SWPP Project, 2020 Spring

1. 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. The machine has a tape-like memory, and its specification is described at spec.pdf.

Each team should

- Use LLVM 10.0 to implement their compiler.
- Use GIT to collaborate and code-review teammates' implementation.

At the end of semester, we'll have a competition to evaluate the performance of compilers.

2. Schedule

Date	**Details**
28 Apr	Idea Presentation Session (each team will be given 15 minutes)
2 May, ~11:59 pm	Documentation
3 ~ 16 May	Sprint 1
17 ~ 30 May	Sprint 2
31 May ~ 13 June	Sprint 3
14 ~ 17 June	Wrap up
18 June	Competition

(1) Idea Presentation Session

Each team will have 5-minutes presentation for introducing their ideas for this project.

(2) Documentation

You must submit two documents (A. requirement and specification, B. planning) before the first sprint and two other documents (C. updated requirement and specification, D. progress report) at the end of each sprint. You should submit the document to TAs via e-mail (swpp@sf.snu.ac.kr) titled "[SWPP] Team N documents".

A. Requirement and specification
    - Describe optimization passes that you are going to implement.
    - For each optimization pass, following information should be included.
        i. Name of optimization
        ii. Description
        iii. Algorithm in pseudo-code
        iv. At least 3 IR programs and the optimized target programs through the suggested optimization
    - Describe the algorithm and IR programs that are going to be implemented in the next sprint only.
        i. If you are implementing optimization A in sprint 3, algorithm and IR programs for A should be included at the end of sprint 2.
    - Please submit it before sprint 1.
    - Update the document and submit it at the end of each sprint.
B. Planning
    - Describe your plan for the project.
    - Must include each member's plan for 3 sprints.
    - Please submit it before sprint 1.
C. Progress report
    - 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 written for checking effectiveness of your optimizations.
        i. Test results should include change in cost before and after optimization
        ii. Write down at least 3 IR programs per optimization pass
    - Submit the document at the end of each sprint.

	2 May	16 May	30 May	13 June
Requirement and specification	O	O (updated doc)	O (updated doc)	O (updated doc)
Planning	O			
Progress report		O	O	O

(3) Sprint 1/2/3

Each sprint consists of a development phase (8 days, Sunday - next Sunday) and code review & revision phase(6 days, Monday - Saturday).

- Development phase: Each student implements a feature & submit a pull request to the main repository of the team.

- Code-review & revision phase: Assigned reviewers review the pull requests, and reviewee should revise the pull request.

For each pull request, 2 reviewers should be assigned. Reviewers should check whether

- The pull request is within 200 lines (except tests).
- It contains unit tests for the added functionality.
- It is readable and works correctly.

For each sprint, one person in a team can be dedicated to implementing the optimizations that already exist in LLVM. One person can work on implementing the existing optimizations only once (not more than one sprint).

(4) Wrap-up & Competition

After the sprints, you will have 4 days to wrap up your implementation.
In the last day, we'll run competition which will estimate:

- Is the compiler correctly compiling input programs?
- Is the compiler generating an efficient assembly, in terms of runtime cost & memory consumption?

3. Evaluation

We will assess two things to evaluate each team:

(1) Quality of Product

- Is the implemented compiler correct?
- Is the performance of the compiler good?
- Is the compiler after each sprint in the consistent state (compiles & runs tests without an error)?

(2) Document

- Is requirements and specification clear?
- Is design and plan reasonable?
- Is the goal of the previous sprint achieved?

Also, we'll assess two things to evaluate each student:

(3) Contribution

- Is a student contributing enough to the project?

(4) Code Review & Revision

- Is a reviewee writing good pull requests (includes tests, less than 200 lines, etc)?
- Is a reviewer reviewing assigned pull requests well?