Assignment 1

Due Date: 11 April 2025

Weighting: 30%

Group or Individual: Individual

This assignment is to assess your knowledge about linear data structures and algorithms and your skills in applying the knowledge in the development of reusable Abstract Data Types (ADTs). This assignment also assesses your linear data structure-based algorithm design and analysis techniques.

In this assignment, you are given two (2) ADT specifications in C# interfaces and an ADT implementation skeleton for each of the ADTs. The two ADTs will be used in the development of a software application for a community tool library to manage the tools there. Your tasks are to complete the two ADT implementations. When implementing the ADTs, you need to design an efficient algorithm to solve a computational problem arising in the ADT implementations and theoretically analyse the time efficiency of your algorithm.

One ADT is *Tool*, which is used to model a hardware tool in the community tool library; another ADT is *ToolCollection*, which is used to store and manipulate a collection of tools in the community too library. In this assignment, you are provided the specifications of the two ADTs in C# interface, *ITool.cs* and *IToolCollection.cs*, a skeleton implementation (an incomplete implementation) of the tool ADT, *Tool.cs,* and a skeleton implementation of the ToolCollection ADT, *ToolCollection.cs.*

1. Your Tasks

a. Refer to the ADT invariants, pre-conditions and post-conditions in the ToolCollection interface, design an efficient algorithm for the ***Search*** method defined in the ADT using the pseudocode introduced in Lecture 1, and document your algorithm in a PDF file. The input of your *Search* algorithm must be an array of tools sorted in ascending order by tool's name.

b. Following the theoretical algorithm analysis presented in Lecture 2, theoretically analyse the worst case time efficiency of your search algorithm and document your analysis details in a separate PDF file.

c. Implement the following methods in light of the corresponding ADT invariants, preconditions and postconditions given in the interfaces:

- *Tool.cs – IncreaseQuantity*
- *Tool.cs – DecreaseQuantity*
- *Tool.cs – AddBorrower*
- *Tool.cs – DeleteBorrower*
- *Tool.cs – SearchBorrower*
- *ToolCollection.cs – Add*
- *ToolCollection.cs – Delete*

- *ToolCollection.cs – Search*
- *ToolCollection.cs – Clear*
- *ToolCollection.cs – IsEmpty*
- *ToolCollection.cs – IsFull*

A precondition is a condition, or a predicate, that must be true before a method runs for it to work. In other words, the method tells clients "This is what I expect from you". So, the method we are calling is expecting something to be in place before or at the point of the method being called. The method is not guaranteed to perform as it should unless the precondition has been met. A postcondition is a condition, or a predicate, that can be guaranteed after a method is finished. In other words, the method tells clients "This is what I promise to do for you". If the method is correct and the precondition is met, then the postcondition is guaranteed to be true. An ADT invariant is something that is always true and won't change. The method tells clients, "If this was true before you called me, I promise it'll still be true when I'm done". An invariant is a combined precondition and postcondition. It must be valid before and after a call to a method.

2. Assignment Requirements

- You must use your search algorithm to implement the *Search* method in the *ToolCollection* ADT; otherwise, a penalty may be applied.
- The programming language used in the ADT implementations must be C#.
- You must not use any third-party C# class libraries. Types and methods defined in the *System.Collections*, *System.Collections.Generic*, and *System.Linq* namespaces are specifically prohibited.
- You must not use any method defined in the Array class.
- You must not make any change to the given C# interfaces.
- You must not add any class field to, or remove any class field from, *Tool.cs* or *ToolCollection.cs*.
- You must not add any method to, or remove any method from, the interfaces.
- You must not add any method to, or remove any method from, the implementation skeletons.
- You must not make any change to the class fields in the class implementation skeletons.

3. Assignment Submission

- In CAB301, we use Gradescope to manage and grade your assignments. You can access Gradescope by clicking on the Gradescope tab in the sidebar (under Assignments). If you did not use Gradescope before, you can find a guideline for how to submit a CAB301 assignment to Gradescope under Modules/About Assignment/Submitting to Gradescope.
- You should submit a zip file containing:
    - Tool.cs
    - ToolCollection.cs
    - Algorithm analysis in a PDF file
    - Theoretical algorithm analysis in a separate PDF file
- You can submit your assignment multiple times before the deadline. But we only grade the last submission before the deadline, if you submit multiple times.