1. **Advantages and Disadvantages of the Microsoft .NET platform**:
    a. **Advantages**: cross-platform compatibility, high performance, support for multiple languages, a large number of libraries.
    b. **Disadvantages**: difficulties with porting old applications, the necessity to learn new technologies.
2. **What is BCL (FCL)?**:
    a. **BCL (Base Class Library)**: a set of libraries that provide basic functionality for working with .NET.
    b. **FCL (Framework Class Library)**: includes BCL and additional libraries for more specific tasks.
3. **Pros and Cons of the C# programming language**:
    a. **Pros**: strong typing, simple syntax, support for parallel and asynchronous programming.
    b. **Cons**: fewer cross-platform capabilities compared to some other languages, resource-intensive.
4. **Purpose of Reflectors and Dotfuscators**:
    a. **Reflector**: allows you to study and modify the structure of the code at runtime.
    b. **Dotfuscator**: obfuscates code to protect it from reverse engineering.
5. **Types of data in C#**:
    a. Simple types (int, float, double, char)
    b. Complex types (classes, structures, arrays)
6. **The keyword var and its usage**:
    a. Allows the compiler to automatically determine the type of a variable based on the assigned value.
7. **Value types and Reference types**:
    a. **Value types**: stored on the stack, copied when assigned.
    b. **Reference types**: stored on the heap, passed by reference.
8. **Input and Output in console applications**:
    a. Using `Console.ReadLine()` for input and `Console.WriteLine()` for output.
9. **Explicit and Implicit type conversion**:
    a. **Explicit**: performed manually using casting operators.
    b. **Implicit**: performed automatically if there is no loss of data.
10. **Operators in C#**:
    a. **Arithmetic**: +, -, *, /, %.
    b. **Logical**: &&, ||, !.
    c. **Comparison**: ==, !=, <, >, <=, >=.
11. **Conditional operators in C#**:
    a. `if`, `else if`, `else`, `switch`.
12. **Looping operators in C#**:
    a. `for`, `while`, `do-while`, `foreach`.
13. **Using arrays in C#**:
    a. Declaration: `int[] array = new int[10];`
    b. Accessing elements: `array[0] = 1;`
14. **The keyword null and its usage**:
    a. Represents the absence of a value for a reference variable.
15. **Strings in C#**:
    a. Using the `String` class and its methods (e.g., `Substring`, `IndexOf`).
16. **Difference between structures and classes**:
    a. **Structures**: value types, stored on the stack.

    b. **Classes**: reference types, stored on the heap.
17. **Method Overloading**:
    a. Creating multiple methods with the same name but different parameters.
18. **Operator Overloading**:
    a. Allows defining new implementations of standard operators for user-defined types.
19. **Using the this keyword**:
    a. Refers to the current instance of the class.
20. **Constructors and Destructors**:
    a. **Constructors**: initialize objects.
    b. **Destructors**: perform cleanup before objects are deleted.
21. **Exceptions. try, catch, throw, finally**:
    a. Error handling using `try`, `catch`, `finally` blocks and throwing exceptions using `throw`.
22. **Using checked and unchecked constructs**:
    a. Control overflow in arithmetic operations.
23. **Namespaces in C# programs**:
    a. Group classes and other types to avoid name conflicts.
24. **Properties. get, set**:
    a. Define access methods for class fields.
25. **Inheritance features in C#**:
    a. Support for single inheritance, ability to use interfaces.
26. **Virtual methods**:
    a. Methods that can be overridden in derived classes.
27. **Abstract class**:
    a. A class that cannot be instantiated and serves as a base for other classes.
28. **Delegates. Purpose and usage**:
    a. References to methods, used for passing methods as parameters.
29. **Interfaces. Purpose and usage**:
    a. Define a contract for classes that implement them.
30. **Enumerations. Purpose and usage**:
    a. A set of named constants used to represent fixed values.
31. **Standard interfaces**:
    a. Built-in interfaces such as `IComparable`, `IEnumerable`.
32. **Using anonymous and lambda functions**:
    a. Anonymous methods: `delegate`.
    b. Lambda expressions: `x => x * x`.
33. **Extension methods**:
    a. Add new methods to existing types without modifying them.
34. **Generics. Purpose and usage**:
    a. Allow creating classes and methods that work with any type.
35. **Collections. Generic collections**:
    a. Using collections such as `List<T>`, `Dictionary<TKey, TValue>`.
36. **Garbage collector**:
    a. Manages automatic cleanup of unused objects from memory.
37. **Working with files**:
    a. Using classes like `FileStream`, `StreamWriter`, `StreamReader`, `BinaryWriter`, `BinaryReader`.
38. **Working with directories**:
    a. Using classes like `Directory`, `DirectoryInfo`, `FileInfo`.
39. **Regular expressions**:

    a. Using the Regex class for pattern matching and text replacement.
40. **LINQ, purpose, and usage**:
    a. Language Integrated Query for working with data collections.
41. **Attributes, serialization**:
    a. Using attributes to add metadata to code, serialization for saving objects to a file.