- A person's telephone number: string, as phone numbers typically include non-numeric characters such as dashes, parentheses, or spaces.

- A person's height: decimal, as this data type provides more precision than float and is commonly used to represent measurements such as height.

- A person's age: byte or short, depending on the expected age range.

- A person's gender: enum or string, depending on the desired storage format and usage.

- A person's salary: decimal or double, depending on the desired precision and range of values.

- A book's ISBN: string, as ISBN numbers typically include both numeric and non-numeric characters.

- A book's price: decimal or double, depending on the desired precision and range of values.

- A book's shipping weight: decimal or double, depending on the desired precision and range of values.

- A country's population: long, as this data type can accommodate large numbers.

- The number of stars in the universe: long, to accommodate very large numbers.

- The number of employees in each of the small or medium businesses in the United Kingdom: short, int or long, depending on the expected number of employees per business.

In C#, there are two main categories of data types: value types and reference types. The main differences between them are:

- Storage location: Value types are stored in the stack memory, while reference types are stored in the heap memory.

- Memory management: Value types are automatically cleaned up when they go out of scope, while reference types require garbage collection to free up memory.

- Default value: Value types have a default value, such as 0 for numeric types or false for bool, while reference types have a default value of null.

- Passing to methods: Value types are passed to methods by value, meaning that a copy of the value is passed to the method, while reference types are passed by reference, meaning that a reference to the object is passed to the method.

- Equality comparison: Value types are compared by their value, while reference types are compared by their memory address.

Boxing and unboxing are used to convert between value types and reference types in C#. Boxing is the process of converting a value type to a reference type by wrapping the value in an object, while unboxing is the process of extracting the value type from the object. Boxing can be useful in cases where a value type needs to be passed to a method that expects a reference type, while unboxing can be useful in cases where a value type needs to be extracted from an object. However, boxing and unboxing can be slow and can have performance implications, so they should be used judiciously.

In .NET, a managed resource is a resource that is managed by the .NET runtime and is automatically allocated and deallocated by the Common Language Runtime (CLR) garbage collector. Examples of managed resources include objects created with the new operator in C# or VB.NET, as well as arrays, strings, and other data structures.

In contrast, an unmanaged resource is a resource that is not managed by the CLR garbage collector and must be manually allocated and deallocated by the programmer. Examples of unmanaged resources include file handles, network connections, database connections, and other system-level resources.

Since unmanaged resources are not automatically freed by the CLR garbage collector, it is the responsibility of the programmer to explicitly release them when they are no longer needed. This is typically done using the Dispose() method or the using statement, which ensures that the unmanaged resource is properly released even in the case of exceptions or other errors.

It's worth noting that .NET provides a mechanism for wrapping unmanaged resources in managed objects, known as "managed wrappers". These wrappers can be used to ensure that unmanaged resources are properly released when the corresponding managed object is no longer needed, and can help simplify memory management in .NET applications. Examples of managed wrappers include the FileStream and NetworkStream classes, which provide a managed interface to unmanaged file and network resources, respectively.

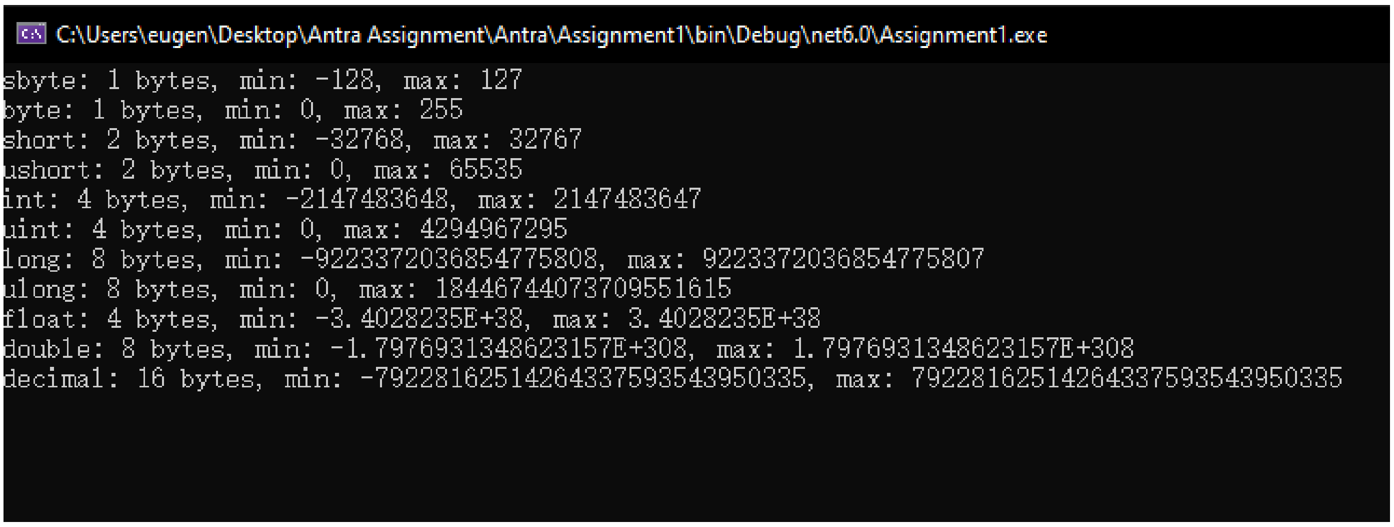
Whats the purpose of Garbage Collector in .NET?

