C# skeleton:

```
using System;
class Hello
{
    public static void Main()
        {
            Console.WriteLine("Hello, World!");
        }
}
```

1. types:

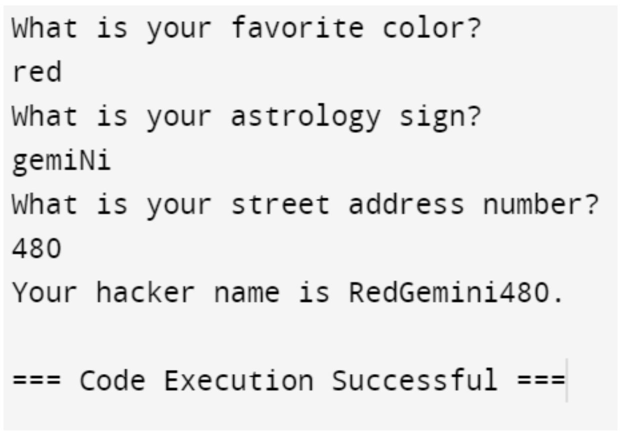
1. Telephone Number: string
2. Height: float
3. Age: int
4. Gender: enum
5. Salary: double
6. ISBN: string
7. Price: double
8. Shipping Weight: float
9. Population: long
10. Number of Stars in Universe: BigInteger
11. Number of Employees in Medium Business Companies: int

2. value type v.s. reference type: value types are on stack, whereas reference types are on heap; value types are automatically deallocated when out of scope, whereas reference types are managed by the garbage collector; value type copy assignment copies the value, whereas reference type copy assignment copies the reference.
Boxing and unboxing: boxing converts a value type to an object type (reference type), and unboxing extracts the value type from the object type.
3. Managed resources are automatically handled by the .NET runtime, while unmanaged resources require explicit cleanup by the programmers.
4. The Garbage Collector manages the allocation and release of memory for applications. It automatically handles the allocation and deallocation of memory for managed objects. It also manages the heap memory and optimizes the use of available memory. When an object is no longer in use, the garbage collector calls the corresponding destructor and frees the memory allocated to them.

Program:

```
using System;
using System.Globalization;

class Hello
{
    public static void Main()
```

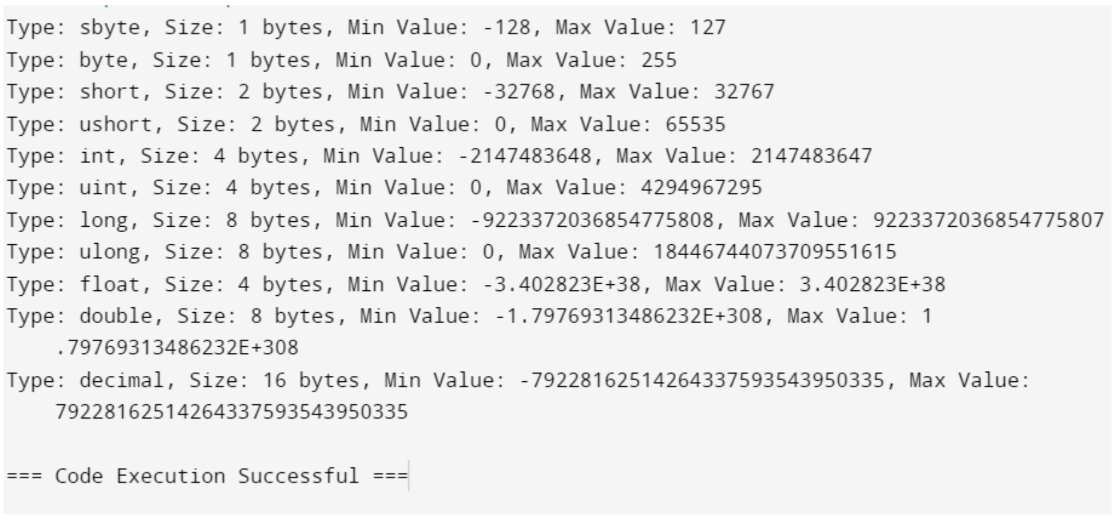
```
	{
		// Ask for user inputs
		Console.WriteLine(“What is your favorite color?”);
		string favoriteColor = Console.ReadLine();

		Console.WriteLine(“What is your astrology sign?”);
		string astrologySign = Console.ReadLine();

		Console.WriteLine(“What is your street address number?”);
		string streetAddressNumber = Console.ReadLine();

		// Combine inputs into a single “hacker name” in Camel Case format
		string hackerName = ToCamelCase(favoriteColor) +
ToCamelCase(astrologySign) + ToCamelCase(streetAddressNumber);

		// Output the hacker name
		Console.WriteLine($”Your hacker name is {hackerName}.”);
	}

	// Helper method to convert a string to Camel Case
	public static string ToCamelCase(string input)
	{
		if (string.IsNullOrEmpty(input))
		{
			return input;
		}

		// Convert the first letter to uppercase and the rest to lowercase
		TextInfo textInfo = CultureInfo.CurrentCulture.TextInfo;
		return textInfo.ToTitleCase(input.ToLower());
	}
}
```



```
What is your favorite color?
red
What is your astrology sign?
gemiNi
What is your street address number?
480
Your hacker name is RedGemini480.

=== Code Execution Successful ===
```

