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 the duration of the class period (1 hour and 15 minutes). Answer the following questions and problems, trying to be as clear and as accurate as possible. 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. You can also abbreviate exceptions' names: for instance, write `AOORE` for `ArgumentOutOfRangeException`.

____ / 15 pts. **Problem 1** Consider the following **incorrect** implementation of binary search:

```
int[] arrayP = {10, 15, 30, 45, 60, 75, 90, 105, 120, 135, 150, 165};
bool fsf = false;            // Found target so far?
int tar = 165;               // Target to find.
int sta = 0;                 // Beginning
int end = arrayP.Length - 1;  // End
Console.WriteLine($"Starting ({sta}-{end})");
while (sta <= end && !fsf){   // Here goes the main loop
  if (tar < arrayP[(sta + end) / 2]) { end = (sta + end) / 2;
           Console.WriteLine($"Going left ({sta}-{end})"); }
  else if (tar == arrayP[(sta + end) / 2]) fsf = true;
  else { sta = (sta + end) / 2;
         Console.WriteLine($"Going right ({sta}-{end})"); }
}
Console.WriteLine("Found: " + fsf);
```

1. Indicate what would be displayed

2. Explain why this implementation is incorrect, and how it can be fixed.

___ / 20 pts. **Problem 2** We implement two methods for a simple `Calculator` **`static`** class and use one of them. The header of the methods is always of the form **`public static`** `int XXX(…)` with XXX standing for the name of the method, and … standing for the argument(s) and datatype(s).

1. Write a `Square` method that takes one `int` argument and returns its value squared if the parameter is less than 46341[1], and throws an `OverflowException` exception otherwise.

2. Write a `Divide` method that takes two `int` arguments and returns the result of dividing the first parameter by the second. If the second parameter is 0, then the method should throw an `ArgumentOutOfRangeException` exception.

3. Write a piece of code (to be inserted into a `Main` method) that asks the user to enter one number. Then, call `Square` with it (no need to loop). Your code should *not* check the value of the `string` entered for the operation, but must catch the exceptions potentially thrown by the `Square` method (*and, as a bonus, by the `Parse` method*). Your code should also display "Thanks!" regardless of whether an exception was thrown.

[1] The square of $46,341$ is greater than the maximum value an `int` can hold, `int.MaxValue`.

___ / 25 pts. **Problem 3** In this problem, we will implement a `Weather` class recording a location and a temperature (in Fahrenheit). Add attributes to your answers if needed.

1. Implement a `Location` `string` property using automatic properties. Its default value should be "Unknown".

2. Implement a `Temperature` `double` property. If the value passed as an argument to the setter is less than -128.6 or greater than 134.1, then the setter should raise an `ArgumentOutOfRangeException` exception. The getter simply returns the value stored.

3. Implement a `TempInC` `double` property with only a getter that returns the temperature in Celsius, using the following formula: $°C = (°F - 32) \div (9/5)$.

4. Implement a `ToString` method that returns a `string` containing the location and the temperature both in Celsius and Fahrenheit.

____ / 25 pts. **Problem 4** *To receive credit for this problem, answer the following questions using a 2-dimensional jagged array **or** a 2-dimensional rectangular array (your choice). To receive bonus points, answer using **both** a jagged array and a rectangular array.*

1. Write statements that declare a 2-dimensional array with at least 2 rows containing values 1 through 6.

2. Write statements that display the values stored in a 2-dimensional array called `arTest`.

3. Write statements displaying "Ranked" if the sum of the values stored in each row is greater than the sum of the values in the preceding row in the `arTest` array. For example, the following rectangular and jagged arrays are both ranked, as 4 < 5 < 6.

```
2 1 1
3 1 1
5 1 0
```

```
4 0
3 1 1
2 2 1 1
```

____ / 15 pts. **Problem 5** Answer the following short questions:

1. What would be displayed by the following?

```
static void Main(){
  int x = 4, y = 3;
  Console.WriteLine($"x is {x}, y is {y}.");
  AddR(ref x, ref y);
  Console.WriteLine($"x is {x}, y is {y}.");
}
static void AddR(ref int a, ref int b){
  int tmp = a;
  a = a + b;
  b = tmp - b;
}
```

2. The following method would *not* compile. Explain why.

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

3. Write, using a generic type parameter, a `First` static method that returns the first element of an array of any datatype (and be careful about **`null`**!).

4. Given an `int` `arrayP` array and an `int` `target` variable, write the code that performs a linear search: it should display "Found it!" if `target` is in `arrayP`.