Project 1 Hints

Jingke Li

Portland State University

Basic Information

- *The Compiler Infrastructure* — Augmented from last term's:
  - `ast --- ` the source-language's AST nodes
  - `astpsr --- ` the AST parser program
  - `ir --- ` the IR tree nodes
  - `irgen0 --- ` the first-version irgen code
  - `tst --- ` a set of test programs

- *IR Tree Nodes:*
  - Top-level abstract class is called `IR`. Two additional abstract classes are at the next level: `STMT` and `EXP`.
  - A group of statement nodes:

    ```
    STMTlist MOVE JUMP CJUMP LABEL CALLST RETURN
    ```
  - A group of expression nodes:

    ```
    EXPlist ESEQ MEM CALL BINOP NAME TEMP FIELD
    PARAM VAR CONST FLOAT STRING
    ```
  - A few more nodes:

    ```
    PROG FUNC FUNClist
    ```

A New Visitor Interface on AST

The IR code generator is to be implemented as a visitor pattern for a new interface over the AST nodes.

```
package ast;
...
public interface TransVI {
  // Note: there is no need to translate type decl nodes
  public PROG visit(Program n) throws Exception;
  public FUNClist visit(ClassDeclList n) throws Exception;
  public FUNClist visit(ClassDecl n) throws Exception;
  ...
  public STMTlist visit(StmtList n) throws Exception;
  public STMT visit(Block n) throws Exception;
  ...
  public EXPlist visit(ExpList n) throws Exception;
  public EXP visit(Binop n) throws Exception;
  ...
  public EXP visit(IntVal n);
  public EXP visit(BoolVal n);
  ...
}
```

Implementation

Guideline:

- Understand the pseudo-code template for each AST node; generate the IR code according to the template.
- Pay attention to the structure of each AST node, and make proper recursive calls to its children if needed.
- Pay attention to the return value type of each node; properly assemble the results from the children.
- Since this is only the first version, some routines' implementation is simplified; their full version will be completed in the next project.

Implementation — Top-Level

```
package irgen0;
...
public class IrgenVisitor0 implements TransVI {
  private NAME cWordSize; // a symbolic name

  public IrgenVisitor0() { cWordSize = new NAME("wSZ"); }

  public PROG visit(Program n) throws Exception {
    FUNClist funcs = n.cl.accept(this);
    return new PROG(funcs);
  }
}
```

Implementation — Declarations

```
public FUNClist visit(ClassDeclList n) throws Exception {
  FUNClist funcs = new FUNClist();
  for (int i = 0; i < n.size(); i++)
    funcs.addAll(n.elementAt(i).accept(this));
  return funcs;
}

public STMTlist visit(VarDeclList n) throws Exception {
  // Each VarDecl returns a STMT or null; need to merge the
  // individual returns into a STMTlist
}

public FUNClist visit(ClassDecl n) throws Exception {
  FUNClist funcs = n.ml.accept(this);
  return funcs;
}

public FUNC visit(MethodDecl n) throws Exception {
  ... return new FUNC(label, 0, 0, stmts); // use 0 for the two
                                           // middle arguments
}
...
```

Implementation — Statements

```
public STMT visit(Block n) throws Exception {
  return n.sl.accept(this);
}
public STMT visit(Assign n) throws Exception {
  EXP lhs = n.lhs.accept(this);
  EXP rhs = n.rhs.accept(this);
  return new MOVE(lhs, rhs);
}
public STMT visit(CallStmt n) throws Exception {
  ... simplified; return CALLST(new NAME(label), ...);
}
public STMT visit(If n) throws Exception {
  ... implement according to the pseudo template
}
public STMT visit(While n) throws Exception {
  ... implement according to the pseudo template
}
public STMT visit(Print n) throws Exception {
  ... return CALLST(new NAME("print"), ...)
}
...
```

Implementation — Expressions

```
public EXPlist visit(ExpList n) throws Exception {
  ... straightforward; return an EXPlist
}
public EXP visit(Binop n) throws Exception {
  ... straightforward; return a BINOP
}
public EXP visit(Relop n) throws Exception {
  ... translate into value representation; return an ESEQ
}
public EXP visit(Unop n) throws Exception {
  ... both ’neg e’ and ’not e’ become ’1 - e’
}
public EXP visit(NewArray n) throws Exception {
  ... implement according to the pseudo template
}
public EXP visit(ArrayElm n) throws Exception {
  EXP array = n.array.accept(this);
  EXP idx = n.idx.accept(this);
  return new MEM(...);
}
public EXP visit(ArrayLen n) throws Exception {
  return new MEM(...);
}
```

Implementation — Expressions (cont.)

```
public EXP visit(NewObj n) throws Exception {
  ... simplified; return CALL(new NAME("malloc"), args)
  ... where 'args' contains a single arg of form
  ... NAME("<class-name>_obj_size")
}
public EXP visit(Field n) throws Exception {
  ... simplified; return a NAME node
}
public EXP visit(Call n) throws Exception {
  ... similar to CallStmt
}
public EXP visit(Id n) throws Exception {
  ... simplified; return a NAME node
}

public EXP visit(IntVal n)   { return new CONST(n.i); }
public EXP visit(BoolVal n)  { return new CONST(n.b); }
public EXP visit(FloatVal n) { return new FLOAT(n.f); }
public EXP visit(StrVal n)   { return new STRING(n.s); }
```

