Projecto Integrador de 1º Ciclo (PIC1)

Goal

The main goal of this project is for students to have a first contact with the real world and have a software engineering experience, including making small changes to a large codebase, estimate implementation effort, having their code reviewed by professional engineers, and have productive technical discussions.

Additional goals include:

- Interacting with an open-source community as open-source software is now commonplace and students should know how it works, where to go when they find a bug or need support, and how to get a patch accepted.
- Build a personal portfolio. Most companies ask for the GitHub/GitLab/etc accounts. But they are not very useful if empty. An account with good patches and good interactions with other developers can be your best CV!

The Project

The project consists in the following tasks:

1. Pick an open-source project of any area you like. This project must be of a medium to large size (minimum 100k lines of code), be active (minimum 50 commits in the past month), and popular (minimum 200 GitHub stars or equivalent).
2. Compile it and run the unit tests successfully. Investigate if not all tests pass.
3. Fix at least one existing bug report and submit the patch for review. Work to get the patch committed.
4. Propose a new feature, implement, and test it. Work to get the patch committed.
5. Write a short (2 pages) report with your experience and give a presentation to the class.

Projects must be carried out in groups of 2 students.

Patches (PRs) submitted for review must use your name and your @tecnico.ulisboa.pt email address and **must be reviewed by a professor before submission**.

Note: If the project you are interested in requires special hardware (e.g., a computer with tons of RAM, an Arduino board, etc) to compile and/or run tests, please ask your lab professor well in advance if it's possible to source it. If not, you will have to pick a different project.

Tips and Suggestions

- For project ideas, see the list of organizations that participated in Google Summer of Code (https://summerofcode.withgoogle.com/programs/2023/organizations). These organizations usually have lists of projects that are doable by newcomers. Additionally, they have tradition of accepting external patches. You can also use these websites to discover projects: https://www.gsocorganizations.dev and https://ovio.org/projects.
- Open-source contributors are not paid to help you! They will help you on a volunteer basis. Be kind, respectful, and always thank them for their time.
- Each time you interact with an open-source community you are being judged and evaluated. Asking a question that is answered in the first line of the developer's manual makes you lose one point. Send a patch that doesn't respect the coding standard, and you lose another point. Asking a good question or committing a patch earns you a point. Lose too many points and people will ignore you. So, read that compiler warning message twice before asking what it means!
- Most open-source developers work at top tech companies. You are being interviewed!
- Don't be afraid of asking questions. But do your homework first! Read the manuals, search for the answer online first. When asking a question, explain what you've found so far and what's missing in your understanding.
- Before submitting a patch for review, make sure it passes all unit tests, that it adheres to the coding standards, and that it has no spelling mistakes in the comments.
- If your feature patch is large, you should split it into several smaller patches, if possible, to make the review process easier and faster.
- When submitting a patch for review, cc the relevant reviewers. Check 'git log' on the files you've modified to find the developers that have worked on those files recently.
- Submit patches as early as possible. Review won't happen overnight!
- If your question/pull request goes unanswered, ping the developers once a week (but not more often). Be mindful of the holidays in other regions.
- Address reviewers' concerns in a timely fashion. Don't leave them waiting for more than a couple of days.
- You are representing our university. Make us proud!

Useful references:

- https://mitchellh.com/writing/contributing-to-complex-projects
- https://www.firsttimersonly.com
- https://stackoverflow.blog/2020/08/03/getting-started-with-contributing-to-open-source/

Milestones

Week	Task
01. 15/February	Introduction to PIC1
02. 23/February	Select a project Read the developer's manual
03. 1/March	Compile code & run unit tests Create a GitHub account **(1 per student)** Sign the CLA (if needed) **(1 per student)**
04. 08/March	Select a bug report to work on & reproduce it **(1 per student)** Read the code style guide
05. 15/March	Work on the bug fix **(1 per student)**
06. 22/March	
07. 29/March	
08. 05/April	Submit bug fix for review **(1 per student)**
09. 12/April	Easter Break
10. 19/April	Submit feature project proposal
11. 26/April	Unit tests written
12. 03/May	
13. 10/May	At least one unit test passes
14. 17/May	The feature is implemented and all initial unit tests pass. Do further tests.
15. 24/May	
16. 27/May	Submit feature patch Submit final report Address issues with the patch (buildbot failures, review comments, etc)
17. 03-14/June	5-minute presentation

Evaluation

Task	Points
02: Select a project	2
04: Select a bug report & reproduce it	2 (individual)
06: Fix the bug	1 (individual
08: Submit the bug fix for review	2 (individual)
Bug fix accepted	1 (individual)
10: Feature project proposal	2
14: Feature development & testing	3
15: Submit the feature patch for review	2
Feature patch accepted	1
16: Final report	3
17: Final presentation	2
Total	21

Penalizations

- Opening a PR without being reviewed by a professor: -5 points
- Opening a PR with the wrong name or a non-IST email: - 5 points
- Use of disrespectful language with reviewers or unprofessional behavior: - 5 points

Deliverable #1: Chose a project

Tasks:

1) Create a GitHub account or associate your IST email address with your existing account.
2) Go to https://pic1dei.tecnico.ulisboa.pt/index.php?page=profile and enter "github:your_username". Ensure that the details on the blue box look ok; change your github profile if not.
3) Go to https://pic1dei.tecnico.ulisboa.pt/index.php?page=listprojects, click on your group and fill in the details. You can submit this form as many times as needed until the deadline. The "repository" field must start with "https://github.com/". Ensure the details on the blue box are ok and that nothing is red.
4) Sign the CLA if necessary (individually).

Note that it's worth spending some time to pick the right project. It should be something that interests you. Should also be an active project that welcomes patches from beginners, and has sufficient easy bug reports open.

What's a CLA (Contributor License Agreement)?

Some projects require contributors to sign a document before they open a PR. Typical CLA clauses include:

- Copyright assignment: you transfer the ownership of the code you wrote to some entity.
- Origin guarantee: you guarantee (legally) that the code you contribute is novel and not copied from somewhere else.
- Patent assignment: if the code is covered by a patent, you give a royalty-free license to all users.

If you choose a project that uses CLAs, you must sign it individually.

Deliverable #2: Select and reproduce a bug

Some projects have bugs marked as for beginners/newbies, or as “good first issues”. Check the labels/filters in the issue tracker.

The bug report should not be too old, otherwise you risk it being already fixed. Also, if it is 5 years old and it wasn’t fixed already, then maybe it’s not that interesting. On the other hand, you may not want to pick a report submitted in the past couple of days, as there is a higher chance of being fixed by someone else in the meantime.

We emphasize that race conditions are possible. It’s your job to minimize the risk. One strategy is to claim the bug by leaving a message saying you’re working on a fix. You can only do this after you reproduce the bug and commit to fixing it. You will be penalized if you claim a bug without reproducing it first or without intention of fixing it. Again, be professional and mindful of other people’s time.

You need to show your lab’s professor that you can reproduce the bug. You can do this in the class before the deadline, or by sending an email to your professor with a small video demonstrating the issue, accompanied by a text description that allows one to understand the problem.

Deliverable #3: Fix a bug

Fixing a bug requires first finding the root cause. Use the scientific method: create a hypothesis, test it, and use the result to create the next experiment. Repeat until you understand the problem. Use a debugger.

Make sure the code you write adheres to the code style of the project!

Once you have a fix and a test case, ensure that the unit tests still pass. To make sure your test is being run, change it in a way that would make it fail and run the test suite.

1) Create a patch

Go to GitHub and fork the project. Clone it to your computer using git. If you are using WSL, clone the project in the WSL drive, not on some windows directory.

Then do "git diff". Is the diff what you expect? Is it beautiful? Are there red things on the screen (indicating extra whitespace that should be removed)? Beautify the patch until it is perfect.

Then do "git add files_you_changed". Check with "git status" that you've included all the relevant files.

Now do "git commit". This will open an editor where you can type the commit message. Check if the project has some guideline. If not, use the following template:

```
Fix #bug_number: short description of the bug (one line)
Longer description of the problem and the fix (several lines, as needed)
```

Confirm again if everything is good with "git show". This will show you both the patch and the commit message. Ensure your name and email are correct. If something is wrong, just remove the commit with "git reset HEAD~1" and start again (this command doesn't delete your changes; only deletes the commit).

Then do "git push". This will upload your patch to your fork on GitHub.

Note: if your project uses DCO, you must do the commit with: "git commit -s". The "-s" flag adds your signature to the commit message.

2) Submit the patch for revision by a professor

Now you need to get approval from a professor before opening a pull request. Go to https://pic1dei.tecnico.ulisboa.pt/index.php?page=patches and create a new patch. For the patch url, you should have something like (can be obtained from the GitHub website on your repository's page): "https://github.com/your_username/repo_name/tree/branch_name".

3) Open a pull request (PR)

Once the patch is approved, you can open a pull request. Go to your fork page on GitHub and click on the create pull request button.

In the description of the PR, you must mention the bug number like #number and include an explanation of the cause and of the fix. The better this is written, the faster it is reviewed and accepted.

If a reviewer suggests some fix, do it quickly. Waiting several weeks is a bad practice.

The reviewer asked me to rebase. What's that?

When you fork a git repository, you create a snapshot of the code at that point in time. In the meantime, the project keeps evolving and maybe your patch doesn't apply cleanly anymore. In these cases, you've got to bring your fork up to date. The best way is through a rebase, which updates your branch and moves your commit to the top.

Commands:

```
git remote add upstream https://github.com/someorg/repo.git # do this once
git fetch upstream
git rebase upstream/master  # some projects use main instead of master
git push -f
```

You may get conflicts if both you and someone else in the project changed the same lines. The code will get annotated and it's your job to figure out how to merge the changes. Read the output of the "git rebase" command and follow the instructions.

The reviewer asked me to squash the commits. What's that?

This is a bad sign. You were supposed to have a single commit. If you have more than one, the reviewer may ask you to merge all the commits into a single one.

The easiest way is to remove the commits and create a new one:

```
git reset HEAD~n  # n is the number of commits to merge
git add changed_files
git commit
git log    # is everything ok?
git show   # is everything ok?
git push -f
```

How can I create a new branch?

This will create a new branch starting at the current commit:

```
git checkout -b branch_name          # creates the branch locally
git push -u branch_name branch_name  # creates the branch on the server
```

Deliverable #4: Feature proposal

The goal of PIC1 is not to develop a lot of code, as students already do that in other courses, but rather to train the software engineering aspects. So, we expect small features, in the order of a few hundred lines of code.

If you have the ambition of getting your feature accepted, you should start by looking at the feature backlog of the project if there's one. If not, look at the GSoC ideas for inspiration if the project participates in this program. If not, you may want to discuss your ideas with the project developers, to realize if your proposed feature is interesting to others, and to better estimate the implementation cost.

Don't simply send an email asking for ideas; do your homework first! Why should someone else waste time with you if you didn't bother thinking about the topic for, say, 1 hour? Or checking the feature backlog?

Deliverable #5: Upstreaming a feature

The work is similar to fixing a bug, with 2 exceptions: 1) you have to write more tests as there are no tests that cover the new functionality, and 2) it's done by more than one person.

In terms of workflow, the only difference is that you are going to use a single fork for the work of the two of you. You may choose to have multiple commits, or just one. If doing just one, and if the work is from both students, one student does the commit, and the other student's details go in the commit message as follows:

```
Implement xpto for foobar (one line explanation of the feature)
longer description with an example

Co-authored-by: The other student's name <email@tecnico.ulisboa.pt>
```

Feature Proposal (Example; max 2 pages)

LLVM introduced a matrix type recently (https://clang.llvm.org/docs/MatrixTypes.html and https://llvm.org/docs/LangRef.html#matrix-intrinsics and https://llvm.org/devmtg/2020-09/slides/Hahn-Matrix_Support_in_LLVM_and_Clang.pdf). Therefore, there are several optimization opportunities available.
We will implement a new middle-end intra-procedural optimizer at the LLVM IR level. It will reassociate sequences of matrix multiplications to pick the best order to reduce the overall cost and memory consumption.

The following example function multiples (a x b) x c:

```
; a = 8 x 3
; b = 3 x 2
; c = 2 x 2
define void @square(<16 x float> %a, <16 x float> %b, <16 x float> %c) {
  %p = call <16 x float> @llvm.matrix.multiply.v16f32.v16f32.v16f32(<16 x
float> %a, <16 x float> %b, i32 8, i32 3, i32 2)
  %q = call <16 x float> @llvm.matrix.multiply.v16f32.v16f32.v16f32(<16 x
float> %p, <16 x float> %c, i32 8, i32 2, i32 2)
  ret void
}

declare <16 x float> @llvm.matrix.multiply.v16f32.v16f32.v16f32(<16 x float>,
<16 x float>, i32, i32, i32)
```

A different operation order is: a x (b x c). Our algorithm will pick the optimal order using the standard dynamic programming algorithm.

Since LLVM has very few unit tests for the matrix type, we will write new unit tests with different chains of multiplications to test our algorithm.

To benchmark our optimization, we will use the benchmarks from the FooBar test suite (https://url) which contain 200 tests with matrix multiplications representative of machine learning inference workloads.
The benchmarks will be run on our laptops.

We've discussed this idea with the LLVM developers, and they confirm it's a useful optimization and that no one else is working on it. They also confirmed it's a feasible project for 2 months for newcomers. Link to our discussion: https://link-to-discourse-discussion

Predicted LoC changes: 200 lines
Predicted unit tests: 500 lines
Predicted implementation effort: 12 hours
Predicted test & debugging effort: 8 hours
Predicted benchmarking effort: 4 hours

Final Report (Template)

This report should have about 2 pages.
Suggested topics to address:

- Description of the proposed feature and what was implemented. If different, why didn't you stick to the original plan?
- Predicted vs actual implementation/testing/debugging effort (work hours) and number of lines.
- Interaction with the open-source community
- Lessons learned: What worked and what didn't? Did you learn new coding patterns? New algorithms? New tools?
- What was the biggest challenge you encountered?
- Is the project documentation good enough for newcomers? What's missing?
- Did you like the experience? Will you continue working on that OSS project? Why? Why not?
- What would you improve in the project you chose?