The use of var is not permitted.
Note: Your program will not compile until you complete the Circle class (Tests 2-5).

Test 1 - Acronym

```
StringBuilder Test1(string[] names)
```

Test1 is similar to Test5 from Lab6A in that you are given an array of string and are to create a string that is an acronym of the words. However, in this lab you are not allowed to use string concatenation. Instead, you will use a StringBuilder object. The `StringBuilder` class is useful for building strings. It is much more efficient than concatenating (`+`) to an existing string object. The StringBuilder class can be found in the `System.Text` namespace. Create an instance of a `StringBuilder` class using its default (no argument) constructor.

Loop through the array of strings, `names`, that has been passed to this function as input. For each string in the array use the `Append()` function of your StringBuilder instance to append the first letter of each string to the `StringBuilder` instance. When you are finished, return the `StringBuilder` instance from this function, and we will use it to verify the letters you appended.

Example Input

```
Peach, Yoshi, Mario, Toad, Luigi
```

Example Output

```
PYMTL
```

Test 2 – Define a new class

```
object Test2(float x, float y, float radius)
```

For this test you will end up writing your own class, called `Circle`, make an instance of it, and return that instance. That is, you will return an object that you make of type `Circle`. In other words, if you were to make an object of type `Circle` called `c`, you would return `c` from this Test. You can put the `Circle` class declaration either: in its own file called `Circle.cs`, or you can put it in the `Submission.cs` file with your `Submission` class within the same namespace scope.

To create your `Circle` class, you will add the following access modifier, fields, constructor and methods.

Your `Circle` class must be declared as public. When you create your class called `Circle`, you must make the class itself `public`. That means, you must put the keyword `public` directly before the words `class Circle`. For example:

```
public class Circle
{
 // fields of this class
 // constructors of this class
 // methods of this class
}
```

Your `Circle` class must have 3 member variables (fields) of type float, which represent:

- the x position of the Circle
- the y position of the Circle
- the radius of the Circle

You can name the fields using any name you want, though you could use the following:

- `float mX;`
- `float mY;`
- `float mRadius;`

The prefix 'm' stands for member variable.

Your `Circle` class must have an overloaded constructor, which takes three variables that will be assigned to the fields of the class:

```
public Circle(float x, float y, float radius)
```

This overloaded constructor should take the variables passed into the class and assign them to the fields. Do not include a default constructor.

Your Circle class must have the following 3 accessor (getter) methods, each of which return one of the 3 fields of the class. Each accessor method must return the appropriate field.

```
public float GetX()
public float GetY()
public float GetRadius()
```

For this Test, when you have finished adding the above requested information to your `Circle` class, you will then make an instance object of type `Circle`, calling its overloaded constructor passing the variables given to you in this Test, and then you will return your class instance object from this Test.
For example:

```
Circle c1 = new Circle(x, y, radius);
return c1;
```

Example Input

```
1, 1, 4
```

Example Output

```
1, 1, 4
```

Test 3

```
object Test3(float x, float y, float radius)
```

Add the following method to your `Circle` class, which computes the area of the circle and returns it. You must use Math.PI (do not define a local version or a literal value for PI).

```
public float GetArea()
```

For this Test, when you have finished adding the above requested information to your `Circle` class, you will then make an instance object of type `Circle`, calling its overloaded constructor passing the variables

given to you in this Test, and then you will return your class instance object from this Test.

For example:
```
Circle c1 = new Circle(x, y, radius);
return c1;
```

Example Input
```
-1, -1, 2
```

Example Output
```
12.57
```

Test 4
```
object Test4(float x, float y, float radius)
```

Add the following method to your `Circle` class, which given another point's x and y coordinates, this method returns whether or not that point is within the bounds of the Circle or not.
```
public bool Contains(float px, float py)
```

To help you accomplish this, on a piece of paper, draw a circle with a center point and radius, then draw two other points, one inside the circle and one outside. How could the computer determine which point is inside the circle and which point is outside the circle? Think about the radius of the circle from its center point compared to the distance between the point and the center of the circle.

For this Test, when you have finished adding the above requested information to your `Circle` class, you will then make an instance object of type `Circle`, calling its overloaded constructor passing the variables given to you in this Test, and then you will return your class instance object from this Test.
For example:
```
Circle c1 = new Circle(x, y, radius);
return c1;
```

Example Input
```
1, -2, 4, -3.04, -3.69
```

Example Output
```
False
```

Test 5
```
object Test5(float x, float y, float radius)
```

Add the following method to your `Circle` class, which computes the circumference of a circle. You must use Math.PI (do not define a local version or a literal value for PI).

```
public float GetCircumference()
```

For this Test, when you have finished adding the above requested information to your `Circle` class, you will then make an instance object of type `Circle`, calling its overloaded constructor passing the variables given to you in this Test, and then you will return your class instance object from this Test.

