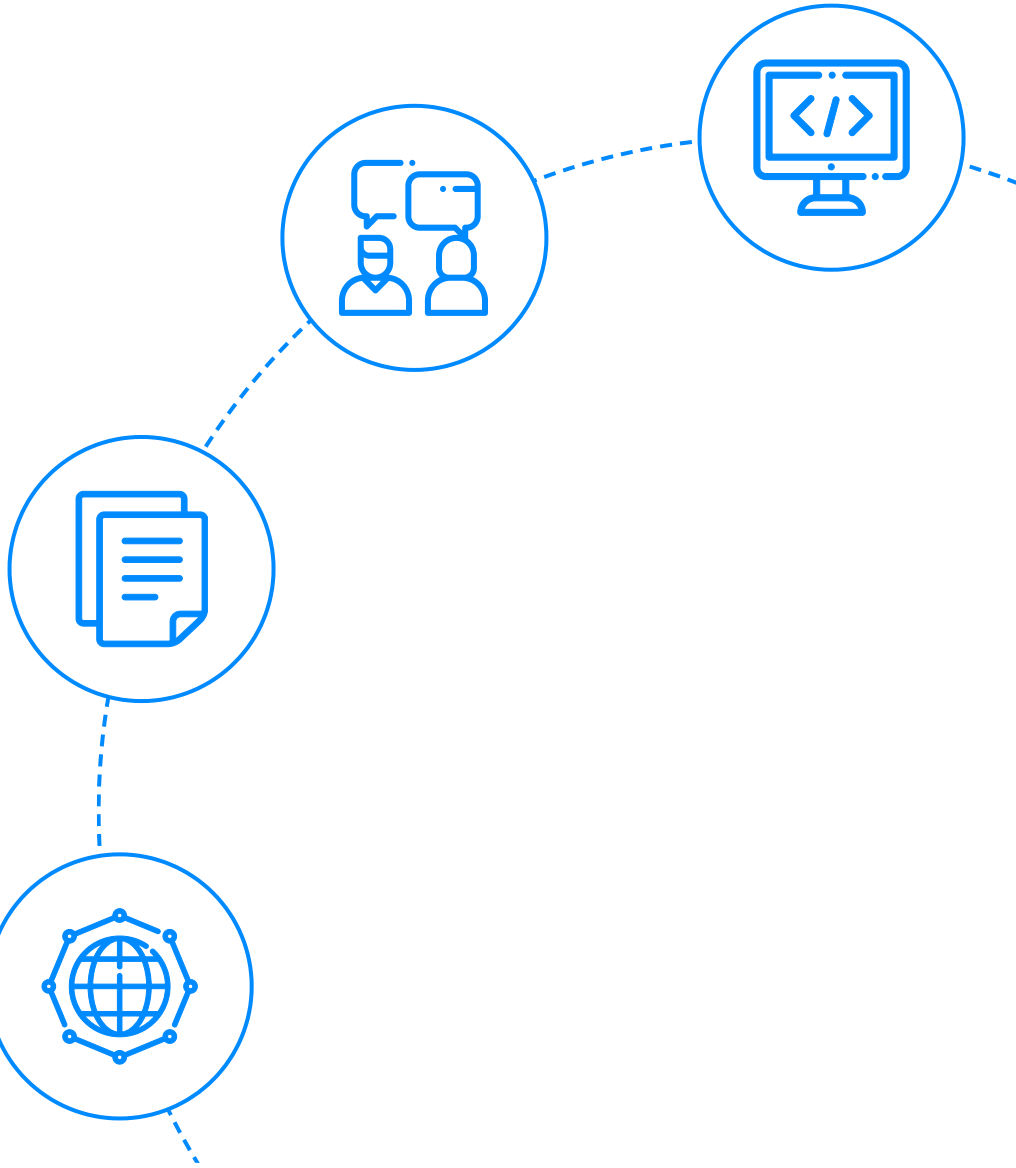
InterviewBit

C# Interview Questions



To view the live version of the page, click here.

