Homework 2 – Custom List

**Due: Sunday, September 29th by 11:59PM**

First, practice using C#’s generic List<> class by creating a small program that allows the user to add, remove, insert and search for string data in a List<string>.

Second, implement your own custom data structure, called CustomList, that mimics the behavior of C#’s List<> class. The class will hold all of its data, aside from its count, in a single array. Using C#’s built-in List<> class inside your CustomList class is ***not* *permitted*** – your job is to make your own version that has the same behavior. Once it’s working, you’ll be completely replacing the List<> you used in part 1 with your custom implementation.

*If you’re not sure exactly how a method or property should behave given particular input, you can always check C#’s built-in List<> class’s version of the method. Make a temporary List<> at the beginning of Main() to test the method/property in question. Your list should exhibit the same behavior as the List<> class.*

Third, once your CustomList is working properly, update it to be generic.

# Task Overview

This is a brief overview of the tasks you must complete for this assignment. Specifics are given in the corresponding sections later in this document. ***Read the entire document before starting.***

### Activity 1

* Create a new C# project in Visual Studio
* Inside your Main method, allow the user to add, remove, insert and search for strings in a List<string>.

### Activity 2

* Create a CustomList class
* You may copy/paste your initial implementation from the Properties practice exercise
* Add the fields, properties, constructors and methods outlined below
* Alter your Main method to use CustomList instead of C#’s List<> class.

### Activity 3

* Make your CustomList class *generic*

# Activity 1

This entire activity should be implemented inside your Main method. Begin by asking the user for the initial capacity for their list, then create a List<string> using the proper constructor. Next, enter a loop that asks the user to enter a word.

If the user enters any of the special command words listed below, perform their corresponding action. If the word is not one of the listed commands, instead add the user’s input to the list.

Note: these commands should be handled regardless of their case (meaning, “indexOf”, “indexof”, “INDEXOF” and “iNdExOf” should all do the same thing). However, if the user’s input is not one of these commands, it should be added to the list without altering its case. You’ll need to structure your code in a way that ensures this functionality.

|  |  |  |
| --- | --- | --- |
| **Command** | **Action** | **Notes** |
| **done** | Ends the loop (and the program) |  |
| **print** | Prints out the elements of the CustomList. | You’ll need to loop through the list and print each element. Do **not** use a foreach loop for this, as your custom implementation later will not function with foreach loops. |
| **indexOf** | Asks the user for a word and prints the index of that word if it exists in the list. | Use the List’s IndexOf() method for this command. |
| **contains** | Asks the user for a word and lets the user know whether or not the word is in the list. | Use the List’s Contains method for this command. |
| **removeAt** | Asks the user for an index, verifies it is valid and removes the data at that index. | Use the List’s RemoveAt() method for this command. |
| **remove** | Asks the user for a word, attempts to remove it from the list (let the user know if it worked). | Use the List’s Remove() method for this command. |
| **insert** | Asks the user for a word, then an index, and inserts that word at that index. | Use the List’s Insert() method for this command. |
| **clear** | Clears the list. | Use the List’s Clear() method for this command. |
| **get** | Asks the user for an index, verifies it is valid and prints the word found at that index. | Use [ ]’s to retrieve the word from the list. |
| **set** | Asks the user for an index, verifies it is valid, asks the user for a word, and then stores the word at the specified index. | Use [ ]’s to store the word in the list. |

See the Sample Run at the end of this document for the exact output expected for this activity.

Once you’ve tested your code to ensure you’re handling each of the commands properly, move on to activity 2. Do not move on until you’ve tested your code.

# Activity 2

For this activity, you’ll be creating your own implementation of a List class using a simple array of strings. Note: you will **not** be using C#’s built-in List<> class anywhere in this activity. Your job is to build the List’s basic functionality yourself.

Below are the specifics of the class itself and the methods you are required to implement. Feel free to include any helper methods you may need.

## The Class – CustomList

Make sure to name your class CustomList, as C# already has a class named List.

## Fields

Your class should have exactly two fields: an **array** to hold the actual data (strings, for now) and a **count** of how much of the array is currently in use. You are not permitted to add any other fields to the class.

Remember: the count represents the amount of the array currently in use, not the length of the array.

## Properties

You’ll need 3 properties for your class:

* **Count** – this will get the current count of the list. There will be no set.
* **Capacity** – this will get the length of the internal array. There will be no set.
* **An *indexer*** – This will get or set individual elements of the array, based upon the specified index. If the index is invalid (below zero, or greater than or equal to the list's *count*), the *get* should return null, and the set should do nothing. Remember, this is a special kind of property that allows you to “index into” the object, not a regular property named indexer.

## Constructors

Create two constructors for the class, each of which initializes the data array.

* Default constructor – Initialize the array with a size of 4.
* Parameterized constructor – The parameter specifies the initial size of the array.

## Methods

Implement the following methods in your CustomList class. You should always aim to reduce duplicate code in your projects. As such, there are several opportunities here to have methods call other methods to simplify their implementation. Read through the following requirements and plan the implementations before starting to code.

#### void Add(String item)

The Add method will accept a string parameter and return nothing. It should put that item into the array at the next “open” position. If the array is full, Add() must create a new array that is double the size, copy the elements from the old array into the new one, and then add the data. Remember to keep track of the amount of valid data (count) in the array.

