Team Project
2024 Spring, SWPP

Updates

4/25

- Add policy for minimum PRs merged per sprint
- Revise policy for maximum PRs merged per sprint
- Revise policy for adding existing LLVM passes
- Revise policy for compiling the project when running the test

4/26

- Revise policy for adding existing LLVM passes

4/28

- Fix incorrect information (documentation deadline, testing command)

5/2

- Add policy for revert/reverted PRs
- Add policy for update PRs
- Revise policy for writing PRs

5/10

- Relax page limits for Req/Spec documentation
- Revise policy for making PRs
- Add policy for unit tests for analysis passes

Goal

In this project, you'll work as a team with other students to write a simple compiler that reads an LLVM IR program and emits an optimized assembly for an imaginary machine.

Each team should

- Use LLVM **18.1.0** built with the script in class repo for your project.
- Use GitHub to collaborate and code-review teammates' implementation.

At the end of the semester, we'll have a competition to evaluate the performance of each team's compiler.

Schedule (subject to change)

Date	**Details**
4/19 - 4/26	Sprint 0 - Read codes, setup development envs and CI, etc.
4/25	Introduce your team in class
4/26	Submit two documents: - Requirement and specification - Planning
4/27 - 5/10	Sprint 1 - ~10 days (From Saturday): development - 3+ days (until Thursday): code review+revision - Last day: progress report & req/spec plans
5/11 - 5/24	Sprint 2
5/25 - 6/7	Sprint 3
6/8 - 6/10	Wrap-up
6/11	Competition

(1) Sprint 0

Introduce your team in the class
Each team will have about 5 minutes to introduce themselves in the class.
- Introduce your team & teammates.
- Explain what you want to learn during the team project about software engineering.
- Suggest 5+ optimizations (either simple or complex) with example codes.
- Prepare short presentation slides.
- The talk is lightweight; your talk does not need to get very technical.

Pre-project Documentation
You should submit two documents (A. requirement and specification, B. planning) before the first sprint. You should submit the document to TAs via eTL. Each team is assigned an eTL group, so anyone can submit the document on behalf of their team. Each document should not be more than 7 pages. You may use Korean or English for these documents. The document should be in ***pdf*** format.

A. Planning
   - File name should be “sprint0-teamN-planning.pdf” (N is your team number)

    - Briefly explain optimizations that your team would like to implement during the 3 sprints.
    - It should include each member's plan for 3 sprints.
    - Each person can implement more than one optimization in a single sprint.
    - Optimizations can be implemented with your teammates.
    - Optimizations can be implemented through multiple sprints.

B. Requirement and specification
    - File name should be "sprint**1**-teamN-reqspec.pdf" (note the sprint number)
    - Describe the optimizations that you are going to implement in sprint 1. For each optimization, describe the following information.
        i. Description
        ii. An algorithm in a high-level pseudo-code
        iii. At least 3 pairs of IR programs, before and after the suggested optimization

Preparing the Project Skeleton

Once the team project repository template has been released, each team should create their own team repository from the template provided. During sprint 0, students can directly push commits that contain project skeleton code, CMakeLists configuration, or GitHub Actions configuration into the team repository. These commits are exempt from PR criteria (described below) and grading.

Especially, automated testing on the GitHub Actions should be fully functional by the beginning of sprint 1. Details about testing are described later in this document.

<u>(2) Sprint 1/2/3</u>

Each sprint consists of

- **Development phase** (Up to 10 days, starting from Saturday)
    - Implement the planned features
    - Send pull requests to the main repository
    - Assign reviewers.
- **Code-review phase** (At least 3 days, until Thursday)
    - Assigned reviewers review the pull requests
    - Reviewee updates the pull request according to the comments.
    - If the update is complete, the pull request is merged into the main repository
- **Documentation phase** (On Friday)
    - Write a progress report for this sprint, and requirements and specifications for the next sprint. Submit them via eTL.

You may use Korean or English in each process.

Pull Requests

- A pull request contains an implementation of a single feature.
- The title of a pull request should start with [Sprint N]. (N is the current sprint number)
- It should contain unit tests to check the added functionality.
- Its line diff should be at most around 300 lines except comments, spaces, tests, and non-C++ code files such as .gitignore or CMakeLists.txt.
    - We'll count the LOC after running the formatter on your code. Formatting rules can be found in included `.clang-format`
- It should satisfy the good pull request conditions that are described in 4/18th's practice session
- **[5/10 Update]** ~~When merging the pull requests, they should be merged to the ***main*** branch.~~ You may merge to any branch on the repository. However, only the PRs that are merged into the **main** branch will be graded.
    - If the opt. pass PR is to be merged to the main branch, it must be reviewed.
    - Otherwise, reviews are optional but recommended.
- Each pull request should be merged using ‘squash and merge’ by the pull request writer (not reviewers!)