The Driver Routine

```
package irgen0;
...
public class TestIrgen0 {
  public static void main(String [] args) {
    try {
      if (args.length == 1) {
        FileInputStream stream = new FileInputStream(args[0]);
        Program p = new astParser(stream).Program();
        stream.close();
        IrgenVisitor0 iv = new IrgenVisitor0();
        PROG ir = iv.visit(p);
        ir.dump();
      } else {
        System.out.println("You must provide an input file name.");
      }
    }
    catch (Exception e) {
      System.err.println(e.toString());
    }
  }
}
```

An Arith Expr Example

```
MINI: int i = 2 + 2 * 4 - 9 / 3;

AST:  (VarDecl (BasicType int) (Id i)
        (Binop - (Binop + (IntVal 2)
                          (Binop * (IntVal 2) (IntVal 4)) )
                 (Binop / (IntVal 9) (IntVal 3)) ) )

IR:   [MOVE (NAME i)
        (BINOP - (BINOP + (CONST 2)
                          (BINOP * (CONST 2) (CONST 4)))
                 (BINOP / (CONST 9) (CONST 3)) ) ]
```

A Boolean Expr Example

```
MINI: boolean b = (1>2) || (3<4) && !false;

AST:  (VarDecl (BasicType boolean) (Id b)
        (Binop || (Relop > (IntVal 1) (IntVal 2))
                  (Binop && (Relop < (IntVal 3) (IntVal 4))
                            (Unop ! (BoolVal false) ) ) ) )

IR:   [MOVE (NAME b)
        (BINOP || (ESEQ
                    [MOVE (TEMP 1) (CONST 1)]
                    [CJUMP > (CONST 1) (CONST 2) (NAME L0)]
                    [MOVE (TEMP 1) (CONST 0)]
                    [LABEL L0]
                    (TEMP 1) )
                  (BINOP && (ESEQ
                              [MOVE (TEMP 2) (CONST 1)]
                              [CJUMP < (CONST 3) (CONST 4) (NAME L1)]
                              [MOVE (TEMP 2) (CONST 0)]
                              [LABEL L1]
                              (TEMP 2) )
                            (BINOP - (CONST 1) (CONST 0))) ) ]
```

An Array Example

```
MINI: int[] a;
      a = new int[2];
      a[0] = 1;

AST:  (VarDecl (ArrayType (BasicType int) ) (Id a) (NullExp) )
      (Assign (Id a) (NewArray (BasicType int) (IntVal 2)) )
      (Assign (ArrayElm (Id a) (IntVal 0) ) (IntVal 1) )

IR: [MOVE (NAME a) (ESEQ
      [MOVE (TEMP 1) (CALL (NAME malloc) ((BINOP * (CONST 3) (NAME wSZ))))]
      [MOVE (MEM (TEMP 1)) (CONST 2)]
      [MOVE (TEMP 2) (BINOP + (TEMP 1) (BINOP * (CONST 2) (NAME wSZ)))]
      [LABEL L0]
      [MOVE (MEM (TEMP 2)) (CONST 0)]
      [MOVE (TEMP 2) (BINOP - (TEMP 2) (NAME wSZ))]
      [CJUMP > (TEMP 2) (TEMP 1) (NAME L0)]
      (TEMP 1)) ]
    [MOVE (MEM (BINOP + (NAME a)
                        (BINOP * (BINOP + (CONST 0) (CONST 1)) (NAME wSZ)))
          (CONST 1) ]
```

An Object Example

```
MINI: Body b = new Body();
      int i = 2;
      b.i = 3;
      System.out.println(i + b.i);

AST:  (VarDecl (ObjType (Id Body) ) (Id b) (NewObj (Id Body) (ExpList)) )
      (VarDecl (BasicType int) (Id i) (IntVal 2)) )
      (Assign (Field (Id b) (Id i) ) (IntVal 3) )
      (Print (Binop + (Id i) (Field (Id b) (Id i)) ) )

IR:   [MOVE (NAME b) (CALL (NAME malloc) ( (NAME Body_obj_size)))]
      [MOVE (NAME i) (CONST 2)]
      [MOVE (NAME i) (CONST 3)]
      [CALLST (NAME print) ( (BINOP + (NAME i) (NAME i)))]
```

An If Statement Example

```
MINI: if ((3*4)==10)
        System.out.println(4);
      else
        System.out.println(5);

AST: (If (Relop == (Binop * (IntVal 3) (IntVal 4)) (IntVal 10) )
         (Print (IntVal 4) )
         (Print (IntVal 5) ) )

IR:  [CJUMP == (ESEQ [MOVE (TEMP 2) (CONST 1)]
                     [CJUMP == (BINOP * (CONST 3) (CONST 4))
                               (CONST 10) (NAME L4)]
                     [MOVE (TEMP 2) (CONST 0)]
                     [LABEL L4]
                     (TEMP 2) )
               (CONST 0) (NAME L3)]
     [CALLST (NAME print) ( (CONST 4))]
     [JUMP (NAME L5)]
     [LABEL L3]
     [CALLST (NAME print) ( (CONST 5))]
     [LABEL L5]
```