**\Custom List Class Project (out of 150 points)**

**User Stories**

The built-in List<T> class is a generic class that acts as a wrapper over the array class. **You cannot use built-in List or Array methods.** For questions regarding *how* to approach a specific feature, please start by referring to the C# List<T> class documentation to get an idea of how the built-in List<T> class handles various situations and methods.

**(20 points)** As a developer, I want to use Test Driven Development (TDD), so that I can write tests for my methods to pass to ensure proper functionality within my application. There needs to be several tests per method.

**(5 points):** As a developer, I want to make good, consistent commits.

**(10 points):** As a developer, I want to use a custom-built list class that stores its values in a private array, so that I can store any data type in my collection.

**(10 points):** As a developer, I want a read-only Count property implemented on the custom-built list class, so that I can get a count of the number of elements in my custom list class instance.

**(10 points):** As a developer, I want a Capacity property implemented on the custom-built list class, so that I can publicly see the size of my private array.

**(10 points):** As a developer, I want to create a C# indexer so that I can make the objects in my list accessible via index. I want to properly ensure that a user cannot access an out-of-bounds index.

**(10 points):** As a developer, I want the ability to add an object to an instance of my custom-built list class by imitating the C# Add() method.

**(10 points):** As a developer, I want the ability to remove an object from an instance of my custom-built list class by imitating the C# Remove() method.

**(10 points):** As a developer, I want to be able to override the ToString method that converts the contents of the custom list to a string.

**(10 points):** As a developer, I want to be able to overload the + operator, so that I can add two instances of the custom list class together.

* CList<int> one = new CList<int>() {1,3,5}; and CList<int> two = new CList<int>() {2,4,6};
* CList<int> result = one + two;
* result has 1,3,5,2,4,6

**(10 points):** As a developer, I want to be able to overload the – operator, so that I can subtract one instance of a custom list class from another instance of a custom list class.

* CList<int> one = new CList<int>() {1,3,5}; and CList<int> two = new CList<int>() {2,1,6};
* CList<int> result = one - two;
* result has 3,5

**(5 points):** As a developer, I want to write documentation in a .txt file that describes the details and functionality of my – operator overload. I want to include details such as “syntax”, “parameters”, “return type”, and an example of it being used, with the output. I want to use the following piece of documentation as a guideline for my own documentation:

https://msdn.microsoft.com/en-us/library/cd666k3e%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396

**(10 points):** As a developer, I want the ability to zip two custom list class instances together in the form of a zipper. An example:

* I have CList<int> odd = new CList<int>() {1,3,5}; and CList<int> even = new CList<int>() {2,4,6};
* odd.Zip(even);
* When lists odd and even are zipped together, your new list will contain values 1,2,3,4,5,6

**(10 points):** As a developer, I want the custom list class to be iterable.

**(10 points):** As a developer, I want to use C# best practices, SOLID design principles, and good naming conventions on the project.

**(Bonus 5 points):** As a developer, I want the ability to sort an instance of my custom-built list class. To be eligible for the bonus points, you may not use Array.Sort() that is already built in and you must tell us what sorting algorithm you used.

**(Bonus 5 points):** As a developer, I want the ability to earn bonus points for an EASTER EGG user story, regarding implementing a specific good practice on one of the methods specified in a user story above.

**NOTICE: Get your unit tests (test methods) checked off by an instructor before you begin writing your methods to ensure you are on the correct path.**

- Grade: 75/150 = 50%

o 5/5 Good commits

o 5/20 TDD

o 5/10 Indexer

o 10/10 Store values in an array

o 10/10 Add method

o 10/10 Remove method

o 0/10 Iterable

o 0/10 ToString

o 0/10 Overload + operator

o 0/10 Overload – operator

o 10/10 Count property

o 10/10 Capacity property

o 0/10 Zip

o 10/10 Use C# best practices, SOLID

o 0/5 Minus operator documentation

- Commits

o Good job with your frequent and descriptive commit messages. With that said, remember to be consistently descriptive. For example, “added 4 tests” and “added 2 more test from study session” are not descriptive. Also, you don’t need to be as specific as “from study session”.

- Add

o Excellent job with your Add method! This is something we worked on together, and you did a nice job answering the questions when I posed them to you.

- Remove

o Well done with the logic for your Remove method!

- Indexer

o With a List, you are not able to access just any index of the underlying array. Instead, you can only access the objects that are in the list, essentially whatever object that is included in the Count. The way you have it set up, it can access any index of the underlying array. To fix this, you would need an if statement in the property that essentially factors in the Count. The else would throw an ArgumentOutOfRangeException.

- Count property

o Good job ensuring this is a read-only property

- Capacity property

o Good job having a get and set for capacity

- Iterable

o Your CustomList class needs to inherit and implement the IEnumerable interface

o CustomList<T> : IEnumerable

o You then would use a for loop and yield return each item in your array. HINT: look back at the Custom Iterator slideshow to see how this is done.

- ToString

o ToString() is a built-in method that every object has in C#. The goal is for you to override the built-in functionality and customize your own way of stringifying the contents of a list. You can use a for loop that iterates over each object in the items array and calls the .ToString() method on each item. Then you can use string concatenation to build out a complete string.

result = “”;

for(int i = 0; i < count; i++)

{

result += items[i].ToString();

}

return result;

o If you look back at our Slack conversation from Sunday, this is what I was getting at. You took my example that I gave you and put it in your project. I wasn’t giving you the code, but instead trying to give you an example of what the end goal is.

- Overload + operator

o You can use your Add method for this. The method signature you are looking for is public static CustomList<T> operator +(CustomList<T> listOne, CustomList<T> listTwo)

- Overload – operator

o You can either use your Add or Remove method for this, depending on your design. The method signature you are looking for is public static CustomList<T> operator -(CustomList<T> listOne, CustomList<T> listTwo)

- Zip

o Calls your Add method in a specific order (like a zipper) inside a loop. The items you are adding are from each index of both CustomList objects. Add(list1[0]) Add(list2[0]) and so forth. Once again, this is done in a loop.

- TDD

o You only have 8 unit tests when we expected somewhere between 25-30. You should have attempted to write the unit tests for the other methods, because that would have helped you understand the expected behavior, which would have helped with writing out the logic.

- The process for coding and solving problems

1. Understand the problem you are about to solve

2. Understand what the end product looks like of the problem you are about to solve

3. Break the problem down into smaller pieces

4. Code out the smaller pieces and bring them together to solve the problem

- I am excited about our plan going forward, especially if that helps you have a better understanding of what you are doing on the projects. Keep up it up man!