CSCI 2275 – Data Structures and Algorithms
Instructor: Hoenigman
Final Project
Due Dates:
Phase 1: Wednesday, November 25, 4pm.
Phase 2: Monday, December 7, 4pm.
Phase 3: Wednesday, December 9, 4pm.
Interview Grading: Week of December 7 through December 12.

In this final project, you will be creating a program of your own design that uses either the data structures we have learned this semester or more complicated data structures and algorithms that we haven't discussed. You are welcome to work in teams of two people or work alone. You will be storing your code on GitHub, which is a popular source code management system.

The project is divided into three phases.

Phase I
The first phase of the project is the design phase. In this phase, you will determine if you will work alone or with another person, where you will store your code, and what you will build. In recitation this week, you will learn about GitHub, a popular system for storing source code, and create a GitHub account and the code repository for your project. You will also contribute a Read Me file to your code repo that describes who is on your project team and what your project is.

Phase 2
The second phase of the project is where you actually create the code that you planned in Phase 1. The majority of the coding you will do for the project occurs in Phase 2.

Phase 3
The third phase of the project is the evaluation and rework phase. You will evaluate one other project by your classmates in your recitation using a grading rubric that will be provided. Your classmates will also be evaluating your project, which gives you an opportunity to fix any bugs they find before we do the final interview grading with the TAs and your instructor.

Phase 1 Requirements:
Recitation 10 is an introduction to GitHub. There is a write-up on Canvas that shows you how to create a GitHub as well as a short video tutorial about GitHub - what it is and how to use it. Work through the steps in the recitation exercise to create a GitHub account. If you already have a GitHub account, you are welcome to use that username.

If you are planning to work with someone else, preferably someone in your recitation, start discussing the project now. The technical requirements for the project are listed under Phase 2 Requirements.

What to submit for Phase 1
Create a project repository under either yours or your teammate's account, you will both contribute to the same repository. Create a Read Me file that includes at least one paragraph that summarizes your project.

Submit the URL for your project repository, and the Github user IDs for members of your project team, to the Canvas Final Project - Phase 1 Submission. In the submission comments, list your project teammate if you have one. **The project repo should be created under one person's account, but everyone on the team should submit the URL for the project.**

Phase 2 Requirements:

In Phase 2, you will be writing the code for your project. There is no set assignment that you need to implement, it's all up to you. Do whatever interests you and satisfies the following technical, design, and documentation requirements.

Phase 2 technical requirements:

- Contains at least two classes.
- Contains at least two data structures that we've studied this semester, or at least one that we studied and one that we didn't.
- One of your classes should contain at least 10 public methods, the other class can have fewer than 10.
- Contains a well-documented driver file to run your library code.
- Your library can't be an assignment from the semester with no additions. However, you can start from an assignment and extend its capabilities.
    - e.g. A Graph class to calculate Dijkstra's algorithm not enough. A graph class with several shortest path algorithms is.

Phase 2 code design requirements:
In Phase 2, your code should be designed to match these stylistic requirements:

- Your class interface and implementation should be in separate files. This is the style we've been using all semester with *Graph.cpp* and *Graph.h* being separate files, for example.
- Your driver file is a separate source file that uses the public methods in your class. In our assignments, an example of a driver file is *Assignment8.cpp*.
- If you have other helper functions, they should be stored in a separate source file and the function prototypes should be listed in a .h header file. For example, if you create a file called *utilities.cpp*, there should also be a *utilities.h* that is included in any .cpp files that use the functions in utilities.

Phase 2 Read Me file requirements:

Your project needs to include a Read Me file that includes all the instructions needed to download your code and run it. Your Read Me file needs to contain the following sections.

Project Summary
One paragraph description of how the program works and what it does.

How to Run
This section contains instructions, with examples, of how to run your program.

Dependencies
This section contains a description of any dependencies that your program requires. For example, if your program relies on another third-party library that needs to be installed, you should provide a description of where to find that library and instructions for how to install it or a link to installation instructions.

System Requirements
Is your program for Windows, Mac, Linux? Are there additional system requirements other than the operating system?

Team Members
List the people who worked on the project.

Contributors
List the people who were not on the project team, but may have contributed comments, enhancements, and bug fixes.

Open issues/bugs
List any known bugs in the project, and any open enhancement requests.

Phase 2 collaboration recommendation:
If you are working with a partner, it is recommended that each person be responsible for a separate file. Merging file changes is difficult in GitHub and it's very easy to lose work by unintentionally overwriting one person's changes.

Phase 3 Requirements:
In Phase 3, you are going to evaluate another project.

