Homework: ArithLang

Learning Objectives:

1. Write programs in ArithLang
2. Understand and extend ArithLang interpreter

Instructions:

1. Total points: 56 pt
2. Early deadline: Feb 8 (Wed) 11:59 pm, Regular deadline Feb 10 (Fri) 11:59 pm
3. Download HW0_FrameworkSetup.zip and HW0_TutorialToSetupTheFramework.pdf from Canvas
4. Set up the programming project according to the instructions in the tutorial
5. How to submit:
   - For questions 1–3, you can write your solutions in latex or word and then convert it to PDF; or you can submit a scanned document with legible handwritten solutions. Please provide the solutions in one PDF file.
   - For questions 4 and 5, please submit your solutions in two different zip files with all the source code files (just zip the complete project's folder).
   - Submit two zip files and one PDF file to Canvas under Assignments, Homework 2

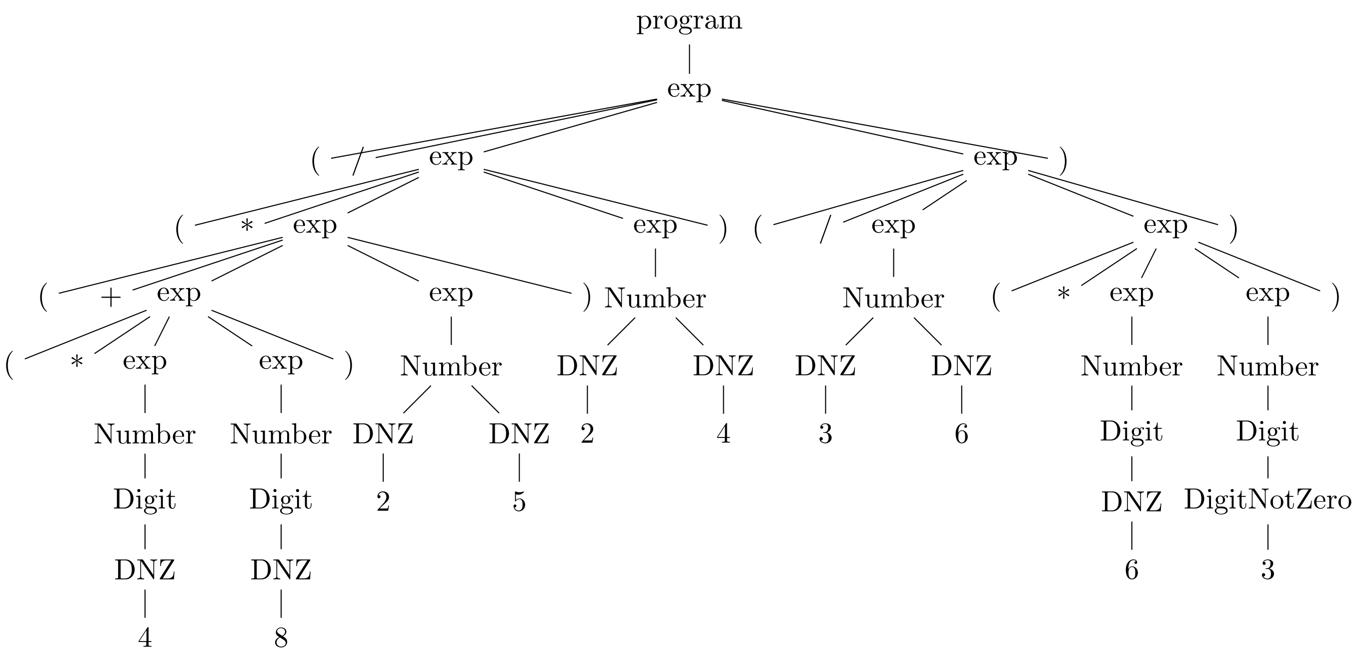
Questions:

1. (8 pt) [Writing ArithLang programs]
   (a) (4 pt) Write two ArithLang programs that compute 342, containing three or more operators.
   Solution:
      i. (+ (* 3 100) (/ 84 (- 279 277)))
      ii. (+ (* 10 30) (/ 12 6) (+ 34 6))
   (b) (4 pt) Convert the following infix notation to ArithLang programs
      i. $(5 - 1)/(2 * 2)$
      ii. $((2 + 3) * (6 - 1))/(7 - 2)$
   Solution:
      i. (/ (- 5 1) (* 2 2))
      ii. (/ (* (+ 2 3) (- 6 1)) (- 7 2))
2. (4 pt) [ArithLang Syntax] Get familiar with the syntax of ArithLang: construct parse trees for the following programs in ArithLang:

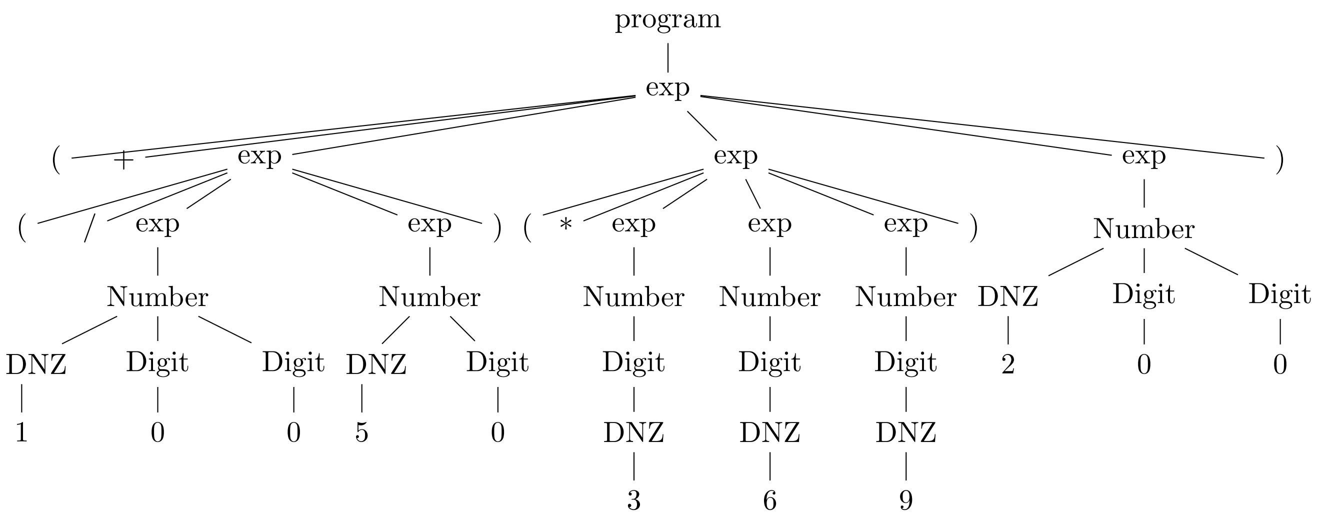
(a) (/ (∗ (+ (∗ 4 8) 25) 24) (/ 36 (∗ 6 3)))

(b) (+ (/ 100 50) (∗ 3 6 9) 200)

Solution:


program
exp
(/ exp exp)
(∗ exp exp) (/ exp exp)
(+ exp exp) Number Number (∗ exp exp)
(∗ exp exp) Number DNZ DNZ DNZ DNZ Number Number
Number Number DNZ DNZ 2 4 3 6 Digit Digit
Digit Digit 2 5 DNZ DigitNotZero
DNZ DNZ 6 3
4 8

(a)


program
exp
(+ exp exp exp)
(/ exp exp) (∗ exp exp exp) Number
Number Number Number Number Number DNZ Digit Digit
DNZ Digit Digit DNZ Digit Digit Digit Digit 2 0 0
1 0 0 5 0 DNZ DNZ DNZ
3 6 9

(b)

3. (3 pt) [ArithLang Interpreter Understanding] Get familiar with the ArithLang source code: List the interpreter code that implements the visitor pattern for evaluating an expression when it processes the expression `(* 2 3)`.

Solution:

- An interface of Visitor should be declared in the AST interface. This interface specifies a method **visit** for each concrete AST node.

```
1 public interface Visitor <T> {
2   public T visit(AST.NumExp e);
3   public T visit(AST.AddExp e);
4   public T visit(AST.SubExp e);
5   public T visit(AST.MultExp e);
6   public T visit(AST.DivExp e);
7   public T visit(AST.Program p);
8   }
```

- Each class in the AST interface includes a method **accept** that takes an object of type **Visitor** as a parameter and invokes method **visit** on that object (e.g., line 3 in the code below).

```
1 public static class MultExp extends CompoundArithExp {
2   public MultExp(List<Exp> args) { super(args); }
3   public Object accept(Visitor visitor) { return visitor.visit(this); }
4   }
```

- Concrete AST traversal functionalities can be implemented by extending the **Visitor** interface in `Evaluator`. The implementation must override visit methods for each type of node (or subclass) of the AST node types (e.g., AddExp). For each sub-expression in each expression, the visitor must call the method accept (line 6), so these sub-expressions are evaluated.

```
1  @Override
2  public Value visit(MultExp e) {
3    List<Exp> operands = e.all();
4    double result = 0;
5    for(Exp exp: operands) {
6      NumVal intermediate = (NumVal) exp.accept(this);
7      result *= intermediate.v();
8    }
9    return new NumVal(result);
10   }
```

- In addition, in the Formatter class under Printer, we use visitors to print the input. For each AST node, the `accept` method will be called once.

```
1 public String visit(MultExp e) {
2   String result = "(*";
3   for(AST.Exp exp : e.all())
4   result += " " + exp.accept(this);
5   return result + ")";
6 }
```

4. (15 pt) [Extend ArithLang Interpreter] Extend the ArithLang to add the *modulus* operation. The operation computes the reminder when performing an integer division. See the example interaction scripts below:

   $ (% 10 4)
   2

```
$ (% 8 3 2)
0
$ (+ (% 7 3) 4)
5
$ (% -2 0)
ERROR // You can report any error message
```

Solution: Found in `hw2code-4-sol.zip`. Files to modify:

- (3 pt) Grammar file (ArithLang.g)
- (3 pt) AST file (AST.java)
- (2 pt) Printer file (Printer.java)
- (7 pt) Evaluator (Evaluator.java)

(3 pt) Grammar file (ArithLang.g)

```
exp returns [Exp ast]:
  n=numexp { $ast = $n.ast; }
  ...
  | md=modexp { $ast = $md.ast; }
  ;
modexp returns [ModExp ast]
        locals [ArrayList<Exp> list]
        @init { $list = new ArrayList<Exp>(); } :
        '(' '%'
            e=exp { $list.add($e.ast); }
                  ( e=exp { $list.add($e.ast); } )+
         ')' { $ast = new ModExp($list); }
        ;
```

(3 pt) AST file (AST.java)

```
public static class ModExp extends CompoundArithExp {
    public ModExp(List<Exp> args) {
        super(args);
    }
    public Object accept(Visitor visitor) {
        return visitor.visit(this);
    }
}

public interface Visitor <T> {
  ...
  public T visit(AST.ModExp e);
}
```

(2 pt) Printer file (Printer.java)

```
public String visit(ModExp e) {
    String result = "(%";
    for(AST.Exp exp : e.all())
        result += " " + exp.accept(this);
    result += ")";
    return result;
}
```

(7 pt) Evaluator (Evaluator.java)

```
@Override
public Value visit(ModExp e) {
    List<Exp> operands = e.all();

    List<Integer> values = new ArrayList<>();
    for(int i = 0; i < operands.size(); i++) {
        NumVal rVal = (NumVal) operands.get(i).accept(this);
        if (rVal.v() <= 0 || rVal.v() % 1 > 0) {
            return new DynamicError("Negative, zero or float operator for mod
                is not permitted. " + ts.visit(e));
        }
        values.add((int) rVal.v());
    }

    int result = values.get(0);
    for(int i = 1; i < values.size(); i++) {
        result = mod(result, values.get(i));
    }

    return new NumVal(result);
}

private int mod(int a, int b) {
      return a%b;
}
```

5. (26 pt) [Interpreter Implementation] Implement an interpreter for AbstractLang described as follows. You can modify from the ArithLang code.