Contents

C# Basic Interview Questions

C# Advanced Interview Questions

C# Advanced Interview Questions

(.....Continued)

19. What are the different ways in which a method can be Overloaded in C#?
20. What is Reflection in C#?
21. What is the difference between constant and readonly in C#?
22. What is the difference between String and StringBuilder in C#?

C# Coding Problems

23. Write a program in C# Sharp to reverse a string?
24. Write a program in C# Sharp to reverse the order of the given words?
25. Write a program in C# Sharp to find if a given string is palindrome or not?
26. Write a C# program to find the substring from a given string.
27. Write a C# program to find if a positive integer is prime or not?

Let's get Started

C sharp is an object-oriented programming language developed by Microsoft. C# is used with the .NET framework for creating websites, applications, and games. C# has a varied reason for being popular:

Comparatively easier: Starting with C# is termed comparatively easier than other programming languages
Wider use of development: Using C#, you are prone to create web applications or gaming apps. C# has some fantastic features like automatic garbage collection, interfaces, etc. which help build better applications.
Larger target audience: Collaboration with Microsoft provides an edge to the applications created using C# since it would have a wider target audience.

Since C# is such a widely-used programming language, a plethora of big and small organizations base their products using it. So, prepare yourself with basic and advanced level C# questions to ace the interviews. Let's look at 50 much asked and a comprehensive set of questions on C#.

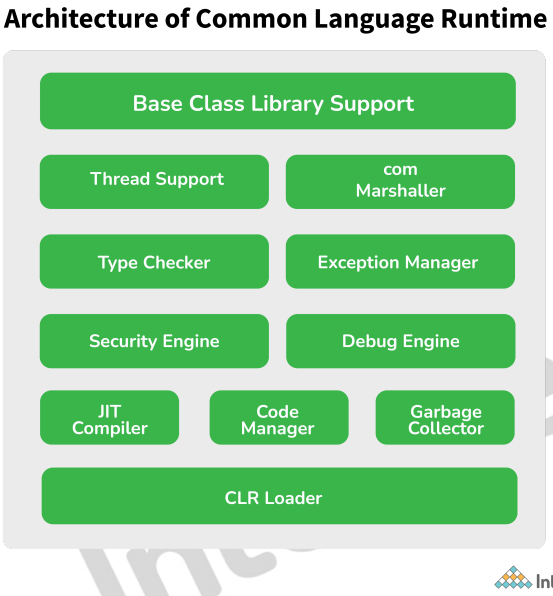
C# Basic Interview Questions

1. How is C# different from C?

You would always know C being the procedural language while C# is a more object-oriented language. The biggest difference is that C# supports automatic garbage collection by Common Language Runtime (CLR) while C does not. C# primarily needs a .NET framework to execute while C is a platform-agnostic language.

2. What is Common Language Runtime (CLR)?

CLR handles program execution for various languages including C#. The architecture of CLR handles memory management, garbage collection, security handling, and looks like:



Architecture of CLR

3. What is garbage collection in C#?

Garbage collection is the process of freeing up memory that is captured by unwanted objects. When you create a class object, automatically some memory space is allocated to the object in the heap memory. Now, after you perform all the actions on the object, the memory space occupied by the object becomes waste. It is necessary to free up memory. Garbage collection happens in three cases:

- If the occupied memory by the objects exceeds the pre-set threshold value.
- If the garbage collection method is called
- If your system has low physical memory

4. What are the types of classes in C#?

Class is an entity that encapsulates all the properties of its objects and instances as a single unit. C# has four types of such classes:

- **Static class:** Static class, defined by the keyword 'static' does not allow inheritance. Therefore, you cannot create an object for a static class.