The purpose of the Garbage Collector (GC) in .NET is to automatically manage the allocation and deallocation of memory used by a .NET application. The GC is responsible for identifying and freeing up memory that is no longer being used by an application, thereby helping to prevent memory leaks and other memory-related issues.

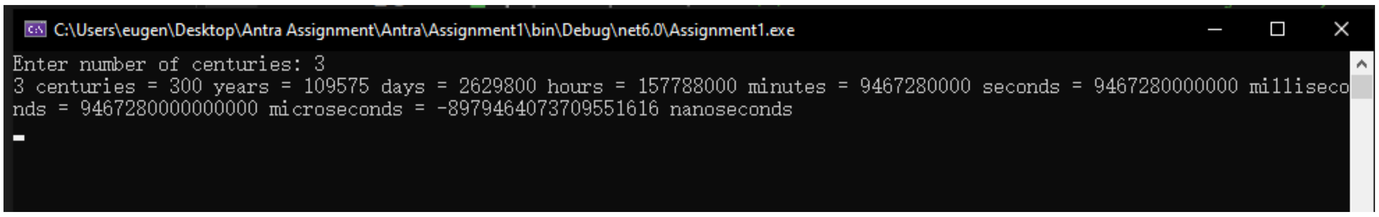
When a .NET application allocates memory for objects and data structures, the GC keeps track of which objects are in use and which ones are no longer being used. It periodically scans the heap memory to identify objects that are no longer referenced by any part of the application, and then frees up the memory associated with those objects.

The GC operates in the background and is completely transparent to the application developer, who does not need to explicitly allocate or deallocate memory. This helps simplify memory management in .NET applications and helps prevent common issues such as buffer overflows, null reference exceptions, and memory leaks.

In summary, the purpose of the Garbage Collector in .NET is to automatically manage memory allocation and deallocation, helping to prevent memory leaks and other memory-related issues, and simplifying memory management for application developers.



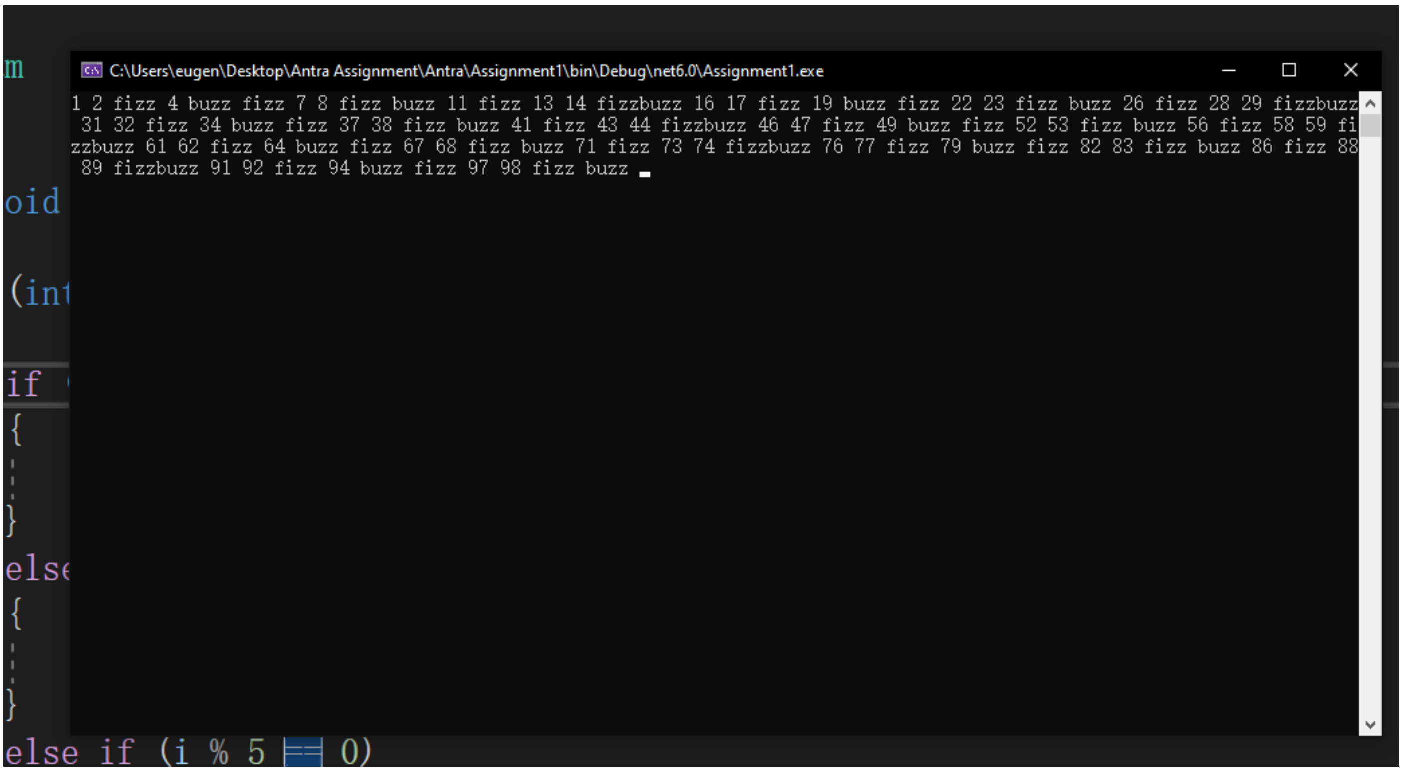
```
C:\Users\eugen\Desktop\Antra Assignment\Antra\Assignment1\bin\Debug\net6.0\Assignment1.exe
sbyte: 1 bytes, min: -128, max: 127
byte: 1 bytes, min: 0, max: 255
short: 2 bytes, min: -32768, max: 32767
ushort: 2 bytes, min: 0, max: 65535
int: 4 bytes, min: -2147483648, max: 2147483647
uint: 4 bytes, min: 0, max: 4294967295
long: 8 bytes, min: -9223372036854775808, max: 9223372036854775807
ulong: 8 bytes, min: 0, max: 18446744073709551615
float: 4 bytes, min: -3.4028235E+38, max: 3.4028235E+38
double: 8 bytes, min: -1.7976931348623157E+308, max: 1.7976931348623157E+308
decimal: 16 bytes, min: -79228162514264337593543950335, max: 79228162514264337593543950335
```



```
C:\Users\eugen\Desktop\Antra Assignment\Antra\Assignment1\bin\Debug\net6.0\Assignment1.exe
Enter number of centuries: 3
3 centuries = 300 years = 109575 days = 2629800 hours = 157788000 minutes = 9467280000 seconds = 9467280000000 milliseco
nds = 9467280000000000 microseconds = -8979464073709551616 nanoseconds
```

