UCT 2007 CSC3003S Compilers

Practical Assignment: IR Trees

Write a program to convert Abstract Syntax Trees into IR trees.

The abstract tree structure is for a simple expression calculator and only contains statements for assignment and printing. The grammar is illustrated by the following trees:

```
program(print(const(1)),print(const(2)))
print(const(1))
program(assign(variable(a),const(1)),if(a,print(const(42)),print(const(21))))
assign(variable(a),variable(b))
assign(variable(a),+(const(1),const(2)))
assign(variable(a),-(const(5),const(2)))
program(assign(variable(a),*(const(1),const(2))),print(variable(a)))
```

The grammar for this language is approximately as follows:

```
program → "program" list_of_statements
list_of_statements → statement list_of_statements |
statement → program | print | assign | if
print → "print" expression
assign → "assign" variable expression
if → "if" expression statement statement
expression → variable | const | expression binop expression
binop → "+" | "-" | "/" | "*"
variable → "variable" NAME
const → "const" INTEGER
```

Your program must read in the abstract syntax tree from a file and generate the equivalent IR representation, following the IR language as discussed in class, with the following changes and restrictions:

- Assume all variables are stack-based, so each variable X should be converted to MEM[+[CONST[k_X],TEMP[fp]]].
- Use the shorthand notation for binary operators instead of BINOP.
- Assume that there is a CALL node with "NAME[print]" and a single expression as parameters, in order to produce output.
- The "if" statement will execute the first statement if the expression is greater than 0 – otherwise the second statement will be executed. There is no comparison operator in the AST language but you will need to use the GT (greater-than) operator in the IR language (look at the example provided).
- You do not need to use expression classes, as there is no expression casting in this language.

Your main program must accept a single command-line parameter that is the name of a data file containing the tree structure, with leading "=" symbols on each line to denote the nesting of child elements.

Your output must be a flattened tree structure, for example:

CALL[NAME[print], *[+[CONST[1],CONST[2]],CONST[3]]]

You may use Java or any other programming language that the TA agrees to. You may use any built-in data structures and any parser tools (though those are not necessary).

Test your code with and provide output for the 3 data files attached (testdata.tar.gz). In each case, use output redirection to capture the output and save it to appropriately-named files (e.g., test1.out). As a sample, the output is provided for the test3.txt case.

Your assignment will be marked according to the following criteria:

- Correctness (60%) (test programs generate correct output for given test cases)
- Documentation (20%) (comments within hand-written source – no report necessary)
- Stress (20%) (test programs generate correct output for hidden test cases)