Assignment 1 README

Here's a description of the first programming assignment for CSE 100, Fall 2013. In this assignment, you will implement a binary search tree as a C++ class template that conforms to the basic conventions of the C++ Standard Template Library.

>>> Due: Fri Oct 11 8:00 PM

>>> Required solution files: BST.hpp BSTIterator.hpp BSTNode.hpp

This means that the file that contains your solution to the assignment must be turned in by the due deadline. The turnin procedure is described below in the section"Turning in your assignment".

It's a good idea to plan to do the assignment well before the deadlines; terminals and tutors can get very busy at the last minute.

In doing this assignment, it's your responsibility to understand the course rules for integrity of scholarship.

The files that contain your solution to the assignment must be turned in by the due deadline.

You may develop your code anywhere, but you must ensure that it compiles and runs correctly on ieng6 (the lab machines), because that's where we'll be testing it.

Part 1 - Getting Started With Github

You must follow these steps to get the starter code and to get set up to do assignment 1. Do not directly copy files from any directory that you find on ieng6, as has been the procedure in previous offerings of this course!

First things first. You will need to generate a public/private SSH-RSA key for use.

Windows or Mac:

If you are using Windows or a Mac for your primary development please see here for details on installing and setting up Github.

Linux and ieng6:

For Linux distributions and for accessing Github from ieng6, you will have two options: HTTPS access or SSH access. SSH access is easier in the long run, but HTTPS access will also work fine.

We strongly recommend you develop on ieng6. We will be testing your code there, so it is your responsibility to ensure that it runs correctly there.

HTTPS Access:

1. Generate an OAuth Token via Github
2. Store that OAuth in a safe place, if you lose it, simply go to Account -> Applications and revoke your old token, and creat
3. In the following instructions, replace all URLs that are '`git@github.com:UCSD-CSE-100/<repository name>.git`' with '`https://username:token@github.com/UCSD-CSE-100/<repository name>.git`'
   NOTE: If you lose your OAuth token, simply go to Account -> Applications and revoke your old token, and create a new one. However you WILL need to edit the remote url's again with the token

SSH Access:

1. Generate an SSH-Key and add it to your Github account, instructions to do so can be found here.

Now that you've gotten that out of the way, you can almost start actually working on your project!

1. Go to `https://github.com/your username`
2. Click on the UCSD-CSE-100 organization below your user picture. Image here
3. There you should see a list of your repositories. Click the one that starts with **`'P1_'`**, if you are working in a pair, select the one that starts with **`'P1_Pair'`**
4. On the right hand side you should see something that looks like this image, select HTTPS or SSH for the clone URL and copy it
5. Go to your terminal and create a directory for your project (eg. '`mkdir P1_Solo`')
6. `cd` to that directory
7. run the following commands:

```
git init
git remote add --track master origin
git@github.com:UCSD-CSE-100/<repository name>.git
```

`git pull`

8. You are now ready to start working on your project!

If you need to change your remote:

1. git remote set-url origin <url>
2. Use `git remote -v` to view the settings.

Pushing Changes to Github (SUPER-IMPORTANT!! THIS IS HOW YOU SUBMIT):

In order for you to get credit for your assignments, you will need to push the changes you make to Github. We highly recommend you commit often and push every few commits. In future assignments, we will require these intermediate pushes through checkpoint deadlines.

1. Run a '`git status`' to see what files have been changed
2. Use '`git add <files>`' to stage certain files for a commit, use '`git add .`' to add all files in the working directory to the commit
3. Use '`git commit -m "MESSAGE"` ' to commit the changes to your repository
   A good rule of thumb for commits is if you need to include the word 'and', you should rethink the commit and either commit fewer files, or make more commits as your work
4. Rinse and repeat steps 2 - 3 as needed to make multiple commits
5. To push your changes onto your repository on Github run '`git push origin master`' You should push regularly. For example, you might push every time you get a new function working. You must push your final working code to Github in order to get credit for the assignment. See the section below on turning in your assignment.

IF YOU ARE WORKING FROM MULTIPLE WORKSTATIONS, here are some extra important instruction to avoid merge conflicts:

1. Always push your changes to Github before leaving your current workstation. Although this is not required when using VCS in general (in fact, parallel editing is one of the strengths of a VCS), this is the workflow we advise in CSE100 to avoid merge conflicts and so that your work is always up to date no matter where you work on it.
2. Always run '`git pull origin master`' to grab any changes you pushed from a separate workstation before you start working in a new place. (And then push when you are done).

