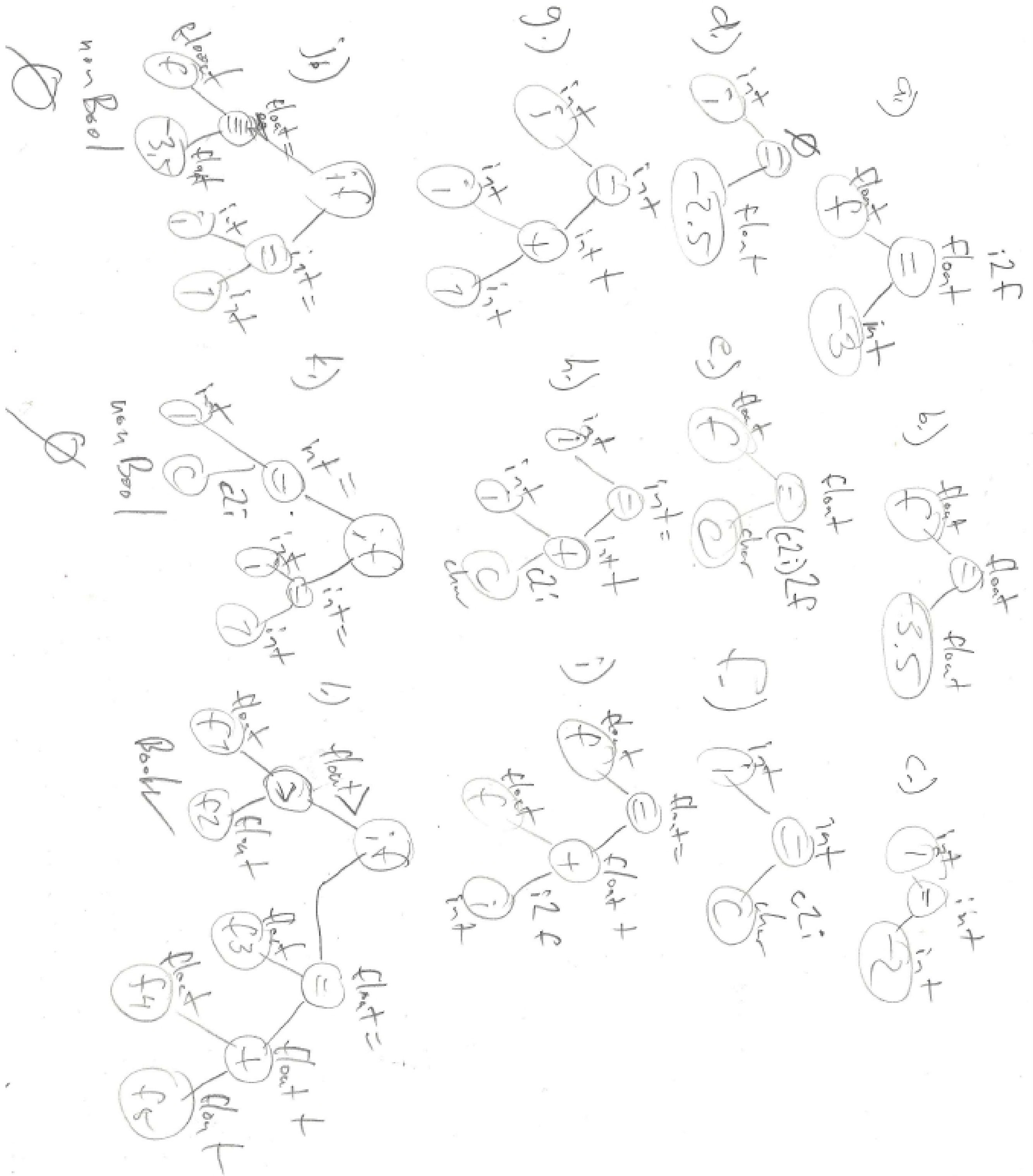
Cnc 2012
Professor Sherri Shulman
assignment 6 -owenMeyer



1. a value is valid
2. a variable is valid if it's id appears in the type map.
2. a binary is valid if all the following are true:
        a. it's expressions term1 and term2 are valid.
        b. if it's binaryOp op is arithmetic then it's term1 must be either int or float and
                if it's term1 is float then it's term2 must be either float or int.
                if it's term1 is int then it's term2 must be either int or char.
c. if it's op is relational then
                if it's term1 is float then it's term2 must be either float or int.
                if it's term1 is int then it's term2 must be either int or char.