Sample code:

```
static class classname
{
  //static data members
  //static methods
}
```

- **Partial class:** Partial class, defined by the keyword 'partial' allows its members to partially divide or share source (.cs) files.
- **Abstract class:** Abstract classes are classes that cannot be instantiated where you cannot create objects. Abstract classes work on the OOPS concept of abstraction. Abstraction helps to extract essential details and hide the unessential ones.
- **Sealed class:** Sealed classes are classes that cannot be inherited. Use the keyword sealed to restrict access to users to inherit that class.

```
sealed class InterviewBit
{
   // data members
   // methods
   .
   .
   .
}
```

5. What is a managed and unmanaged code?

Managed code lets you run the code on a managed CLR runtime environment in the .NET framework.
Managed code runs on the managed runtime environment than the operating system itself.
Benefits: Provides various services like a garbage collector, exception handling, etc.

Unmanaged code is when the code doesn't run on CLR, it is an unmanaged code that works outside the .NET framework.
They don't provide services of the high-level languages and therefore, run without them. Such an example is C++.

6. What is the difference between an abstract class and an interface?

Let's dig into the differences between an abstract class and an interface:

- Abstract classes are classes that cannot be instantiated ie. that cannot create an object. The interface is like an abstract class because all the methods inside the interface are abstract methods.
- Surprisingly, abstract classes can have both abstract and non-abstract methods but all the methods of an interface are abstract methods.
- Since abstract classes can have both abstract and non-abstract methods, we need to use the Abstract keyword to declare abstract methods. But in the interface, there is no such need.

An abstract class has constructors while an interface encompasses none.

Ex.

Abstract class:

```
public abstract class Shape{
public abstract void draw();
}
```

Interface:

```
public interface Paintable{
void paint();
}
```

7. What are the differences between ref and out keywords?

C# ref keywords pass arguments by reference and not value. To use the 'ref' keyword, you need to explicitly mention 'ref'.

```
void Method(ref int refArgument)
{
   refArgument = refArgument + 10;
}
int number = 1;
Method(ref number);
Console.WriteLine(number);
// Output: 11
```

C# out keywords pass arguments within methods and functions.
'out' keyword is used to pass arguments in a method as a reference to return multiple values. Although it is the same as the ref keyword, the ref keyword needs to be initialised before it is passed. Here, The out and ref keywords are useful when we want to return a value in the same variables that are passed as an argument.

```
public static string GetNextFeature(ref int id)
{
   string returnText = "Next-" + id.ToString();
   id += 1;
   return returnText;
}
public static string GetNextFeature(out int id)
{
   id = 1;
   string returnText = "Next-" + id.ToString();
   return returnText;
}
```

8. What are extension methods in C#?

Extension methods help to add new methods to the existing ones. The methods that are added are static. At times, when you want to add methods to an existing class but don't perceive the right to modify that class or don't hold the rights, you can create a new static class containing the new methods. Once the extended methods are declared, bind this class with the existing one and see the methods will be added to the existing one.

```
// C# program to illustrate the concept
// of the extension methods
using System;

namespace ExtensionMethod {
static class NewMethodClass {

   // Method 4
   public static void M4(this Scaler s)
   {
       Console.WriteLine("Method Name: M4");
   }

   // Method 5
   public static void M5(this Scaler s, string str)
   {
       Console.WriteLine(str);
   }
}

// Now we create a new class in which
// Scaler class access all the five methods
public class IB {

   // Main Method
   public static void Main(string[] args)
   {
       Scaler s = new Scaler();
       s.M1();
       s.M2();
       s.M3();
       s.M4();
       s.M5("Method Name: M5");
   }
}
}
```

Output:

Method Name: M1

Method Name: M2

Method Name: M3

Method Name: M4

Method Name: M5

9. What are Generics in C#?

In C# collections, defining any kind of object is termed okay which compromises C#'s basic rule of type-safety. Therefore, generics were included to type-safe the code by allowing re-use of the data processing algorithms. Generics in C# mean not linked to any specific data type. Generics reduce the load of using boxing, unboxing, and typecasting objects. Generics are always defined inside angular brackets <>. To create a generic class, this syntax is used:

```
GenericList<float> list1 = new GenericList<float>();
GenericList<Features> list2 = new GenericList<Features>();
GenericList<Struct> list3 = new GenericList<Struct>();
```

Here, GenericList<float> is a generic class. In each of these instances of GenericList<T>, every occurrence of T in the class is substituted at run time with the type argument. By substituting the T, we have created three different type-safe using the same class.