- When you divide an int variable by 0 in C#, a System.DivideByZeroException is thrown at runtime.
- When you divide a double variable by 0 in C#, the result is a special value called "infinity" or "negative infinity" depending on the sign of the dividend. No exception is thrown in this case.
- When you overflow an int variable in C#, the value wraps around and starts from the minimum value again. For example, if you set an int variable to the value 2,147,483,647 and then increment it by 1, it will overflow and become -2,147,483,648.
- x = y++; assigns the value of y to x and then increments the value of y. x = ++y; increments the value of y and then assigns the incremented value to x.

- break terminates the loop immediately and continues execution with the next statement after the loop. continue skips the current iteration of the loop and continues with the next iteration. return exits the function and returns a value to the caller.
- The three parts of a for statement are the initialization, condition, and increment/decrement. The initialization and increment/decrement are optional, but the condition is required.
- The = operator is the assignment operator, which assigns the value on the right-hand side to the variable on the left-hand side. The == operator is the equality operator, which compares two values for equality and returns a boolean value indicating whether they are equal or not.
- Yes, the statement compiles. It is an infinite loop with no initialization or increment/decrement, and the condition true is always true.
- In a switch expression in C# 8.0 and later, the underscore _ is a discard pattern that matches any value and can be used to provide a default case.
- An object must implement the IEnumerable or IEnumerable<T> interface to be enumerated over by using the foreach statement.



