Assignment: Code Generation
Due Friday, November 01, 2019 at 11:59 PM

1 Introduction

In this assignment, you will implement a smaller version of a code generator for Cool.

The code generator makes use of the AST constructed in the previous assignments. Your code generator should produce LLVM-IR that implements any program that have the given constraints. There is no error recovery in code generationall erroneous Cool programs have been detected by the front-end phases of the compiler.

As with the semantic analysis assignment, this assignment has considerable room for design decisions. Your program is correct if the code it generates works correctly; how you achieve that goal is up to you. We will suggest certain conventions that we believe will make your life easier, but you do not have to take our advice. As always, explain and justify your design decisions in the README file. This assignment requires you to write a lot of code, maybe more than what you wrote for semantic analysis, though they share much of the same infrastructure. **So start early!**

Critical to getting a correct code generator is a thorough understanding of both the expected behavior of Cool constructs and the interface between the runtime system and the generated code. The expected behavior of Cool programs is defined by the operational semantics for Cool given in Section 13 of the Cool Reference Manual. Recall that this is only a specification of the meaning of the language constructsnot how to implement them. *Please read it thoroughly.*

For this assignment, you can assume that the input programs do not have any SELF TYPE type, case expressions, let expressions or dynamic dispatch (static dispatch still exists). These assumptions lead a to substantial reduction of the code that you have to write, making the assignment doable in around 2 weeks.

CS3423 registrants can work in a group for this assignment, where a group consists of two students; in special circumstances, three per team may be allowed. For CS6240 registrants, it is a choice whether they choose to do it individually or in a team of two. Work with the same groups as the previous assignments

2 Files and Directories

To get started, simply run the following in the root folder of your assignment directory. tar xjvf codegen.tar.gz

- src/java/cool/Codegen.java
  This file will contain almost all your code for the code generator. The entry point for your code generator is the **Codegen(AST.program, PrintWriter)** constructor. You will write code to traverse the AST appropriately and emit the LLVM-IR code.

- src/java/cool/CodegenTest.java
  This file contains the code for testing your code generator. You are recommended not to touch this file.

- src/java/cool/Semantic.java
  This is a place-holder file for Semantic analysis. **Use your Semantic analysis code, written in the last assignment**. A working semantic analysis code would be posted later, if required.

- src/java/codegen
  Use this to test your compiler (Yes, we can call it a compiler now). Type ./codegen <filename>
- README
  This file will contain the write-up for your assignment. It is critical that you explain design decisions, how your code is structured, and why you believe your design is a good one (i.e., why it leads to a correct and robust program). It is part of the assignment to explain things in text as well as to comment your code.

3 Design

In considering your design, at a high-level, your code generator will need to perform the following tasks:

1. Determine and emit code for global constants, such as prototype objects.
2. Determine and emit code for global tables, such as the string constants, etc.
3. Determine and emit code for initialization method for each class.
4. Determine and emit code for each method definition.

There are a number of things you must keep in mind while designing your code generator:

- You should have a clear picture of the runtime semantics of Cool programs. The semantics are described informally in the first part of the Cool Reference Manual, and a precise description of how Cool programs should behave is given in Section 13 of the manual.
- You should understand the LLVM-IR. You can refer to http://llvm.org/docs/LangRef.html for help.

  You **do not** have to handle the following:

  - Let expressions
  - case expressions
  - Regular Dispatch

  Assume that the input will never have such constructs
  Note: You have to handle Static dispatch

3.1 Runtime Error Checking

The end of the Cool manual lists six errors that will terminate the program. Of these, your generated code should catch the first two. **Divide by zero** and **static dispatch on void** and print a suitable error message before aborting.

4 Testing and Debugging

You will need a working scanner, parser, and semantic analyzer to test your code generator. You are provided with all the above.

You will run your code generator using codegen, a shell script that uses CodegenTest.class which glues together the generator with the rest of compiler phases. This will create a file with the .ll extension in the current working directory. You can convert this file to a .out executable binary with

clang <filename>.ll

5 Final Submission

Make sure to complete the following items before submitting to avoid any penalties.

- Include your write-up in README
- Include your test cases that test your code generator in src/test cases/.
- Make sure to submit and name the tar ball in the mentioned format. As automated scripts would be used for evaluation, your submission may not be evaluated otherwise.
- Make sure to use `Antlr-4.5` as the submissions would be graded comparing with the mentioned version.
- If you are in a group, only one should submit and the others can just "Turn in" in the classroom page.
- The last submission you do will be the one graded.
- The burden of convincing us that you understand the material is on you. Obtuse code, output, and write-ups will have a negative effect on your grade.
- **A flat-rate penalty would be applied for the submissions made within one week after the deadline, after which the submissions would not be evaluated.**
- Take time to clearly (and concisely!) explain your results.

Submit the final code as <your_roll_no>_codegen.tar.gz

Plagiarism policy followed by the department: https://cse.iith.ac.in/academics/plagiarism-policy.html