10. What is the difference between an Array and ArrayList in C#?

An array is a collection of similar variables clubbed together under one common name. While ArrayList is a collection of objects that can be indexed individually. With ArrayList you can access a number of features like dynamic memory allocation, adding, searching, and sorting items in the ArrayList.

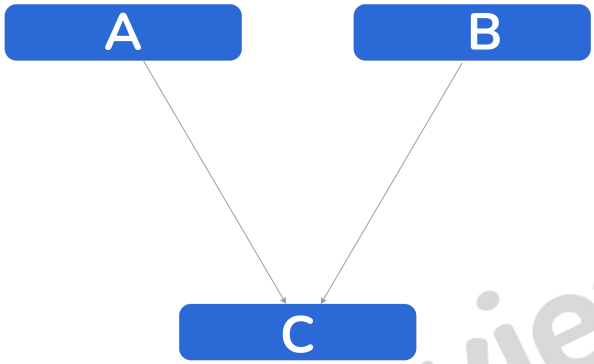
- When declaring an array the size of the items is fixed therefore, the memory allocation is fixed. But with ArrayList, it can be increased or decreased dynamically.
- Array belongs to system.array namespace while ArrayList belongs to the system.collection namespace.
- All items in an array are of the same datatype while all the items in an ArrayList can be of the same or different data types.
- While arrays cannot accept null, ArrayList can accept null values.

For ex.:

```
// C# program to illustrate the ArrayList
using System;
using System.Collections;

class IB {

   // Main Method
   public static void Main(string[] args)
   {

       // Create a list of strings
       ArrayList al = new ArrayList();
       al.Add("Bruno");
       al.Add("Husky");
       al.Add(10);
       al.Add(10.10);

       // Iterate list element using foreach loop
       foreach(var names in al)
       {
           Console.WriteLine(names);
       }
   }
}
```

11. What is inheritance? Does C# support multiple inheritance?

Inheritance means acquiring some of the properties from a master class.



Multiple Inheritance

Multiple Inheritance in C#

Here, class C can inherit properties from Class A and Class B. But, C# doesn't support multiple inheritances.
Instead, you can use interfaces to inherit the properties using the class name in the signature.

```
// C# program to illustrate
// multiple class inheritance
using System;
using System.Collections;

// Parent class 1
class Scaler {

  // Providing the implementation
  // of features() method
  public void features()
  {

      // Creating ArrayList
      ArrayList My_features= new ArrayList();

      // Adding elements in the
      // My_features ArrayList
      My_features.Add("Abstraction");
      My_features.Add("Encapsulation");
      My_features.Add("Inheritance");

      Console.WriteLine("Features provided by OOPS:");
      foreach(var elements in My_features)
      {
          Console.WriteLine(elements);
      }
  }
}

// Parent class 2
class Scaler2 :Scaler{

  // Providing the implementation
  // of courses() method
  public void languages()
  {

      // Creating ArrayList
      ArrayList My_features = new ArrayList();

      // Adding elements in the
      // My_features ArrayList
      My_features.Add("C++");
      My_features.Add("C#");
      My_features.Add("JScript");


      Console.WriteLine("\nLanguages that use OOPS concepts:");
      foreach(var elements in My_features)
      {
          Console.WriteLine(elements);
      }
  }
}

// Child class
```

C# Advanced Interview Questions

12. What is Boxing and Unboxing in C#?

The two functions are used for typecasting the data types:

Boxing: Boxing converts value type (int, char, etc.) to reference type (object) which is an implicit conversion process using object value.

