Team Project
2023 Spring, SWPP

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 **15.0.7** 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
18 Apr	Introduce your team in class
17 Apr - 23 Apr	Sprint 0 Read codes, setup development envs and CI, etc.
23 Apr, - 11:59 pm	Submit two documents: Requirement and specification Planning
24 Apr - 7 May	Sprint 1 8 days: development 6 days: code review+revision, write progress report
8 May - 21 May	Sprint 2
22 May - 4 Jun	Sprint 3
5 Jun - 7 Jun	Wrap up
8 Jun	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 6+ 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.

(2) Sprint 1/2/3

Each sprint consists of

- **Development phase** (8 days, Monday - next Monday)
    - Students implement the planned features
    - Send pull requests to the main repository of the team
    - Assign reviewers.
- **Code-review & documentation phase** (6 days, Tuesday - Sunday)
    - 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
    - The team writes a progress report for this sprint, and requirements and specifications for the next sprint. They are submitted 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 200 lines except comments, spaces, tests, and non-C++ code files such as .gitignore or CMakeLists.txt.
    - **Once per semester**, each student is allowed to write a longer pull request (about 300 LOC).
- It should satisfy the good pull request conditions that are described in Apr. 13th's practice session
- All pull requests should be merged into the ***main*** branch.
- Each pull request should be merged using 'squash and merge' by the pull request writer (not reviewers!)

Policy for Writing Pull Requests & Reviews

- One student can make at most 2 pull requests per sprint.
- For each pull request, 2 or more reviewers should be assigned.
- If necessary, for each iteration a team can write one pull request that does ***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 2 pull request restriction.
    - This pull request can be merged during the development phase.

Existing Optimization

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

You may use existing optimizations but in a restricted way.

For each sprint, at most *one* person in a team can add optimizations that already exist in LLVM. The person shouldn't have worked on adding existing optimization during all previous sprints. Each existing optimization should be added via separate PR.

Writing & Running Unit Tests

- 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.

Main Repository

- The main repository should contain a **main** branch.
- After each commit, the project should work correctly. TA will check

  ```
  cmake -Bbuild
  cmake --build build --target swpp-compiler
  ctest --test-dir
  ```

  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 a 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 results of all the existing tests as well as your own tests to show the effectiveness of your optimizations.
    - If there is an update in planning, please submit the updated part as well.

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.
    - Please submit it before the next sprint.

You should submit two documents on eTL. Each document should not be more than 5 pages. You may use Korean or English.

	~ Sprint 0	~ Sprint 1	~ Sprint 2	~ Sprint 3
	23 Apr 11:59 pm	7 May 11:59 pm	21 May 11:59 pm	4 June 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.