                If it's term1 is char then it's term2 must be char.
                If it's term1 is bool then it's term2 must be bool.
…

```
   public static void V (Expression e, TypeMap tm) {
       if (e instanceof Value)
          return;
       if (e instanceof VariableRef) {
                          if (e instanceof Variable){
                                  Variable v = (Variable)e;
                                  check( tm.containsKey(v), "undeclared variable: " + v);
                                  return;
                          }
                          else if (e instanceof ArrayRef){
                                  System.out.println("checking array ref.");
                                  ArrayRef a = (ArrayRef)e;
                                  check( tm.containsKey(a), "undeclared array: " + a);
                                  check( typeOf(a.index(),tm)==Type.INT,"non integer index value");
                                  return;
                          }
       }
       if (e instanceof Binary) {
          Binary b = (Binary) e;
          Type typ1 = typeOf(b.term1, tm);
          Type typ2 = typeOf(b.term2, tm);
          V (b.term1, tm);
          V (b.term2, tm);
          if (b.op.ArithmeticOp( )){
```

```
                                       //added ->
                                       if(typ1 == Type.FLOAT)
                                                 check( typ2 == Type.FLOAT || typ2 == Type.INT,
                                                                                       "type error for" + b.op);
                                       else if(typ1 == Type.INT)
                                                 check( typ2 == Type.FLOAT || typ2 == Type.INT || typ2 ==
Type.CHAR,
                                                                                       "type error for" + b.op);
                                       else throw new IllegalArgumentException("type error for" + b.op);
                             }
                                       //added <-
                             /*
             check( typ1 == typ2 &&
                   (typ1 == Type.INT || typ1 == Type.FLOAT)
                   , "type error for " + b.op);
                             */
                             else if (b.op.RelationalOp( )){
                                       //added ->
                                       if(typ1 == Type.FLOAT)
                                                 check( typ2 == Type.FLOAT || typ2 == Type.INT,
                                                                                       "type error for" + b.op);
                                       else if(typ1 == Type.INT) //added
                                                 check( typ2 == Type.INT || typ2 == Type.CHAR,
                                                                                       "type error for" + b.op);
                                       else if(typ1 == Type.CHAR)
                                                 check(typ2 == Type.INT || typ2 == Type.CHAR,
                                                                                       "type error for" + b.op);
                                       else throw new IllegalArgumentException("type error for" + b.op);
                             }
                                       //added <-

          //    check( typ1 == typ2 , "type error for " + b.op);

          else if (b.op.BooleanOp( ))
             check( typ1 == Type.BOOL && typ2 == Type.BOOL,
                   b.op + ": non-bool operand");
          else
             throw new IllegalArgumentException("should never reach here");
          return;
      }
      // student exercise
                    if (e instanceof Unary) {
                             Unary u = (Unary) e;
                             Type t = typeOf(u.term,tm);
                             V (u.term, tm);
                             if (u.op.NotOp() )
                                       check(typeOf(u.term,tm)==Type.BOOL,
                                                                             "Booleon Operator, NonBoolean Operand");
                             else if (u.op.NegateOp() )
```

```
                                    check(typeOf(u.term,tm)==Type.INT ||
typeOf(u.term,tm)==Type.FLOAT,
                                                                        "Negate Operator, NonNumeric Operand");
                           else if (u.op.charOp() || u.op.floatOp())
                                    check(typeOf(u.term,tm)==Type.INT,"Illegal Cast");
                           else if (u.op.intOp() )
                                    check(typeOf(u.term,tm)==Type.CHAR ||
typeOf(u.term,tm)==Type.FLOAT,
                                                                        "Illegal Cast");
                           return;
                  }
                                    //student end
                  else throw new IllegalArgumentException("should never reach here");
   }
```

6.9)

```
import java.util.*;

public class TypeTransformer {

   public static Program T (Program p, TypeMap tm) {
      Block body = (Block)T(p.body, tm);