Example:

```
int num = 23; // 23 will assigned to num
Object Obj = num; // Boxing
```

Unboxing: Unboxing converts reference type (object) to value type (int, char, etc.) using an explicit conversion process.

Example:

```
int num = 23;          // value type is int and assigned value 23
Object Obj = num;     // Boxing
int i = (int)Obj;     // Unboxing
```

13. What are Properties in C#?

Properties in C# are public members of a class where they provide the ability to access private members of a class. The basic principle of encapsulation lets you hide some sensitive properties from the users by making the variables private. The private members are not accessible otherwise in a class. Therefore, by using properties in C# you can easily access the private members and set their values.

The values can be easily assigned using get and set methods, also known as accessors. While the get method extracts the value, the set method assigns the value to the variables.

14. What are partial classes in C#?

Partial classes implement the functionality of a single class into multiple files. These multiple files are combined into one during compile time. The partial class can be created using the partial keyword.

```
public partial Clas_name
{
        // code
}
```

You can easily split the functionalities of methods, interfaces, or structures into multiple files. You can even add nested partial classes.

15. What is the difference between late binding and early binding in C#?

Late binding and early binding are examples of one of the primary concepts of OOPS: **Polymorphism.**

For ex: one function calculateBill() will calculate bills of premium customers, basic customers, and semi-premium customers based on their policies differently. The calculation for all the customer objects is done differently using the same function which is called polymorphism.

When an object is assigned to an object variable in C#, the .NET framework performs the binding.

When the binding function happens at compile-time, it is called early binding. It investigates and checks the methods and properties of the static objects. With early binding, the number of run-time errors decreases substantially and it executes pretty quickly.

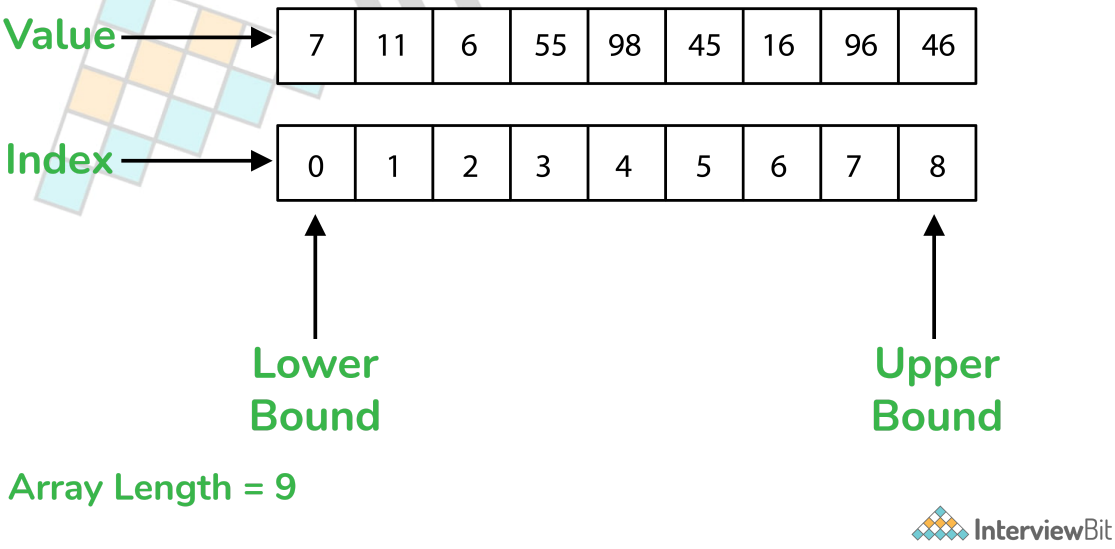
But when the binding happens at runtime, it is called late binding. Late binding happens when the objects are dynamic (decided based on the data they hold) at run-time. It is slower as it looks through during run-time.

16. What are the Arrays in C#?

When a group of similar elements is clubbed together under one name, they are called arrays.

