Instructions: This exam is to be taken in silence, without notes, books, or electronic devices (including "smart" watches or earbuds). The time limit to complete it is 2 hours. Answer the following questions and problems, trying to be as clear and as accurate as possible, and remembering that a partial answer is better than no answer at all. Take your time to read the statements carefully before trying to answer them. If you need more space, write on the back of your test page and indicate it clearly. When writing code, make sure your special punctuation characters are legible, and your lowercase and uppercase letters are easy to distinguish. As usual, every statement or series of statement is assumed to be in a valid class and method, and you can use the `C.RL()`, `C.W()` and `C.WL()` abbreviations.

Please, after you indicated your name above, tear off the last page and read it.

Program Execution Examples: On some problems, we provide an example program execution in `this font` (typewriter), underlining the user input and representing "return" with ↵. You do not need to reproduce it exactly; it is just to illustrate the expected behavior.

```
This is a prompt by the program.
This is what the user enters.↵
```

____ **/ 35 pts.** **Problem 1** *Complete Problem 1 (displayed on the last page) below.*

____ / 45 pts. **Problem 2** Suppose given a 2-dimensional array of `char` called `words`.

1. Write a program that displays the `words` array, with a new line at the end of each row.

2. Write a program that displays “Letter found” if the `char` variable `letter` occurs in the `words` array.

3. Write a condition that evaluates to **`true`** if `words` is a square (that is, if it has the same number of rows and columns).

From now on, we assume that `words` is a square 2-dimensional array.

4. Write a program that displays “Word match found” if the word on the first row is the same as the word on the first column, as in the following example:

```
't' 'e' 's' 't'
'e' '*' '*' '*'
's' '*' '*' '*'
't' '*' '*' '*'
```

5. Write a condition that displays “Word match found” if the word on the first row is the same as the word on the diagonal, as in the following example:

```
't' 'e' 's' 't'
'*' 's' '*' '*'
'*' '*' 's' '*'
'*' '*' '*' 't'
```

___ **/ 50 pts.** **Problem 3** Consider the UML class diagram displayed on the last page.

1. Write the complete implementation of the `Shape` abstract class. The `ToString` method should simply return the string `"This shape is "`.

2. Write an implementation for the `Width` property of the `Rectangle` class such that setting the width to a negative value should result in an `ArgumentOutOfRangeException` (that you can shorten to `AOORE`) exception being thrown. Add an attribute if needed.

3. Write an implementation for the `ToString` method of the `Rectangle` class that returns a `string` containing what was returned by the `Shape`'s `ToString` method, the width, length and area of the calling object. For example, for a `Rectangle` with width 10 and length 5, it should be of the form "This shape is a rectangle (W: 10, L: 5, Area: 50)".

4. Write the constructor for the `WeightedRectangle` class.

5. Write the `Equals` method for the `Rectangle` class. It should return **`true`** if the calling object and the parameter have the same lengths and same widths, or if one can be obtained by rotating the other.

6. Write the `Equals` method for the `WeightedRectangle` class. It should return **`true`** if the calling object and the parameter are equal rectangles with the same weight.

___ / 35 pts. **Problem 4** In the following, assume that only the `System` namespace is available, and consider the usual implementation of `CList`:

```
public class CList<T>{  private Cell first;
                        public CList(){first = null;}
                        private class Cell{
                            public T Data { get; set; }
                            public Cell Next { get; set; }
                            public Cell(T dataP, Cell nextP)
                                {Data = dataP;  Next = nextP;}
                        }
}
```

1. Write the `AddF(T dataP)` method that adds `dataP` "at the beginning" of the calling object.

2. Write the `RemoveF()` method that deletes the "first" element of the calling object.

3. Write a `Size()` method that returns the number of elements in calling object.

4. Write a series of statement that creates a `CList` object containing `int`, adds cells containing 10 and 20, and display its size.

____ / 20 pts. **Problem 5** Write a `DisplayR(int valueP)` *recursive* method that displays all the numbers between `valueP` and `0`. For example, `DisplayR(3)` should display `3 2 1 0`, and `DisplayR(-2)` should display `-2 -1 0`.

____ / 25 pts. **Problem 6** Answer the following questions, *checking all that apply*:

1. What is the correct way of creating a 2-dimensional rectangular array of `int` with 5 rows and 2 columns named `myMatrix`?