Policy for Writing Pull Requests & Reviews

- Each student can merge up to 3 pull requests per sprint.
    - **[4/25 Update]** If your team has less than four teammates for any reason, you may take the ‘merge chance’ of the missing teammate(s) as well.
- **[4/25 Update]** Each student should merge at least one pull request per sprint.
    - **[5/2 Update]** Only ‘graded’ PRs counts; otherwise, we cannot grade your performance on a sprint and have to give you zero points.
    - **[5/2 Update]** We will count co-authored PR as one PR per each person.
- For each pull request, 2 or more reviewers should be assigned.
- **[5/2 Update]** Every PR must pass the CI at the time of merge.
    - Merging the PR that fails the CI will be penalized, with no exception.
- If necessary, a team can write pull requests that make ***non-functional changes*** such as directory structure changes / source file splitting / etc.
    - Please add “[Sprint N, NFC]” at the beginning of the title.
    - This pull request should be reviewed but not as much as functional ones.
    - This pull request is exempt from the 3 pull request restriction.
    - We'll be generous about what counts as ‘non-functional’:
      As long as your PR does not change the function of the compiler code, you may call it NFC.
- **[5/2 Update]** If necessary, a team can make reverting pull requests that ***revert previous PR***
    - Please add “[Sprint N, Revert]” at the beginning of the title.
    - Please add “[Sprint N, Reverted]” at the title of the reverted PR.
    - This pull request should be reviewed but not as much as functional ones.

    - ‘Revert’ and ‘Reverted’ pull requests are exempt from the 3 pull request restriction.
    - While ‘Revert’ PR will not be graded, we will still use ‘Reverted’ PR for grading.

- **[5/2 Update]** ***TAs’ codebase update*** should be applied in the form of PR as well.
    - Please add “[Sprint N, Update]” at the beginning of the title.
    - This pull request does not have to be reviewed, but it should still pass the CI!
    - You may apply Update PRs at any moment of the project period, but all updates must be applied to the team repository by the end of wrap-up period, as they may contain critical bug fixes.

Existing Optimizations

LLVM has many optimizations, and simply calling these functions from your project can sometimes bring large performance improvements.

You may use existing passes but in a restricted way.

For each sprint, each person in a team can add at most one pass that already exists in LLVM. Each existing optimization should be added via separate PR, and should come with unit tests as well. **[4/26 Update]** This pull request is exempt from the 3 pull request restriction.

[4/25 Update] Or, you may include an arbitrary number of existing passes as part of your implementation, if it is closely related to the optimization you’re working on. ‘Relation’ includes, but is not limited to, complex analysis or canonicalization necessary for one’s optimization pass.

Writing & Running Unit Tests for Optimization Pass

- Each optimization pass PR should contain at least 3 unit tests.
- Unit tests are responsible for checking the functionality of ‘single’ optimization pass.
    - Testing the output of `opt --<new-opt-pass>` is therefore recommended
- Each unit test should be an LLVM IR program with FileCheck comments.
- Running `ctest --test-dir build` should run all added unit tests.
- Each commit and pull request to the team repository should trigger GitHub Actions and run these tests.
- Adding Alive2 tests is not mandatory but is highly recommended.

[5/10 Update] Writing & Running Unit Tests for Analysis Pass

- Each analysis pass PR should contain at least 3 unit tests.
- Unlike optimization passes, analysis results are not visible in IR programs, and requires direct access into LLVM’s internal data structure to check its functionality.
- Therefore, each unit test should be a C++ code that loads an LLVM IR program, runs the program via analysis pass, and asserts that your pass functions as expected.
- Other than the implementation method, the rule for optimization pass unit tests apply the same.

Main Repository

- The main repository should contain a **main** branch.
- **[4/25 Update]** After each commit, the project should work correctly. TA will check
  ```
  ./compile.sh # we will modify directory variables as necessary
  ctest --test-dir build
  ```
  from the root directory of the branch.

Using GitHub Issue

- Please use GitHub issue to show that you are actively communicating with people.
- If there was an offline discussion and it wasn't written as comments at Pull Request, please record it at GitHub Issue.

End-of-the-Sprint Documentation

At the end of each sprint, each team should submit the progress report and the updated requirements/specifications for the next sprint. The document should be in ***pdf*** format.

A. Progress report
   - File name should be "sprintN-teamM-progress.pdf" (N is the **current** sprint)
   - Describe each member's progress at the end of the sprint.
   - If you did not finish what you have planned, explain why.
   - Include the results of applying your passes to all benchmarks (distributed at the beginning of the project) as well as your own tests to show the effectiveness of your optimizations. Examples may include performance enhancements, applicability (how many cases do your passes have effects), etc.
   - If there is an update in planning, please submit the updated part as well.
   - **[5/10 Update]** This document should not exceed 5 pages.

B. Requirement and specification
   - File name should be "sprintN-teamM-reqspec.pdf" (N is the **upcoming** sprint)
   - Describe optimizations that you are going to implement in the next sprint. For each optimization, description / pseudo-code / IR programs should be included.
   - **[5/10 Update]** This document should not exceed 7 pages.

You should submit two documents on eTL. You may use Korean or English.

	~ Sprint 0	~ Sprint 1	~ Sprint 2	~ Sprint 3
	4/26 11:59 pm	5/10 11:59 pm	5/24 11:59 pm	6/7 11:59 pm
Planning	O			
Requirement and specification	O (1)	O (2)	O (3)	
Progress report		O (1)	O (2)	O (3)

(3) Wrap-up & Competition

After 3 sprints, you will be given a few days to wrap up your implementation.
In this period, you may commit codes as much as you want (no code review, no line diff constraint, …)

On the last day, we'll run a competition, estimating the correctness & efficiency of the compiler.