1.

```
using System;

namespace _02UnderstandingTypes
{
    class Program
    {
        static void Main(string[] args)
        {
            PrintTypeInfo<sbyte>("sbyte");
            PrintTypeInfo<byte>("byte");
            PrintTypeInfo<short>("short");
            PrintTypeInfo<ushort>("ushort");
            PrintTypeInfo<int>("int");
            PrintTypeInfo<uint>("uint");
            PrintTypeInfo<long>("long");
            PrintTypeInfo<ulong>("ulong");
            PrintTypeInfo<float>("float");
            PrintTypeInfo<double>("double");
            PrintTypeInfo<decimal>("decimal");
        }

        static void PrintTypeInfo<T>(string typeName) where T : struct,
IComparable
        {
            int size = System.Runtime.InteropServices.Marshal.SizeOf<T>();
            string minValue = string.Empty;
            string maxValue = string.Empty;

            switch (typeName)
            {
                case "sbyte":
                    minValue = sbyte.MinValue.ToString();
                    maxValue = sbyte.MaxValue.ToString();
                    break;
                case "byte":
                    minValue = byte.MinValue.ToString();
                    maxValue = byte.MaxValue.ToString();
                    break;
                case "short":
                    minValue = short.MinValue.ToString();
                    maxValue = short.MaxValue.ToString();
                    break;
                case "ushort":
                    minValue = ushort.MinValue.ToString();
                    maxValue = ushort.MaxValue.ToString();
                    break;
                case “int”:
                    minValue = int.MinValue.ToString();
                    maxValue = int.MaxValue.ToString();
                    break;
                case “uint”:
                    minValue = uint.MinValue.ToString();
                    maxValue = uint.MaxValue.ToString();
                    break;
                case “long”:
                    minValue = long.MinValue.ToString();
                    maxValue = long.MaxValue.ToString();
                    break;
                case “ulong”:
                    minValue = ulong.MinValue.ToString();
                    maxValue = ulong.MaxValue.ToString();
                    break;
                case “float”:
                    minValue = float.MinValue.ToString();
                    maxValue = float.MaxValue.ToString();
                    break;
                case “double”:
                    minValue = double.MinValue.ToString();
                    maxValue = double.MaxValue.ToString();
                    break;
                case “decimal”:
                    minValue = decimal.MinValue.ToString();
                    maxValue = decimal.MaxValue.ToString();
                    break;
                default:
                    minValue = “N/A”;
                    maxValue = “N/A”;
                    break;
            }

            Console.WriteLine($”Type: {typeName}, Size: {size} bytes, Min
Value: {minValue}, Max Value: {maxValue}”);
        }
    }
}
```

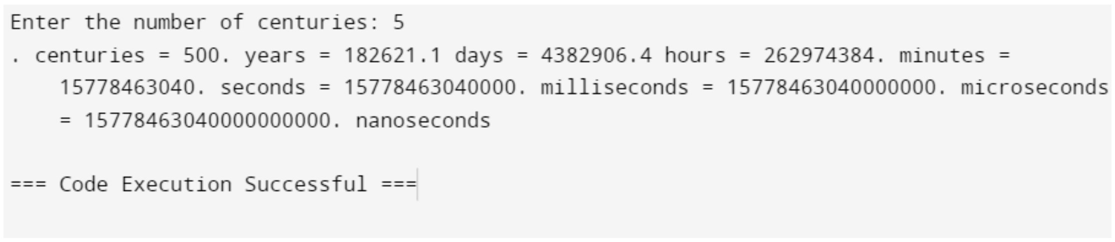


```
Type: sbyte, Size: 1 bytes, Min Value: -128, Max Value: 127
Type: byte, Size: 1 bytes, Min Value: 0, Max Value: 255
Type: short, Size: 2 bytes, Min Value: -32768, Max Value: 32767
Type: ushort, Size: 2 bytes, Min Value: 0, Max Value: 65535
Type: int, Size: 4 bytes, Min Value: -2147483648, Max Value: 2147483647
Type: uint, Size: 4 bytes, Min Value: 0, Max Value: 4294967295
Type: long, Size: 8 bytes, Min Value: -9223372036854775808, Max Value: 9223372036854775807
Type: ulong, Size: 8 bytes, Min Value: 0, Max Value: 18446744073709551615
Type: float, Size: 4 bytes, Min Value: -3.402823E+38, Max Value: 3.402823E+38
Type: double, Size: 8 bytes, Min Value: -1.79769313486232E+308, Max Value: 1
    .79769313486232E+308
Type: decimal, Size: 16 bytes, Min Value: -79228162514264337593543950335, Max Value:
    79228162514264337593543950335

=== Code Execution Successful ===
```

