Overview: In this lab, you will work in groups of three, experimenting in the use of git (for your local version of a repository) and GitHub (where, via git, the three of you will combine your work). You will ultimately implement a working solution to a collection of C++ classes/programs that includes `CharacterVector.{h,cpp} DoubleVector.{h,cpp} IntegerVector.{h,cpp} VectorTester.cpp`

Each class depends on the other two classes, and `VectorTester.cpp` will eventually test all three classes.

Groups: You have been assigned at random into the groups below, with associated GitHub repository listed:

<table>
<thead>
<tr>
<th>team</th>
<th>netids*</th>
<th>repository URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>delta</td>
<td>mh2 : dm9 : an7</td>
<td><a href="https://github.com/dszajda/cmsc240_f2020_lab6_delta">https://github.com/dszajda/cmsc240_f2020_lab6_delta</a></td>
</tr>
<tr>
<td>foxtrot</td>
<td>nm7 : dm2 : hj9</td>
<td><a href="https://github.com/dszajda/cmsc240_f2020_lab6_foxtrot">https://github.com/dszajda/cmsc240_f2020_lab6_foxtrot</a></td>
</tr>
<tr>
<td>hotel</td>
<td>js4 : nc4 : hp8</td>
<td><a href="https://github.com/dszajda/cmsc240_f2020_lab6_hotel">https://github.com/dszajda/cmsc240_f2020_lab6_hotel</a></td>
</tr>
<tr>
<td>juliet</td>
<td>ca5 : ca3 : dm4</td>
<td><a href="https://github.com/dszajda/cmsc240_f2020_lab6_juliet">https://github.com/dszajda/cmsc240_f2020_lab6_juliet</a></td>
</tr>
<tr>
<td>kilo</td>
<td>ka2 : md5 : mb5</td>
<td><a href="https://github.com/dszajda/cmsc240_f2020_lab6_kilo">https://github.com/dszajda/cmsc240_f2020_lab6_kilo</a></td>
</tr>
<tr>
<td>lima</td>
<td>ch3 : kw9 : sh5</td>
<td><a href="https://github.com/dszajda/cmsc240_f2020_lab6_lima">https://github.com/dszajda/cmsc240_f2020_lab6_lima</a></td>
</tr>
<tr>
<td>oscar</td>
<td>as6 : na2 : yz7</td>
<td><a href="https://github.com/dszajda/cmsc240_f2020_lab6_oscar">https://github.com/dszajda/cmsc240_f2020_lab6_oscar</a></td>
</tr>
</tbody>
</table>

Your assigned class will be `CharacterVector`, `DoubleVector`, or `IntegerVector` based on the order in which your abbreviated netid appears above. For example, for team “delta”: mh2 takes `CharacterVector`, dm9 takes `DoubleVector`, and an7 takes `IntegerVector`.

Getting Started:

- First, you need to have a GitHub user name. If you already have one, please email it to me (so that I can grant you permission to collaborate on the appropriate repository). If you don't have one, go to https://github.com and create a user name, then email it to me.
- Next remotely log in to one of the mathcs* machines, and then create a new lab6 directory within your own directory structure for this course. Then change into your lab6 directory.
- Begin by cloning the original version of your repository, using your URL given in the table above:
  ```
git clone https://github.com/dszajda/cmsc240_f2020_lab6_{yourTeam}
  ```
- Change into the resulting directory.
Process:

- First, and very important! In class, I used the term master for the “primary” branch. As of October 1, the primary branch for any repository created will by default be called main. Older repositories will retain master as the primary branch.

- Using the commands below, you will show the current local git branch(es), create a new branch, and change to that branch. Use your NetID or name (or similar) for BRANCH_NAME below.

  ```
  git branch # show which branch you’re on -- master
  git branch BRANCH_NAME # create a new branch
  git branch # show you’re still on master
  git checkout BRANCH_NAME # change to your new branch
  git branch # show you’re on your new branch
  ```

- Now do your work on your own non-master branch. Specifically:
  - In the two .cpp files that are not assigned to you, add code to the non-void stubs sufficient to make them compile (just return 0).
  - Add the appropriate code to the methods in your assigned .cpp file.
  - Add code in the appropriate spot in VectorTester.cpp to fully test your class’s implementation. (Do not yet add tester code in the “appended-to” sections.) Because one of the classes relies on characters, make sure that you are inserting integer/double/character values that correspond to printable ASCII characters (see https://www.ascii-code.com/).
  - Use make to compile your code, and fix any errors, repeating until your implementation is correct.