Reverting To A Previous Commit (you may never need to do this, but just in case...):

Sometimes you might have made a change that broke something and you don't remember how to fix it, if you've been committing often, like we suggest, that will be no problem!
To revert your master branch to previous commit do one of the following:

To permenantly rollback to a previous commit:

1. Run '`git log`' to see a history of your commits
2. Copy the commit id, the long SHA-1 id, of the commit you want to rollback to
3. Run '`git reset --hard <commit id>`'
4. Run '`git push origin -f`' to commit the thrown away commits

To revert commits to a previous commit without throwing those commits away:

1. Run '`git log`' to get the history your commits
2. Copy the commit ids of the commits before the commit you want to rollback to
3. Run '`git revert <commit ids>`'
4. Edit the commit message, and exit to create a commit that undoes the listed commits
5. Run '`git push origin master`' to save those changes to Github

Part 2: Implementing a BST in C++

Read Drozdek, Chapter 6 and make sure you understand the basic properties of binary search trees. In this assignment you will be implementing the basic "insert" and "find" operations, as well as implementing the iterator pattern, using concepts from the C++ Standard Template Library. In a later assignment, you will extend the classes you define in this assignment in order to implement a more sophisticated, high-performance search tree structure. So don't be confused by the keyword `virtual` in the provided code. This is there so that you can subclass your BST class in the next assignment. (If you are not sure what a virtual function is in C++, you will want to understand this before PA2, so you might want to look it up. Assuming you understand the concepts of polymorphism and dynamic dispatch from Java, a simple Google search on "virtual function c++" should get you to some understandable resources, or you can ask the course staff).

Assuming you have successfully completed part 1, you will have your repository that contains the provided code for this assignment. In it you will find the following files:

```
BST.hpp  BSTIterator.hpp  BSTNode.hpp  Makefile  test_BST.cpp
```

You will modify the three `*.hpp` files, providing full definitions of member functions marked with `// TODO`, and turn them in. In completing the implementation, you shouldn't remove anything from those files (except the `// TODO` comments!). If design considerations dictate it, you can add additional functions or variables, but they should be in a `private:` section, indicating that those additions are part of your particular implementation, and not part of the public or protected interface to these classes.

The `test_BST.cpp` file is a simple C++ application which partially tests some aspects of the BST class. You should definitely add code here, in order to test your program as thoroughly as you can (we will certainly test it as thoroughly as we can when grading it!).

The `Makefile` is provided for your convenience. Running `make` in the directory will compile

the `test_BST.cpp` application, creating an executable named `bst` which can be run from the command line. Modify this Makefile as needed, or create your own, to support your development process. Of course, it is entirely possible on this assignment to find that the provided Makefile works perfectly fine for your purposes.

Note that your first task is to *understand the code*. Do not attempt to start implementation until you have a firm grasp of what the provided code does and what you are supposed to do with it. Almost the entire specification of the functionality is given in the code and the comments, so you'll need to read everything carefully. Unlike in previous classes (particularly my offerings of CSE8A and 8B) we are not providing too much scaffolding to walk you through what this code does, outside of what we have already provided in class. This is a gateway course, and in the upper division you will be given at times very vague assignment specifications and expected to make sense of them. We've tried to provide some guidance, but not as much as you may have received in previous classes. We also encourage test-driven development here. Make sure you can understand the provided test file, and then ADD YOUR OWN TESTS BEFORE YOU START IMPLEMENTATION.

If you have any doubts at all about the provided code, come to tutoring hours in B220, one of the TAs in their office hours, or Prof. Alvarado's office hours. See the tutoring calendar for when these hours are offered. If you do not understand what you are doing, you will not learn anything from this assignment and the whole point of it will be lost. We are happy to help, but you must seek this help. And START EARLY. Read the code RIGHT NOW.

Turning in your assignment

When you have completed the assignment, you must submit it. You will do this by pushing your changes to Github. Follow these steps (which are more or less the same as the steps from the "pushing changes to Github" section above:

1. Run a '`git status`' to see what files have been changed
2. Use '`git add <files>`' to stage certain files for a commit, use '`git add .`' to add all files in the working directory to the commit (hopefully you will not be doing this at this late stage--all files should already be in your repository!)
3. Use '`git commit -m "FINAL SUBMISSION"`' to commit the changes to your repository and indicate that you are done and this is your final submission.
4. To push your changes onto your repository on Github run '`git push origin master`' Don't forget this push! You must do this to get credit.
5. As an added safety check, you should log on to Github on the web and verify that your push went through.

A couple of notes about this process:

- If you are submitting with a partner, just push to the joint repository on behalf of both

partners. That's your joint submission.

- Use the commit message "FINAL SUBMISSION" in ONLY ONE of your individual and partner repositories. You shouldn't be using both anyway, but if you happen to have pushed changes to both, just make sure only one contains the "FINAL SUBMISSION" commit message.
- If you decide you want to resubmit your assignment before the deadline, just do another commit with the message "FINAL SUBMISSION" (and don't forget to push to Github!!). We will grade the last commit before the deadline (or the slip day deadline if you use slip day(s)) that is tagged as the "FINAL SUBMISSION".
- If you submit the assignment within 24 hours after the deadline (i.e. if there is a commit tagged "FINAL SUBMISSION") you will be charged a slip day. If it is more than 24 but within 48 hours, you will be charged 2 slip days. If you are out of slip days, or after 48 hours, we'll roll back to the last commit tagged as final that was submitted before the deadline.

Grading

There are 25 possible points on the assignment. If your solution files do not compile and link error-free, you will receive 0 points for this assignment. We will compile and test your program using the g++ compiler and runtime environment on ieng6. To get full credit, style and comments count.

Comments

You do not have permission to add comments.