Lab12

1. Lab Topics

This lab primarily covers git collaboration, arrays, structs inside struct, and usage of the chained dot operator.

2. Groupings

In this lab (lab12), you will be grouped together to work on a project. Each group will consist of four (4) students and the project is divided into two (2) major tasks. The workload should be split between each group member as the group deem fit. I'll advice to assign two members to each task. Such that you create a subgroup.

You are provided with the coding guidelines in the repository.

3. Working with Git.

Follow the instructions below:

1. Each group member should clone the repository of their group (G1-G5) to their own computer. This is entirely different from the normal repository you usually work with. It follows this format below but remember replacing the *??* with your group name.

```
git clone https://git-classes.mst.edu/2023-FS-CS1580-305/f23_cs1580_lab12_??.git
```

2. Before making changes, create a new branch for the task you are working on. As seen below. Make sure to replace `teacher_portal` with `student_portal` depending on your chosen task. This ensures that you are not making changes directly to the main branch.

```
git checkout -b teacher_portal
```

3. Write your code on this branch you are on. Make necessary changes to the templated code provided. Add and commit your changes. Refer to section 4 for what you are to code.

```
git add .
git commit -m "Descriptive commit message"
```

4. Push to your own branch, which can be `teacher_portal` or `student_portal`. An example is shown below, make sure to edit `teacher_portal` according to the branch you are working on.

```
git push origin teacher_portal
```

5. Open a Merge Request aka Pull request (PR) for the code you just pushed to your branch. This can be done through the pop-up or by creating it yourself. View Figure 1 below.

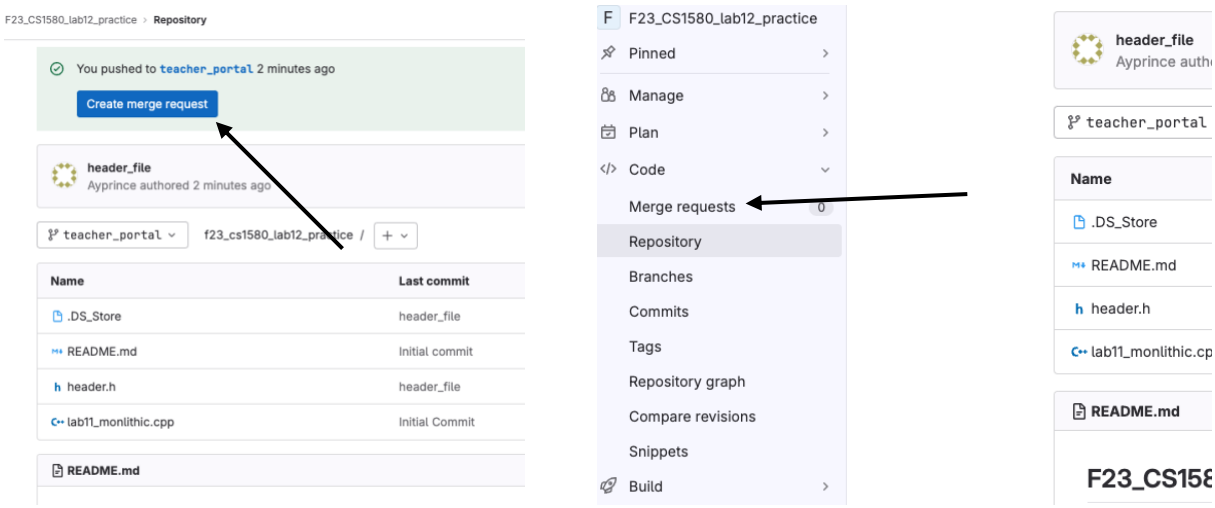

Figure 1: (a) shows the automatic pop up for a merge request. (b) shows how you can create a merge request by navigating to the menu.

6. If you are creating a new merge request, select Source (the branch containing your changes) and target branch (the branch where you want to merge your changes, for this lab, the main branch). See figure 2 below.

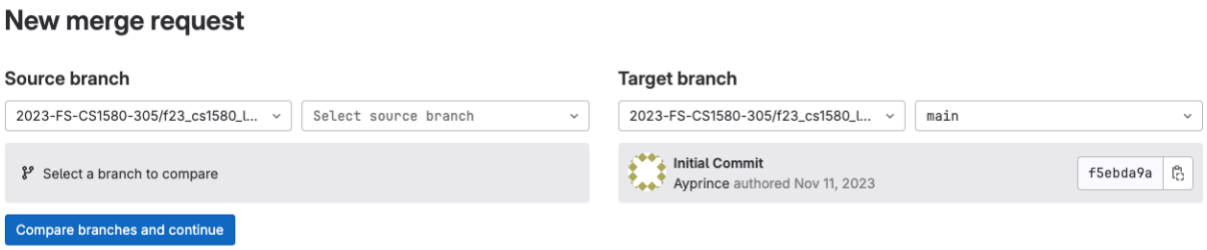

Figure 2 shows the merge request.