For example:

```
Circle c1 = new Circle(x, y, radius);
return c1;
```

Example Input

```
-1, 1, 3
```

Example Output

```
18.85
```

Test 6

```
int Test6(string str1, string str2, bool ignoreCase)
```

For this Test, all you have to do is call the static `Compare()` method of the `String` class. There is a method called `Compare`, which is declared as static, inside the `String` class. Remember that you call a static method by using the class name, not an instance name, and then the method name that you want to call (e.g. `String.Compare()`). You can see which parameters the `Compare()` method takes by looking at the Intellisense in Visual Studio or by searching online (https://msdn.microsoft.com/en-us/library/zkcaxw5y(v=vs.110).aspx). Notice that there is an overload of the `Compare()` method that takes the exact same parameters as those passed to this Test. So all you have to do is pass those parameters to the `Compare()` method and return the result of that method call.

If you are curious as to the output of the `String.Compare()` method, it returns 0 when the 2 strings it is comparing are considered equal, it returns +1 when the first string comes after the second string in an alphabetic sort (using the ASCII table), and it returns -1 when the first string comes before the second string in an alphabetic sort. The MSDN docs have more information.

Example Input

```
Beemo, beemo, True
```

Example Output

```
0
```

Test 7

```
string Test7(sbyte offset, string message)
```

For this test you will end up writing your own class, called `TextCodec`, make an instance of it, call the `Encode()` method (described below), and return the result of that method. You can put the `TextCodec` class declaration either: in its own file called `TextCodec.cs`, or you can put it in the `Submission.cs` file with your `Submission` class within the same namespace scope.

To create your `TextCodec` class, you will add the following access modifier, fields, constructor and methods.

Your `TextCodec` class must be declared as public. When you create your class called `TextCodec`, you must make the class itself public. That means, you must put the keyword `public` directly before the words `class TextCodec`. For example:

```
public class TextCodec
{
 // fields of this class
 // constructors of this class
 // methods of this class
}
```

Your `TextCodec` class must have 1 member variable (field), which is:

- an `sbyte`, which holds a numeric offset

You can name the field any name you want.

Your `TextCodec` class must have an overloaded constructor, which takes the offset variable, that will be assigned to the field of the class:

```
public TextCodec(sbyte offset)
```

Your TextCodec class must have the following method:

```
public string Encode(string message)
```

The purpose of the `Encode()` method is take the `message` passed into the method and encode it into an unreadable string using the offset field stored in the class. To accomplish this, we will provide you with the following code to use as the body of your Encode method:

```
StringBuilder sb = new StringBuilder(message);
for (int i = 0; i < message.Length; i++)
{
    sb[i] = (char)(sb[i] + mOffset);
}
return sb.ToString();
```

This method uses a `StringBuilder` class because it is more efficient to construct strings with a StringBuilder than simply adding strings together. The StringBuilder class is declared in the `System.Text` namespace.

For this Test, when you have finished adding the above requested information to your `TextCodec` class, you will then make an instance object of type `TextCodec`, calling its overloaded constructor passing the `offset` variable given to you in this Test, and then you will call the `Encode()` method passing the `message` variable given to you in this Test, and return the result of the Encode() method.

For example: `TextCodec codec = new TextCodec(offset);`

```
// call the Encode() method passing message and return the result
```

Example Input

```
-3, "A winner is you."
```

Example Output

```
">↔tfkkbo↔fp↔vlr+"
```

Test 8

```
string Test8(sbyte offset, string message)
```

Add the following method to your `TextCodec` class:

```
public string Decode(string message)
```

The purpose of the `Encode()` method is take the `message` passed into the method and decode it back to a readable string using the offset field stored in the class. The decoding code works nearly exactly the same as the encoding code, except you will subtract the offset from each character in the StringBuilder, instead of adding it.

For this Test, when you have finished adding the above requested information to your `TextCodec` class, you will then make an instance object of type `TextCodec`, calling its overloaded constructor passing the `offset` variable given to you in this Test, and then you will call the `Decode()` method passing the `message` variable given to you in this Test, and return the result of the `Decode()` method.
For example:

```
 TextCodec codec = new TextCodec(offset);
// call the Decode() method passing message and return the result
```

Example Input

```
-3, ">↔tfkkbo↔fp↔vlr+"
```

Example Output

```
"A winner is you."
```