**Assignment Instructions** Please read the following instructions carefully. Failure to follow the instructions will results in losing at least 5 points.

- This is a group assignment. You may choose to form your own group of **exactly two students**. If you do not build your own group, you will be assigned a group randomly. Groups may be different from assignment 4. If you choose to work in the same pair, make sure you indicate that.
  Here are the general rules for teams:
  1. You should form the pair on Canvas by **Monday December 9 at 5 pm** using one of the groups under *People* → *assignment5-groups*.
  2. Build your team even if it is the same as in Assignment 4. Groups are not curried over.
  3. You should work on the whole assignment together.
  4. You should **not** just partition the work between you.
  5. You should upload only one submission to canvas.
  6. You may pair up with any student across the two sections.
- Hand in all program header/source files softcopy using Canvas and be sure that they properly execute. Failure to do so will mean that your program is not graded.
- All header/source files should be compressed into a *ZIP* archive using the following naming format: `firstname1_lastname1_firstname2_lastname2_asmt5.zip`.
- Your ZIP file should include the exact same files that have been provided to you, in the same structure. You should add `User.h` and `User.cpp`. You may replace the existing `LinkedBagDS` with the modified implementation from Assignment 2 and 3, but make sure it is included as `LinkedBagDS`.
- Your answer for **Part 3** should be submitted as a single PDF file **through Canvas**. Do **not include it in the ZIP** file.
- Please include your full names in the PDF.
- Please refer to the university integrity policy on the syllabus, and remember that **you should be able to explain and reproduce any code you write and submit as part of this assignment.**

## Assignment Goals and Objectives

1. Programming methodology:
   - Examine graph implementation and design.
2. C++ implementation:
   - Analyze an adjacency list-based graph implementation
   - Modify existing code to adapt it to application requirements

## Part 1: Adjacency LinkedBag – 40 points

You are provided with an adjacency list implementation of the graph ADT. This implementation is based on two STL data structures: Vector and List.

1. You should change the implementation to use `LinkedBag` data structure instead of `list`. Your changes should be reflected both in the header file (i.e. `Graph.h`) and source file (i.e. `Graph.cpp`).
2. You should modify the implementation such that it does not allow the addition of self-edges.
3. The initial version of the `LinkedBag` implementation is provided in `LinkedBagDS` (i.e. it does not include `Node<ItemType>* findKthItem(const int& k)`). You may replace it with the modified implementation from Assignment 2 and 3 to be able to use `findKthItem`.

## Part 2: Building a Graph – 25 points

We would like to allow Instagram340 Users to add each other as friends. We will implement this task independently of the project you have implemented so far. All you need to include from your previous submissions is the class `User`.

In the file `Instagram340_Graph_main.cpp`, create 10 distinct users and add them to the vector `users`. Create a graph using the modified implementation of `Graph.h/.cpp`, where users are friends with each other.

1. The graph should include all 10 users
2. It should include 15 to 20 edges.
3. Each Instagram340 User will be represented by their index in the vector (i.e. vertex 0 in the graph is the User stored at index 0 in the vector).
4. The graph should either be directed or undirected. You should make that decision and explain your choice in a comment.

5. Assign random single value weights to all edges. The weight could represent different information: frequency of interaction, length of friendship, etc.

***Note:*** Do not worry about adding posts. However, since we are not including the `Post.h/.cpp` files in this submission, you may encounter compilation errors. There are several ways to resolve these errors: you can either remove all `Post`-related functionality from the `User` class, change the post type from `Post` to `string`, or choose any other approach that works for you.

**Part 3: Graph Topology** – 10 points
Provide a plot of your graph. You may draw it by hand or use any app/software (e.g. `https://draw.io/`). For each node, you should indicate the integer/index used to represent the User as well as the User's username.

**The figure should be uploaded separately to Canvas as a PDF file (i.e. do not include it in the ZIP file).**

**Part 4: Recursive Depth-First Traversal** – 10 points
A recursive implementation of Depth-First Traversal is provided in `Graph.cpp`. This implementation prints the node's value, which is the index of a User. We would like it to print the User's information instead.

1. Change the function prototype both in the header file and source file.
2. Change the implementation of the function to be able to print out Users' information instead of the indices.

**Part 5: Recursive Depth-First Search** – 15 points
Add two new functions: `DFS` and `DFSRecursive`. You may use the implementation of Depth-First Traversal provided in `Graph.cpp` and modify it such that it searches for a certain value instead of processing all nodes.

1. The function should return `true` if the value is found and `false`, otherwise.
2. The function should search for a User by `username`.