2.

```
using System;
using System.Numerics; // Include this for BigInteger

namespace CenturiesConverter
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Enter the number of centuries: ");
            if (int.TryParse(Console.ReadLine(), out int centuries))
            {
                // Conversion constants
                const long yearsInCentury = 100;
                const double daysInYear = 365.2422; // Average considering leap
years
                const long hoursInDay = 24;
                const long minutesInHour = 60;
                const long secondsInMinute = 60;
                const long millisecondsInSecond = 1000;
                const long microsecondsInMillisecond = 1000;
                const long nanosecondsInMicrosecond = 1000;

                // Calculations
                long years = centuries * yearsInCentury;
                long days = (long)(years * daysInYear);
                long hours = days * hoursInDay;
                long minutes = hours * minutesInHour;
```

```
                    // Using BigInteger for seconds and smaller units
                    BigInteger seconds = (BigInteger)minutes * secondsInMinute;
                    BigInteger milliseconds = seconds * millisecondsInSecond;
                    BigInteger microseconds = milliseconds *
microsecondsInMillisecond;
                    BigInteger nanoseconds = microseconds *
nanosecondsInMicrosecond;

                    // Output
                    Console.WriteLine($"{centuries} centuries = {years} years =
{days} days = {hours} hours = {minutes} minutes = {seconds} seconds =
{milliseconds} milliseconds = {microseconds} microseconds = {nanoseconds}
nanoseconds");
                }
                else
                {
                    Console.WriteLine("Invalid input. Please enter a valid integer
number of centuries.");
                }
            }
        }
}
```



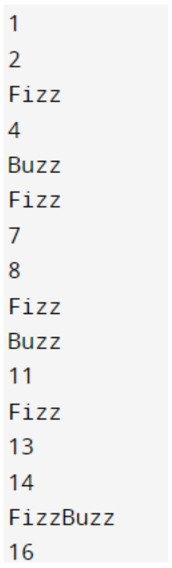
```
Enter the number of centuries: 5
. centuries = 500. years = 182621.1 days = 4382906.4 hours = 262974384. minutes =
    15778463040. seconds = 15778463040000. milliseconds = 15778463040000000. microseconds
    = 15778463040000000000. nanoseconds

=== Code Execution Successful ===
```

1. Division of integer by 0 throws a DivdeByZeroException.
2. Division of a double by 0 yields Infinity or -Infinity depending on the sign of the original double.
3. Overflow beyond the maximum int value yields a negative integer, and overflow beyond the minimum int value yields a positive integer.
4. In both cases, y is incremented by 1; however, x=y++; assigns the original y value to x, whereas x=++y; assigns the incremented y value to x.
5. Break exits a loop prematurely, continue skips the current iteration and proceeds to the next one, while return exits the method with that for loop prematurely, optionally returning a value to the caller.
6. Three parts of a statement: the expression, the semicolon, and the comment. The first two parts are required.
7. = is used to assign a value to a variable, whereas == is used to check if two values are equal.

8. Yes.
9. _ represents the default case in a switch statement.
10. IEnumerator Interface.

1.

```
using System;

class Exercise03
{
	static void Main()
	{
		for (int i = 1; i <= 100; i++)
		{
			if (i % 3 == 0 && i % 5 == 0)
			{
				Console.WriteLine(“FizzBuzz”);
			}
			else if (i % 3 == 0)
			{
				Console.WriteLine(“Fizz”);
			}
			else if (i % 5 == 0)
			{
				Console.WriteLine(“Buzz”);
			}
			else
			{
				Console.WriteLine(i);
			}
		}
	}
}
```



```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
```

