Project 8

File You Will Create:

Generator.java

Important Files:

Generator0.java
IR.java

Other Files:

Lexer.class
Parser.class
Checker.class
Main.java
Token.java
StringTable.java
SymbolTable.java
LogicError.java
FatalError.java
PrintAst.java
PrettyPrint.java

makefile
tst/...
run
runAll
go
Main.jar ("Black box" solution)

IR Opcodes

```
OPiadd
OPisub
...
OPfadd
...
OPlabel
OPgoto
OPgotoiLT
OPgotoiLE
...
OPgotofLT
OPgotofLE
...
OPassign
OPloadAddr
OPstore
OPloadIndirect
```

Output from "printIR()"

```
        x := y + z  (integer)
        x := y - z  (integer)
        ...
        x := y + z  (float)
        ...
    Label_47:
        goto Label_47
        if x < y then goto Label_43  (integer)
        if x <= y then goto Label_43  (integer)
        ...
        if x < y then goto Label_43  (float)
        if x < y then goto Label_43  (float)
        ...
        x := y
        x := &y
        *x := y
        x := *y
```

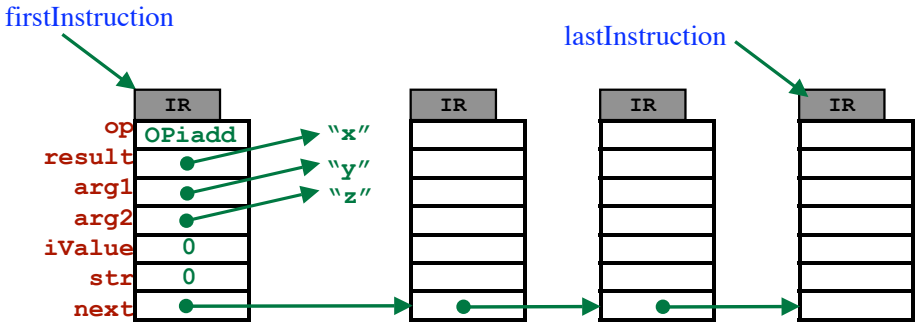
IR Opcodes

```
OPparam
OPcall
OPresultTo
OPprocEntry
OPreturnExpr
OPreturnVoid
OPmainEntry
OPmainExit
OPcomment
```

Output from "printIR()"

```
param  4,x
call foo
resultTo z
procEntry  foo,lexLevel=7,frameSize=120
return x
return
mainEntry
mainExit
! ...string...
```


Linked List of IR Instructions




```
class IR {
  int op;
  AstNode result;
  AstNode arg1;
  AstNode arg2;
  int iValue;
  String str;
  IR next;
  ...
}
```

Will be either VarDecl or Formal

Will be either VarDecl, Formal, IntegerConst, RealConst

Static Methods in IR

```
IR.printIR ()
```
Main will call
Prints all IR instructions

Example Output:

```
Label_43:
  t3 := &x
  t2 := y + z  (integer)
  *t3 := t2
  goto Label_43
```

```
IR.iadd (x,y,z)
IR.isub (x,y,z)
   ...
IR.returnVoid ()
IR.go_to (str)
```

One for each op-code

Your code will look a little like this:

```
lab = NewLabel ();
IR.label (lab);
IR.loadAddr (..., ...);
IR.iadd (..., ..., ...);
IR.store (..., ...);
IR.go_to (lab);
```

Note: "goto" is a Java keyword (use "go_to" here)

Comments

Your Code:

```
IR.iadd (..., ..., ...);
IR.comment ("hello");
IR.isub (..., ..., ...);
```

Result:

```
x := y + z   (integer)
! hello
a := b - c   (integer)
```

Typical Usage:

```
IR.comment ("IF STATEMENT...");
```

Labels

Starter file contains a method

```
newLabel
```

Your Code:

```
String lab = newLabel();
...
IR.go_to (lab);
...
IR.label (lab);
...
```