   - ☐ `int[][] myMatrix = new int[5][2];`
   - ☐ `int[,] myMatrix = new int[5, 2];`
   - ☐ `int[][] myMatrix = new int[2][5];`
   - ☐ `int[,] myMatrix = new int[2, 5];`

2. What is the difference between **`ref`** and **`out`**?

   - ☐ **`ref`** variables are "read-only", their value cannot change inside a method.
   - ☐ There isn't any: they are both used to pass a value to a method.
   - ☐ **`out`** variables may not be initialized going into the method, but have to receive a value inside the method.
   - ☐ **`ref`** is a keyword, **`out`** is not.

3. The following method would *not* compile. Why?

```
public static void Test(int a, out int b){
    if (a > 0) { b = 12; }
}
```

   - ☐ A method cannot be **`static`** and have `void` as a return type.
   - ☐ Any **`out`** parameter must have its value set in the body of the method.
   - ☐ The **`else`** is missing.
   - ☐ The keyword **`out`** cannot be used in the header of a method.
   - ☐ An **`out`** parameter cannot be assigned a value.

4. A **`try-catch-finally`** block…

   - ☐ … will execute its **`catch`** block only if an exception is thrown inside the **`try`** block.
   - ☐ … will execute all of the statement in the **`try`** block even if an exception is thrown.
   - ☐ … will always execute its **`finally`** block.
   - ☐ … will execute its **`catch`** block only if no exception is thrown inside the **`try`** block.

Problem 1 Suppose given and initialized a `MAXWIDTH` integer constant, and a `filePath` `string` variable, write (on the first page) a program that

1. Asks the user to enter a message,
2. Stores that message in a file located at `filePath`, making sure that no line go over `MAXWIDTH` characters, "wrapping" it if needed (spreading on as many lines as needed).

Your program should be able to handle graciously unexpected issues.

Example execution

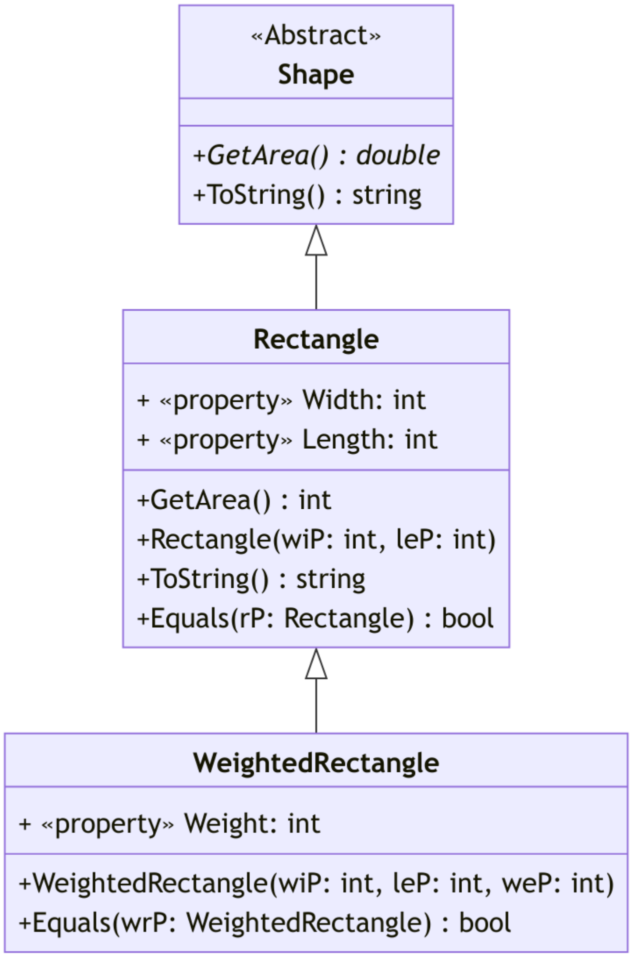
Assume `MAXWIDTH` is set to 70 and `filePath` to "C:/Users/test/Documents/msg.txt".

```
Enter a message
Text wrapping, also known as line wrapping, word wrapping or line breaking, is breaki
Now creating a file at C:/Users/test/Documents/msg.txt.
```

After execution, `C:/Users/test/Documents/msg.txt` contains:

```
Text wrapping, also known as line wrapping, word wrapping or line brea
king, is breaking a section of text into lines so that it will fit int
o the available width of a page, window or other display area.
```

Problem 3 Consider the following diagram:


«Abstract»
Shape
+GetArea() : double
+ToString() : string
Rectangle
+ «property» Width: int
+ «property» Length: int
+GetArea() : int
+Rectangle(wiP: int, leP: int)
+ToString() : string
+Equals(rP: Rectangle) : bool
WeightedRectangle
+ «property» Weight: int
+WeightedRectangle(wiP: int, leP: int, weP: int)
+Equals(wrP: WeightedRectangle) : bool