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 3: IR Code Generation

In this lab, you are going to use syntax-directed translation scheme to implement an IR code generator from a simple AST source, AST0, which represents common expressions and statements.

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

`ir0/` — a directory containing the target IR representation

`tst/` — a directory containing some test programs

`IR0Interp.jar` — an interpreter for the IR language

`IR0Gen0.java` — a starter version of the IR code-gen

`SC0GenAG0.txt` — a template for adding SC0 code-gen attribute grammars

`Makefile` — for compiling programs

`gen, run` — scripts for testing 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 IR Language: IR0

IR0 is defined in correspondence to AST0. Its program representation is in `ir0/IR0.java`, and its grammar is shown here:

```
Program -> {Inst}

Inst     ->  Binop  | Unop  | Move  | Load | Store
         |   Malloc | Print | CJump | Jump | LabelDec
Binop    ->  Src BOP Src
Unop     ->  Dest "=" UOP Src
Move     ->  Dest "=" Src
Load     ->  Dest "=" Addr
Store    ->  Addr "=" Src
Malloc   ->  Dest "=" "malloc" "(" Src ")"
Print    ->  "print" "(" [Src] ")"
CJump    ->  "if" Src ROP Src "goto" Label
Jump     ->  "goto" Label
LabelDec ->  Label ":"

Addr  -> [<IntLit>] "[" Src "]"
Src   -> <Id> | <Temp> | <IntLit> | <BoolLit>
Dest  -> <Id> | <Temp>
Label -> <Id>

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

<Temp:  "t" (<digit>)+>
<Id:    (<letter> (<letter>|<digit>|"_")*)>
```

IR CodeGen Implementation

The code-gen program, `IR0Gen.java`, follows the syntax-directed translation scheme. The `main` method reads in an AST program through an AST parser. It then 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. (Recall this week's lecture.)

Here is a summary of the specific features of `IR0Gen.java` code setup:

- For an `Ast0.Stmt` node, the `gen` routine returns a list of IR0 instructions (`List<IR0.Inst>`). This instruction list corresponds to the `Stmt.c` attribute discussed in class.
- For an `Ast0.Exp` node, the `gen` routine returns a `CodePack` object which contains two components, a list of IR0 instructions and an `IR0.Src` object (for holding the `Exp`'s value). These two components, likewise, correspond to the attributes, `Exp.c` and `Exp.v`, discussed in class.
- For an `Ast0.Exp` node that may appear on the left-hand side of an `Ast0.Assign` statement, there is also a `genAddr` routine, which returns an `AddrPack` object with a list of instructions and an `IR0.Addr` object (representing the `Exp`'s value as a memory address). For AST0, only one node, `Ast0.ArrayElm`, needs to have this routine defined.

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. After finishing coding, you can compile and test your `IR0Gen` program by using the following commands:

```
linux> make gen
linux> ./gen tst/test*.ast
linux> ./run tst/test*.ir
```

Attribute Grammars for Stack Code Generation

Now we are switching to think of stack-machine IR code. We'd like to see how to generate SC0 code from the AST0 language. The file `SC0GenAG0.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.

For your convenience, SC0's instruction list is shown below.

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.