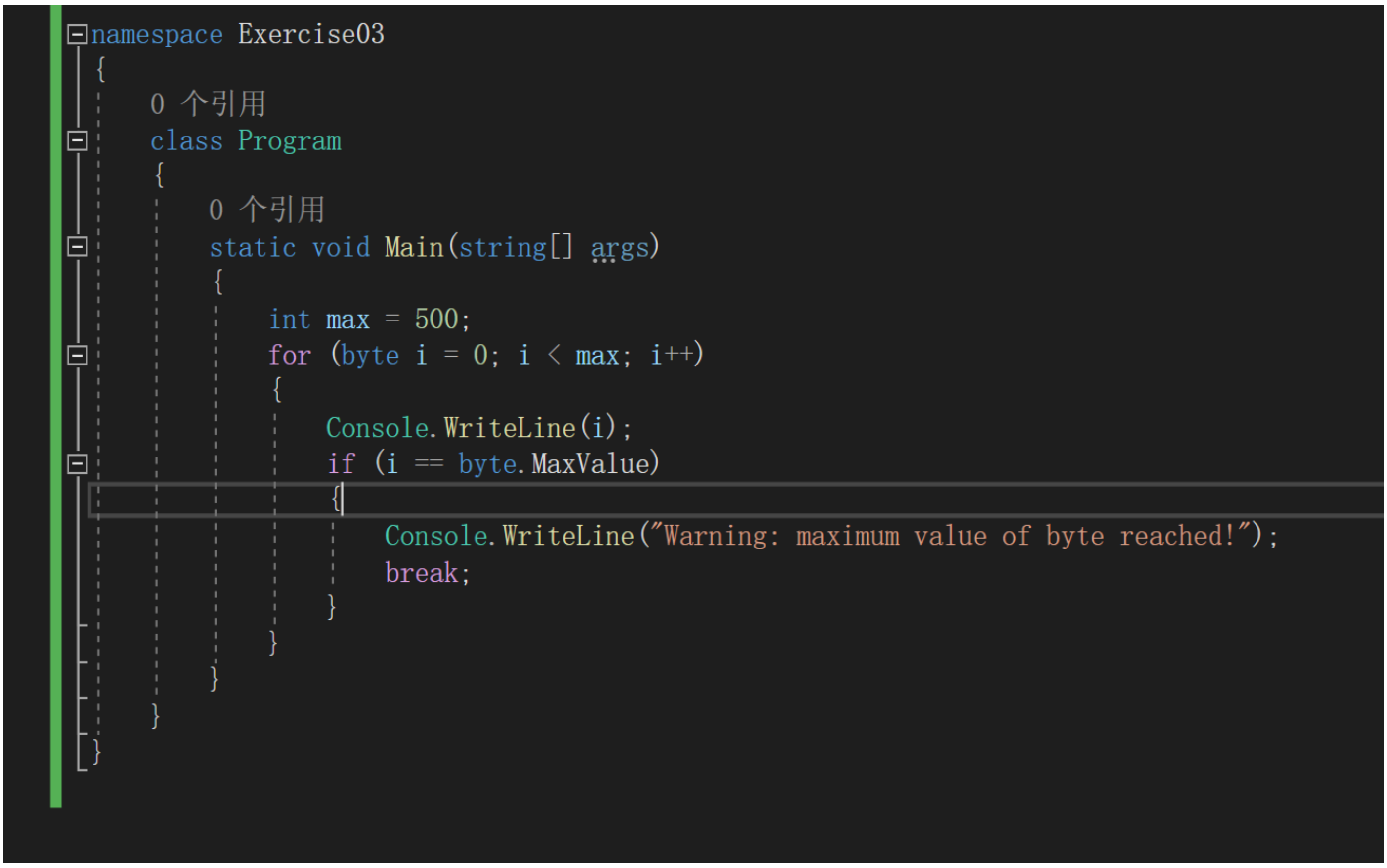
C:\Users\eugen\Desktop\Antra Assignment\Antra\Assignment1\bin\Debug\net6.0\Assignment1.exe

```
1 2 fizz 4 buzz fizz 7 8 fizz buzz 11 fizz 13 14 fizzbuzz 16 17 fizz 19 buzz fizz 22 23 fizz buzz 26 fizz 28 29 fizzbuzz
 31 32 fizz 34 buzz fizz 37 38 fizz buzz 41 fizz 43 44 fizzbuzz 46 47 fizz 49 buzz fizz 52 53 fizz buzz 56 fizz 58 59 fi
zzbuzz 61 62 fizz 64 buzz fizz 67 68 fizz buzz 71 fizz 73 74 fizzbuzz 76 77 fizz 79 buzz fizz 82 83 fizz buzz 86 fizz 88
 89 fizzbuzz 91 92 fizz 94 buzz fizz 97 98 fizz buzz
```