For ex. An array of tea Atea[4]: [green tea, chamomile tea, black tea, lemon tea]. The length of the array defines how many elements are present in the array.

In C#, the memory allocations for the elements of the array happen dynamically. This is how values are stored in an array sequentially.



Arrays in C#

A few pointers for arrays in C#:

- The memory allocation is DYNAMIC.
- Arrays in C# are treated as objects.
- The length of the array is easy to find by detecting the number of members in the array.
- The members in the array are ordered and begin with the index value=0.
- The array types are reference types derived from the base array type.

Syntax: `< Data Type > [ ] < Name_Array >`

17. What are Indexers in C#?

Indexers are called smart arrays that allow access to a member variable. Indexers allow member variables using the features of an array. They are created using the Indexer keyword. Indexers are not static members.

For ex. Here the indexer is defined the same way.

```
<return type> this[<parameter type> index]
{
   get{
       // return the value from the specified index of an internal collection
   }
   set{
       // set values at the specified index in an internal collection
   }
}
```

18. Difference between the Equality Operator (==) and Equals() Method in C#?

Although both are used to compare two objects by value, still they both are used differently.

For ex.:

```
int x = 10;
int y = 10;
Console.WriteLine( x == y);
Console.WriteLine(x.Equals(y));
Output:
True
True
```

Equality operator (==) is a reference type which means that if equality operator is used, it will return true only if both the references point to the same object.

Equals() method: Equals method is used to compare the values carried by the objects. int x=10, int y=10. If x==y is compared then, the values carried by x and y are compared which is equal and therefore they return true.

Equality operator: Compares by reference

Equals(): Compares by value

19. What are the different ways in which a method can be Overloaded in C#?

Overloading means when a method has the same name but carries different values to use in a different context. Only the main() method cannot be overloaded.

In order to overload methods in C#,

- Change the number of parameters in a method, or
- Change the order of parameters in a method, or
- Use different data types for parameters
  In these ways, you can overload a method multiple times.

For ex.

```
public class Area {
   public double area(double x) {
       double area = x * x;
       return area;
   }
   public double area(double a, double b) {
       double area = a * b;
       return area;
   }
}
```

Here, the method Area is used twice. In the first declaration, one argument is used while in the second one, there were two arguments are used. Using different parameters in the same method, we were able to overload the method area().

20. What is Reflection in C#?

Reflection in C# extracts metadata from the datatypes during runtime.

To add reflection in the .NET framework, simply use System.Refelction namespace in your program to retrieve the type which can be anything from:

- Assembly
- Module
- Enum
- MethodInfo
- ConstructorInfo
- MemberInfo
- ParameterInfo
- Type
- FieldInfo
- EventInfo
- PropertyInfo

21. What is the difference between constant and readonly in C#?

A **const** keyword in C# is used to declare a constant field throughout the program. That means once a variable has been declared **const**, its value cannot be changed throughout the program.

In C#, a constant is a number, string, null reference, or boolean values.

For ex:

```
class IB {

   // Constant fields
   public const int xvar = 20;
   public const string str = "InterviewBit";

   // Main method
   static public void Main()
   {

       // Display the value of Constant fields
       Console.WriteLine("The value of xvar: {0}", xvar);
       Console.WriteLine("The value of str: {0}", str);
   }
}
Output:
The value of xvar is 20.
The value of string is Interview Bit
```

On the other hand, with readonly keyword, you can assign the variable only when it is declared or in a constructor of the same class in which it is declared.

Ex:

```
public readonly int xvar1;
   public readonly int yvar2;

   // Values of the readonly
   // variables are assigned
   // Using constructor
   public IB(int b, int c)
   {

       xvar1 = b;
       yvar2 = c;
       Console.WriteLine("The value of xvar1 {0}, "+
                      "and yvar2 {1}", xvar1, yvar2);
   }

   // Main method
   static public void Main()
   {
     IB obj1 = new IB(50, 60);
   }
}

Output:
The value of xvar1 is 50, and yvar2 is 60
```

Constants are static by default while readonly should have a value assigned when the constructor is declared.
Constants can be declared within functions while readonly modifiers can be used with reference types.