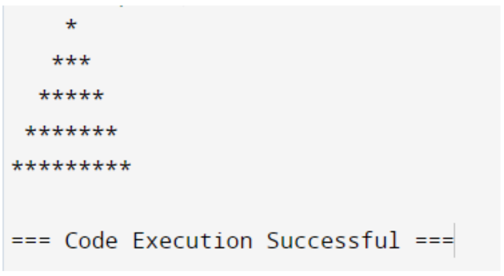
After outputting 255, the next output overflows to 0.

```
int max = 500;
int previousValue = -1; // Initialize with a value that won’t be seen in the loop
for (byte i = 0; i < max; i++)
{
    if (i == 0 && previousValue == 255) // Check for overflow
    {
        Console.WriteLine(“Warning: Byte overflow detected.”);
    }
    Console.WriteLine(i);
    previousValue = i; // Store the current value for comparison in the next iteration
}
```

```
using System;

class Program
{
    static void Main()
    {
        // Generate a random number between 1 and 3
        int correctNumber = new Random().Next(3) + 1;

        // Prompt the user to guess the number
        Console.WriteLine(“Guess a number between 1 and 3:”);

        // Read the user’s input and convert it to an integer
        int guessedNumber;
        try
        {
            guessedNumber = int.Parse(Console.ReadLine());
        }
        catch (FormatException)
        {
            Console.WriteLine("Invalid input. Please enter a valid number
between 1 and 3.");
            return;
        }

        // Check if the guessed number is within the valid range [1, 3]
        if (guessedNumber < 1 || guessedNumber > 3)
        {
            Console.WriteLine("Your guess is out of range. Please enter a number
between 1 and 3.");
            return;
        }

        // Compare the guessed number with the correct number
        if (guessedNumber < correctNumber)
        {
            Console.WriteLine("Your guess is too low.");
        }
        else if (guessedNumber > correctNumber)
        {
            Console.WriteLine("Your guess is too high.");
        }
        else
        {
            Console.WriteLine("Congratulations! You guessed the correct
number.");
        }
    }
}
```

2.

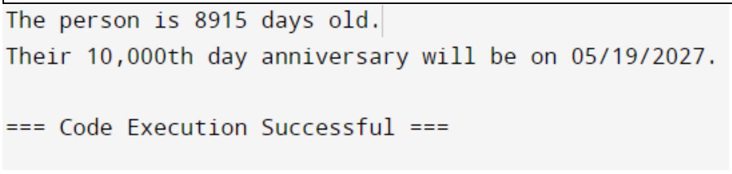
```
using System;

class Program
{
        static void Main()
        {
            int levels = 5;

            for (int k = 1; k <= levels; k++)
            {
                // Print leading spaces
                for (int i = 0; i < levels - k; i++)
                {
                    Console.Write(" ");
                }

                // Print asterisks
                for (int i = 0; i < 2 * k - 1; i++)
                {
                    Console.Write("*");
                }

                // Move to the next line
                Console.WriteLine();
            }
        }
}
```



```
    *
   ***
  *****
 *******
*********

=== Code Execution Successful ===
```

3. Already done. See above.
4.

```
using System;

class Program
{
    static void Main()
    {
        // Define the birth date
        DateTime birthDate = new DateTime(2000, 1, 1);

        // Calculate the number of days since the birth date
        TimeSpan age = DateTime.Today - birthDate;
        int daysOld = (int)age.TotalDays;

        // Output the person's age in days
        Console.WriteLine($"The person is {daysOld} days old.");
        // Calculate the date of their 10,000th day anniversary
        int daysToNextAnniversary = 10000 - (daysOld % 10000);
        DateTime                    anniversaryDate                    =
DateTime.Today.AddDays(daysToNextAnniversary);

        // Output the date of their 10,000th day anniversary
        Console.WriteLine($"Their  10,000th  day  anniversary  will  be  on
{anniversaryDate:d}.");
    }
}
```



```
The person is 8915 days old.
Their 10,000th day anniversary will be on 05/19/2027.

=== Code Execution Successful ===
```

5.

```
using System;

class Program
{
    static void Main()
    {
        // Get the current local time in Eastern Time
        TimeZoneInfo                    easternTime                    =
TimeZoneInfo.FindSystemTimeZoneById("Eastern Standard Time");
        DateTime  currentTime  =  TimeZoneInfo.ConvertTime(DateTime.Now,
easternTime);

        // Get the hour component of the current time
        int hour = currentTime.Hour;

        // Determine the appropriate greeting based on the hour
        string greeting;
        if (hour >= 6 && hour < 12)
        {
            greeting = "Good Morning";
        }
        else if (hour >= 12 && hour < 18)
        {
            greeting = "Good Afternoon";
        }
        else if (hour >= 18 && hour < 21)
```

```
            {
                  greeting = “Good Evening”;
            }
            else
            {
                  greeting = “Good Night”;
            }

            // Output the greeting to the user
            Console.WriteLine($”{greeting}”);
      }
}
```

6.

```
using System;

class Program
{

          static void Main()
      {
            for (int i = 1; i <= 4; i++)
            {
                  for (int j = 0; j <= 24; j+=i)
                  {
                        if(j == 24){
                              Console.Write(j);
                        }
                        else{
                              Console.Write(j + “,”);
                        }
                  }
                  Console.WriteLine();
            }
      }

      }
```