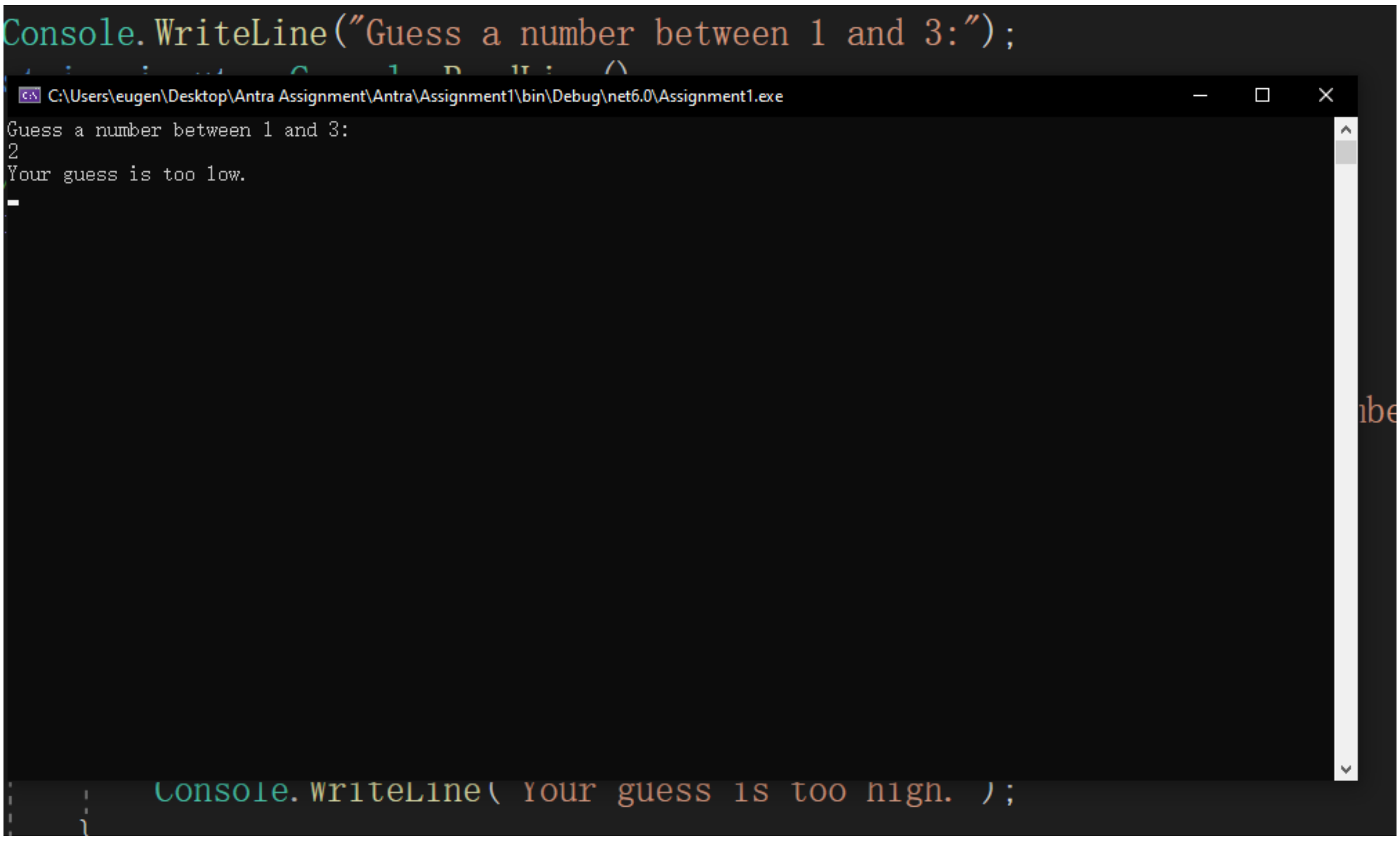
```
m

oid

(int

if
{
}
else
{
}
else if (i % 5 == 0)
```

The code will compile and execute without any error, but it will result in an infinite loop because the byte type has a maximum value of 255, so when i reaches 255 and is incremented, it will wrap around to 0 and continue counting up. Since the loop condition i < max will always be true, the loop will continue indefinitely.

To warn about this problem, we could add a check for the maximum value of i inside the loop and display a warning message when it is reached



```
namespace Exercise03
{
    0 个引用
    class Program
    {
        0 个引用
        static void Main(string[] args)
        {
            int max = 500;
            for (byte i = 0; i < max; i++)
            {
                Console.WriteLine(i);
                if (i == byte.MaxValue)
                {
                    Console.WriteLine("Warning: maximum value of byte reached!");
                    break;
                }
            }
        }
    }
}
```