- Commit your work on your non-main branch. It is good practice to regularly check the status of your commits and to regularly pull from the remote (GitHub) master branch to pull in any updates others have made. (If you receive conflict issues when trying to pull, see the “Handling Pull Conflicts” section below.) Do not add any .o files nor the VectorTester executable for committing.

  ```
  git status # check status of commits
  git pull origin main # updates your local main and your branch
  git commit -a -m "BRIEF MESSAGE INDICATING YOUR CHANGES"
  git status # should show your changes have been committed
  git pull origin main # in case others made changes to remote main
  ```

- Now switch back to the main branch, then merge your non-main branch to the main. (By using cat, you’ll see that your updates aren’t seen in the main branch until you merge.)

  ```
  git branch # should show currently on your branch
  git checkout main # switch to the main
  git branch # should show now on main
  cat YOUR_CLASS.cpp # show main’s (older) version of .cpp file
  git merge BRANCH_NAME # merge your branch into the (current) main
  cat YOUR_CLASS.cpp # show main’s updated version of .cpp file
  ```

- Now push your changes to the shared repository on GitHub:
CMSC 240, Prof. Szajda, Fall 2020 Lab 6: git & GitHub Due: Wed, Oct. 14, 5:00 pm

The repository on GitHub should now reflect the changes seen on your local repositories. Visit your GitHub repository in a browser and you should see your (and perhaps others') code changes.

- Now continue to do work on your non-main branch, adding the additional “appended-to” tests in VectorTester.cpp corresponding to your assigned class. Merge your non-main branch into your main, push, and repeat the process until your team has a fully-implemented version of all classes and the tester — on GitHub and in each of your local repositories.

Handling Pull Conflicts: When doing a fetch-and-merge from the remote repository (via `git pull origin main`), git will do its best to merge your local repository’s code with (potentially different) code that exists on the remote repository. In many cases — particularly if you make a practice of regularly keeping your local repo up-to-date via regular pulls — git can itself handle merging using the “recursive strategy”. In such cases, you should see a message similar to the following:

```
Merge made by the 'recursive' strategy.
```

In some cases, however, git’s recursive strategy will not work, and conflicts will result that you must handle explicitly. (One such situation is a conflict of a teammate’s “good” code in their assigned class versus your “stub” code in their assigned class.) When git cannot automatically merge, you will see a message similar to the following:

```
% git pull origin main
From https://github.com/dszajda/cmsc240_f2020_lab6_alpha
  * branch main -> FETCH_HEAD
Auto-merging VectorTester.cpp
Auto-merging IntegerVector.cpp
CONFLICT (content): Merge conflict in IntegerVector.cpp
Auto-merging DoubleVector.cpp
CONFLICT (content): Merge conflict in DoubleVector.cpp
Auto-merging CharacterVector.cpp
CONFLICT (content): Merge conflict in CharacterVector.cpp
Automatic merge failed; fix conflicts and then commit the result.
```

You will then need to edit each file for which there is a conflict:

- Remove any of the following lines added by git:
  ```
  <<<<<< HEAD
  ========
  >>>>>> (hash)
  ```
- Delete any incorrect lines of code (e.g., old stubs).
• Keep the correct lines of code.

Then commit your code on your non-main branch, merge into your main branch, and then push:

```bash
git commit -a  # (will offer a merge comment in vim – just :wq to accept)
git branch
git checkout main
git branch
git merge BRANCH_NAME
git push origin main
git remote show origin
```

---

**Submitting and Naming:** After all team members have pushed all their final changes, each person should do a final pull from the origin main. In this way, you should all have an (otherwise) identical local main branch within your own directory.

Package your `lab6` directory (containing your local repo) into a gzipped tarball a la `cmsc240_lab6_neitd.tgz` and submit as usual using the email address provided below. Note that each member of your team needs to submit their own gzipped tarball, since part of the grading of the lab will be ascertaining that you all end up with the same repository! Your lab is due by 5:00 on Wednesday, October 14.

The the email address for this lab is

Lab6.klxe6rwpkohzjzac@u.box.com.