Plan for today

Translating program features to the Tree IR

This time:
- call expressions
- less than operator
- if statement
- while statements

Lvalues versus rvalues

Call expression

... this . otherFunc(y,f)...

Call expression example with recursive call expressions

class MethodCalls {
    public static void main(String[] a) {
        System.out.println(
            new Bar().getBaz().getFoo().testing());
    }
}
class Foo {
    public int testing() {
        return 42;
    }
}
class Bar {
    public Baz getBaz() {
        return new Baz();
    }
}
class Baz {
    public Foo getFoo() {
        return new Foo();
    }
}

outACallExp

Steps
- Figure out the class type for the receiver
  - if (node.getExp() instanceof AThisExp)
  - else if (node.getExp() instanceof ANewExp)
  - else if (node.getExp() instanceof AIdExp)
  - else if (node.getExp() instanceof ACallExp)
- look up class in current symbol table
- look up method in class’ symbol table
- create an ExpNAME for the method, get name from the methods frame
- collect Tree.Exp from argument children
- create a StmMOVE that stores the result of the call
- collect statements from children and place before StmMOVE in stmtlist
- map stmt list and Tree.Exp for temp to ACallExp node
operator <

Low level pseudocode for result of translation
if (lhs < rhs) goto truebody else goto falsebody
falsebody:
    temp = 0
    goto endif
truebody:
    temp = 1
    goto endif
endif:

Tree.Exp for ALTExp node
– ExpTEMP( Temp instance for temp )

---

If statement

Low level pseudocode for result of translation
if (test == true) goto truebody else goto falsebody
falsebody:
    ...
    goto endif
truebody:
    ...
    goto endif
endif:

Tree.Exp for AIfStatement node
– none, AIfStatement is a statement

---

while statement

Low level pseudocode for result of translation
loop:
if (test != true) goto endloop else goto body
body:
    ...
    goto loop

Tree.Exp for AWhileStatement node
– none, AWhileStatement is a statement

---

StmCJUMP

public class StmCJUMP extends Stm {
    public int relop;
    public Exp left, right;
    public Label iftrue,iffalse;
    public StmCJUMP(int rel, Exp l, Exp r, Label t, Label f) {
        ...
    }

    public final static int EQ=0,
    NE=1,
    LT=2,
    GT=3,
    ...
}
Tree intermediate representation

ExpCONST(int i) - The integer constant i
ExpNAME(Label n) - Constant address. Corresponds to label in assembly.
ExpTEMP(Temp t) - Temporary t. “Infinite” Temps in Tree IR.
ExpBINOP(int binop, Exp left, Exp right) - left binop right
ExpMEM(Exp exp) - If left child of move, then store into address calculated by exp.
Otherwise, fetch value at address calculated by exp.
ExpCALL(Exp func, List<Exp> args) - evaluate func to find func address, then evaluate args left to right.

StmMOVE(Temp t, e) - Eval e and put result in t.
StmMOVE(ExpMEM(e1), e2) - Eval e2 and store into address e1.
StmEXP(Exp e) - Eval e and ignore result
StmJUMP(Label targ) - Transfer control to given label.
StmLABEL(Label l) - Label in assembly.

Lvalues versus Rvalues

Lvalue
- “result of an expression that can occur on the left of an assignment statement”
- examples: *(&a), b.membervar, p->membervar

Rvalue
- an expression whose result can only appear as a subexpression or on the rhs of a statement
- examples: &a, 3*4, new Foo()

Why?
- explains compiler errors you might see in the future, e.g. non-lvalue assignment
  - x+3 = 5
  - 4 = 5
- emphasizes the difference between equality in math and assignment in programming languages
- think about where values are stored during the progression of a program
- this is why the lhs of a StmMOVE must be an ExpMEM or ExpTEMP
- what if we could pass around user defined types (not just references to them)?