22. What is the difference between String and StringBuilder in C#?

The major difference between String and StringBuilder is that String objects are immutable while StringBuilder creates a mutable string of characters. StringBuilder will make the changes to the existing object rather than creating a new object.

StringBuilder simplifies the entire process of making changes to the existing string object. Since the String class is immutable, it is costlier to create a new object every time we need to make a change. So, the StringBuilder class comes into picture which can be evoked using the System.Text namespace.

In case, a string object will not change throughout the entire program, then use String class or else StringBuilder.

For ex:

```
string s = string.Empty;
for (i = 0; i < 1000; i++)
  {
    s += i.ToString() + " ";
  }
```

Here, you'll need to create 2001 objects out of which 2000 will be of no use.

The same can be applied using StringBuilder:

```
StringBuilder sb = new StringBuilder();
for (i = 0; i < 1000; i++)
 {
   sb.Append(i); sb.Append(' ');
 }
```

By using StringBuilder here, you also de-stress the memory allocator.

C# Coding Problems

23. Write a program in C# Sharp to reverse a string?

```
internal static void ReverseString(string str)
{

  char[] charArray = str.ToCharArray();
  for (int i = 0, j = str.Length - 1; i < j; i++, j--)
  {
      charArray[i] = str[j];
      charArray[j] = str[i];
  }
  string reversedstring = new string(charArray);
  Console.WriteLine(reversedstring);
```

24. Write a program in C# Sharp to reverse the order of the given words?

```
internal static void ReverseWordOrder(string str)
{
  int i;
  StringBuilder reverseSentence = new StringBuilder();
   int Start = str.Length - 1;
  int End = str.Length - 1;
   while (Start > 0)
  {
      if (str[Start] == ' ')
      {
          i = Start + 1;
          while (i <= End)
          {
              reverseSentence.Append(str[i]);
              i++;
          }
          reverseSentence.Append(' ');
          End = Start - 1;
      }
      Start--;
  }
   for (i = 0; i <= End; i++)
  {
      reverseSentence.Append(str[i]);
  }
  Console.WriteLine(reverseSentence.ToString());
}
```

25. Write a program in C# Sharp to find if a given string is palindrome or not?

```
internal static void chkPalindrome(string str)
{
   bool flag = false;
   for (int i = 0, j = str.Length - 1; i < str.Length / 2; i++, j--)
   {
       if (str[i] != str[j])
       {
           flag = false;
           break;
       }
       else
           flag = true;
   }
   if (flag)
   {
       Console.WriteLine("Palindrome");
   }
   else
       Console.WriteLine("Not Palindrome");
```

Output:

Input: Key Output: Not Palindrome
Input: step on no pets Output: Palindrome

26. Write a C# program to find the substring from a given string.

```
internal static void findallsubstring(string str)
{
   for (int i = 0; i < str.Length; ++i)
   {
       StringBuilder subString = new StringBuilder(str.Length - i);
       for (int j = i; j < str.Length; ++j)
       {
           subString.Append(str[j]);
           Console.Write(subString + " ");
       }
   }
}
```

27. Write a C# program to find if a positive integer is prime or not?

```
 static void Main(string[] args)
{
    if (FindPrime(47))
    {
        Console.WriteLine("Prime");
    }
    else
    {
        Console.WriteLine("Not Prime");
    }
    Console.ReadLine();
}
internal static bool FindPrime(int number)
{
    if (number == 1) return false;
    if (number == 2) return true;
    if (number % 2 == 0) return false;
     var squareRoot = (int)Math.Floor(Math.Sqrt(number));
     for (int i = 3; i <= squareRoot; i += 2)
    {
        if (number % i == 0) return false;
    }
     return true;
}
```

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions