;
            Console.WriteLine(x);
        }
        Console.WriteLine(x);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 9

Выберите верные утверждения (укажите все верные ответы):

- 1) Тип decimal не допускает инициализации без суффикса m.
- 2) На тип int отведено 4 байта памяти.
- 3) Если явно не определён тип данных числа с плавающей точкой, его тип будет double.
- 4) Чтобы явно определить тип данных ulong всегда нужно добавлять к числу литерал UL.
- 5) Типы данных с плавающей точкой допускают наличие отрицательной или положительной бесконечностей.

Задачи

Ответ 1: **12**

Ответ 2: **44**

byte переполнится и значения будут считаться по кругу, начиная снова с 0.

Ответ 3: **3**

Ответ 4: **3250**

Ответы

Ответ 5: 235

Float не инициализируется без суффикса f, int не поддерживает деление на 0.

Ответ 6: 156

Если вы не внесли ответ 6, настоятельно рекомендуем вам... читать задание.

Ответ 7: 345

Ответы

Варианты 1 и 2 в обоих случаях приводят к переполнению.

Ответ 8: 720

Ответ 9: 235

String C#

В языке C# строка – ссылочный тип, а сами строки – **неизменяемые** объекты, являющиеся последовательностью символов Unicode. Тем не менее, вы могли заметить, что в C# для строк перегружен оператор «+». Он обозначает вызов операции конкатенация (склеивание строк). На самом деле, в момент применения «+» создаётся новый объект типа String.

В случае применения + для набора, состоящего из строки и нескольких значений целого или вещественного типа, работает конкатенация. Например, строка когда *Console.WriteLine("abc" + 4.4 + 3);* выведет abc4,43

Object и Dynamic в C#

System.Object – базовый класс всех управляемых типов в языке C#. Вы можете создавать переменные типа `object`, которые могут ссылаться на объект любого типа. При этом, когда вы присваиваете переменной типа `object` какой-либо значимый тип, происходит упаковка (вокруг значения создаётся объект-контейнер). Эта операция работает и в обратную сторону – объект можно распаковать (Пример: `int i = 10; object o = i; i = (int)o;`). К двум переменным типа `object` не получится применить операции/вызвать методы, не определённые для `System.Object`.

Хотя `dynamic` тоже является `System.Object`, он имеет важные отличия: выражения, которые содержат операции над `dynamic`, проверяются (или полностью не допускаются) при выполнении программы. Как результат, некорректные операции с более высокой вероятностью могут приводить к исключениям. При этом сами переменные типа `dynamic` компилируются в `object`.

Задание 1

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        dynamic a = 10;
        dynamic b = "abc";
        Console.WriteLine(a + b);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 2

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        dynamic a = null;
        dynamic b = 10;
        object c = a + b;
        Console.WriteLine(c);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 3

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        dynamic x = null;
        x = 9;
        x = "233";
        x %= 10;
        Console.WriteLine(x);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 4

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        dynamic x = null;
        x = 5;
        x++;
        ++x;
        x = "123";
        x += 1;
        x = int.Parse(x);
        x %= 10;
        Console.WriteLine(x);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задачи

Ответ 1: 10abc

Будет определено компилятором как конкатенация строк.

Ответ 2: ---

В с будет храниться значение null. Без разницы, что прибавлять к null.

Ответ 3: +++

Для компилятора операция % не определена для строк. Тем не менее, так как при использовании dynamic операции применяются в runtime, выбрасывается исключение.

Ответ 4: 1

Ответы

Специфика Math. Pow, Round, DivRem

Math.Pow() в C#:

- Принимает и возвращает double.
- Для дробных степеней необходимо передавать именно вещественные числа типа double (хотя бы одно из чисел должно быть double), в противном случае Math.Pow(x, 1/3), например, вернёт 1 (т. к. это x в степени 0)
- decimal не может использоваться в Pow, так как не имеет в явном виде определённой перегрузки.

Math.Round() в C#:

- По умолчанию округляет пограничные значения в сторону ближайшего чётного (то есть, Math.Round(5.5) == 6 и Math.Round(6.5) == 6
- Принимает 2 параметр – сторону округления числа (MidpointRounding) в сторону ближайшего чётного числа (ToEven) или до ближайшего числа дальше от нуля (AwayFromZero)

Math.DivRem() в C#:

- Принимает на вход два int/long и передаёт в out параметр остаток от деления первого числа на второе; возвращает целую часть от деления.

Parse и TryParse в C#

Parse и **TryParse** являются статическими методами (мы не создаём никаких объектов для того, чтобы вызвать эти методы)

TryParse – безопасный метод с `out` параметром, имеет возвращаемое значение `bool` (`true` если преобразование возможно). Безопасность заключается в том, что данный метод не выбрасывает исключений.

Parse имеет возвращаемое значение того типа, к которому преобразуется переданный аргумент.

`Parse` выбрасывает следующие исключения: `ArgumentNullException`, `ArgumentException`, `FormatException`, `OverflowException`.

ref и out

Ключевое слово `ref` в языке C# позволяет передавать параметры по ссылке. То есть параметры, переданные с `ref` изменятся и вне метода при их изменении внутри.

Особенности `ref`:

- Параметры, передаваемые по ссылке с помощью `ref` должны быть обязательно инициализированы.

Ключевое слово `out` является ещё одним способом работать с аргументами по ссылке.

Особенности `out`:

- Всем переменным с `out` должно быть присвоено какое-то значение внутри метода.
- В отличие от `ref`, `out` допускает передачу в него неинициализированных переменных, однако до инициализации `out`-параметра попытка чтения из него приведёт к ошибке компиляции.

Общее для `ref` и `out`:

- При вызове метода перед передаваемым значением обязательно должно указываться ключевое слово `ref/out`. В противном случае код не скомпилируется.
- Объявить 2 метода с одинаковой сигнатурой и параметрами, отличающиеся только `ref` и `out` нельзя, с точки зрения полученного IL-кода это одно и то же.

Задание 1

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int x = 0;
        int y = Meth(ref x);
        Meth(ref x);
        Console.WriteLine(x + y * 2);
    }

    static int Meth(ref int vrr) {
        vrr += 10 ^ 4;
        return 20 ^ 8;
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 2

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int x = 10;
        int y = 20;
        Console.WriteLine($"{RefOut(ref x, out y)}{x}{y}");
    }

    static int RefOut(ref int a, out int b) {
        int res = 3;
        for (int i = 0; i < a; i++) {
            res += b = a = 5;
        }
        b = res - a;
        return res;
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 3

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int x = 5;
        Meth(ref x);
    }

    static void Meth(ref int x) {
        x--;
        if (x == 0)
            return;
        Meth(ref x);
        Console.WriteLine(x);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 4

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int x = 5;
        Meth(out x);
        Console.WriteLine(x);
        int.TryParse(Console.ReadLine(), out x);
    }

    static void Meth(out int x) {
        x--;
        Console.WriteLine(x);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 5

Выберите верные утверждения (указывайте все верные ответы):

- 1) Параметр метода, помеченный модификатором ref, должен быть проинициализирован до передачи его в метод.
- 2) Нельзя объявить 2 метода с одинаковой сигнатурой и параметрами, отличающимися только модификаторами ref и out.
- 3) Параметр метода, помеченный модификатором out должен быть либо проинициализирован до передачи в метод, либо проинициализирован в самом методе.
- 4) Модификатор out и ref должны быть указаны при вызове метода.
- 5) С помощью модификатора ref можно передавать в метод числовые аргументы.

Задачи

Ответ 1: **84**

Ответ 2: **28523**

Ответ 3: **0000**

Ответ 4: *******

Попытка чтения значения из out-параметра до его присваивания чревата ошибкой компиляции.

Ответ 5: **124**

Ответы

Массивы: Основные Определения

Элементы – отдельные единицы данных, содержащиеся в массиве.

Ранг (размерность) – положительное число измерений массива.

- **Длина измерения** – число элементов в указанном измерении.
- **Размер массива** – общее количество элементов массива (`length`).

Помните, что массивы – ссылочный тип. При этом сами массивы могут хранить как ссылки, так и значения. Количество измерений и длина каждого из них задаются при инициализации и не могут меняться (метод `Resize` принимает массив через `ref` и создаёт новый массив, изменяя значение ссылки).

Длина измерения массива может быть равна 0. Во многих сценариях удобнее передавать пустой массив, а не `null`.

Существует 3 крупные категории массивов: одномерные, многомерные и «зубчатые» (массивы массивов).

Элементы массивов нумеруются с 0, то есть последний элемент имеет номер $N-1$. При попытке обратиться по индексу, не попадающему в данный диапазон возникнет исключение `IndexOutOfRangeException`.

Создание Массивов

<тип>[] <идентификатор ссылки>;

int[] a0; – без инициализации.

int[] a1 = new int[2]; – неявная инициализация

int[] a2 = { 1, 2, 3 }; – явная инициализация

int[] a3 = new int[4] { 4, 5, 6, 7 }; – явная инициализация

int[] a4 = new int[] { 0, 0, 0 }; – явная инициализация

var a5 = new[] { 8, 9, 10 }; – это тоже явная инициализация

int[] weirdArray = { (int)3.14 }; – так тоже можно

Важно: Так нельзя:

int[5] a;

int[] a = new int { 0, 1, 2 };

var[] a; – массива var не может быть

Задание 1

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int[] args = new int[10];
        for (int i = 0; i <= args.Length; i++) {
            args[i] = i;
            Console.Write(i + args[i]);
        }
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

ЗАДАЧИ

Задание 2

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int[] args = new int[] { 1, 2, 3, 4, 5 };
        Array.Resize(args, 3);
        for (int i = 0; i < args.Length; i++) {
            Console.Write(args[i] + i);
        }
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 3

В результате выполнения фрагмента программы:

```
using System;
```

```
class Program {
    static void Main() {
        int[] args = { 1, 6, 3, 9, 2 };
        int[] args2 = args;
        for (int i = 0; i < args.Length; i++) {
            args[i] = args2[args2.Length - i - 1];
        }
        args2 = null;
        Console.Write(args[0] + args[args.Length - 1]);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 4

В результате выполнения фрагмента программы:

```
using System;
```

```
class Program {
    static void Main() {
        int[] args = { 0, 4, 8, 1, 3 };
        int[] args2 = args;
        Array.Reverse(args);
        for (int i = 0; i < args.Length; i++) {
            Console.Write(args[i] + args2[i]);
        }
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Ответ 1: +++

>= args.Length – IndexOutOfRangeException.