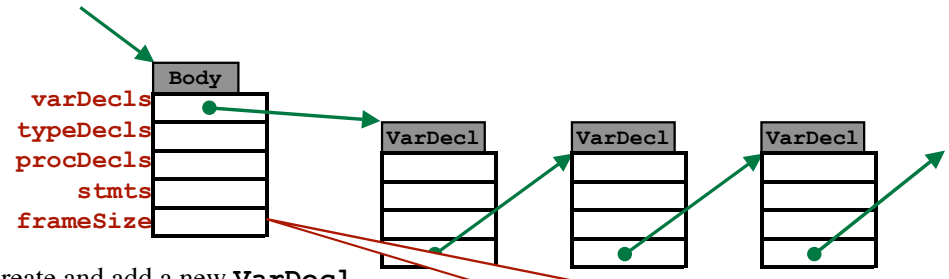
Result:

```
        ...
        goto Label_53
        ...
Label_53:
        ...
```


Temporary Variables

Starter File Contains:

```
Ast.VarDecl newTemp () { ... }
Ast.Body currentBody;
```



- Create and add a new **`VarDecl`** to end of this list.
- Return a pointer to it.

As if the source had included a declaration.
["**`typeName`**" and "**`initExpr`**" will be NULL.]
You must update **`currentBody`**.

Example PCAT Program

```
program is
   procedure foo1 (...) is
      begin
         ...
      end;
   procedure foo2 (...) is
      procedure foo2a (...) is
         begin
            ...
         end;
      procedure foo2b (...) is
         begin
            ...
         end;
      procedure foo2c (...) is
         begin
            ...
         end;
      begin
         ...
      end;
   begin
      ...
   end;
```

```
mainEntry
  ...
mainExit
procEntry foo1
  ...
return
  ...
return
procEntry foo2
  ...
return
procEntry foo2a
  ...
procEntry foo2b
  ...
procEntry foo2c
  ...
```

Each procedure will have one or more return instructions
returnVoid
returnExpr

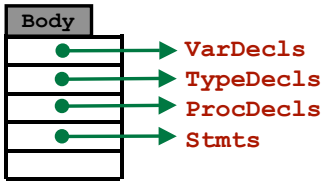
Example PCAT Program

```
program is
   procedure foo1 (...) is
      begin
         ...
      end;
   procedure foo2 (...) is
      procedure foo2a (...) is
         begin
            ...
         end;
      procedure foo2b (...) is
         begin
            ...
         end;
      procedure foo2c (...) is
         begin
            ...
         end;
      begin
         ...
      end;
   begin
      ...
   end;
```

```
mainEntry
  ...
mainExit
procEntry foo1
  ...
return
  ...
return
procEntry foo2
  ...
return
procEntry foo2a
  ...
procEntry foo2b
  ...
procEntry foo2c
  ...
```

Code for the Main Program:

```
mainEntry
<code for variable initialization>
<code for statements>
mainExit
<code for procedures>
```



PCAT Source:

```
procedure foo2 (a:int, b:int,...) is
    procedure foo2a (...) ...
    procedure foo2b (...) ...
    procedure foo2c (...) ...
  begin
    ...
  end;
```

Code for Each Procedure:

```
procEntry foo2
formal  1,a
formal  2,b
...
<code for variable initialization>
<code for statements>
<code for nested procedures>
```

Will include several "return" instructions

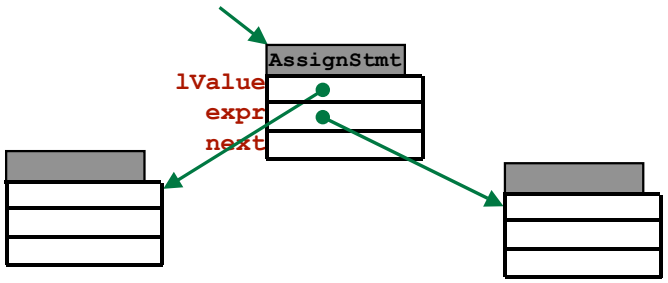
Assignment Statements

Source Code:

```
a := (3 * (b + c)) ;
```

L-Value *Expr*

AssignStmt



Assignment Statements

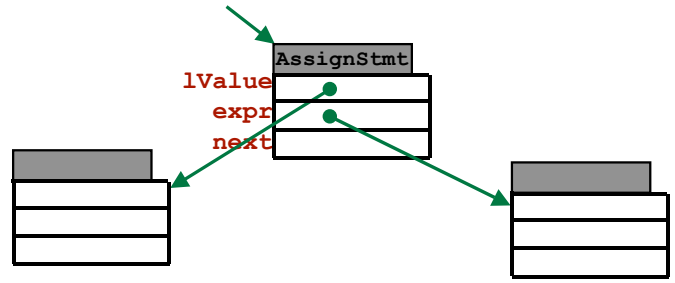
Source Code:

```
a := (3 * (b + c)) ;
```