#### int IndexOf(String item)

This method will take a string parameter and will return an integer. It should search through the internal array for the specified data and return that data’s index. If the data is not found, return -1.

#### bool Contains(String item)

Contains takes a string parameter and returns a Boolean specifying whether or not the data is present in the array.

#### void RemoveAt(int index)

RemoveAt will take an integer specifying the index of the data to be removed from the array. If the index is invalid (negative, or larger than or equal to the count of valid data), do nothing. When removing valid data, do not leave any “gaps” in the array; data after the “removed” index should be shifted over to fill the gap. Update the count accordingly.

In other words, **do not just replace the data with null**. That is not an acceptable way to remove it!

#### bool Remove(String item)

Remove will take a string parameter and return a Boolean. It will look for the first instance of the specified data in the array. If the data is found, remove it and return true. If it’s not found, return false. Again, do not leave any gaps in the array.

#### void Insert(int index, String item)

Insert will take two parameters: an index, and a string. The data should be inserted into the array at the specified index. You’ll need to “make room” for the data in the array, so as to not overwrite any data. If there is not enough room in the array, you’ll need to perform the size-increase operation outlined in the Add() method’s description. If the index is negative, do nothing. If the index is greater than the count, simply add the data to the array at the next valid position (exactly as the Add() method would).

#### void Clear()

This method takes no parameters and returns nothing. This should “clear” the list. (Hint: the easiest way to do this is to simply reset the count so new data is added starting at the beginning of the array.)

# Main Method

Update your Main method by replacing all uses of C#’s List<> class with your own CustomList. Ideally, this will only require you to change the variable definition, as you should have implemented all of the methods and properties you used with C#’s List<>.

Your program’s output and functionality should be the same as in Activity 1. The only difference is that your program is now using your own custom data structure instead of a built-in one. If the functionality has changed, you’ll need to identify inconsistencies in your implementation and fix them before moving on to Activity 3.

# Activity 3

Alter your CustomList class to make it generic. This means it will be able to hold any type of data, not just strings (although you’ll still be using strings in the Main method).

## Class Definition

Update the class definition itself to be generic. This is where you use the < >’s and define the type specifier.

## Field & Indexer Updates

Update your internal data structure (the array) to use the generic type instead of hardcoding it as an array of strings.

Make sure the indexer uses the generic type, as well. If the index for the *get* section is invalid, you should return a default value of the type specified. Remember: there is a special C# keyword for this!

A more advanced class would throw an exception when an invalid index is specified, as we've asked for something that doesn't exist. However, since we haven't covered throwing our own exceptions yet, returning a default value is fine for this homework.

## Method Updates

Any method that accepted strings as a parameter should be updated to accept parameters of the generic type specifier instead. For example, the Add method’s parameter should match the type specifier, rather than always accepting strings. Depending on how you wrote them, methods like Clear and Print will probably not need to be updated at all.

## Main Method

Update the CustomList you created in your Main method to be a CustomList <String>. Your program should still run correctly at this point.

To demonstrate that you’ve correctly made your class generic, you should also create a CustomList<int> at the end of Main. Call several of the CustomList methods and print the results to ensure it works as intended.

# Sample Run

Enter a word: ***insert***

Which word to insert? ***Seven***

Which index? ***1***

Inserted “Seven” into index 1

Enter a word: ***print***

Printing list contents:

hello

Seven

Bees?

GDAPS

Enter a word: ***get***

Which index to retrieve? ***1***

Word at 1 is “Seven”

Enter a word: ***get***

Which index to retrieve? ***99***

99 is not a valid index

Enter a word: ***set***

Which index to set? ***17***

17 is not a valid index

Enter a word: ***set***

Which index to set? ***2***

What is the new word? ***BEES!!!***

“BEES!!!” has been placed at index 2

Enter a word: ***print***

Printing list contents

hello

Seven

BEES!!!

GDAPS

Enter a word: ***clear***

List has been cleared

Enter a word: ***print***

List is empty

Enter a word: ***Chair***

“Chair” added to list

Enter a word: ***print***

Printing list contents:

Chair

Enter a word: ***done***

Quitting program.

*<Output testing List<int> here>*

Welcome to the CustomList homework!

Specify the initial capacity of the list: ***2***

Enter a word: ***bob***

“bob” added to list

Enter a word: ***hello***

“hello” added to list

Enter a word: ***Bees?***

“Bees?” added to list

Enter a word: ***print***

Printing list contents:

bob

hello

Bees?

Enter a word: ***INDEXOF***

Word to search for: ***hello***

The index of “hello” is 1

Enter a word: ***inDEXof***

Word to Search for: ***RIT***

The index of “RIT” is -1 (it does not exist)

Enter a word: ***Contains***

Word to search for: ***goodbye***

Goodbye is NOT in the list

Enter a word: ***RemoveAt***

Which index to remove? ***-10***

That is not a valid index

Enter a word: ***RemoveAt***

Which index to remove? ***0***

Data at index 0 removed

Enter a word: ***dragon***

“dragon” added to list

Enter a word: ***GDAPS***

“GDAPS” added to list

Enter a word: ***remove***

Word to remove: ***dungeon***

“dungeon” was not found

Enter a word: ***remove***

Word to remove: ***dragon***

“dragon” was removed

Enter a word: ***print***

Printing list contents:

hello

Bees?

GDAPS