Ответ 2: ***

Resize принимает массив по ссылке с помощью ref, а не саму ссылку.

Ответ 3: 4

Ответы

Ответ 4: **621680**

Многомерные Массивы

Чтобы создать N-мерный массив, необходимо при его инициализации в первых квадратных скобках указать символ «», ». Количество измерений задаётся количеством запятых – int[,] – двухмерный, int[,,] – трёхмерный и т. д.

int[,,] multiArray;

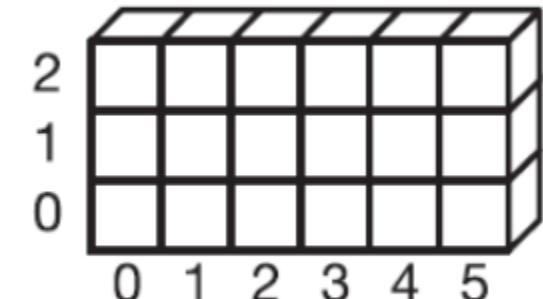
Ранг многомерного массива равен количеству измерений, а длина – количеству элементов всех измерений массива.

Пример:

```
int[,] a4 = new int[4,3,2] {  
    { {8, 6}, {5, 2}, {12, 9} },  
    { {6, 4}, {13, 9}, {18, 4} },  
    { {7, 2}, {1, 13}, {9, 3} },  
    { {4, 6}, {3, 2}, {23, 8} }  
};
```

Rank: 3
Length: 24

Rectangular Arrays



Two-Dimensional
int[3,6]

Создание Многомерных Массивов

<тип>[,] <идентификатор ссылки>;

int[,] a0; – двухмерный массив без инициализации

int[,] a1 = new int[2,3,5]; – трёхмерный массив

int[,] a2 = new int[2,2] { { 2, 3 }, { 2, 3 } }; – двухмерный массив

int[,] a3 = { { 0, 0}, {0, 1}, {1, 0}, {1, 1} }; – укороченный синтаксис

```
int[,] a4 = new int[4,3,2] {  
    { {8, 6}, {5, 2},{12, 9} },  
    { {6, 4}, {13, 9}, {18, 4} },  
    { {7, 2}, {1, 13}, {9, 3} },  
    { {4, 6}, {3, 2}, {23, 8} }  
};
```

Массивы Массивов

Массив массивов представляет собой ссылку на некоторые другие массивы соответствующего типа. Создаётся массив массивов при указании нескольких пар квадратных скобок. Могут быть массивами многомерных массивов.

```
int[ ][ ] jaggedArray;
```

Ранг массива массивов также равен количеству измерений, а длина – количеству элементов.

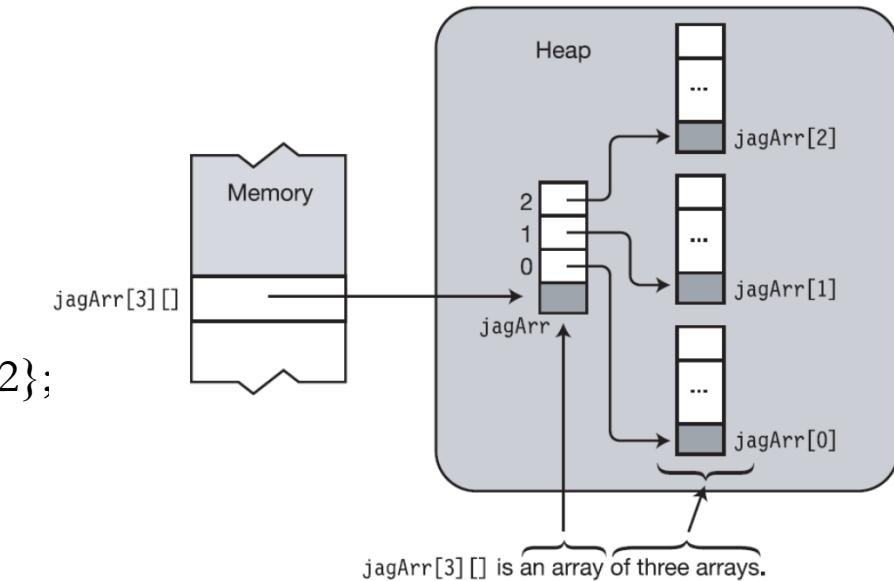
Пример:

```
int[ ][ ] arr= new int[3][ ];  
arr[0] = new int[ ] {1, 2, 3};  
arr[1] = new int[ ] {4, 5, 6, 7};  
arr[2] = new int[ ] {8, 9, 10, 11, 12};
```

Rank a – 1

Length a – 3

Rank a[0] – 1
Length a[0] – 3



Создание Массивов Массивов

<тип>[][] <идентификатор ссылки>;

Пример 1:

int[][][] a0; – МММ (Массив Массивов Массивов)

Пример 2:

```
int[] a1 = new int[3][];
a1[0] = new int[ ] {1, 2, 3};
a1[1] = new int[ ] {4, 5, 6, 7};
a1[2] = new int[ ] {8, 9, 10, 11, 12};
```

Пример 3:

```
int[,] a2;
a2 = new int[3][];
a2[0] = new int[,] { { 10, 20 }, { 100, 200 } };
a2[1] = new int[,] { { 30, 40, 50 }, { 300, 400, 500 } };
a2[2] = new int[,] { { 60, 70, 80, 90 }, { 600, 700, 800, 900 } };
```

Так нельзя:

int[][] aWrong = new int[2][3];

Цикл foreach

Цикл foreach в C# позволяет перебирать элементы коллекций. Без дополнительной вложенности работает с одномерными и многомерными массивами. Для работы с массивами массивов потребуется вложенность. Имеет синтаксис:

```
foreach (<Тип значения> <Имя> in <Коллекция/Массив>){ };
```

При работе с foreach удобно использовать неявно типизированную итерационную переменную:

```
foreach (var <Имя> in <Коллекция/Массив>){ };
```

Важно: по умолчанию итерационная переменная цикла foreach доступна только для чтения, ей нельзя изменить значение внутри цикла – в противном случае возникнет ошибка компиляции.

Тем не менее, если итерация проходит по списку объектов, через итерационную переменную можно обращаться ко всему доступному функционалу объектов.

Члены класса Array

Свойства

Rank – возвращает значение типа int – количество измерений массива

Length – возвращает количество элементов во всех измерениях массива

Статические Методы

Clear – приравнивает все элементы массива к значениям по умолчанию.

Sort – сортирует элементы одномерного массива в соответствии с правилами сравнения для типов. Таким образом, числа сортируются в порядке возрастания, string – в лексико-графическом порядке по символам и т. д.

BinarySearch – использование алгоритма бинарного поиска для одномерного массива.

IndexOf – возвращает индекс первого вхождения элемента массива и -1, если указанное значение в массиве не встречается.

Reverse – меняет порядок элементов массива на обратный.

Resize – создаёт новый массив из старого указанной длины. **Важно:** Требует передачи массива по ссылке с помощью ref.

Методы

GetLength – возвращает длину указанного измерения массива.

GetUpperBound - возвращает индекс верхней границы указанного измерения массива

GetLowerBound – возвращает индекс нижней границы указанного измерения массива

Задание 1

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int[,] args = { { 1, 2, 3 }, { 3, 4, 5 } };
        Console.WriteLine(args.GetLength(1) + args.GetUpperBound(0) +
args.GetLowerBound(1));
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 3

Выберите корректные инициализации массива (укажите все верные ответы):

- 1) int[] args = new int[] { 1, 2, 3, 4 };
- 2) double[] args = new { 1, 2.4, 5 };
- 3) int[,] args = { { 1, 2 }, { 3, 4 } };
- 4) var[] args = new object { new Exception() };
- 5) int[][] args = { new int[3], new int[5], new int[7] };

Задание 2

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int res = 0;
        int[][] args = { new int[5], new int[3], new int[1],
new int[4], new int[2] };
        for (int i = 0; i < args.Length; i++) {
            for (int j = 0; j < args[i].Length; j++) {
                args[i][j] = i + j;
                res += args[i][j];
            }
        }
        Console.WriteLine(res);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 4

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int[] args = { 0, 1, 2, 3, 4, 5 };
        Array.Reverse(args);
        Array.Resize(ref args, 3);
        Array.Reverse(args);
        Console.WriteLine(args[2] / args[1] * args[0]);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задачи

Задание 5

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int sum = 0;
        int[] arg = { 0, 1, 2, 3, 4, 5 };
        int[][] args = new int[5][];
        for (int i = 0; i < args.Length; i++) {
            args[i] = arg;
            for (int j = 0; j < args.Length; j++) {
                args[i][j] = i + j;
                sum += args[i][j];
            }
        }
        Console.WriteLine(sum);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

