Prof. Jingke Li (FAB120-06, lij@pdx.edu), Tue 10:00-11:50 @UTS 203, Labs: Tue 12:00-13:50 & Wed 12:00-13:50 @FAB 88-10

Lab 4: Stack Code Generation

This lab is a follow-up to Lab 3. In this lab, you are going to use syntax-directed translation scheme to implement an stack-machine IR code generator. The input to this code generator is a program in the AST0 representation.

Preparation

Download `lab3.zip` and unzip it. You'll see a `lab3` directory with the following contents:

`ast0/` — a directory containing the AST representation and its parser

`SC0Gen0.txt` — a starter version of attribute grammars for the SC0 code-gen

`SC0Gen0.java` — a starter version of the SC0 code-gen

`SC0Interp.jar` — an interpreter for the SC0 language

`Makefile` — for compiling programs

`gen, run` — scripts for testing programs

`tst/` — a directory containing some test programs

The Source Language: AST0

A program in this simple language consists of just a list of statements. There are no variable declarations, functions, or other complex constructs. The program representation of AST0 is in `ast0/Ast0.java`. Its grammar is shown here:

```
Program -> {Stmt}

Stmt -> "Assign" Exp Exp
     |  "If" Exp Stmt ["Else" Stmt]
     |  "While" Exp Stmt
     |  "Print" (Exp | "(" ")")

Exp  -> "(" "Binop" BOP Exp Exp ")"
     |  "(" "Unop" UOP Exp ")"
     |  "(" "NewArray" <IntLit> ")"
     |  "(" "ArrayElm" Exp Exp ")"
     |  <Id>
     |  <IntLit>
     |  <BoolLit>

BOP  -> "+" |"-" | "*" | "/" | "&&" | "||" |
        "==" | "!=" | "<" | "<=" | ">" | ">="
UOP  -> "-" | "!"
```

The Target Stack-Machine IR Language: SC0

Instruction	Sematics	Stack Top (before *vs* after)
`CONST n`	load constant `n` to stack	→ `n`
`LOAD n`	load `var[n]` to stack	→ `val`
`STORE n`	store `val` to `var[n]`	`val` →
`ALOAD`	load array element	`arrayref,idx` → `val`
`ASTORE`	store `val` to array element	`arrayref,idx,val` →
`NEWARRAY`	allocate new array	`count` → `arrayref`
`NEG`	`- val`	`val` → `result`
`ADD`	`val1 + val2`	`val1,val2` → `result`
`SUB`	`val1 - val2`	`val1,val2` → `result`
`MUL`	`val1 * val2`	`val1,val2` → `result`
`DIV`	`val1 / val2`	`val1,val2` → `result`
`AND`	`val1 & val2`	`val1,val2` → `result`
`OR`	`val1 | val2`	`val1,val2` → `result`
`GOTO n`	`pc = pc + n`	
`IFZ n`	`if (val == 0) pc = pc + n`	`val` →
`IFNZ n`	`if (val != 0) pc = pc + n`	`val` →
`IFEQ n`	`if (val1 == val2) pc = pc + n`	`val1,val2` →
`IFNE n`	`if (val1 != val2) pc = pc + n`	`val1,val2` →
`IFLT n`	`if (val1 < val2) pc = pc + n`	`val1,val2` →
`IFLE n`	`if (val1 <= val2) pc = pc + n`	`val1,val2` →
`IFGT n`	`if (val1 > val2) pc = pc + n`	`val1,val2` →
`IFGE n`	`if (val1 >= val2) pc = pc + n`	`val1,val2` →
`PRINT`	print `val`	`val` →

Note: For the jump instructions, the operand `n` represents the *relative* displacement from the the current instruction position. `n` can be either positive or negative.

Attribute Grammars for SC0 Generation

(We didn't get to this part in Lab 3.) We'd like to see how to generate SC0 code from the AST0 language. The file `SC0Gen0.txt` contains a copy of the AST0 grammar. Your tasks are

1. decide what attributes are needed, and
2. add attribute definitions to each production to generate SC0 code.

SC0 CodeGen Implementation

Like the `IR0Gen.java` counter part, this code-gen program, `SC0Gen.java`, also follows the syntax-directed translation scheme. The `main` method reads in an AST program through an AST parser, and invokes the `gen` routine on the top-level `Ast0.Program` node. The rest of the program is a collection of (overloaded) `gen` routines, one for each type of AST nodes. Each individual `gen` routine follows an attribute grammar developed specifically for the corresponding AST node.

Your Task Walk through the program to get familiar with the code setup. Use the attribute grammars as guidance, complete the `gen` routine implementation for all AST0 nodes.