University of Arizona, Department of Computer Science

CSc 340 Foundations of Computer Systems — Assignment 3

Christian Collberg

Due Thursday, March 8, at 11:59pm

1 Assignment

In the last assignment we executed programs written in *stack code*. This is what the Java interpreter does. In this assignment we'll examine what a Java *compiler* does, namely translate infix expressions (such as $4 * 3 + 9 * (5 + 4)$) into postfix expressions, i.e. code for a stack machine.

The file `/home/cs340/spring01/prog3/ExprParse.java` contains a simple Java implementation of an expression parser and evaluator. The program

1. takes a string such as `"4*3+9*(5+4)"` as input,
2. parses the string and issues an error message if it is not a valid expression,
3. builds a binary tree representing the expression,
4. recursively traverses the tree to emit the stack code, and
5. recursively traverses the tree, evaluating the expression.

Your task is to re-implement this program in MIPS assembly code. The file `/home/cs340/spring01/prog2/eval.s` contains a skeleton program to get you started.

The skeleton file contains a simple implementation of `malloc`, a dynamic memory allocator. You should translate calls to Java's `new` with calls to `malloc`.

You will be given more freedom than in previous assignments to implement your program as you see fit. In other words, you don't have to translate `ExprParse.java` statement-by-statement into MIPS assembly code. However, the purpose of this assignment is for you to practice writing MIPS procedure calls that follow the MIPS calling convention, and I therefore want you to keep the recursive structure of `ExprParse.java`.

You don't have to pay much attention to error handling. If the parser fails due to a syntax error in the input, it's sufficient to print a simple `"parse error"` message and stop parsing.

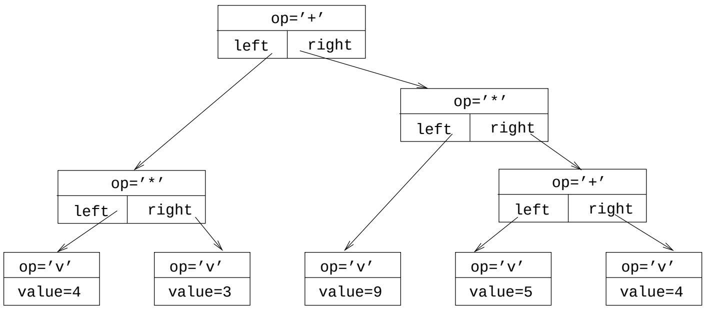
2 Honors section

You can do this assignment on wonka or spim, whichever you prefer.

You should implement assignment 1 in x86 assembly code. You will get 10% extra credit for this.

3 Example

The expression $4 * 3 + 9 * (5 + 4)$ would be parsed into the expression tree



4 Extension 1 [10% extra credit]

Extend the program so that it produces reasonable error messages, rather than just "parse error".

5 Extension 2 [10% extra credit]

Create a new program `comp.s` (which should be submitted separately) containing your interpreter from assignment 2 and the parser from assignment 3. Join them together so that the parser produces bytecodes that the interpreter can execute directly.

6 Extension 3 [10% extra credit]

Extend extension 2 by adding Java-style local variables, `assignment`-, `if`-, and `while`-statements to the parser and generate the appropriate byte-codes for the interpreter. Write some simple test programs.

7 Turnin and the Makefile

A sample makefile for your use can be found in file `/home/cs340/spring01/prog3/Makefile`.

When you have completed the program, submit your program file by typing ⌜`make turnin`⌝. Or, to do it manually type ⌜`turnin cs340prog3 eval.s`⌝. If you have implemented any of the extensions you should describe them in the README file, so that the graders know what to test for.

This is an individual assignment. Don't show your code to anyone, don't read anyone's code, don't discuss the details of the code with anyone. If you need help with the assignment see the TA or the instructor.