ЗАДАЧИ

Задание 6

В результате выполнения фрагмента программы:

```
using System;
```

```
class Program {
    static void Main() {
        int[,] args = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
        Array.Clear(args, 3, 3);
        for (int i = 0; i < args.GetLength(0); i++) {
            for (int j = 0; j < args.GetLength(1); j++) {
                Console.Write(args[i, j]);
            }
        }
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Ответ 1: 4

Длина 1 измерения – 3, верхняя граница нулевого – 1, а минимальный индекс – 0.

Ответ 2: 45

Ответ 3: 135

Во 2 использование new без указания типа – так нельзя. 4 – массив var? Шта?

Ответ 4: 3

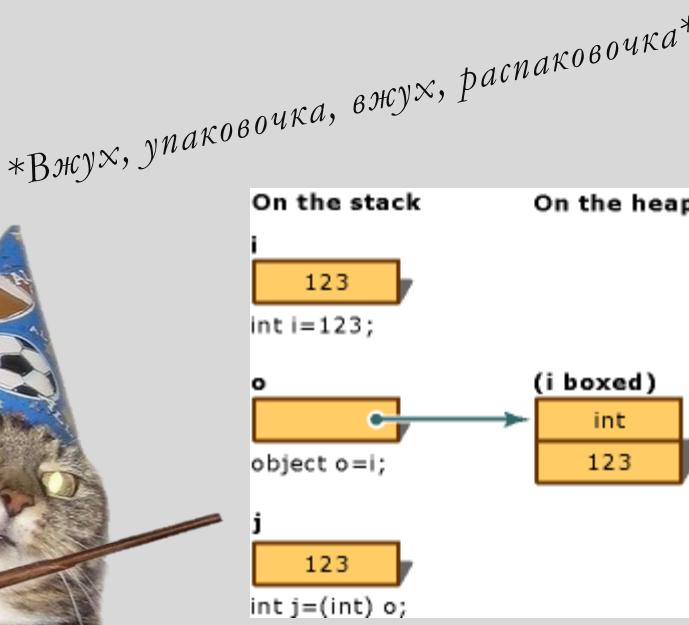
Ответы

Ответ 5: **100**

Ответ 6: **123000789**

Ответы

Упаковка. Распаковка



Упаковка – процесс приведения значимого типа к `Object` или любому другому типу интерфейса, реализуемого данным значимым типом. В процессе упаковки тип значения инкапсулируется внутри экземпляра `System.Object` и сохраняется в куче общеязыковой средой выполнения (CLR).

Распаковка – процесс, обратный упаковке, извлекает значение из объекта.

Важно: упаковка происходит неявно, а распаковка требует обязательного явного приведения к исходному типу. Всегда нужно помнить, что упаковка и распаковка являются весьма затратными с точки зрения вычислений процессами (при упаковке нужно создать объект и разместить его в куче). При успешной распаковке будут выполнены 3 шага:

- Проверка инвариантности (совпадения) типов. Тип, к которому приводится упакованный объект должен строго совпадать с изначальным типом упакованного значения, иначе возникнет **InvalidCastException**.
- Проверка на null. При попытке распаковки null вы получите **NullReferenceException**.
- Копирование значения из экземпляра в указанную значимую переменную, если два прошлых шага были пройдены успешно.

Примеры Упаковки и Распаковки

При упаковке и распаковке всегда нужно помнить тип упакованного объекта, иначе вы получите **InvalidCastException**. Тем не менее, вы можете узнать тип упакованного объекта при помощи оператора **is** (ошибки возможны только в случае с nullable-тиปами). Оператор **as** для типов значений неприменим.

Примеры:

```
int int1 = 4;  
  
object container1 = int1; // значение int1 упаковано как Int32  
  
object container2 = (byte)int1; // значение int1 упаковано как byte  
  
short short1 = (short)container1; // Исключение – short != int1  
  
int int2 = 5 + (byte)container1; // Исключение – container1 содержит int, а не byte  
  
int int3 = (int)container2; // Исключение – container2 содержит byte, а не int  
  
int int4 = (int)container1 + 228; // OK  
  
byte byte1 = (byte)container2; // OK
```

Задание 1

Выберите верные утверждения: (укажите все верные ответы):

- 1) Любой тип значения может быть упакован в Object;
- 2) Упаковка является явным приведением типов;
- 3) При распаковке вы можете привести объект только к типу, из которого он был изначально запакован;
- 4) Вы можете распаковать объект при помощи метода Unbox() класса Object;
- 5) Распаковка является неявным приведением типов;

© Сагалов Даниил: vk.com/mrsagel

Задание 2

Из указанных строк выберите те, написание которых НЕ приведёт к ошибке компиляции или исключению (укажите все верные ответы):

```
using System;

public class Program {
    static void Main() {
        int myInt = 255;
        object obj1 = myInt;
        object obj2 = (byte)myInt;
        int resInt1 = 4 + (int)obj2; // 1
        int resInt2 = (byte)obj2 + (int)obj1; // 2
        int resInt3 = obj1 as int; // 3
        long resLong = (int)obj1 + 33; // 4
        byte resByte = obj2 is byte b ? b : (byte)0; // 5
    }
}
```

© Сагалов Даниил: vk.com/mrsagel

Задачи

Ответы

Ответ 1: 13

Ответ 2: 245

Начиная с C# 7.0, Вы можете использовать **кортежи значений** как способ группировки нескольких элементов в одну простую структуру (**System.ValueTuple**). Они являются **типами значений** и могут содержать сколько угодно много элементов. Кортежи могут использоваться в качестве возвращаемых значений в случаях, когда Вам необходимо вернуть несколько элементов в качестве результата метода.

C# предоставляет специальный синтаксис, позволяющий более лаконично объявлять кортежи значений – в качестве объявления типа в круглых скобках нужно перечислить типы всех элементов:
(<Тип 1>, <Тип 2>, ...) <Идентификатор> = (<Значение 1>, <Значение 2>, ...);

После создания кортежа Вы можете напрямую изменять его поля, по умолчанию компилятор генерирует для них имена автоматически: **Item<Number>**, где – number – натуральное число, номер элемента (начиная с 1).

Кортежи Значений

```
(int, double) valueTuple1 = (10, 15.55);
// Выведется (10, 15.55) - значения в круглых
// скобках, перечисленные через запятую.
Console.WriteLine(valueTuple1);
// Копирование кортежа, тип
// определяется автоматически.
var valueTuple2 = valueTuple1;
valueTuple2.Item1++;
// Выведется число 11.
Console.WriteLine(valueTuple2.Item1);
```

Вы также можете явно указывать имена полей кортежа как при объявлении типа, так и при его инициализации (при использовании `var`). Тем не менее, помните, что данные имена существуют только до этапа компиляции и впоследствии заменяются именами по умолчанию – они недоступны на этапе выполнения программы. Имена полей по умолчанию могут использоваться одновременно с явно указанными.

Начиная с C# 7.1, компилятор может выводить имена переменных кортежей из имён соответствующих переменных при инициализации.

На имена полей кортежей накладываются ограничения:

- Они не могут повторяться.
- Они не могут дублировать имена уже существующих для кортежей имён – `Item3`, `ToString` и т. д.

Важно: Имена полей кортежей не учитываются при сравнении на равенство и неравенство. Для присваивания значения одного кортежа другому достаточно совпадения количества параметров и их типов.

Именование Полей Кортежей

```
// Явное именование параметров при объявлении.  
(double Perimeter, double Area) valueTuple1 = (11.1, 22.2);  
// Автоматическое определение кортежа с именованными полями.  
var valueTuple2 = (x: 1, y: 2);  
// Использование явно заданного имени и имени по умолчанию.  
Console.WriteLine(valueTuple2.x + valueTuple2.Item2);  
// Хотя имена различные, типы полностью соответствуют.  
(double Pushistostb, double Milota) valueTuple3 = valueTuple1;  
// C# 7.1: Определение имён полей по переданным переменным.  
int mass = 30, energy = 9000;  
var valueTuple4 = (mass, energy);  
Console.WriteLine(valueTuple4.mass + valueTuple4.energy);
```

DateTime в C#

DateTime – структура в языке C#, которая представляет собой текущее время в формате дата + время суток. Значения времени измеряются в тактах – каждый такт 100-наносекунд

Содержит определение для полей MinValue и MaxValue – DateTime определена в диапазоне с 00:00 1 января 1 года до 23:59:59 31 декабря 9999 года

Экземпляры класса DateTime можно сравнивать с помощью оператора сравнения

Конструктор DateTime выбросит ArgumentOutOfRangeException в случае передачи некорректных входных данных (например, попытке задать 13 месяц)

Некоторые конструкторы:

public DateTime (long ticks);

public DateTime (int year, int month, int day);

public DateTime (int year, int month, int day, int hour, int minute, int seconds);

Работа с Каталогами Файлами в C#

В общем виде путь фала и его расположение выглядит так:
`<путь>/<имя файла>.<расширение>` для UNIX и `<путь>\<имя файла>.<расширение>` для Windows.

По умолчанию Visual Studio сохраняет .exe файл в `bin\Debug\<версия .NET>` вашего проекта

Данные символы запрещены в имени файла: > < | ? * / \ : “

- Для использования имён файлов настоятельно рекомендуется писать префикс буквального строкового литерала `@`.
- Комбинация символов в пути файла `..\` означает, что используется файл на уровень выше исполняемого модуля.
- Для использования функционала классов, связанных с работой с файлами в C# необходимо дописать `using System.IO;` или писать каждый раз писать `System.IO.<необходимый элемент>`.

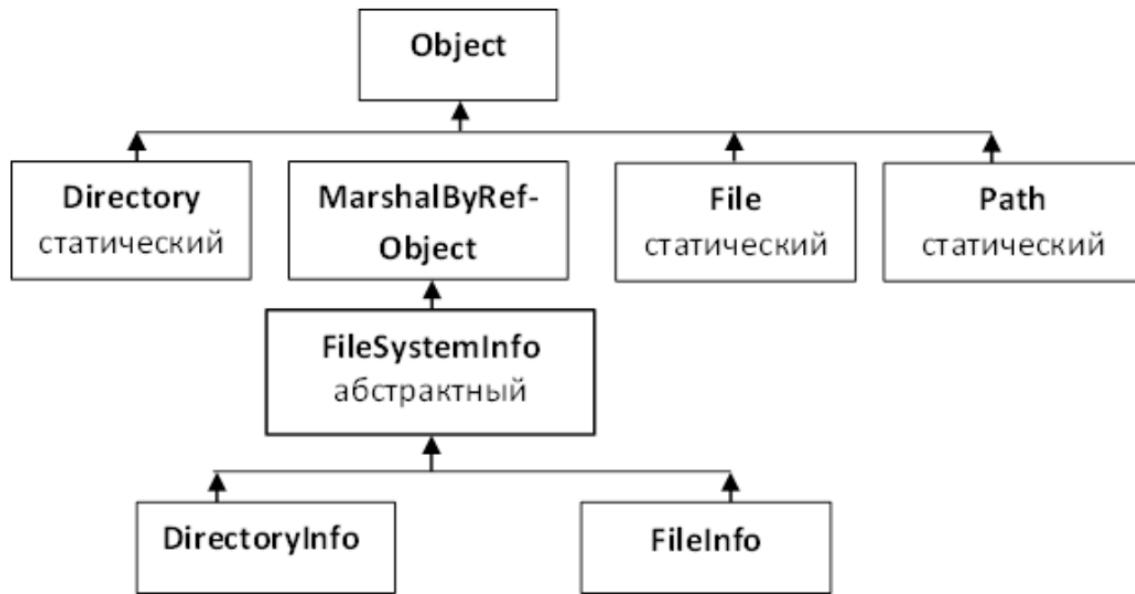
Исключения При Работе с Файлами

При работе с файлами могут возникать различные исключения: программист не может гарантировать наличие каталога или конкретного диска. По этой причине, необходимо заключать сценарии работы с вводом-выводом в блоки try-catch.

Ниже представлены некоторые из исключений, возникающие при работе с **System.IO**:

- **IOException** – базовый класс всех ошибок ввода-вывода.
- **DriveNotFoundException** – ошибка при обращении к недоступному диску/хранилищу данных.
- **DirectoryNotFoundException** – не удается найти часть пути к файлу/каталогу.
- **PathTooLongException** – путь к файлу или каталогу оказался длиннее максимального, разрешенного системой.
- **UnauthorizedAccessException** – ошибка, возникающая в связи с отсутствием доступа.

