Advanced Programming Tutorial
Worksheet 1: Warming up the engines

Keywords: C++ basics, building code, version control with Git, collaboration with GitLab

Welcome to the Advanced Programming tutorial! Apart from this document, you will find more information on Moodle: short guides to the tools involved, as well as introductory presentations on Linux and C++.

The tasks marked with "**Idea**" are meant as optional ways to further explore a concept. It is fine if you don't get enough time to solve them or if you don't yet know the prerequisites, but we can also not present detailed solutions to these tasks.

Prepare at home

- ☐ Follow the "System requirements" page on Moodle to prepare your system.
- ☐ Write and run a simple "Hello world" C++ program, to ensure everything is working.
- ☐ Install Git and make sure you can login into `https://gitlab.lrz.de`.
- ☐ Read the exercises and refresh your memory on C++ syntax (see course CSE Primer). You don't need to solve the exercises, we will do that in class. We will have a short introduction to Git.
- ☐ On the day of the tutorial, bring your laptop (or join online).

Exercise 1: Simple Calculator

In this exercise, we want to implement a very simple calculator, freshening up some C++ memories you or your neighbor may already have, while learning how to collaborate with Git and GitLab. Talk to your neighbor and decide who is going to be "Alice" and who "Bob". It would be simpler to use the command line interface of Git for this exercise.

If you are working alone, try to at least use two branches and resolve a merge conflict. If you are joining remotely, find a collaborator in the lecture chatroom and arrange a communication channel, or use the provided platform (announced during the tutorial). Synchronize with the tutor at the points they announce.

Alice

1. Create a project on `https://gitlab.lrz.de` and clone the repository. Your current branch should be `main`.
2. Create a file `simpleCalculator.cpp`, add the main function, and publish your changes.
3. Invite Bob as a developer to your project.
4. In a branch `sum-subtract`, implement the following functions for `a` and `b` of type `double`:
    - `sum(a,b)`, which returns $a + b$
    - `subtract(a,b)`, which returns $a - b$

   All functions should return their results as doubles.
5. Synchronize with Bob and explain your implementations to each other. Merge both branches (`sum-subtract` and Bob's `multiply-divide`) to "main" and push (only Alice does this now).
6. In a branch named `mean`, implement an additional function, `mean(a,b)`, calculating the average value of two `double` arguments. Reuse the previously defined functions. Commit and push your changes.
7. Now that you know how to merge, help Bob merge the new branches `mean` (from you) and `menu` (from them) to the `main` branch.

Bob

1. Wait for Alice to create a project (do it together!).
2. Wait for Alice to add the file `simpleCalculator.cpp` to the repository.
3. Wait for Alice to invite you to the project and clone the repository.
4. In a branch named `multiply-divide`, implement the following functions for `a` and `b` of type `double`:
    - `multiply(a,b)`, which returns $a * b$.
    - `divide(a,b)`, which checks if b is non-zero and returns a/b.
      Hint: `if (condition) {...} [else if (condition) {...}] else {...}`.

   All functions should return their results as doubles.
5. Synchronize with Alice and explain your implementations to each other. Wait for Alice to merge & push to the main branch and pull.
6. In a branch named `menu`, implement a menu in which the user is prompted to specify an operation $(+, -, /, *, m)$ and to give 2 values. Print the result and finish the run. Use an `if(){...}` structure to call the respective function.
7. This time, take care of merging the two branches `mean` (from Alice) and `menu` into `main` (only Bob).

Follow-up ideas

- Instead of merging branches locally, use Merge Requests. Hint: Look at the suggestions on GitLab after pushing a new branch.
- Intead of synchronizing with your neighbor to discuss the changes, give a code review in the merge request.
- Do you have further ideas, or did you find a bug? Open an issue on your project.
- You don't want to use the teminal all the time? Learn how to use version control in VSCode or in your favorite editor.

Idea: A more sophisticated Calculator

Extend your simple calculator from the previous exercise with methods to support:

1. the square operator (x^2).
2. the trigonometric operations ($sin(\pi x), cos(\pi x), tan(\pi x)$).

Make sure to also add these in your menu and check them for correctness.

Discussion

- Which library files do you need to include in `simpleCalculator.cpp`?
- How did you define / get the value of π?
- What would you have to do if you wanted to implement the main function in another file?

Idea: Base Conversion

Extend your Calculator with an operation `base(a, b)` to convert an integer (or other type) `a` from the decimal system to another base, `b` and print the result (no return value).

Note: Integer division returns *truncated* integers, which is useful here. You may also use the *modulo* operator `%`.

Epilogue

This code was clearly very short and fast. However, it was only meant as an example to remind you some basics of C++ syntax and let you try out Git with some help. In our everyday development, we almost always use version control, from very small to very large and complicated projects. The workflow is mostly the same, but we also extensively use issues and merge requests.

Chair of Scientific Computing in Computer Science – Technical University of Munich
Last updated on October 25, 2023