

# Экзамен АиП C Sharp

## C#

### 1. C#. Платформа .NET. Процесс выполнения программы.

**.NET** - это кроссплатформенный фреймворк от компании Microsoft, ранее известная как .NET Framework, когда она ещё была ориентированно только на ОС Windows. Она поддерживает множество языков, главный из которых **C#**, разработанный специально для неё.

Для выполнения кода в .NET служит **CLR** (Common Language Runtime, общезыковая исполняющая среда). Она выполняет **CIL** (Common Intermediate Language, байт-код) - специальный промежуточный язык, который представляет собой "высокоуровневый ассемблер" виртуальной машины .NET. В него компилируются программы, написанные на .NET совместимых языках, получаются исполняемые файлы в формате `exe` или `dll`. При выполнении программы происходит преобразование CIL кода в машинный код, так называемая **Just-In-Time (JIT)** компиляция. Производительность повышается за счёт того, что во время выполнения компилируется лишь часть кода, к которой происходит обращение.

### 2. C#. Структура программы. Пространство имен. Сборка.

#### Структура программы

Базовая структура программы на языке C# включает в себя главный класс и статическую функцию `Main`, принимающую в качестве параметра массив аргументов командной строки:

```
using System;
namespace program
{
    class Program
    {
        public static void Main(string[] argc)
        {

        }
    }
}
```

В новых версиях появилась поддержка операторов верхнего уровня, что позволяет писать код программы не в функции `Main`, а сразу в файле, главный класс и функция `Main` тогда создаются автоматически.

Также каждый проект включает файл проекта с расширением **.csproj**, который содержит описание свойств проекта.

#### Пространства имён

Пространства имён (в примере выше - `program`) используются для объединения кода в логические блоки и предотвращения конфликтов имён. Пространство имён объявляется с помощью специального ключевого слова и последующего блока кода: `namespace <имя> { ... }`. Обращаться к членам пространства имён можно с помощью оператора точки: в примере выше полное имя функции `Main` будет `program.Program.Main`. Чтобы не обращаться к объектам по полному имени, можно подключить пространство имён с помощью директивы `using`. В примере выше так подключается пространство

имён `System`. Пространства имён допускают вложенность. Можно объявить пространство имён для всего файла, добавив в начале: `namespace <имя>;`, оно применится к нему целиком.

## Сборка

Сборка - это файл, который содержит программу или библиотеку кода, элементы управления, ресурсы или другую информацию, которая может быть использована программой. Сборка создается как единый проект и компилируется в исполняемый файл (`exe`) или динамическую связанную библиотеку (`dll`). Каждая сборка включает в себя манифест, метаданные типов, код приложения и ресурсы. Атрибуты сборки, такие как версия, название, автор и прочая информация о продукте, содержатся в специальном файле `AssemblyInfo.cs`.

## 3. C#. Типы данных. Различия типов-значений и типов-ссылок.

Базовые типы в языке C#:

- **bool** - хранит значение `true` или `false`, представлен системным типом `System.Boolean`
- **byte** - хранит целое число от 0 до 255, системный тип - `System.Byte`
- **sbyte** - знаковый байт, целое число от -128 до 127, системный тип - `System.SByte`
- **short** - знаковое целое число от -32768 до 32767, занимает 2 байта, системный тип - `System.Int16`
- **ushort** - беззнаковое целое число от 0 до 65535, 2 байта, системный тип - `System.UInt16`
- **int** - знаковое целое число от -2147483648 до 2147483647, занимает 4 байта, системный тип - `System.Int32`, тип по умолчанию для численных литералов
- **uint** - беззнаковое целое число от 0 до 4294967295, занимает 4 байта, системный тип - `System.UInt32`
- **long** - знаковое целое число от -9223372036854775808 до 9223372036854775807, 8 байт, системный тип - `System.Int64`
- **ulong** - беззнаковое целое число от 0 до 18446744073709551615, 8 байт, системный тип - `System.UInt64`
- **float** - число с плавающей точкой от  $-3.4 \cdot 10^{38}$  до  $3.4 \cdot 10^{38}$ , 4 байта, системный тип - `System.Single`
- **double** - число с плавающей точкой от  $\pm 5.0 \cdot 10^{324}$  до  $\pm 1.7 \cdot 10^{308}$ , 8 байт, системный тип - `System.Double`
- **decimal** - десятичное дробное число, без десятичной запятой имеет значение от  $\pm 1.0 \cdot 10^{-28}$  до  $\pm 7.9228 \cdot 10^{28}$ , 16 байт, системный тип `System.Decimal`
- **char** - одиночный символ в кодировке Unicode, 2 байта, системный тип - `System.Char`, тип по умолчанию для символьных литералов
- **string** - набор символов Unicode, системный тип - `System.String`, тип по умолчанию для строковых литералов
- **object** - может хранить значения любого типа данных, занимает 4 байта на 32-разрядной системе и 8 на 64-разрядной, системный тип - `System.Object`, являющийся базовым для всех других типов и классов .NET.

