EECS665

Compiler Construction

Drew Davidson
Andrew Riachi
Koyel Pramanick

Lecture: LEEP2 2300
M/W/F 3:00 PM - 3:50 PM

Project 6

Due on November 22nd 11:59 PM (Accepted with no penalty).

Accepted late by November 23rd 11:59 PM for 1 penalty

Accepted late by November 24th 11:59 PM for 2 penalties

Not accepted on or after November 25th 12:00 AM

Each penalty incurs a 15% stacking reduction on the project grade or uses 1 "penalty token". Penalty tokens are applied by course personnel to your maximum benefit.

Updates

Classes have been dropped for this project, you don't need to worry about MemberIndexNodes or any class-related functionality.

Resources

- The starter code is available here.
- The oracle is available to consult here.

Objective

For this assignment you will covert the Abstract-Syntax Tree of the A program to a flat 3AC representation. Your main task will be to write conversion functions for the nodes of the AST.

Your compiler must accept an option

```
-a <3acFile>
```

that outputs the 3AC code for a given input program to the provided file.

Assessment

Your grade will be based on your compiler's ability to build a 3AC representation of an input program. In order to demonstrate the correctness of your transformation, we will apply a number of test programs to your submitted compiler and compare the output to what we expect. We will invoke your compiler in a way similar to the below sequence of commands:

```
tar -xf p6.tgz
cd p6_files
make
ac <case1.a> -a <case1.out>
ac <case2.a> -a <case2.out>
...
ac <caseN.a> -a <caseN.out>
```

Where `<case1a>` ... `<caseNa>` are input files and `<case1.out>` ... `<caseN.out>` are the corresponding output files.

We will expect your compiler to output 3AC code that preserves the semantics of the original program in a straightforward way. However, since there are many methods to represent a program, we will not require an exact match to our reference implementation. To ensure that your output is easy to understand, you should adhere to the specifications listed in the Output Specifications section, below.

Output Specifications

The 3AC format that we will use for this project will track closely with that presented in class (Instructions/quads must adhere to the 3AC rules, operands must be atomic values, and instructions are organized into procedures).

Instruction Formats

The instruction format is provided to you as part of the 3ac_quads files (as constructed by the `toString()` function of the abstract Quad class, with assistance from the various Quad subclasses.

In general, you should follow the instruction format and overall 3AC listing that is already present in the `3ac_quads.cpp` output functions (toString and repr). the oracle. If you feel that the oracle's output is correct but unreasonable, please consult course staff for approval to output code in a different way. If you feel that the oracle's output is incorrect, please report the issue on Piazza. If course staff confirms your report as the first to identify a particular error in the oracle, you will be given extra credit (if you report the bug completely anonymously, we will not be able to award extra credit - but you can report is anonymously to students).

Your Assignment

To complete this assignment, you should fill out the `to3AC` and `flatten` functions that are provided in skeleton form in `3ac_output.cpp`.

If you would like to change the format of a 3AC instruction or you would like to add a 3AC instruction to the set, you may do so with approval of course staff. Please post your request on Piazza.

You are free to modify any and all other files provided to you, provided you follow the basic

rules of the course (your code must be your own, must compile and run on the cycle servers, and should not unduly affect the system).

Hints and Advice

Believe in yourself! Also, make sure to watch the Project 6 video to get an overview of the project.

Instructor KU EECS