1. Retrofit the OurArrayList class to use generics.

Extra Credit – if you want more to do, add these methods to the OurArrayList class (no need to send to me) these are all methods in the  [IList<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.ilist-1?view=net-6.0) interface that List<T> inherits from.

1. The **Contains** method is used to test for the presence of an item in the list, true if item is found in the OurArrayList; otherwise, false
2. The **Insert** method is used to insert a new item in the middle of the list at the index provided (not greater then Count), shifting everything over, If index == Count, the item is added at the end of the list. If this insert brings Count to the capacity of the array a new array needs to be allocated and all the items copied over (like in the add method). Remember to adjust Count if needed.
3. The [**Remove**](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.remove?view=net-6.0) method is used to remove the first occurrence of a specific item from the OurArrayList, The method should return a boolean, true if item is successfully removed; otherwise, false. This method also returns false if item was not found in the [List](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-6.0). Keep in mind the need to shift the data into the slot where the item is removed from and the Count adjusted.
4. The **IndexOf** method is used to Determine the index of a specific item in the OurArrayList. Return the index of item if found in the list; otherwise, -1.
5. The **TrimExcess** method, this is the method a few of you were looking for it is part of the List<T>class (not the IList<T> interface).

The documentation describes it as such. It sets the capacity (the size of the array) to the actual number of elements in the [List](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-6.0) (the Count). “*This method can be used to minimize a collection's memory overhead if no new elements will be added to the collection. The cost of reallocating and copying a large List<T> can be considerable, however, so the TrimExcess method does nothing if the list is at more than 90 percent of capacity. This avoids incurring a large reallocation cost for a relatively small gain.”*

If you are still bored, Google how to overload the operator +, in our Shapes.Square class. Meaning add to the Square class the ability to add 2 Squares together and the resulting Square will have the length of both Squares combined. You should be able to write this code after you are done:

Square s1 = new Square(10);

Square s2 = new Square(20);

Square s3 = s1 + s2;

s3.length == 30

Still bored…...

Write or use a person class that has an age in it and overload the ++operator (incrementer) and --operator (decrementer) to increment or decrement the person’s age.

You should be able to write this code after you are done:

Person p = new Person;

p.Age = 20;

p++;

p.Age == 21

p--;

p.Age == 20;