В C# возможна неявная типизация с использованием ключевого слова `var` (аналог `auto` в C++):

```
// можно писать так:  
var hello = "hello world";  
var a = 1;  
var b = 12.345;  
var c = 'A';
```

```
// но так нельзя!  
var n = null;
```

Все типы данных подразделяются на типы-значения и типы-ссылки в зависимости от того, как для них происходит организация памяти. Память подразделяется на стек и кучу.

- Объекты типов-значений размещаются в стеке, по адресу непосредственно хранится само значение переменной или параметра функции. К таким типам относятся все целочисленные типы, типы чисел с плавающей запятой, типы `decimal`, `bool`, `char`, перечисления `enum` и структуры `struct`. (но поля типов-значений объектов располагаются в куче)
- Объекты ссылочных типов хранятся в куче, в стеке на них хранится лишь ссылка. Ссылочными типами являются `object`, `string`, все классы, интерфейсы и делегаты.

Процесс копирования разных типов различается: типы-значения копируются по значению, а для типов-ссылок копируется лишь ссылка на объект, то есть 2 ссылки начинают указывать на одну и ту же область в памяти. Это также относится к более сложным ситуациям, когда структура содержит поле с типом класса и при копировании этой структуры, получится, что все поля с типами-значениями скопировались по значению, а это поле класса по ссылке, то есть 2 разных объекта будут иметь поля с ссылкой на один и тот же объект. Таким образом, ссылки работают как указатели и дают те же возможности.

Стоит учитывать, что при передаче объекта класса в функцию через параметры, передаётся копия ссылки на исходный объект, то есть функция получает к нему доступ и может изменить его поля. При этом функция не может изменить сам объект, так как передана лишь копия ссылки. Чтобы это сделать, нужно использовать ключевое слово `ref`, тогда станет возможно, например, создание нового другого объекта и сохранение его по исходной ссылке.

## 4. С#. Литералы. Примеры. Переменные. Примеры. Область действия переменной.

### Литералы

Литералы (константы) - неизменяемые значения. Их можно передавать переменным в качестве значения. Виды литералов в С#:

- логические литералы: `true` и `false`
- целочисленные литералы:
  - в десятичной форме: `1`, `-2`, `123`
  - в двоичной форме: `0b11`, `0b1011`, `0b100001`
  - в шестнадцатеричной форме: `0x0A`, `0xFF`, `0xA1`
- вещественные литералы:
  - в десятичной форме: `12.34`, `-0.28`
  - в экспоненциальной форме: `3.2e3` ( $3.2 \cdot 10^3 = 3200$ ), `1.2E-1`
- символьные литералы: `'A'`, `'\n'`, `'\x75'`, `'\u0420'`
- строковые литералы: `"hello world"`
- `null` - ссылка, которая не указывает ни на какой объект

### Переменные

Для хранения данных в программе применяются переменные. Переменная представляет именованную область памяти, в которой хранится значение определенного типа. Переменная имеет тип, имя и значение. Тип определяет, какого рода информацию может хранить переменная. Чтобы объявить переменную, нужно указать её тип и имя, например, переменная целого типа с именем `a`: `int a;`,

строка `str: string str;`

Имя переменной может содержать любые цифры, буквы и символ подчёркивания, при этом первый символ должен быть буквой или подчёркиванием. В нём не должно быть пробелов и знаков пунктуации и оно не должно являться ключевым словом.