```
using System.IO;
```



Иерархия Классов

Работа с файлами и каталогами осуществляется в основном с использованием 3 статических классов – **Directory**, **File** и **Path**, а также ещё 2 нестатических – **FileInfo** и **DirectoryInfo**.

Это позволяет обращаться к конкретному экземпляру при необходимости.

Работа с Каталогами

Проверка существования каталога:

static bool Directory.Exists(string name) – возвращает true, если указанный каталог существует на диске и false в противном случае.

Создание каталогов:

static DirectoryInfo Directory.CreateDirectory(string path)
Directory.CreateDirectory(string path, DirectorySecurity) – статический метод, создающий каталог по заданному пути, если каталог ещё не существует. Имеет перегрузку, позволяющую получить на вход некоторые заданные параметры безопасности каталога в Windows.

Удаление каталогов:

static void Directory.Delete(string path)
Directory.Delete(string path, bool recursive) – удаляет каталог по указанному пути. Имеет перегрузку, позволяющую удалить все подкаталоги и файлы.

DirectoryInfo.Delete()

DirectoryInfo.Delete(bool recursive) – экземплярный метод, аналог Directory.Delete

Работа с Каталогами

Перемещение каталогов:

static void Directory.Move(string source, string destination) –
перемещает указанный каталог или файл по заданному пути.

DirectoryInfo.MoveTo(string destination) – *экземплярный* метод,
перемещающий объект DirectoryInfo по указанному пути.

Создание подкаталога в указанном:

DirectoryInfo.CreateSubdirectory(string path)

DirectoryInfo.CreateSubdirectory(string path, DirectorySecurity) –
экземплярный метод, создающий подкаталог по в указанном. Имеет
перегрузку, позволяющую получить на вход некоторые заданные
параметры безопасности каталога в Windows.

Получение списка каталогов:

static string[] Directory.GetDirectories(string name) – возвращает
массив строк – пути всех каталогов, вложенных в указанный.

DirectoryInfo.GetDirectories() – *экземплярный* метод, аналог
Directory.GetDirectories().

Получение списка файлов и каталогов:

DirectoryInfo.GetFileSystemInfos() – *экземплярный* метод,
возвращающий массив объектов FileInfo – все файлы и каталоги
внутри указанного.

Работа с Файлами

Проверка существования файла:

static bool File.Exists(string name) – возвращает true, если указанный файл существует на диске и false в противном случае.

Создание файлов:

static FileStream File.Create(string path)

File.Create(string path, int bufSize)

File.Create(string path, int bufSize, FileMode) – открывает поток и с его помощью создаёт файл. Важно закрыть поток для дальнейшей работы с файлом через другие источники!

Удаление файлов:

static void File.Delete(string path) – удаляет файл по указанному пути.

Копирование файлов:

static void File.Copy(string source, string dest)

File.Copy(string source, string dest, bool canOverride) – копирует файл по указанному пути. Не позволяет перезаписать файл, если он уже существует без использования перегрузки.

FileInfo.CopyTo(string dest) – экземплярный аналог Copy.

Работа с Файлами

Перемещение файлов:

- **File.Move(string source, string dest)** – статический метод, перемещает файл по указанному пути, возможно с изменением названия.
- **FileInfo.MoveTo(string destination)** – экземплярный метод, перемещает файл по указанному пути, возможно с изменением названия.

Добавление текста в файл:

- **File.AppendAllText(string file, string content)**
File.AppendAllText(string file, string content, Encoding targ) – статический метод, открывает файл, *добавляет* в него переданный текст, затем закрывает файл. Если файла не существует, пробует создать его. Может принимать кодировку.
- **File.WriteAllText(string file, string content)**
File.WriteAllText(string file, string content, Encoding targ) – статический метод, аналог AppendAllText с одним ключевым отличием – перезаписывает содержимое файла во всех случаях.
Важно: **File.AppendText(string path)** – другой метод, создаёт объект StreamWriter для записи или создания текста в файл в кодировке UTF-8.

Работа с Файлами

Чтение текста из файла:

File.ReadAllText(string path)

File.ReadAllText(string path, Encoding) – статический метод, считывает весь текст из файла и возвращает его в виде единственной строки. Возможно указать кодировку.

File.ReadAllLines(string path)

File.ReadAllLines(string path, Encoding) – статический метод, аналог ReadAllText, позволяющий вернуть массив строк, каждая строка файла равна одной возвращённой строке.

ЗАДАЧИ

Задание 1

Выберите методы, содержащиеся в классе File (укажите все верные ответы):

- 1) Remove();
- 2) Move();
- 3) Copy();
- 4) Delete();
- 5) Create();

Задание 2

Выберите методы, содержащиеся в классе Directory (укажите все верные ответы):

- 1) Remove();
- 2) Move();
- 3) Copy();
- 4) DeleteDirectory();
- 5) CreateDirectory();

Задание 3

Выберите символы, которые НЕ допускаются в имени файлов (укажите все верные ответы):

*Учитывается только OS Windows

- 1) |
- 2) <
- 3) ; (точка с запятой)
- 4) %
- 5) @

Задание 4

Программа была запущена 3 раза, в результате выполнения 3-го запуска:

```
using System;
using System.IO;

class Program {
    static void Main() {
        int[] args = { 1, 2, 3, 4, 5 };
        for (int i = 0; i < 3; i++) {
            File.AppendAllText("args.txt",
args.GetValue(i).ToString());
        }
        Console.WriteLine(File.ReadAllText("args.txt"));
    }
}
```

на экран будет выведено (до 1-го запуска файла args.txt в корневом каталоге не было):

Примечание:
Если возникнет ошибка компиляции, введите: ***
Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---
Если возникнет ошибка исполнения или исключение, введите: +++

ЗАДАЧИ

Задание 5

В результате выполнения фрагмента программы:

```
using System;
using System.IO;

class Program {
    static void Main() {
        File.Create("test.txt");
        int x = 0;
        for (int i = 0; i < 10; i++) {
            x += i;
        }
        File.WriteAllText("test.txt", x.ToString());
        Console.WriteLine(File.ReadAllText("test.txt") + 10);
    }
}
```

на экран будет выведено:

Примечание:
Если возникнет ошибка компиляции, введите: ***
Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---
Если возникнет ошибка исполнения или исключение, введите: +++

Задание 6

Программа была запущена 3 раза, в результате выполнения 3-го запуска:

```
using System;
using System.IO;

class Program {
    static void Main() {
        int[] args = { 1, 2, 3, 4, 5 };
        for (int i = 0; i < 3; i++) {
            File.WriteAllText("args.txt",
args.GetValue(i).ToString());
        }
        Console.WriteLine(File.ReadAllText("args.txt"));
    }
}
```

на экран будет выведено (до 1-го запуска файла args.txt в корневом каталоге не было):

Примечание:
Если возникнет ошибка компиляции, введите: ***
Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---
Если возникнет ошибка исполнения или исключение, введите: +++

Ответы

Задание	Ответ
1	2345
2	25
3	12
4	123123123 (<i>AppendAllText добавляет текст</i>)
5	+++ (<i>попытка обратиться к файлу, открытому другим потоком</i>)
6	3

Методы в C#

В языке C# метод является именованным блоком кода, состоящем из заголовка и тела.

Заголовок метода состоит из: атрибутов, модификаторов, типа возвращаемого значения, имени и набора параметров в круглых скобках, перечисленных через запятую.

Сигнатура метода в C# учитывает: имя метода, количество параметров и их тип, модификаторы параметров (кроме `ref` и `out`, они эквивалентны), порядок их следования в списке. В C# внутри одного типа не может быть методов с одинаковой сигнатурой

Важно: тип возвращаемого значения и модификаторы метода не входят в сигнатуру.

Примеры с различной сигнатурой:

```
public int Example(int a, double b) {...}  
public int Example(double b, int a) {...}  
public int Example(ref int a, double b) {...}
```

Примеры с одинаковой сигнатурой:

```
public int BadExample(ref int a, ref double b) {...}  
public void BadExample(ref int a, ref double b) {...}  
public static void BadExample(ref int a, ref double b) {...}  
public void BadExample(out int a, ref double b) {...}
```

Аргументы и Параметры. `params`.

Важно разбираться в терминологии языка. Помните, что:

1) **аргументы** – это то, что вы передаёте в метод при его вызове.

2) **параметры** – локальные «псевдонимы», по которым вы будете обращаться к переданным параметрам.

Имена параметров могут совпадать с именами передаваемых им аргументов, но это необязательно.

params – модификатор параметров метод, позволяющий передать переменное число аргументов в виде одномерного массива.

При использовании params стоит помнить о его особенностях:

- Только один параметр может быть помечен модификатором params
- Параметр с params должен стоять в конце списка параметров, после него нельзя добавлять другие параметры.
- При вызове метода можно НЕ передавать params аргументов, тогда он просто будет нулевой длины.
- При вызове метода аргументы для params можно передавать как в виде массива, так и как перечисление значений через запятую.
- Нельзя создать метод, отличающийся только модификатором params у массива – из-за неоднозначности вызова возникнет ошибка компиляции.

Примеры Использования params

Объявление метода с params:

```
public static int Sum(int startValue, params int[] members)
{
    foreach (var item in members)
    {
        startValue += item;
    }
    return startValue;
}
```

Возможные варианты его вызова:

Sum(4); // params нулевой длины

Sum(4, 5, 6, 7, 8, 9); // 5, 6, 7, 8, 9 – params

int[] myArray = new int[] { 1, 2, 3, 4, 5 };

Sum(0, myArray); // с передачей инициализированного массива

Задание 1

Выберите пары методов с различной сигнатурой (укажите все верные ответы):

- 1) static int Meth(); - static void Meth();
- 2) void Meth(); - void Math();
- 3) public static int Meth(); - static int Meth();
- 4) static int Meth(ref int a); - static int Meth(int a);
- 5) int Meth(double a); - int Meth(int a);

Задание 2

В сигнатуру метода входит (укажите все верные ответы):

- 1) Имя метода.
- 2) Модификатор **static**.
- 3) Количество входных параметров метода.
- 4) Имена входных параметров метода.
- 5) Тип возвращаемого значения.

Задание 3

Выберите верные утверждения (укажите все верные ответы):

- 1) Параметр метода с модификатором **params** всегда должен быть последним в списке параметров метода.
- 2) Если в списке параметров метода присутствуют параметры с модификатором **ref** или **out**, то при вызове метода, с параметром, соответственно, надо указывать данные модификаторы.
- 3) В качестве аргумента с модификатором **params** можно передать как разное число аргументов, так и массив целиком.
- 4) При объявлении метода, можно инициализировать аргументы прямо в списке параметров.
- 5) Аргументы можно передать через определённое имя аргумента, в таком случае порядок не имеет значения.

ЗАДАЧИ

Задание 4

Выберите пары методов, имеющие различную сигнатуру (укажите все верные ответы):

- 1) void Meth(ref int a); - void Meth(out int a);
- 2) int Meth(); - public int Meth();
- 3) void Meth(int a); - void Meth(double a);
- 4) int Meth(int a); - int Meth(int a, int b);
- 5) void Meth(int[] a); - void Meth(params int[] a);

Задание 5

В результате выполнения фрагмента программы:

```
using System;

class Program {
    static void Main() {
        int[] args = { 1, 3, 5, 7, 9 };
        Arr(args);
        Console.WriteLine(args[2]);
    }

    public static void Arr(params int[] args) {
        Array.Reverse(args);
        for (int i = 0; i < args.Length; i++) {
            args[i] += i;
        }
    }
}
```

на экран будет выведено:

Примечание:
Если возникнет ошибка компиляции, введите: ***
Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---
Если возникнет ошибка исполнения или исключение, введите: +++

Ответ 1: **245**

Ответ 2: **13**

Ответ 3: **12345**

Ответ 4: **34**

Ответ 5: **7**

Ответы

Операторы Переходов

- За их неразумное использование дают бан по codestyle на СР, аккуратнее.
- **break** – выход из текущего цикла или switch.
- **continue** – прорабатывает на следующую итерацию цикла.
- **goto** – переводит выполнение кода к указанной именованной метке или блоку case. Является хорошим способом выхода из глубоко вложенных циклов.

Пример использования goto:

```
for (int i = 0; i < 100000; i++)  
{  
    Console.WriteLine(i);  
    goto Label;  
  
}  
Label:  
    Console.WriteLine("Вылезаем из норы");
```

- **return** – завершает выполнение метода/switch и возвращает значение.

const в C#

В языке C# модификатором **const** помечаются константы – неизменяемые именованные значения.

Важной особенностью констант является то, что их можно инициализировать только в той строке в которой они были объявлены, в отличие от `readonly` полей, которые можно инициализировать в конструкторах.

Стоит также отметить, что константы могут быть объявлены как локально, так и внутри типов.

Подобно полям, помеченным `static`, константы можно использовать в объекте без создания экземпляра, обращаясь к константе по имени её класса. При этом помните, что `static` и `const` – не одно и то же: в момент компиляции все использованные константы заменяются их значениями и не хранятся в памяти.

Важно: `const` не сочетается с модификаторами `static` и `readonly`.

Концепция ООП в C#

Как мы уже знаем, **C# – объектно-ориентированный язык программирования**. Для того чтобы получить рабочий код необходимо объявить хотя бы один класс.

На самом деле при создании класса мы по сути описываем новый ссылочный тип объекта.

Помните, что хоть понятия класс и объект тесно связаны, они не являются одним и тем же. **Класс** – описание того, каким должен быть объект данного типа. **Объект (экземпляр класса)** – конкретная сущность, созданная по правилам, описанным в классе.

Сам объект класса начинает существовать, когда вы непосредственно создаёте его при помощи new. В C# реализована автоматическая сборка мусора – объекты существуют до тех пор, пока на них указывает хотя бы одна ссылка.

Основными принципами объектно-ориентированного подхода являются: **инкапсуляция, полиморфизм, наследование** (иногда ещё отдельно выделяется **абстракция**).

Виды Членов Классов

Члены Данных	Функциональные Члены	
Поля	Методы	Операции
Константы	Свойства	Индексаторы
	Конструкторы	События
Деструкторы		

Члены
Классов в C#

Модификаторы Доступа

В C# существует 6 модификаторов доступа:

public – элемент полностью доступен из любой части кода

protected internal – элемент полностью доступен внутри текущей сборки, а также для всех наследников

protected – элемент доступен для наследников из любой сборки

internal – элемент доступен для любого класса в рамках текущей сборки

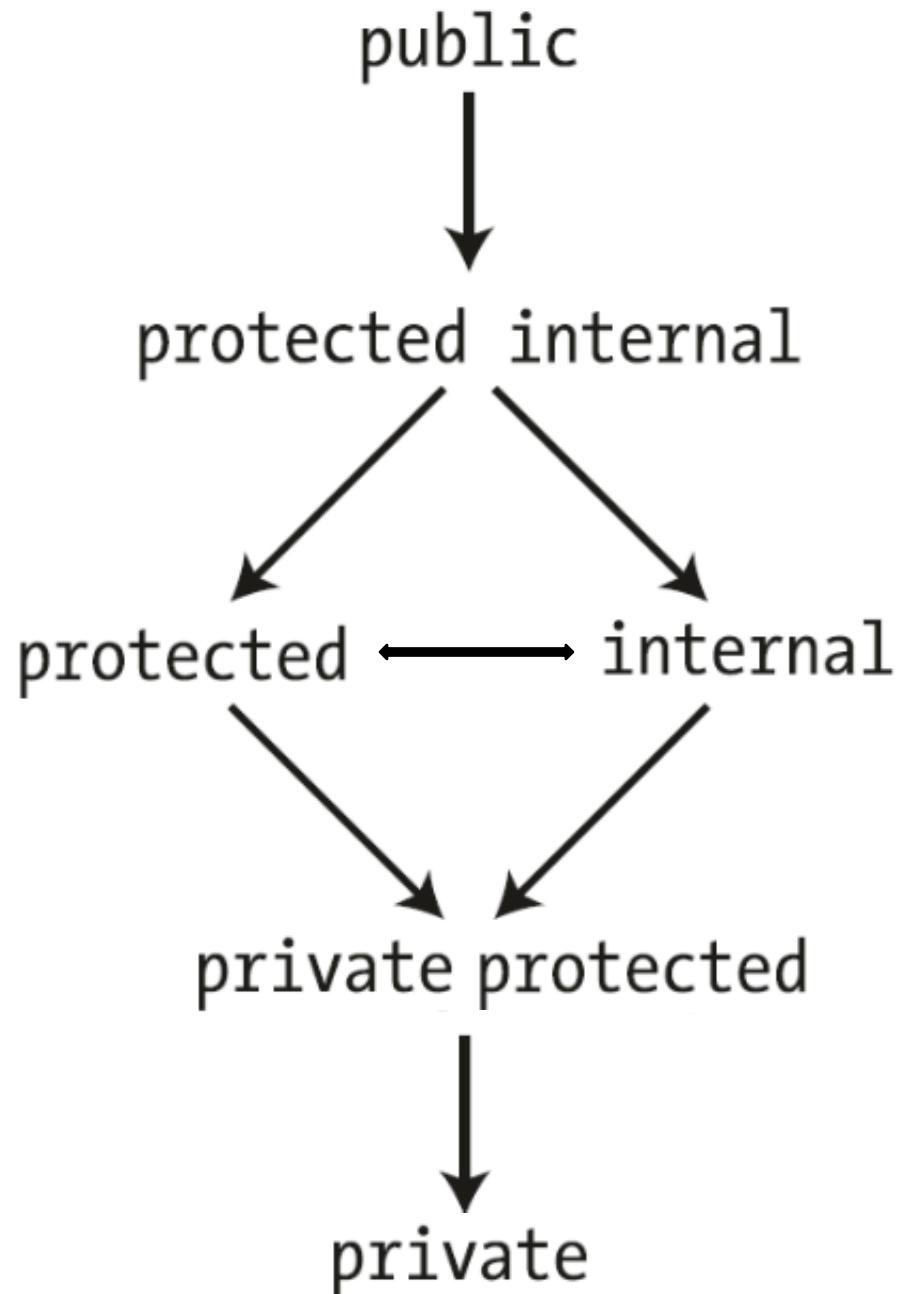
private protected – элемент доступен только для наследников текущей сборки

private – элемент доступен только в том типе, в котором он содержится

Помните, что все остальные модификаторы кроме **internal** и **public** в контексте классов применимы для вложенных типов.

Доступность Членов Классов

	Классы в Той Же Сборке		Классы в Других Сборках	
	Наследники	Не Наследники	Наследники	Не Наследники
private				
internal	✓	✓		
protected	✓		✓	
protected internal	✓	✓	✓	
private protected	✓			
public	✓	✓	✓	✓



*protected и internal не могут сужать доступ относительно друг друга.

Классы в C#

Порядок объявления класса: <Атрибуты> <модификаторы> <Имя>{ }

Помните, что порядок модификаторов класса не важен – public static и static public являются допустимыми конструкциями.

Важно: Без указания модификатора доступа по умолчанию все классы, вне зависимости от наличия пространства имён являются internal. При этом разрешается для невложенных классов указывать только модификаторы **public** и **internal**.

Все члены классов в C#, включая вложенные классы по умолчанию private.

Также, в отличие от С и С++ в языке C# не существует глобальных переменных и функций! Вы не можете создавать переменные и методы вне классов.

Члены Класса

Без указания модификатора static члены класса принадлежат каждомуциальному экземпляру. Для того, чтобы обратиться к любому нестатическому члену для начала необходимо создать объект соответствующего типа:

Пример:

```
public class Kotik {  
    public int murCounter = 0;  
}  
  
Class Program {  
    static void Main( ) {  
        // Kotik.murCounter = 10;           СЕ – murCounter не static  
        Kotik kis1 = new Kotik( ); // создадим 1ый объект  
        Kotik kis2 = new Kotik( ); // создадим 2ой объект  
        kis1.murCounter++; // значения murCounter для kis1  
        kis2.murCounter = 5; // kis2 не связаны – разные объекты  
    }  
}
```

Статические Члены Класса

Нестатический класс может содержать статические члены, причём работать с ними можно и при отсутствии экземпляра. Доступ к статическим членам всегда осуществляется через имя класса. Помните, что всегда существует только одна копия статического члена, общая для всех экземпляров.

Статические методы/свойства не могут обращаться к нестатическим полям и событиям в содержащем их типе. Также они не могут обращаться к переменным экземпляра объекта кроме случаев, когда они переданы в параметрах метода явно.

Важно: Статические методы можно перегружать, однако они несовместимы с `abstract`, `virtual` и `override`. Также нельзя комбинировать `static` и `const`.
Локальные переменные статическими быть не могут

Статические члены инициализируются перед первым доступом к ним и перед вызовом статического конструктора (если он определён в классе)

Static Поля. Пример

```
public class Dragon {  
    public static int communismGold = 100; // общее золото для всех драконов  
    public static int dragonCount = 0;  
    public Dragon( ) {  
        Dragon.communismGold -= 100; // золото у драконов общее – обращаемся через класс  
        Dragon.dragonCount++; // при создании каждого дракона увеличиваем количество  
    }  
}  
class Program {  
    static void Main( ) {  
        Dragon drago1 = new Dragon( ); // создадим 1ый объект  
        Dragon drago2 = new Dragon( ); // создадим 2ой объект  
    }  
}
```

Статические Классы

C# позволяет добавлять статические классы, экземпляры которых нельзя создавать. Статические классы могут содержать только статические члены, от них нельзя наследоваться, они неявно запечатанные.

По этим причинам, следует запомнить, что статические классы:

- Могут содержать только статический конструктор.
- Не могут содержать члены, помеченные модификаторами `protected internal`, `protected` и `private protected`.

Задание 1

В результате выполнения фрагмента программы:

```
using System;

class MyClass {
    public static int x = 10;
}

class Program {
    static void Main() {
        MyClass obj = new MyClass();
        obj.x = 5;
        Console.WriteLine(MyClass.x);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 2

В результате выполнения фрагмента программы:

```
using System;

class MyClass {
    int val = 10;
}

class Program {
    static void Main() {
        MyClass obj = new MyClass();
        obj.val++;
        Console.WriteLine(obj.val--);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

ЗАДАЧИ

Задание 3

Верно, что класс может быть помечен следующим модификатором доступа
(укажите все верные ответы):

- 1) public;
- 2) private;
- 3) internal;
- 4) protected;
- 5) protected internal;

Задание 4

Укажите номера строк кода, которые приведут к выводу в консоль числа 5:
using System;

```
class MyClass {
    public int x = 5;
    int y = 6;
    protected int z = 4;
    internal int w = 4;
    protected internal int t = 6;
}
```

```
class Program {
    static void Main() {
        MyClass obj = new MyClass();
        Console.WriteLine(obj.x); //1
        Console.WriteLine(obj.y - 1); //2
        Console.WriteLine(obj.z++); //3
        Console.WriteLine(obj.w++); //4
        Console.WriteLine(--obj.t); //5
    }
}
```

- 1) 1
- 2) 2
- 3) 3
- 4) 4
- 5) 5

Ответ 1: ***

Обращение к статическому полю через ссылку.

Ответ 2: ***

Поля по умолчанию private.

Ответ 3: 12345

Ответ 4: 15

Ответы

Свойства

<модификаторы> <тип возвр. значения> <Имя> { <get... set...>}

Свойство – член класса, позволяющий задавать некоторые правила определения и получения значений полей. Как правило, свойства каким-либо образом ограничивают доступ к непубличным полям, что делает их ключевым инструментом для инкапсуляции. Перегрузить свойства не получится в силу отсутствия параметров.

На самом деле свойство представляет из себя 2 специальных метода *get* и *set*, первый из которых должен возвращать значение типа свойства, второй позволяет присвоить значение при помощи контекстно-ключевого слова *value*. Вызвать аксессоры *get*/*set* явно нельзя.

Помните, что свойство может иметь только *get* или только *set*, что сделает его доступным только для чтения или только для записи.

Для *get* и *set* внутри могут быть заданы модификаторы доступа, более узкие, чем модификатор самого свойства. Однако, хотя бы один из модификаторов должен совпадать с заданным при определении свойства, в противном случае возникнет ошибка компиляции. Также, модификатор *get*/*set* не может иметь менее ограничивающий модификатор доступа – это тоже ошибка компиляции. Как и методы, свойства могут быть *abstract*, *virtual* или *static*.

Автореализуемые Свойства

<модификаторы> <тип возвр. значения> <Имя> { <get... set...>}

В определённых сценариях у нас возникает необходимость задавать присваивание/чтение через get и set без дополнительной логики. В таких случаях не нужно дополнительно создавать поля.

Вместо этого можно воспользоваться автоматически реализуемыми свойствами. Для этого вам при объявлении свойства необходимо объявить get и/или set без тела. Помните, что вы не можете комбинировать обычное свойство с автореализуемым.

На самом деле, компилятор предоставляет вам резервное поле при создании автоматически реализуемых свойств. По этой причине, у них тоже есть умалчиваемое значение.

Свойства. Пример

```
public class Hedgehog {  
    private int flightSpeed = 300; // Задаём логику определения значения при помощи свойств  
    public int Speed {  
        get { return flightSpeed / 3600; }  
        set {  
            if (value > 1000000)  
                throw new ArgumentOutOfRangeException("2 speed 4 u");  
            else flightSpeed = value;  
        }  
    }  
}  
class Program {  
    static void Main() {  
        Hedgehog Sonic = new Hedgehog(); // создадим 1ый объект  
        Console.WriteLine($"Скорость Соника составляет {Sonic.Speed}");  
    }  
}
```

Конструкторы

Конструктор – специальный функциональный член, позволяющий создать объект указанного типа. По этой причине: 1) конструктор не имеет типа возвращаемого значения 2) его имя обязательно должно совпадать с именем класса.

Конструкторы не наследуются и, как следствие, не могут быть abstract, virtual или override. При наследовании конструкторы автоматически вызываются от самого базового до наследника.

Если вы явно не создаёте конструкторов в классе, компилятор делает это неявно – создаётся public конструктор без параметров, задающий всем членам значения по умолчанию.

По сути конструктор тоже схож с методом: он также может иметь модификаторы доступа, может иметь параметры и перегружаться. Помните, что если вы явно объявили конструктор с параметрами, то компилятор не создаст конструктора по умолчанию, а попытка создать объект без передачи аргументов вызовет ошибку компиляции.

У конструкторов может быть инициализатор – с помощью base/this вы можете из конструктора вызвать специфический родительский или собственный конструктор. Также, при вызове вы в фигурных скобках можете инициализировать значения доступных полей значимого или ссылочного типа, свойствам.

Конструкторы. Пример.

```
public class StarWars {  
    private int budget;  
    public bool willBeAGoodMovie;  
    private bool lukeSkywalkerIsAlive;  
    public StarWars(int budget, bool lukeSkywalkerIsAlive) {  
        this.budget = budget; // так как имена параметров и полей совпадают, мы используем this  
        this.lukeSkywalkerIsAlive = lukeSkywalkerIsAlive; // для присваивания значений полям  
    } // именно создаваемого объекта. Подробнее об этом немного позже.  
    public StarWars() : this(100000, true){} // конструктор без параметров вызывает другую перегрузку  
}  
  
class Program {  
    static void Main() {  
        StarWars returnOfTheJedi = new StarWars() { willBeAGoodMovie = true};  
    }  
}
```



// инициализация доступного поля

Статические Конструкторы

C# позволяет создавать статические конструкторы. Помните, что они могут быть как в статическом, так и в нестатическом классах. Класс или структура может иметь только один статический конструктор. У статического конструктора не может быть модификаторов доступа или параметров – они вызываются только один раз автоматически при первом обращении к типу в коде.

В нестатическом классе может быть одновременно как статический, так и нестатический конструктор, причём при обращении к классу статический конструктор выполнится до экземплярного. Как следствие, мы получаем ситуацию, что при наследовании статические конструкторы вызываются в обратном порядке по сравнению с нестатическими – от наследника к базовому (при создании объектов, при обращении к статическим методам наследника это не так).

Помните об особенностях статических конструкторов:

- Могут задавать static readonly поля, в отличие от экземплярных конструкторов.
- Не могут напрямую работать с нестатическими полями.

Задание 1

В результате выполнения фрагмента программы:

```
using System;

class MyClass {
    public int x = 10;

    public MyClass(ref int x) {
        x = 5;
    }
}

class Program {
    static void Main() {
        int x = 7;
        MyClass obj = new MyClass(ref x);
        Console.WriteLine(obj.x - x);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 2

В результате выполнения фрагмента программы:

```
using System;
```

```
class MyClass {
    public static const int val = 10;
}
```

```
class Program {
    static void Main() {
        Console.WriteLine(MyClass.val ^ 5);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 3

В результате выполнения фрагмента программы:

```
using System;

class MyClass {
    public readonly int x = 10;

    public MyClass(int y) {
        x = y;
    }
}
```

```
class Program {
    static void Main() {
        int x = 7;
        MyClass obj = new MyClass(x);
        Console.WriteLine(obj.x + x);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 4

В результате выполнения фрагмента программы:

```
using System;
```

```
class MyClass {
    public static readonly int x = 10;

    public MyClass(int y) {
        x = y;
    }
}
```

```
class Program {
    static void Main() {
        int x = 7;
        Console.WriteLine(MyClass.x + x);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 5

В результате выполнения фрагмента программы:

```
using System;

class MyClass {
    public int x;
    public MyClass() : this("5", 5) {
        x -= 10;
    }

    public MyClass(int x) {
        this.x = x;
    }

    public MyClass(string line, int x) : this(int.Parse(line + x)) {
        this.x += int.Parse(line + 10);
    }
}

class Program {
    static void Main() {
        MyClass obj = new MyClass();
        Console.WriteLine(obj.x);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 6

Выберите верные утверждения (укажите все верные ответы):

- 1) В классе неявно всегда определяется конструктор по умолчанию.
- 2) Для вызова другого конструктора этого же класса требуется использование ключевого слова `this`.
- 3) Конструкторы НЕ могут быть непубличными.
- 4) Конструктор имеет тип возвращаемого значения.
- 5) Конструктор – функциональный член класса.