L-Value — *Expr* — *AssignStmt*

This code is in the starter file

```
void genAssignStmt (Ast.AssignStmt p) {
   IR.comment ("ASSIGNMENT STMT...");
   AstNode x = genLValue (p.lValue);
   AstNode y = genExpr (p.expr, null, null);
   IR.store (x, y);
}
```



Assignment Statements

Source Code:

```
a := (3 * (b + c)) ;
```

L-Value — *Expr* — *AssignStmt*

This code is in the starter file

```
void genAssignStmt (Ast.AssignStmt p) {
   IR.comment ("ASSIGNMENT STMT...");
   AstNode x = genLValue (p.lValue);
   AstNode y = genExpr (p.expr, null, null);
   IR.store (x, y);
}
```

Code Generated:

```
t5 := &a
```

genLValue returns t5

Assignment Statements

Source Code:

```
a := (3 * (b + c)) ;
```

L-Value — *Expr* — *AssignStmt*

This code is in the starter file

```
void genAssignStmt (Ast.AssignStmt p) {
   IR.comment ("ASSIGNMENT STMT...");
   AstNode x = genLValue (p.lValue);
   AstNode y = genExpr (p.expr, null, null);
   IR.store (x, y);
}
```

Code Generated:

```
t5 := &a
t6 := b + c
t7 := 3 * t6
```

genLValue returns t5

genExpr returns t7

Assignment Statements

Source Code:

```
a := (3 * (b + c)) ;
```

L-Value — *Expr* — *AssignStmt*

This code is in the starter file

```
void genAssignStmt (Ast.AssignStmt p) {
   IR.comment ("ASSIGNMENT STMT...");
   AstNode x = genLValue (p.lValue);
   AstNode y = genExpr (p.expr, null, null);
   IR.store (x, y);
}
```

Code Generated:

```
t5 := &a
t6 := b + c
t7 := 3 * t6
*t5 := t7
```

genLValue returns t5

genExpr returns t7

genAssignStmt

Assignment Statements

Source Code:

```
a := (3 * (b + c)) ;
```

L-Value — `a`
Expr — `(3 * (b + c))`
AssignStmt — `a := (3 * (b + c)) ;`

This code is in the starter file

```
void genAssignStmt (Ast.AssignStmt p) {
   IR.comment ("ASSIGNMENT STMT...");
   AstNode x = genLValue (p.lValue);
   AstNode y = genExpr (p.expr, null, null);
   IR.store (x, y);
}
```

Code Generated:

genLValue → returns t5

genExpr → returns t7

```
*t5 := t7
```

genAssignStmt

The "genLValue" Method

Passed a pointer to:

`Variable`	`x`
`ArrayDeref`	`a[i+j]`
`RecordDeref`	`r.name`

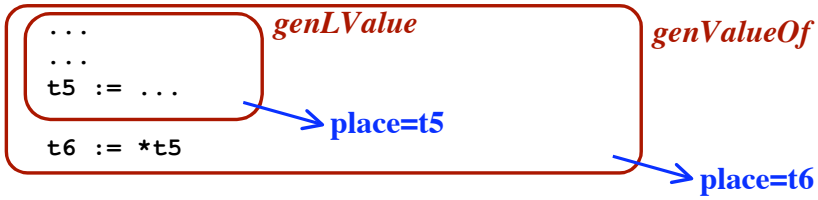
These will be done in project 9

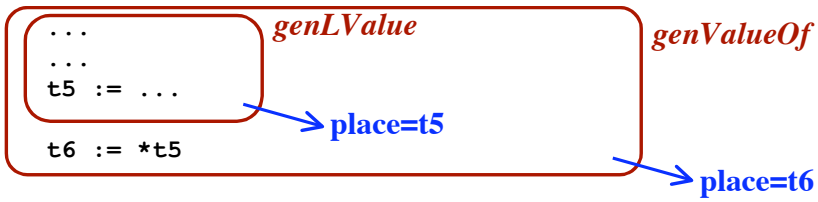
- Create a temporary variable.
- Generate IR code to move an address into this temp.
- Return the temp.