Чтобы инициализировать переменную, ей нужно присвоить значение:

```
int a; // объявляем переменную целого типа
a = 5; // присваиваем значение
char c = 'a'; // можно объявить переменную и сразу же инициализировать её
string name = "Max";
```

Для переменных существует модификатор `const`, который запрещает изменение значения после инициализации (она при этом обязательна при объявлении переменной):

```
// константы принято называть большими буквами
const int SIZE = 10; // объявление константы с значением 10
SIZE = 5; // ошибка!!! константы нельзя изменить
```

## Области действия (видимости) переменных

Каждая переменная существует только в определённом контексте:

- Контекст класса - переменные, объявленные на уровне класса, доступны во всех его методах, их называют глобальными или полями класса.
- Контекст метода - переменные, объявленные на уровне метода, являются локальными и доступны только в рамках этого метода.
- Контекст блока кода - переменные, объявленные на уровне блока кода, также являются локальными и доступны только в этом блоке кода.

```
class Obj
{
    int n = 10; // переменная уровня класса
    // в этом месте доступна только переменная n
    public void Print()
    {
        char c = 'A'; // локальная переменная метода
        // в этом месте доступны переменные n и c
        { // блок кода
            string str = "hello"; // локальная переменная блока кода
            // здесь доступны все 3 переменные: n, c и str
        }
    }
    public void Save()
    {
        // здесь доступно только поле класса n
    }
}
```

## 5. C#. Выражения. Преобразование типов при выполнении операций. Примеры.

Выражение - правило вычисления значения. Состоит из операндов (переменные, литералы и поименованные константы) и операций. По приоритету:

1. Первичные: `.`, `()`, `[]`, `x++`, `x--`, `new`, `typeof`, `checked`, `unchecked`
2. Унарные: `+`, `-`, `!`, `~`, `++x`, `--x`, `(<Тип>)`
3. Мультипликативные: `*`, `/`, `%`
4. Аддитивные: `+`, `-`
5. Сдвига: `<<`, `>>`
6. Отношения порядка и проверки типа: `<`, `>`, `≤`, `≥`, `is`, `as`
7. Проверка на равенство: `==`, `≠`
8. Поразрядные логические: `&`, `^`, `|`
9. Условные логические: `&&`, `||`
10. Условная операция: `?:`
11. Присваивания: `=`, `*=`, `/=`, `%=`, `+=`, `-=`, `<=<=`, `>=>=`, `&=`, `^=`, `|=`

Если операнды, участвующие в операции имеют разные типы, то неявно выполняются преобразования типов, исключающие потерю данных, то есть к более длинным типам и от целых чисел к вещественным: `char`  $\rightarrow$  `int`  $\rightarrow$  `long`  $\rightarrow$  `float`  $\rightarrow$  `double`. Преобразования, которые могут приводить к потере данных, например, вещественных чисел в целые, требуют явного преобразования типов. Явное преобразование можно выполнить:

- с помощью операции `(<типа> <имя переменной> :`

```
int k = 100;
char c = (char)k;
long b = 300;
int a = (int)b;
byte d = (byte)a; // произойдёт потеря данных
```

- соответствующим методом класса `Convert` :

```
int i = Convert.ToInt32("123");
bool b = true;
double d = Convert.ToDouble(b);
```

Если преобразование невозможно, то генерируется исключение `FormatException`

## 6. C#. Ввод-вывод консольного приложения. Примеры.

### Вывод

Для выполнения операций вывода используют статические методы класса `System.Console`, перегруженные для всех стандартных типов данных и форматного вывода:

- `Console.Write()` – вывод строки без перехода на следующую;
- `Console.WriteLine()` – вывод строки с переходом на следующую.

```
Console.Write("Целое число: ");
Console.WriteLine(123);
```

Эти функции принимают строку и список аргументов для форматирования. Синтаксис форматирования: `{<номер аргумента>[, <мин. ширина поля>]:(<спецификатор формата[<число>]> | <шаблон>)]}`

```
Console.WriteLine("Имя: {0}, возраст: {1}, рост: {2}", "Петя", 14, 170);
```

## Спецификаторы форматирования:

- C или c - форматирование как национальной валюты
- F или f - форматирование как число с фиксированной точкой, можно указать число знаков после запятой
- G или g - форматирование в общем виде
- P или p - форматирование как процента
- X или x - форматирование как шестнадцатеричного числа

```
Console.WriteLine("{0:C}", 10);
Console.WriteLine("{0:F2}", 10); // 2 знака после запятой
Console.WriteLine("{0:G}", 12.234M); // Decimal
Console.WriteLine("{0:P}", 0.97); // 97,00 %
Console.WriteLine("{0:X}", 12754); // 31D2
```