      return new Program(p.decpart, body);
   }

   public static Expression T (Expression e, TypeMap tm) {
      if (e instanceof Value)
         return e;
      if (e instanceof VariableRef){
                           if (e instanceof Variable)
                                    return e;
                           if (e instanceof ArrayRef)
                                    return e;
                  }
      if (e instanceof Binary) {
         Binary b = (Binary)e;
         Type typ1 = StaticTypeCheck.typeOf(b.term1, tm);
         Type typ2 = StaticTypeCheck.typeOf(b.term2, tm);
         Expression t1 = T (b.term1, tm);
         Expression t2 = T (b.term2, tm);
         if (typ1 == Type.INT){
                                    if (typ2 == Type.FLOAT)
                                             return new Binary(b.op.intMap(b.op.val),t1,
                                                               new Unary(Operator.floatMap(Operator.INT),t2));
                                    else if (typ2 == Type.INT)
                  return new Binary(b.op.intMap(b.op.val), t1,t2);
                                    else if (typ2 == Type.CHAR)
                                             return new Binary(b.op.intMap(b.op.val), t1,
                                          new Unary(Operator.charMap(Operator.INT),t2) );
                  }
      else if (typ1 == Type.FLOAT){
                        if (typ2 == Type.FLOAT)
            return new Binary(b.op.floatMap(b.op.val), t1,t2);
                        else if (typ2 == Type.INT)
                              return new Binary(b.op.floatMap(b.op.val), t1,
                                          new
Unary(Operator.intMap(Operator.FLOAT),t2) );
                  }
      else if (typ1 == Type.CHAR){
                        if (typ2 == Type.CHAR)
            return new Binary(b.op.charMap(b.op.val), t1,t2);
                        else if (typ2 == Type.INT)
                              return new Binary(b.op.charMap(b.op.val), t1,
                                          new Unary(Operator.intMap(Operator.CHAR), t2));
                  }
      else if (typ1 == Type.BOOL)
        return new Binary(b.op.boolMap(b.op.val), t1,t2);
      throw new IllegalArgumentException("should never reach here");
            }
            // student exercise begin
            if (e instanceof Unary){
                  Unary u = (Unary) e;
                  Type typ = StaticTypeCheck.typeOf(u.term, tm);
                  Expression t = T (u.term, tm);
                  if (typ == Type.INT)
                        return new Unary(u.op.intMap(u.op.val), t);
                  else if (typ == Type.FLOAT)
                        return new Unary(u.op.floatMap(u.op.val), t);
                  else if (typ == Type.CHAR)
                        return new Unary(u.op.charMap(u.op.val), t);
                  else if (typ == Type.BOOL)
                        return u;// new Unary(new Operator(Operator.NOT), t);
                  else throw new IllegalArgumentException("unary dissillusion");
            }
    // student exercise end
    throw new IllegalArgumentException("should never reach here!!");
  }

  public static Statement T (Statement s, TypeMap tm) {
    if (s instanceof Skip) return s;
    if (s instanceof Assignment) {
      Assignment a = (Assignment)s;
                  VariableRef target = null;
                  if (a.target instanceof Variable)
                        target = (Variable)a.target;
                  if (a.target instanceof ArrayRef)
                        target = (ArrayRef)a.target;
```

```
                                Expression src = T (a.source, tm);
                                Type ttype = (Type)tm.get(a.target);
                                Type srctype = StaticTypeCheck.typeOf(a.source, tm);
        if (ttype == Type.FLOAT) {
            if (srctype == Type.INT) {
                src = new Unary(new Operator(Operator.I2F), src);
                srctype = Type.FLOAT;
            }
        }
        else if (ttype == Type.INT) {
            if (srctype == Type.CHAR) {
                src = new Unary(new Operator(Operator.C2I), src);
                srctype = Type.INT;
            }
        }
        StaticTypeCheck.check( ttype == srctype,
                  "bug in assignment to " + target);
        return new Assignment(target, src);
    }
    if (s instanceof Conditional) {
        Conditional c = (Conditional)s;
        Expression test = T (c.test, tm);
        Statement tbr = T (c.thenbranch, tm);
        Statement ebr = T (c.elsebranch, tm);
        return new Conditional(test,  tbr, ebr);
    }
    if (s instanceof Loop) {
        Loop l = (Loop)s;
        Expression test = T (l.test, tm);
        Statement body = T (l.body, tm);
        return new Loop(test, body);
    }
    if (s instanceof Block) {
        Block b = (Block)s;
        Block out = new Block();
        for (Statement stmt : b.members)
            out.members.add(T(stmt, tm));
        return out;
    }
    throw new IllegalArgumentException("should never reach here");
}


public static void main(String args[]) {
    Parser parser  = new Parser(new Lexer(args[0]));