The "genValueOf" Method

Passed a pointer to a ValueOf node

- Call "genLValue" to get address of variable
- Generate a "loadIndirect" instruction to get the data.

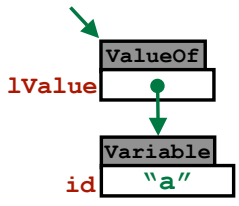


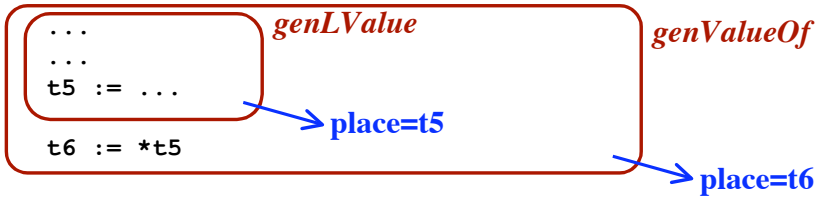


Special Case: The R-Value is a Simple Variable

Example Source

```
x := ... a .... ;
```

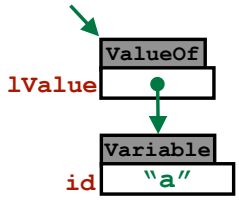




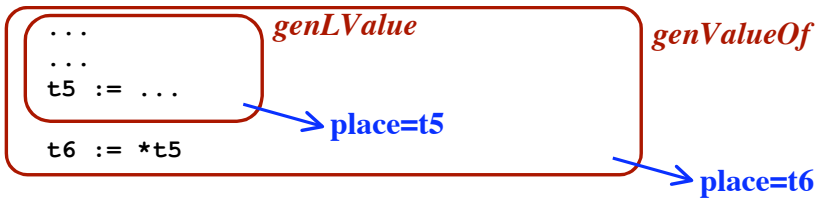
Special Case: The R-Value is a Simple Variable

Example Source

```
x := ... a .... ;
```



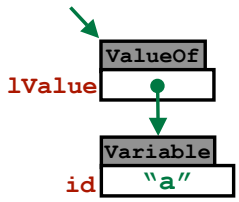
Avoid generating this:



Special Case: The R-Value is a Simple Variable

Example Source

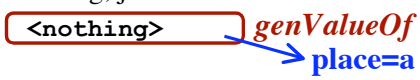
```
x := ... a .... ;
```



Avoid generating this:

Modify "genValueOf" to recognize this case.

- Avoid calling "genLValue".
- Generate nothing; just return the variable.



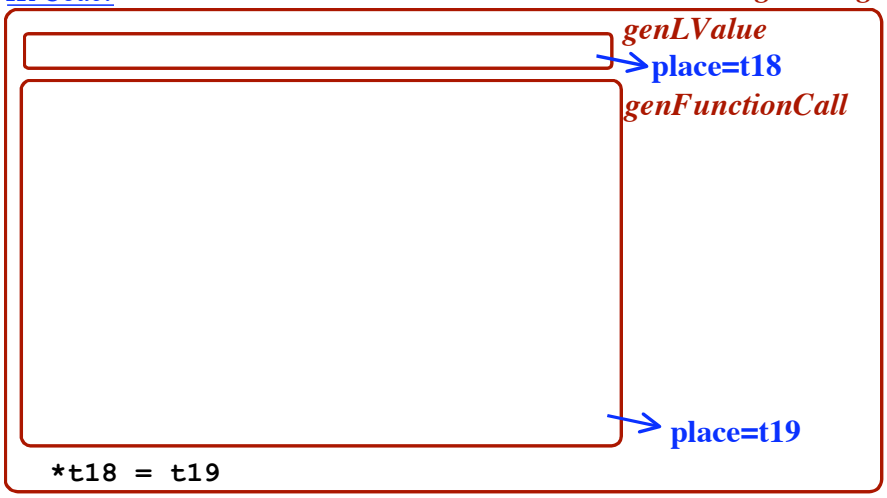
Procedure Invocation

Source:

```
x := foo ( ..., ..., ... );
```

IR Code:

genAssignStmt



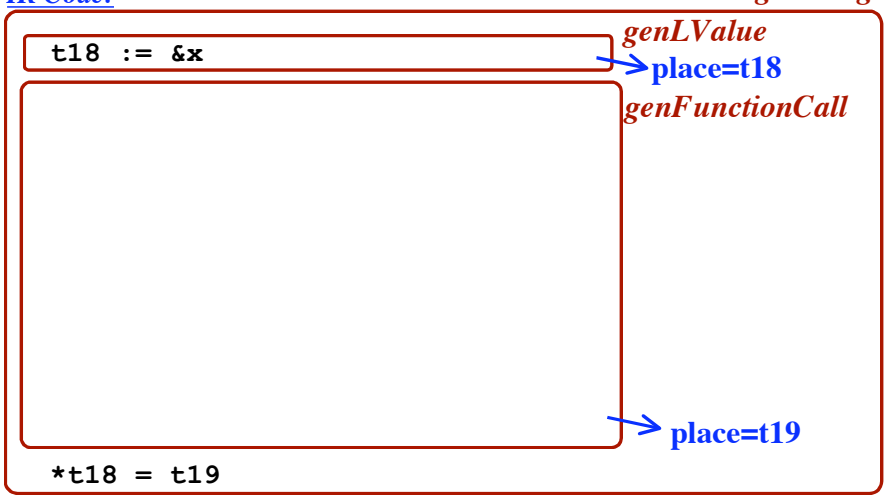
Procedure Invocation

Source:

```
x := foo ( ..., ..., ... );
```

IR Code:

genAssignStmt



Procedure Invocation

Source:

```
x := foo ( ..., ..., ... );
```

IR Code:

genAssignStmt

```
t18 := &x
```

genLValue → place=t18

genFunctionCall

genExpr → place=t23

genExpr → place=t26

genExpr → place=t28

```
param 1,t23
param 2,t26
param 3,t28
call foo
resultTo t19
```

→ place=t19

```
*t18 = t19
```


Procedure Invocation

Source:

```
x := foo ( ..., ..., ... );
```

IR Code:

genAssignStmt

```
t18 := &x
```

genLValue → place=t18

genFunctionCall

```
t23 := ...
```

genExpr → place=t23

```
t26 := ...
```

genExpr → place=t26

```
t28 := ...
```

genExpr → place=t28

```
param 1,t23
param 2,t26
param 3,t28
call foo
resultTo t19
```

→ place=t19

```
*t18 = t19
```

Code for Procedures

Source:

```
procedure foo ( x, y, z: int ) : int is
    begin
        ...
        return w;
    end;
```

IR Code:

```
procEntry foo,lexLevel=2,frameSize=0
formal 1,x
formal 2,y
formal 3,z
...
returnExpr w
```

OPreturnVoid
OPreturnExpr

How To Begin?

Create a skeleton program that walks every part of the AST
by modifying "PrettyPrint.java"

```
prettyPrintAst (body)
ppBody (indent, body)
ppVarDecls (indent, varDecls)
...
ppStmts (indent, stmts)
...
ppExpr (p)
...
```

```
generateIR (body)
genBody (body)
genVarDecls (varDecls)
...
genStmts (stmts)
...
genExpr (p)
...
```

REMOVE:
Everything related to printing

AND PLEASE:
Alter the comments!

Generating Code For Expressions

All the methods that generate code for expressions...

```
genExpr
genBinaryOp
genUnaryOp
```
...etc...

...must do two things:

- Generate IR code to evaluate the expression and place the value into some variable
- Return the variable (i.e., return the synthesized "place" attribute)

The place will be:
Temporary or normal variable
VarDecl
Formal

Generating Code For Expressions

All the methods that generate code for expressions...

```
genExpr
genBinaryOp
genUnaryOp
```
...etc...

...must do two things:

- Generate IR code to evaluate the expression and place the value into some variable
- Return the variable (i.e., return the synthesized "place" attribute)

The place will be:
Temporary or normal variable
VarDecl
Formal

To handle short-circuit code, will add 2 additional parameters.

```
void genExpr (Ast.Expr p)
void genBinaryOp (Ast.Expr p)
void genUnaryOp (Ast.Expr p)
...
```

```
Ast.Node genExpr (Ast.Expr p, String trueLabel, String falseLabel)
Ast.Node genBinaryOp (Ast.Expr p, String trueLabel, String falseLabel)
Ast.Node genUnaryOp (Ast.Expr p, String trueLabel, String falseLabel)
...
```