## Шаблоны форматирования:

- # - вставка значащей цифры
- 0 - вставка значащей цифры или незначащего нуля

```
Console.WriteLine("{0,7:000.###}", 12.345); // 012.345
```

Возможна интерполяция строк, в которую можно вставлять интерполированные выражения:

{<выражение>[, <длина>]}[:<формат>]}. Длина - константа, если она отрицательная, используется выравнивание по левому краю. Интерполированная строка обозначается знаком \$" ... "

```
Console.WriteLine($"Результат: {Res}");
Console.WriteLine($"|{"Left",-7}|{"Right",7}|");
Console.WriteLine($"{{Math.PI,10:F5}}");
```

## Ввод

Для выполнения операций вывода используют статические методы класса `System.Console`:

- `Console.ReadLine()` - ввод строки типа `string`, включая перенос строки
- `Console.Read()` - возвращает код введенного символа типа `int` или `-1`, если значащих символов в потоке нет

```
string s = Console.ReadLine();
char ch = (char)Console.Read();
Console.ReadKey(); // ввод символа или функциональной клавиши
```

Преобразование введенной строки в число осуществляют:

- методы класса `Convert`, возвращающие `FormatException` в случае ошибки
- методы `Parse` каждого числового класса, возвращающие `FormatException` в случае ошибки
- методы `TryParse`, при ошибке возвращающие `false` в случае ошибки

```
int a = Convert.ToInt32("123");
int b = int.Parse("123");
double c = int.Parse("12.345"); // разделитель определяется операционной системой
```

```
int d;  
bool res = int.TryParse("123", out d);
```

## 7. C#. Исключения. Примеры.

Исключения - это аварийные ситуации, возникающие в процессе выполнения программы, например, деление на 0, переполнение, невозможность преобразования строки в число и т.д.

C# позволяет обрабатывать синхронные исключения, которые могут возникнуть в конкретных местах программы, когда какая-либо её часть не смогла выполнить то, что от неё требовалось. Они генерируются средой выполнения или предусматриваются программистом в опасных местах с помощью оператора `throw`. Классы исключений в C# организованы в иерархию, обработчики родительских классов перехватывают исключения всех производных классов. Собственные классы исключений следует наследовать от класса `ApplicationException`.

Для перехвата и обработки исключений используется конструкция `try ... catch`:

```
try <Оператор>  
catch ([<Тип исключения> [<Имя>]]) <Оператор>  
{catch ([<Тип исключения> [<Имя>]]) <Оператор>}  
[finally <Оператор>]
```

- `catch(<Тип>)` - перехватывает все исключения заданного и производных классов
- `catch(<Тип> <Имя>)` - то же самое, но позволяет передать объект исключения в обработчик как параметр
- `catch` - обрабатывает любые исключения
- `finally` - выполняется, если в блоке `try` не возникло исключений

Пример:

```
int[] nums = { 1, 2, 3, 4, 5, 6, 7};  
Console.WriteLine("Введите индекс от 0 до 6: ");  
try {  
    int i = int.Parse(Console.ReadLine());  
    Console.WriteLine(nums[i]);  
} catch (IndexOutOfRangeException) {  
    Console.WriteLine("Неверный индекс!");  
} catch (FormatException) {  
    Console.WriteLine("Введено не число!");  
} catch {  
    Console.WriteLine("Неизвестная ошибка");  
} finally {  
    Console.WriteLine("Ошибок не возникло");  
}
```

Класс `Exception` содержит полезные свойства:

- `Message` - текстовое описание ошибки
- `Source` - имя объекта, сгенерировавшего ошибку
- `TargetSite` - метод, сгенерировавший ошибку
- `InnerException` - ссылка на исключение, послужившее причиной текущего

Для генерации ошибки используется оператор `throw <Выражение>;`. Форма без параметра используется для возобновления исключения.

```
throw new DivideByZeroException(); // генерация исключения
...
throw; // возобновление исключения
```

Операторы `checked` и `unchecked` используются для включения и отключения проверки числовой арифметики на исключения:

```
// вариант checked для одного выражения
checked(a + b); // если произойдет переполнение, будет выброшено исключение
OverflowException
checked {
    byte a = 255;
    ++a; // будет выброшено исключение из-за переполнения типа byte
}
// в таком случае его нужно обработать
checked // вариант checked для блока кода
{
    byte a = 255; // максимальное значение для типа byte
    try
    {
        ++a;
    }
    catch (OverflowException)
    {
        Console.WriteLine("Произошло переполнение");
    }
}
```

Оператор `unchecked` наоборот отключает эту проверку и допускает переполнения.