Phase 3 evaluation requirements
Your TA will assign you one other project to evaluate from your recitation. If you would like to evaluate more than one project, you are welcome to do so. There is a quiz on Canvas called **Phase 3 Project Quiz: Part A** that you need to complete for the project you evaluate. Your evaluation needs to include whether the project runs, the completeness of the code and documentation, and general interestingness of the project. Additionally, any issues you encounter will be added as issues to the project

repo that the owner can fix. The evaluation for the project needs to be completed by **Wednesday, December 9**.

Other Project Details

Grading
Phase 1 and Phase 3 contribute 25% to the grade and Phase 2 is 50% of the final project grade.

Teams
You can work in teams of up to two students. You can also work alone if you would like. If you want to work on multiple project teams that's also fine, but you will only get one grade for the project. Each teammate is responsible for evaluating another project.

Phase 3 Evaluation Guidelines
Evaluating your peers

As part of Phase 3 of your final project, everyone will be evaluating two other projects. Evaluating the work of your peers, and having them evaluate you, can be extremely educational for all parties involved. It can also be extremely Terrifying! I mean, seriously, someone else other than a TA or Instructor is going to look at your code, search for bugs, comment on what works and what doesn't, and it can be hard not to take it personally.

The first rule of peer evaluation is to be respectful. This doesn't mean that you should give high marks to bad work, or overlook the fact that the code doesn't run. It means that your comments should focus on the work and offer useful suggestions for how to fix the problem, if you have suggestions.

Don't be like Stack Overflow
There are several examples on Stack Overflow of how NOT to interact with others. We've all seen responses that are obnoxious, focused on petty details, and just generally unhelpful. For some unknown reason, people tolerate this behavior. It might be because the forum is effectively anonymous, anyone from anywhere in the world can comment and you never have to see these people face-to-face. In this class, you have classmates who you may be in classes with for the next couple of years. Be nice.

Put yourself in their place
Before you give feedback, imagine you are the person receiving that feedback and ask yourself what you would find the most helpful. If you think it wouldn't be very nice to have to read it, then the other person is probably going to feel the same way.

Example: After downloading and testing the code, you find a bug in the code that causes a segmentation fault. Before entering an issue on the code, you look at the code and notice that the code is using a linked list. Examples of good and bad text to include in the issue are shown here:
Good response: The code generates a segmentation fault when I delete the last node from the list. I added five integers to the list, 1, 2, 3, 4, 5, and when I tried to delete the 5 I got a seg fault.

Bad response: The code generates a segmentation fault when I delete the last node from the list. I added five integers to the list, 1, 2, 3, 4, 5, and when I tried to delete the 5 I got a seg fault. You really should have used a vector, I'm not sure why you didn't.

The text in the good response tells the programmer the behavior that you observed and when you observed it. The text also includes the steps that you took to generate the error, including the values added to the linked list before the delete operation. The bad response also includes this useful information, but also includes some advice that doesn't really help.

Evaluation guidelines
Technical components to evaluate

1. Follow the instructions in the project Read Me for installing their code. (If it doesn't work the first time, verify that you followed the steps exactly.) If the code generates an error, go to the project URL in GitHub and open a new issue by selecting the Issue link on the right-hand side of the repo, and then clicking the New Issue button.
    a. For the Title, enter a description of the problem. It should be descriptive. For example, "Code broken" is a bad title, but "Seg fault when deleting node" is a good title.
    b. In the comment box, enter a description of the steps you took to get the error, including values entered, and the error message or unexpected behavior you observed. If there is an error message, you can even copy and paste the error text into the box.
2. If there are problems, such as dependency not found, file missing, or directions unclear, these are also items that can be entered as issues.
3. Once the code builds, determine from the Read Me, code, or other documentation which file has the main function that is the entry point to the code. Run the code.
4. Try a few different options in the code, select a few menu items if it has a menu, play the game if it is a game, etc.
5. **Try to break the code.** Can you imagine corner cases or unusual inputs that the developer may not have considered? These are great places to find bugs. If you find a bug, enter an issue.
6. Open the .cpp and .h files. Check the documentation in the .cpp file. Do the methods have comments that describe the method's functionality, including a

brief description of what it does? If documentation is missing or incomplete, enter an issue.

7. Can you follow the documentation, including the Read Me, source comment blocks, or other documentation to know what to expect from the code?
8. Does the driver file include enough examples of how to use the library code? For example, if the library code has methods for building, inserting, and deleting nodes from a binary search tree, does the driver file provide examples of how to do all of these things? If you think more examples are needed, enter an issue.

Phase 3 –Responding to your peers

Once your project is evaluated and issues are entered, you have the opportunity to respond to the issues. Take the issues seriously and fix them before the TAs look at your project for final grading. You can mark an issue as Resolved once you have fixed it. The TAs will evaluate how many issues are marked as Resolved by either you or someone else who decides to contribute to your project.