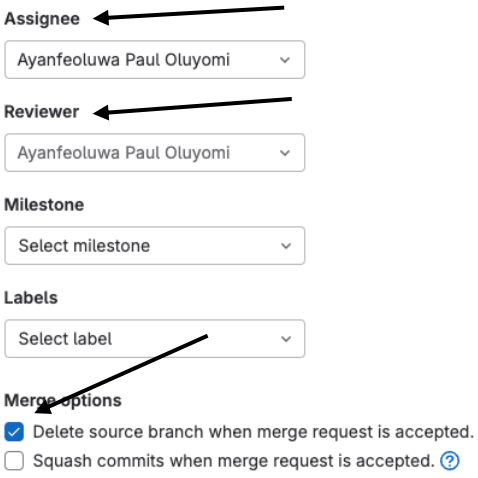

Figure 3 Shows what you are expected to uncheck and how you are to assign to an assignee and reviewer.

7. When creating the merge request for your subgroup, assign it to a member of the other subgroup, and let the other member of the subgroup be the reviewer. Make sure to **UNCHECK** the "Delete source branch when merge request is accepted". Part of your grades depends on this. This is shown in figure 3.

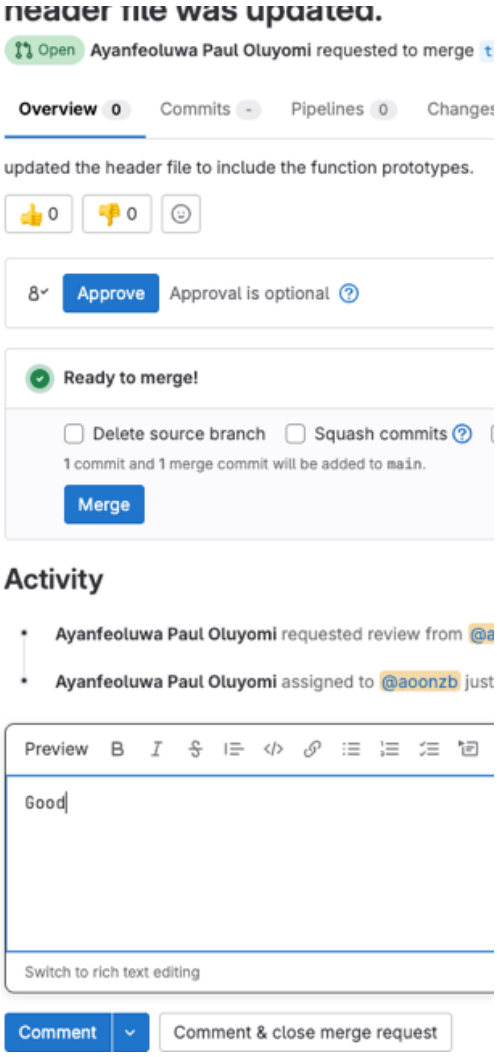

Figure 4a

8. The reviewer of the merge request **must leave a comment** before merging.

To know you are on track, you can click on the merge request tab as shown in step 5, fig 1b. What you have should be similar to figure 4a&b below.

If you view your main branch, you should see the changes made in the commit branch.

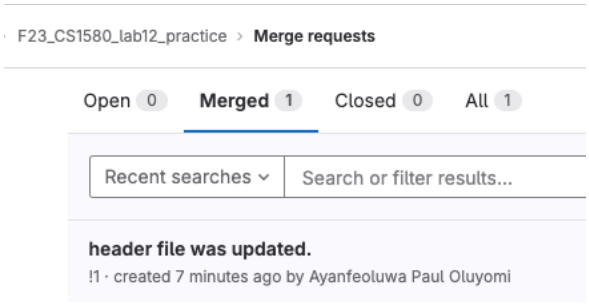

Figure 4b

Fig. 4. (a) Demonstration on how the reviewer will leave a comment. (b) viewing the merge request.