## 8. C#. Объявление классов и их компонентов. Примеры.

Классы в языке C# описываются в привычном виде: `[Атрибуты][Спецификаторы] class <Имя>[: Предки] { Тело класса}`.

Компонентами класса могут быть поля и методы:

- поля - хранят данные:
  - константы - общие для всех объектов
  - переменные:
    - поля объекта - свои для каждого объекта
    - поля класса (статические поля) - общие для всех объектов
- методы - процедуры и функции, обрабатывающие данные

Пример:

```
// абстрактный класс, доступный в данной сборке
internal abstract class A {
    public int x; // переменная-поле объекта
    protected static int y; // статическая переменная
    public A(int _x, int _y) {
        x = _x;
        y = _y;
    }
    public void print() // метода класса
```



```

    {
        Console.WriteLine("x = {0}, y = {1}", x, y);
    }
}
// класс, наследуемый от класса A, доступный в данной сборке и не допускающий наследование
от себя самого
internal sealed class B : A {
    const double eps = 1e-10; // константа
    public B(int _x=0, int _y=0) : base(_x, _y) { } // вызов конструктора базового класса
    new public void print() // переопределение метода с помощью спецификатора new
    {
        Console.WriteLine("x = {0}, y = {1}, eps = {2}", x, y, eps);
    }
}

```

Объявление класса по сути является созданием собственного типа, объект класса может быть создан с помощью ключевого слова `new`:

```
B b = new B(1, 2);
```

## 9. C#. Спецификаторы доступа классов, структур и их компонентов.

Эти спецификаторы доступны и одинаково работают как для классов, так и для полей и методов (за исключением включающих `protected` для структур, так как они не допускают наследование):

- `public` - доступ не ограничен
- `protected` - доступ из производных классов
- `private` - доступ только из данного класса (по умолчанию для вложенных классов и полей класса)
- `internal` - доступ из классов сборки (по умолчанию для всех классов, кроме вложенных)
- `protected internal` - доступ из производных классов этой и других сборок, а также классов данной сборки
- `private protected` - доступ из производных классов данной сборки

Для классов также:

- `abstract` - абстрактный класс
- `static` - статический класс
- `new` - для вложенных классов задаёт новое описание класса взамен унаследованного
- `sealed` - запрещает наследование от данного класса

Для полей доступны дополнительно:

- `const` - делает поле неизменяемым, такие поля требуют инициализации при компиляции и общие для всех членов класса, возможно обращение через имя класса
- `readonly` - доступ только для чтения, инициализируются при компиляции или конструктором
- `static` - статическое поле, общее для всех объектов, возможно обращение через имя класса
- `new` - задаёт новое описание поля, скрывающее унаследованное
- `volatile` - значение, которое может быть изменено другим процессом

```

class A
{
    private int i = 1;           // скрытое поле объекта
    double z;                   // скрытое поле объекта (по умолчанию)
}

```

```
public int v;           // общедоступное поле объекта
public const double w = 32.517; // константное поле класса
public static string s = "D";  // статическое поле класса
}
```

10. C#. Конструкторы классов. Примеры.
11. C#. Поля: константные, объекта, класса, только для чтения. Примеры.
12. C#. Методы: конструкторы объектов, статические, деструкторы. Примеры.
13. C#. Методы: объектов, классов. Примеры.
14. C#. Параметры методов. Передача параметров по значению и по ссылке. Выходные параметры. Примеры.
15. C#. Одномерные массивы с элементами типов-значений и ссылочных типов. Примеры объявления.
16. C#. Оператор foreach. Примеры применения для массивов разных типов.
17. C#. Массивы прямоугольные и ступенчатые. Различие. Примеры
18. C#. Строка String. Примеры создания и использования.
19. C#. Регулярные выражения. Примеры.
20. C#. Структуры. Примеры.
21. C#. Наследование. Пример.
22. C#. Полиморфное наследование. Абстрактные классы. Пример.
23. C#. Композиция и агрегация. Пример.
24. C#. Интерфейсы. Пример.
25. C#. Свойства. Пример.