    Program prog = parser.program();
     prog.display();           // student exercise
    System.out.println("\nBegin type checking...");
    System.out.println("Type map:");
```

```
        TypeMap map = StaticTypeCheck.typing(prog.decpart);
         map.display();    // student exercise
        StaticTypeCheck.V(prog);
        Program out = T(prog, map);
        System.out.println("Output AST");
         out.display();    // student exercise
    } //main

    } // class TypeTransformer
```

6.10)

```
…
= V (e.term1, tm) and V (e.term2, tm)
and     (typeOf (e.term2, tm) is elem of {float, int}
                if typeOf (e.term1, tm) = float
        or typeOf (e.term2, tm) is elem of {int, char}
                if typeOf (e.term1, tm) = int)
                        if e is binary and e.op is elem {arithmeticOp}
…
```

6.11

```
   public static void V (Statement s, TypeMap tm) {
       if ( s == null )
          throw new IllegalArgumentException( "AST error: null statement");
       if (s instanceof Skip) return;
       if (s instanceof Assignment) {
          Assignment a = (Assignment)s;
                        if (a.target instanceof Variable){
                                System.out.println("check variable ref: "+a.target.toString());
                                check( tm.containsKey((Variable)a.target)
               , " undefined Vtarget in assignment: " + a.target);
                        }
                        else if (a.target instanceof ArrayRef){
                                System.out.println("checking array ref: "+ a.target.toString());
                                check(tm.containsKey((ArrayRef)a.target),
                                        "undefined Atarget in assignment:" + a.target);
                                V (a.target,tm);
                        }
          V(a.source, tm);
          Type ttype = (Type)tm.get(a.target);
          Type srctype = typeOf(a.source, tm);
          if (ttype != srctype) {
             if (ttype == Type.FLOAT)
                check( srctype == Type.INT
                      , "mixed mode assignment to " + a.target);
             else if (ttype == Type.INT)
                check( srctype == Type.CHAR
                    , "mixed mode assignment to " + a.target);
            else
               check( false
                    , "mixed mode assignment to " + a.target);
        }
        return;
      }
                  if (s instanceof Conditional){
                          //System.out.println("Looking at Conditional.");
                          Conditional c = (Conditional) s;
                          V(c.test, tm);
                          check(typeOf(c.test, tm)==Type.BOOL,"NonBoolean Test Expression.");
                          V (c.thenbranch, tm);
                          V (c.elsebranch, tm);
                          return;
                  }
                  if (s instanceof Loop){
                          //System.out.println("Looking at Loop.");
                          Loop l = (Loop) s;
                          V (l.test, tm);
                          check(typeOf(l.test, tm)==Type.BOOL,"NonBoolean Test Expression.");
                          V (l.body, tm);
                          return;
                  }
                  if (s instanceof Block){
                          //System.out.println("Looking at Block.");
                          Block b = (Block) s;
                          for (Statement stm : b.members){
                                  V (stm, tm);
                          }
                          return;
                  }
                  else throw new IllegalArgumentException("should never reach here");
  }
```

6.12

```
public static Type typeOf (Expression e, TypeMap tm) {
    if (e instanceof Value) return ((Value)e).type;
    if (e instanceof VariableRef) {
                    if (e instanceof Variable){
                            Variable v = (Variable)e;
                            check (tm.containsKey(v), "undefined variable: " + v);
                            return (Type) tm.get(v);
                    }
                    else if (e instanceof ArrayRef){
                            System.out.println("fetching array type");
                            ArrayRef a = (ArrayRef)e;
                            Type ty = (Type) tm.get(a);
```

```
                                 check (tm.containsKey(a), "undefined array: " + a);
                                 return ty;
                        }
   }
   if (e instanceof Binary) {
      Binary b = (Binary)e;
      if (b.op.ArithmeticOp( ))
         if (typeOf(b.term1,tm)== Type.FLOAT)
            return (Type.FLOAT);
         else return (Type.INT);
      if (b.op.RelationalOp( ) || b.op.BooleanOp( ))
         return (Type.BOOL);
   }
   if (e instanceof Unary) {
      Unary u = (Unary)e;
      if (u.op.NotOp( ))        return (Type.BOOL);
      else if (u.op.NegateOp( )) return typeOf(u.term,tm);
      else if (u.op.intOp( ))    return (Type.INT);
      else if (u.op.floatOp( )) return (Type.FLOAT);
      else if (u.op.charOp( ))  return (Type.CHAR);
   }
   throw new IllegalArgumentException("should never reach here");
}
```