9. Stay Updated by regularly pulling the latest changes from the main branch to keep your local copy up-to-date.

```
git pull origin main
```

10. In a situation where conflicts arise during the pull, resolve them locally before pushing your changes.
11. Keep your main repository organized, follow a consistent folder structure, and use a clear naming conventions for branches and commits.

NOTE: Students are free to research more on any concept they might need.

4. Coding assignments.

Step 1:
There are two major parts, completing the code for the student_portal.h and student_portal.cpp. The second task is completing the code for the teacher_portal.h and teacher_portal.cpp. These tasks can be divided among the group members.
See the TODO in the files provide for coding requirements.

Step 2:
Each subgroup should test their code before pushing it to the main branch by copying their file to the unit_test_?? folder they are working with. Make sure to also change how the header is being included in the copied .cpp file.

a. For the student_portal, you should have the following after running your code.

```
Course Statistics:
Course Name: C++ Programming
Instructor: Dr. Smith
Number of Students: 3
Highest Score: 95.5
Lowest Score: 70
Average Score: 85.2
```

b. For the teacher_portal, you should have the following after compiling.

```
Course Name: Test Course
Instructor: Dr. Test
Number of Students: 3
Student ID: 123
Course Major: ABC Engineering
Student ID: 456
Course Major: XYZ Science
Student ID: 789
Course Major: LMN Arts
Score: 85.5
Score: 90
Score: 75.3
```

Step 3.
All/Any group member can work on the main function. Follow the instructions provided in the main.cpp file. The main idea of the main function is for you to declare a new *CourseDetails* and *CourseStatistics,* insert necessary details from the *CourseDetails* into the *CourseStatistics*. From the scores arrays in the *CourseDetails,* obtain the highestScore and lowestScore along with the averageScore. Insert this into the *CourseStatistics*. Finally, call the overloaded insertion operator to output the *CourseStatistics* in a good format.

Step 4.
After confirming that the code is working as expected, each subgroup is to merge their branches **without deleting** these branches. Each member can pull, but a designated member should compile all the codes written and the output should look like this.

```
Enter details for a course:
Enter course name: Comp_sci
Enter instructor name: San
Enter the number of students: 2
Enter student ID for student 1: 19WE
Enter course major for student 1: Math
Enter score for student 1: 45
Enter student ID for student 2: 18ER
Enter course major for student 2: CompSci
Enter score for student 2: 89

Course Statistics:
Course Name: Comp_sci
Instructor: San
Number of Students: 2
Highest Score: 89
Lowest Score: 45
Average Score: 67
```

Step 4.
Take the following screenshots and push them along with your updated code to the group's git repository.

1. The result of the two unit test which should be **exactly** as what is given in step 2.
2. The outcome after completing the exercise as what is given in step 4.
3. Two (2) screenshots similar to what is in fig. 4a & b that shows the activity on that repo and the number of merges you've done as a group which should be at least two (2).

5. How to get full marks

To get a 100% on this lab, follow this requirement:

1. **Make sure to follow all instructions given in this document.**
2. There must be at least 2 branches in the repository of your group. Which is the student_portal and the teacher_portal.
3. When pushing your final code, make sure to put the names of all group members at the top of the main file.
4. Each student will inherit the score of the entire group, unless I receive a report that a student did not participate, and this report is attested to by the other members of the group. In that case, the student shall receive a grade of zero (0).

More explanation of the code structure.

1. Define a Student Structure:

Begin by creating a structure named Student that holds information about a student. This structure should have two members: studentID (a string) and courseMajor (also a string).

2. Create a CourseDetails Structure:

Develop another structure called CourseDetails, which encompasses the details of a course. It should include the following members:

- students: an array of Student structures.
- courseName: a string representing the course name.
- instructor: a string indicating the instructor's name.
- scores: an array of doubles for storing student scores.

Additionally, overload the extractor operator (>>) to enable the input of course details.

3. Implement CourseDetails Extraction Operator:

In the overloaded operator>>, ensure the correct input of course details, including student information and scores.

4. Code a CourseStatistics Structure:

Define a new structure named CourseStatistics to manage statistics for a course. Its members should include:

- courseName: a string.
- highestScore: a double.
- lowestScore: a double.
- averageScore: a double.
- numStudents: an integer.
- instructor: a string.

Further, overload the insertion operator (<<) to facilitate the output of course details in a readable format.

5. Implement CourseStatistics Insertion Operator:

In the overloaded operator<<, format and output the course statistics in a visually appealing manner.

6. In the main function, declare as the need arise and calculate the statistics, assigning it the course statistics.