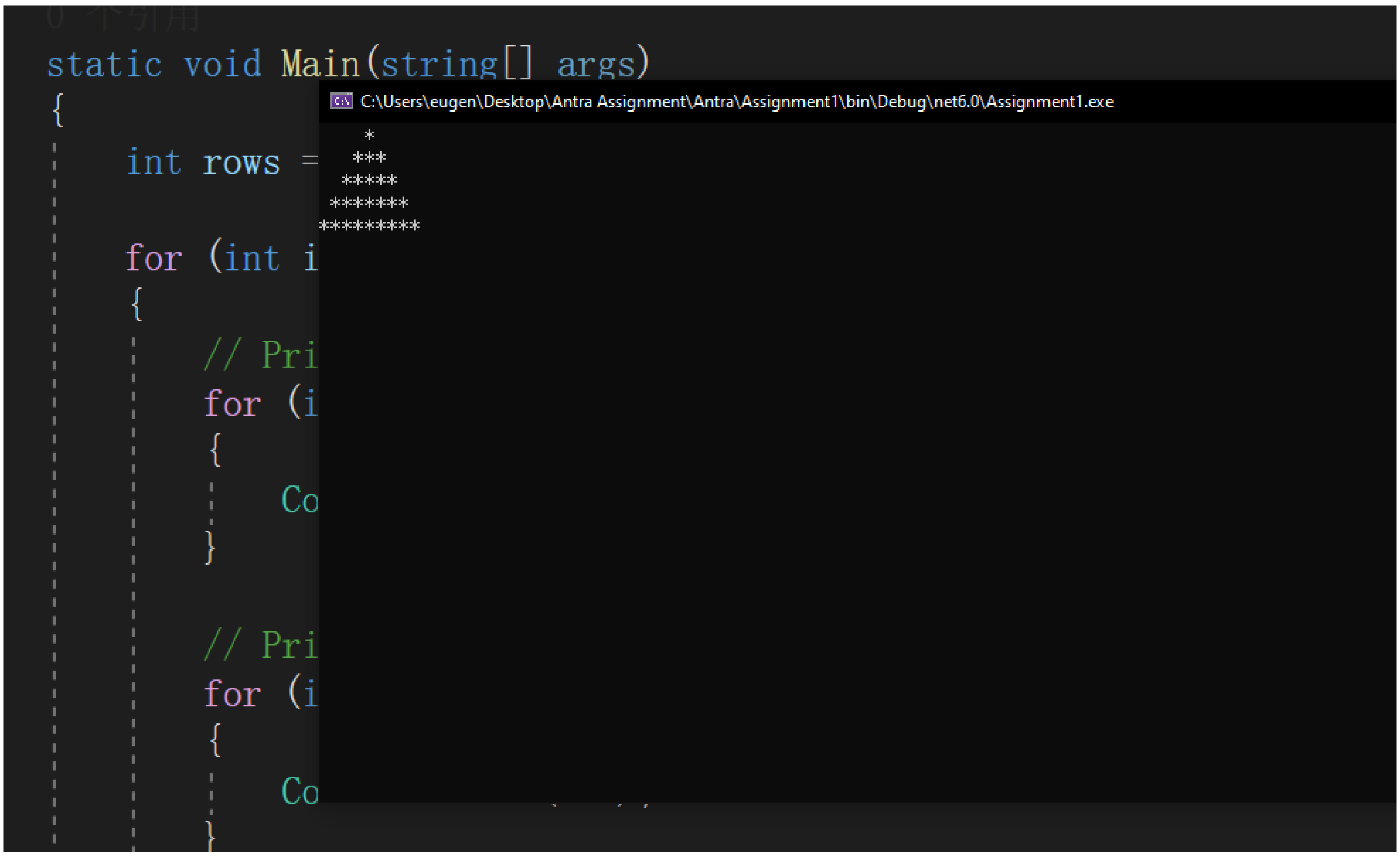


```
Console.WriteLine("Guess a number between 1 and 3:");
```

C:\Users\eugen\Desktop\Antra Assignment\Antra\Assignment1\bin\Debug\net6.0\Assignment1.exe

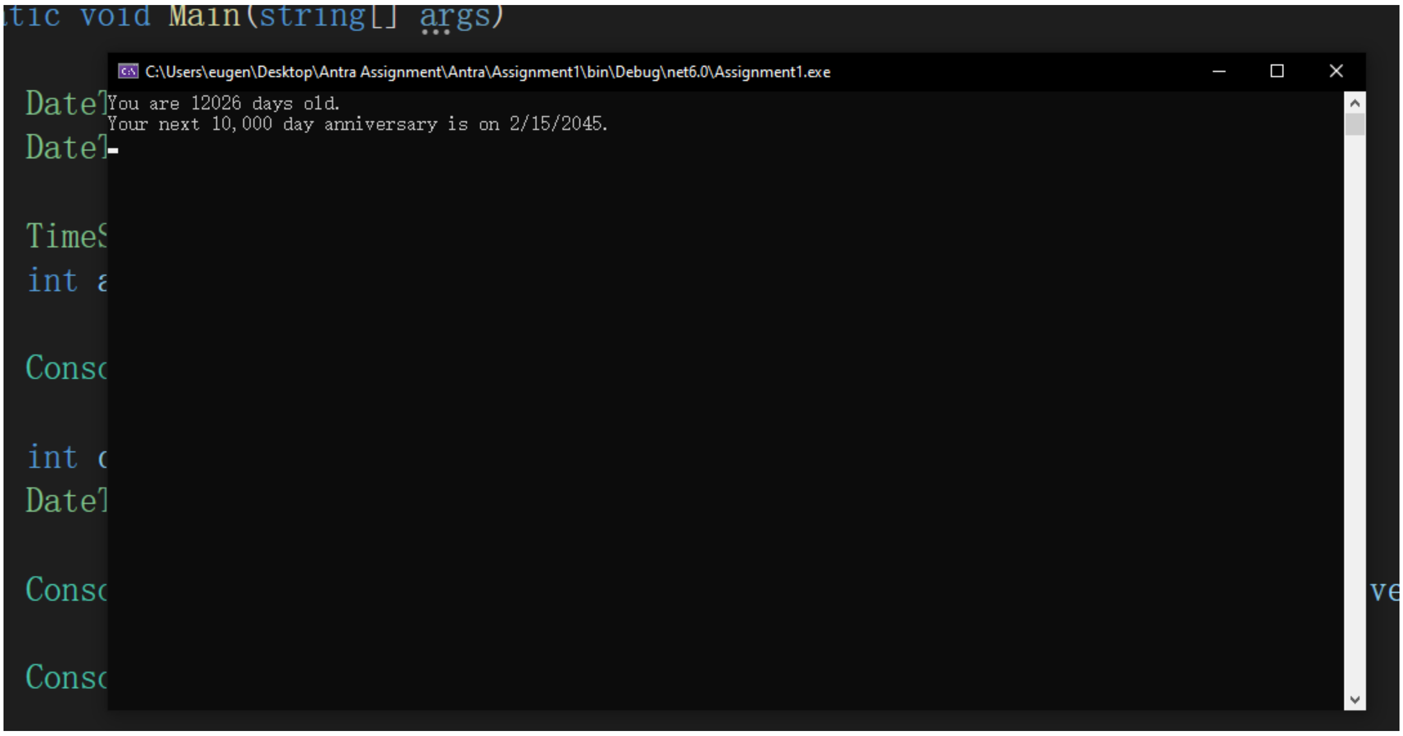
```
Guess a number between 1 and 3:
2
Your guess is too low.
```

```
Console.WriteLine("Your guess is too high.");
```