ЗАДАЧИ

Задание 7

Верно, что свойство (укажите все верные ответы):

- 1) Может быть перегружено.
- 2) Может иметь тип возвращаемого значения `void`.
- 3) Может иметь лишь один аксессор.
- 4) Обязательно должен содержать аксессор `get`.
- 5) Может иметь аксессоры с явно определёнными модификаторами доступа.

Ответ 1: 5

Ответ 2: ***

Попытка объявить const как static – так нельзя.

Ответ 3: 14

readonly полям можно присваивать значения в конструкторе.

Ответ 4: ***

static readonly поля можно определять в статическом конструкторе, но не в экземплярном.

Ответы

Ответ 5: 555

Ответ 6: 25

Ответ 7: 35

Ответы

Деструкторы

Формат объявления: `~<Имя типа>() { }`

Для того, чтобы определить деструктор необходимо написать перед именем класса символ «`~`». Также как и в случае с конструктором, деструктор имеет такое же имя, как и сам класс.

Деструктор можно создать только в классе. В структурах объявить деструктор нельзя. Деструкторы не имеют параметров или модификаторов, они не наследуются и не перегружаются.

В отличие от C++, вызвать деструктор явно нельзя. Он вызывается автоматически при удалении экземпляров сборщиком мусора. Так как сборка мусора осуществляется в недетерминированный момент времени, рекомендуется использовать **IDisposable**, о нём будет говорится позже в курсе программирования.

this this this



Ключевое слово **this** ссылается на текущий экземпляр класса, а также используется для первого параметра методов расширений.

Самые частые случаи применения this:

- Для работы с элементами с одинаковыми именами. Например, у вас в классе есть поле item, а в конструкторе вы объявляете параметр с таким же именем и типом. Для того, чтобы задать полю значение параметра нужно будет написать: this.item = item;
- Для передачи текущего объекта в качестве параметра в метод.
- Для объявления индексаторов.

Индексаторы

Формат объявления: <мод> <тип> this[<параметры>] { <get...> } set...> }

Индексаторы позволяют обращаться к экземплярам классов/структур также, как и элементы массивов – с помощью квадратных скобок. По сути работы индексаторы похожи на свойства: для работы они также используют аксессоры get/set, имеют непустой тип возвращаемого значения, на один из аксессоров get/set может быть установлено ограничение доступа.

Обращение к индексаторам осуществляется через экземпляры типа, они не бывают статическими и не могут быть автоматически реализованы. Кроме того, у индексатора обязан быть хотя бы один параметр, в противном случае возникнет ошибка компиляции. Параметры индексатора не могут иметь модификаторов ref/out, однако могут содержать params.

Индексаторы безымянны, при их объявлении вместо имени используется this. У индексаторов можно объявить набор параметров (**важно:** обязательно в квадратных скобках); их можно перегружать, а также делать virtual или abstract.

Индексы. Пример (Часть 1)

```
public class HSE_SE {  
    private string student;  
    private string lecturer; // создаём 3 поля, которые мы будем индексировать  
    private string seminarist;  
    public string this[int index] { // объявляем индексатор, принимающий индекс типа int  
        get {  
            switch (index) { // возвращаем соответствующий элемент по индексу или выбрасываем исключение  
                case 0: return student; case 1: return lecturer; case 2: return seminarist;  
                default: throw new ArgumentOutOfRangeException("Индекс вышел за границу перечисления.");  
            }  
        }  
        set {  
            switch (index) { // задаём соответствующий элемент по индексу или выбрасываем исключение  
                case 0: student = value; break; case 1: lecturer = value; break; case 2: seminarist = value; break;  
                default: throw new ArgumentOutOfRangeException("Индекс вышел за границу перечисления.");  
            }  
        }  
    }  
}
```

Индексы. Пример (Часть 2)

```
public class Program {  
    static void Main( ) {  
        HSE_SE group = new HSE_SE( ); // создаём объект  
        for (int i = 0; i < 3; i++) {  
            group[i] = Console.ReadLine( ); // заполняем поля введёнными строками  
        }  
        for (int i = 0; i < 3; i++) {  
            Console.WriteLine(group[i]); // возвращаем значение  
        }  
    }  
}
```

Задание 1

Выберите корректные объявления индексатора (укажите все верные ответы):

- 1) public int this[int x] { get; set; }
- 2) public double this[int x, int y] { get { return x; } }
- 3) public int this[int x, int y] { set { x = value; } }
- 4) public void this[int x] { set { x = 5; } }
- 5) public int this[] { get { return 5; } }

Задание 2

Верно, что индексатор (укажите все верные ответы):

- 1) Может быть перегружен.
- 2) Может иметь тип возвращаемого значения void.
- 3) Может иметь лишь один аксессор.
- 4) Обязательно должен содержать аксессор get.
- 5) Может быть статическим.

Задание 3

Свойство, в отличие от индексатора (укажите все верные ответы):

- 1) Может иметь имя.
- 2) Имеет тип возвращаемого значения.
- 3) Может быть перегружено.
- 4) Имеет аксессоры get и set.
- 5) Может быть инициализированным в объявлении.

ЗАДАЧИ

Задание 4

В результате выполнения фрагмента программы:

```
using System;

class Program {

    int z;
    public int this[int x, int y] {
        get { return z + x + y; }
        set { z = value + this[Math.Max(x, y)]; }
    }
    public int this[int i] {
        get { return i + z - this[i, i]; }
        set { z = value; }
    }

    static void Main(string[] args) {
        Program obj = new Program();
        obj[5, 10] = 20;
        Console.Write(obj[7, 3]);
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выводится ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Ответ 1: 23

Ответ 2: 13

Попытка объявить const как static – так нельзя.

Ответ 3: 15

readonly полям можно присваивать значения в конструкторе.

Ответ 4: 20

Ответы

Наследование как Часть Объектно-Ориентированной Парадигмы

Наследование – один из основных механизмов объектно-ориентированных языков программирования. Оно позволяет определять некоторые классы, которые используют, дополняют или переопределяют функционал своего родительского класса. Для наследования выполняется транзитивность, то есть, если В наследуется от А и С наследуется от В, то С наследуется от А.

Помните, что для вложенных классов доступны private члены – так как они сами находятся формально внутри класса.

Есть вещи, которые не наследуются в C# – конструкторы и деструкторы.

Чтобы наследоваться от класса нужно указать после имени с двоеточием имя того класса, от которого вы хотите наследоваться.

Важно: множественное наследование запрещено в C#. Также, нельзя наследоваться взаимно, то есть если класс А наследует класс В, то одновременно В не может наследовать А.

Наследование и Конструкторы

```
class Program {  
    static void Main() {  
        B b = new B();  
    }  
}
```

ВЫВОД:

B will get static!
A will get static!
A is here!
B is here!

Хотя конструкторы и деструкторы в C# не наследуются, стоит помнить о важной особенности языка: перед вызовом экземплярного конструктора самого объекта снизу вверх вызываются все конструкторы родителей, начиная с Object (помните также о статических конструкторах...).

Со статическими конструкторами и деструкторами ситуация наоборот – они вызываются сверху вниз – от ребёнка по порядку к Object. Для статических конструкторов несложно объяснить это тем, что они вызываются при первом обращении к классу.

Пример:

```
public class A {  
    public A() { Console.WriteLine("A is here!"); }  
    static A() { Console.WriteLine("A will get static!"); }  
}  
public class B : A {  
    public B() { Console.WriteLine("B is here!"); }  
    static B() { Console.WriteLine("B will get static!"); }  
}
```

Наследование от Object. Методы Object.

Все классы в C# неявно наследуются от Object – его методы всегда доступны в любом классе. От dynamic наследоваться нельзя. Помните, что для вложенных классов доступны private члены – так как они сами формально внутри класса.

Методы Object:

public virtual string ToString() – возвращает строку – полное имя типа указанного объекта.

public virtual bool Equals(object)

public static bool Equals(object1, object2)

public static bool ReferenceEquals(object1, object2) – все три по умолчанию возвращают true, если ссылки указывают на один и тот же объект и false в противном случае.

protected object MemberwiseClone(object) – возвращает неполную копию переданного объекта. То есть, все значимые элементы нового объекта копируются (как и сам объект), однако все ссылочные члены всё также будут ссылаться на объекты копируемого объекта.

public virtual int GetHashCode() – возвращает результат хэш функции по умолчанию.

Задание 1

Про наследование верно (укажите все верные ответы):

- 1) Класс может унаследовать несколько классов, помимо `Object`.
- 2) Классы могут иметь взаимное наследование (производный класс наследует базовый и базовый наследует производный одновременно).
- 3) При скрытии поля или метода обязательно нужно использовать ключевое слово `new`.
- 4) Все классы унаследованы от `Object`, вне зависимости от того, `sealed` ли класс или нет.
- 5) Можно унаследоваться от статического класса.

Задание 2

В результате выполнения фрагмента программы:

```
using System;

class A {
    static A() {
        Console.WriteLine("SA");
    }

    public A() {
        Console.WriteLine("A");
    }
}

class B : A {
    static B() {
        Console.WriteLine("SB");
    }

    public B() {
        Console.WriteLine("B");
    }
}

class Program {
    static void Main() {
        A a = new B();
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выведется ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 3

Укажите строки кода, вставка которых вместо пропуска в программу:

```
class A { }
class B : A { }
class C : A { }
class D : C { }

class Program {
    static void Main() {
    }
}
```

НЕ вызовет ошибок компиляций:

- 1) A a = new C();
- 2) B b = new D();
- 3) B b = (B)new C();
- 4) D d = (D)new C();
- 5) A a = new A();

Задание 4

Укажите строки кода, которые вызовут ошибки компиляции или исключение:

```
class A { }
class B : A { }
class C : A { }
class D : C{ }
class E : B{ }

class Program {
    static void Main() {
        A a = new B(); //1
        C c = new E(); //2
        A e = (A)new E(); //3
        C d = (A)new D(); //4
        B b = (B)new A(); //5
    }
}
```

- 1) 1
- 2) 2
- 3) 3
- 4) 4
- 5) 5

Задачи

Ответ 1: 4

Ответ 2: **SBSAAB**

В начале отработают статические конструкторы, затем – экземплярные.