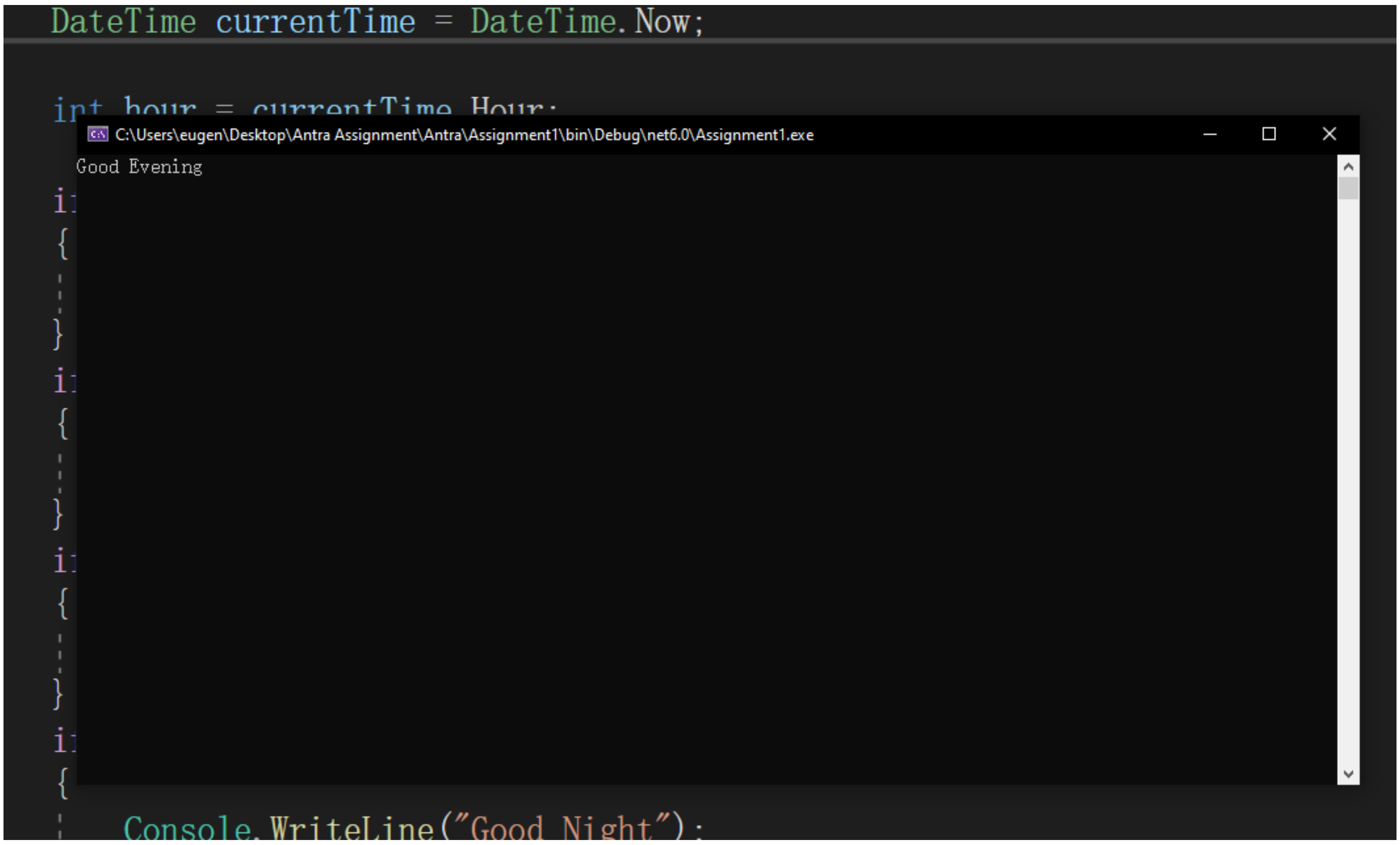
```
0 个引用
static void Main(string[] args)
{
    int rows =

    for (int i
    {
        // Pri
        for (i
        {
            Co
        }

        // Pri
        for (i
        {
            Co
        }
```

```
C:\Users\eugen\Desktop\Antra Assignment\Antra\Assignment1\bin\Debug\net6.0\Assignment1.exe
    *
   ***
  *****
 *******
*********
```



```
tic void Main(string[] args)

  DateT
  DateT

  TimeS
  int a

  Conso

  int c
  DateT

  Conso                                                                     ve

  Conso
```

```
C:\Users\eugen\Desktop\Antra Assignment\Antra\Assignment1\bin\Debug\net6.0\Assignment1.exe
You are 12026 days old.
Your next 10,000 day anniversary is on 2/15/2045.
```