Ответ 3: 145

Ответ 4: 245

Ответы

Ковариантность типов в C#

C# позволяет размещать объекты классов наследников по ссылкам типа родительского класса. Это называется **ковариантностью**. При этом помните, что вы не можете положить объект класса родителя в массив типа дочернего класса (любая кошка – животное, но не любое животное – кошка). То есть, если вы объявляете ссылку типа A, то она может указывать на объект типа B, где B – наследник A. В обратную сторону это не работает – возникнет ошибка компиляции.

Важно: Помните, что из ссылки типа A у вас не будет доступа к функционалу членов класса B (кроме переопределённых через override).

Помните, что в классы-наследники вы можете добавлять методы, которые повторяют сигнатуру метода родителя и поля, с такими же именами, как и родителей – без виртуальности такие действия будут приводить к скрытию (сокрытию) соответствующих членов..

Контравариантность в C# поддерживается только в отдельных случаях для обобщённых типов и делегатов, что будет позже рассматриваться в курсе.

Экранирование (Сокрытие) Членов Класса в C#

Процесс создания члена в классе наследника с таким же именем, что и у родителя называется экранированием. По сути при экранировании мы скрываем члены базового класса и заменяем их версией дочернего класса. Если не указано явно, компилятор выдаст предупреждение, однако скрытие будет работать. Для явного экранирования в заголовке члена используйте new. На заметку: new и override не сочетаемы.

Скрытие членов может иметь одну из следующих форм:

- Поля, константы, свойства и типы, введённые в классе/структуре, скрывают члены родителя с таким же именем. Причём член, скрывающий родительский может иметь другой тип (например, string вместо int). Однако, есть и ограничения мы не можем скрыть метод одним из перечисленных членов. При попытке сделать это в классе одновременно будет и новый член, и метод родителя.
- Метод, введённый в классе или структуре, скрывает свойства, константы, поля и типы с тем же именем в базовом классе. Кроме того, он также скрывает все методы базового класса, имеющие такую же сигнатуру.
- Индексатор, представленный в классе или структуре, скрывает все индексаторы базового класса, имеющие одинаковую сигнатуру.

Важно отметить, что в случае с экранированием при создании ссылки родительского типа и вызове метода, будет вызываться версия родительского класса.

base и sealed

Ключевое слово `base` используется для получения доступа к членам класса-родителя: а) при вызове переопределённого в наследнике метода б) при выборе определённого конструктора базового класса при создании экземпляров производного в) при обращению к одноимённым членам базового класса при использовании склонения.

Помните, что доступ к базовому классу разрешён только в конструкторе, методе экземпляра или аксессоре нестатического свойства.

Важно: использование `base` со статическими членами невозможно, возникнет ошибка компиляции.

Модификатор `sealed`: а) запрещает наследоваться от класса б) запрещает осуществлять дальнейшее переопределение методов с помощью `override`.

virtual и override в Языке C#

По принципу работы виртуальные методы очень похожи на экранированные, однако у них есть ключевое отличие, благодаря которому в C# возможна реализация полиморфизма. При вызове метода наследника по ссылке родительского типа для виртуальных методов используется корректное переопределение. Для того, чтобы объявить метод переопределяемым наследниками используется ключевое слово **virtual**, а для переопределения – ключевое слово **override**.

override-члены сами по себе тоже **virtual**, их тоже можно переопределять. Вызов подходящей версии метода наследника для виртуальных методов не зависит от типа ссылки родителя – всегда будет вызываться самая близкая к наследнику.

Виртуальными в C# могут быть: методы, свойства, события, индексаторы, причём модификатор **virtual** не сочетаем с **static**, **abstract**, **private** и **override**. Вы можете объявить новый виртуальный метод во время экранирования при помощи комбинации **new** и **virtual**.

Абстрактные Классы в C#

Порой в C# возникает необходимость в классе описать общую категорию признаков каких-либо вещей. Например, вы хотите создать класс мебель, однако не разрешается создавать объекты типа мебель в общем виде.

Для реализации такого поведения можно использовать **абстрактные классы**, члены которых могут не иметь реализаций. Экземпляры абстрактных классов создавать запрещается, а любой неабстрактный наследник обязан реализовать весь функционал абстрактных родителей.

Абстрактные члены неявно виртуальные. Модификатором `abstract` могут быть помечены методы, свойства, индексаторы и события. Помните, что `sealed` и `abstract` друг с другом не сочетаемы, так как в комбинации дают логически бессмысленную конструкцию.

Исключения

Исключения в C# тоже являются объектами. Главное, что делает объект исключением – это наследование от **System.Exception** или любого из его производных типов.

Главной особенностью исключений является то, что их можно выбрасывать при помощи оператора `throw` (в отличие от языка C++, в котором `throw` может использоваться с произвольными типами).

Исключения могут создаваться средой выполнения (CoreCLR), платформой .NET Core, сторонними библиотеками или кодом самого приложения.

Зачастую исключения создаются не самим методом, который вызывается в вашем коде, а одним из последующих в стеке вызовов. В этом случае CoreCLR разворачивает стек полностью в поисках обработчика соответствующего типа. Если подходящий блок `catch` не будет обнаружен во всём стеке вызовов, CoreCLR досрочно завершит процесс и выведет сообщение об ошибке.

В библиотеках обработка исключений зачастую – не лучшая идея).

Предопределённые Исключения

ArgumentException	Недопустимое значение при вызове метода.
ArithmeticException	Базовый класс для исключений, которые возникают при выполнении арифметических операций. Например, <code>DivideByZeroException</code> или <code>OverflowException</code> .
ArrayTypeMismatchException	Посыпается при попытке присвоить элементу массива значение, тип которого не совместим с типом элементов массива.
DivideByZeroException	Посыпается при попытке деления на ноль целочисленного значения.
FormatException	Несоответствие параметра спецификации форматирования.
IndexOutOfRangeException	Посыпается при выходе индекса массива за пределы граничной пары (индекс отрицателен или больше верхней границы).
InvalidCastException	Посыпается при неверном преобразовании от базового типа или базового интерфейса к производному типу.
NullReferenceException	Посыпается при попытке применить ссылку со значением <code>null</code> для обращения к объекту.
OutOfMemoryException	Посыпается при неудачной попытке выделить память с помощью операции <code>new</code> (недостаточно свободной памяти).
OverflowException	Посыпается при переполнениях в арифметических выражениях, выполняемых в контексте, определённом ключевым словом <code>checked</code> .
RanException	Несоответствие размерностей параметра и аргумента при вызове метода.
StackOverflowException	Посыпается при переполнении рабочего стека. Возникает, когда вызвано слишком много методов, например, при очень глубокой или бесконечной рекурсии.
TypeInitializationException	Посыпается, когда исключение посылает статический конструктор и отсутствует обработчик исключений (блок <code>catch</code>) для перехвата исключений.

Свойства Исключений

Свойство	Тип	Описание
Message	string	Сообщение содержит описание источника ошибки
StackTrace	string	Содержит информацию о том, где возникло исключение
InnerException	Exception	Если текущее исключение было вызвано другим исключением, это свойство будет содержать ссылку на предыдущее.
Source	string	Если исключение было вызвано вне приложения, содержит информацию о сборке, в которой оно возникло.

Обработка Исключений

Для обработки исключений необходимо поместить код, который может вызвать исключение внутрь блока **try { }**. Если внутри блока try возникает исключение, то выполнение кода в нём тут же остановится, а CLR начнёт поиск соответствующего обработчика.

Блок try не может быть определён без хотя бы одного catch или блока finally.

Помните, что код внутри блока finally выполнится всегда, вне зависимости от того, возникло ли при выполнении блока finally исключение.

Существует 4 различных формы обработчика исключений:

- **catch { // действия }**
- **catch (<Тип исключения>) { // действия }**
- **catch (<Тип исключения> <Имя>) { // действия }**
- **catch (<Тип исключения> <Имя>) when (<спецификация>){// действия}**

Особенности Работы с Исключениями

Помните, что также, как и в блоке switch, обработчики исключений должны идти от более частного случая к более общему. Например, если вы попытаетесь после обработчика catch для Exception указать catch NullReferenceException, возникнет ошибка компиляции.

При наличии нескольких обработчиков исключений отработает только самый конкретный. Также, ничто не запрещает внутри блоков try-catch использовать return, continue, break или throw.

continue полезно использовать внутри цикла после возникновения исключения, чтобы повторить какое-либо действие после возникновения ошибки. Блок finally является одним из наиболее подходящих мест выполнения очистки ресурсов.

Важно: избегайте сценариев, когда в блоках catch/finally повторно выбрасывается исключение при наличии существующего: при таком сценарии объект исключения теряется, заменяясь новым.

C# допускает использование throw без ссылки на исключение внутри блока catch (но не finally), что позволяет пробросить исключение дальше

Создание Классов Исключений

Вы можете без затруднений создать собственный класс исключения, отметив его наследником `Exception` или одного из производных `Exception`.

Существует стандартное правило создания классов исключений – так как главное в исключении – его имя, на конце имён типов исключений пишут `Exception`. Вы можете быстро реализовать исключение при помощи синтетиков Visual Studio.

Основной функционал исключения – реализация 4 конструкторов базового класса:

- `public <Имя>()` – инициализирует новый экземпляр исключения.
- `public <Имя>(String message)` – инициализирует новый экземпляр исключения с указанным сообщением.
- `public <Имя>(SerializationInfo, StreamingContext)` – инициализирует новый экземпляр исключения с сериализованными данными. Предназначается для реализации поддержки обработки исключений в различных процессах.
- `public <Имя>(String message, Exception innerException)` - инициализирует новый экземпляр исключения с ссылкой на вложенное исключение, вызвавшее данное.

Выбрасывание Исключений

Для того, чтобы выбросить новое исключение во время исполнения программы, используется ключевое слово **throw**. Если вы генерируете новое исключение, используйте new.

Обратите внимание, что вы можете прородить исключение на уровень выше при помощи throw; без аргументов на уровень выше.

При попытке использовать пустой throw вне блока catch возникнет ошибка компиляции.

Пример:

```
Class Program {  
    public static void Main() {  
        try {  
            throw new Exception("RIP"); // кидаем  
        }  
        catch (Exception ex) {  
            Console.WriteLine(ex.Message); // сообщаем  
        }  
    }  
}
```



FINITO
LA
COMÉDIE

ВСЕМ СЛАСИ БО ЗА ВНІМАННЯ!