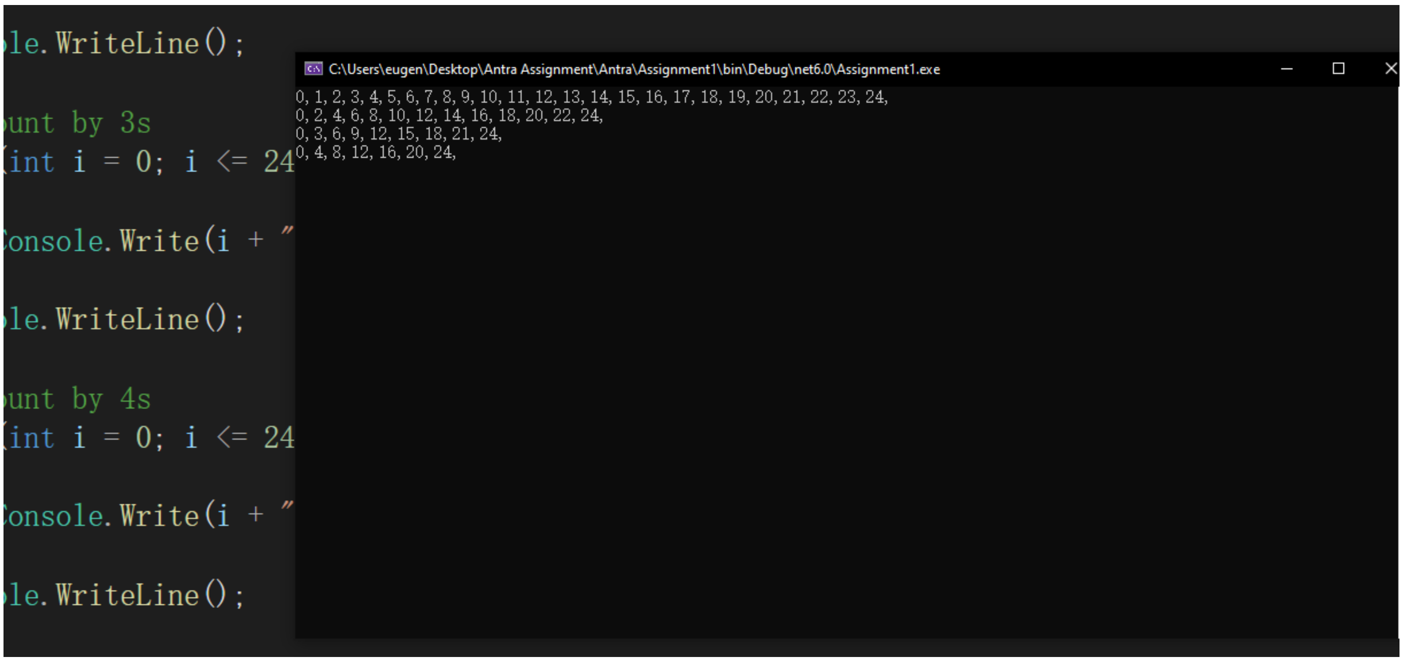


```
DateTime currentTime = DateTime.Now;

int hour = currentTime.Hour;
```

C:\Users\eugen\Desktop\Antra Assignment\Antra\Assignment1\bin\Debug\net6.0\Assignment1.exe

```
Good Evening
```

```
i
{

}
i
{

}
i
{

}
i
{
    Console.WriteLine("Good Night");
```



```
le.WriteLine();

unt by 3s
(int i = 0; i <= 24

Console.Write(i + "

le.WriteLine();

unt by 4s
(int i = 0; i <= 24

Console.Write(i + "

le.WriteLine();
```

C:\Users\eugen\Desktop\Antra Assignment\Antra\Assignment1\bin\Debug\net6.0\Assignment1.exe

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24,
0, 3, 6, 9, 12, 15, 18, 21, 24,
0, 4, 8, 12, 16, 20, 24,
```