   (a) This language contains only four terminals: `0, p, n, u`.

   (b) `p` represents positive numbers, `n` represents negative numbers and `u` represent unknown values.

   (c) There are three operators $*$, $-$, $+$ that can be applied on the terminals, and their syntactic rules are similar to $*$, $-$ and $+$ in ArithLang, except that each operator only can take two operands.

   (d) The semantics of AbstractLang are defined by the following rules (the first column in the table represents the first operand and the first row in the table represents the second operand of the operation):

+	**0**	**p**	**n**	**u**
0	0	p	n	u
p	p	p	u	u
n	n	u	n	u
u	u	u	u	u

−	**0**	**p**	**n**	**u**
0	0	n	p	u
p	p	u	p	u
n	n	n	u	u
u	u	u	u	u

∗	**0**	**p**	**n**	**u**
0	0	0	0	0
p	0	p	n	u
n	0	n	p	u
u	0	u	u	u

See the example interaction scripts below:

$ (+ p p)
p
$ (∗ p n)
n
$ (+ (− p p) (∗ n n))
u
$ (∗ p p p)
ERROR // You can report any error message (Hint: Look into ANTLRErrorStrategy)

Sol: Found in hw2code5-sol.zip. You will need to modify the following files:

- (6 pt) Grammar file (ArithLang.g)

```
exp returns [Exp ast]:
     n=numexp { $ast = $n.ast; }
       | a=addexp { $ast = $a.ast; }
       | s=subexp { $ast = $s.ast; }
       | m=multexp { $ast = $m.ast; }
       ;

numexp returns [NumExp ast]:
   n0=Num { $ast = new NumExp($n0.text); }
     ;

addexp returns [AddExp ast]
       :
   '(' '+'
       e1=exp
       e2=exp
   ')' { $ast = new AddExp($e1.ast, $e2.ast); }
   ;

subexp returns [SubExp ast]
       :
   '(' '-'
       e1=exp
           e2=exp
   ')' { $ast = new SubExp($e1.ast, $e2.ast); }
   ;

multexp returns [MultExp ast]
       :
   '(' '*'
       e1=exp
           e2=exp
   ')' { $ast = new MultExp($e1.ast, $e2.ast); }
   ;
```

- (4 pt) AST file (AST.java)

```
public static class Program extends ASTNode {
    Exp _e;

    public Program(Exp e) {
        _e = e;
    }

    public Exp e() {
        return _e;
    }

    public Object accept(Visitor visitor) {
        return visitor.visit(this);
    }
}
public static abstract class Exp extends ASTNode {

}

public static class NumExp extends Exp {
    String _val;

    public NumExp(String v) {
        _val = v;
    }

    public String v() {
        return _val;
    }

    public Object accept(Visitor visitor) {
        return visitor.visit(this);
    }
}

public static abstract class CompoundArithExp extends Exp {
    Exp _lop;
    Exp _rop;
    public CompoundArithExp(Exp lop, Exp rop) {
        _lop = lop;
        _rop = rop;
    }

    public Exp getLOp() {
        return _lop;
    }

    public Exp getROp() {
        return _rop;
    }
}

public static class AddExp extends CompoundArithExp {
    public AddExp(Exp lop, Exp rop) {
        super(lop, rop);
```

```java
    }
    public Object accept(Visitor visitor) {
        return visitor.visit(this);
    }
}

public static class SubExp extends CompoundArithExp {
    public SubExp(Exp lop, Exp rop) {
        super(lop, rop);
    }
    public Object accept(Visitor visitor) {
        return visitor.visit(this);
    }
}

public static class MultExp extends CompoundArithExp {
    public MultExp(Exp lop, Exp rop) {
        super(lop, rop);
    }
    public Object accept(Visitor visitor) {
        return visitor.visit(this);
    }
}

public interface Visitor <T> {
    // This interface should contain a signature for each concrete AST
       node.
    public T visit(AST.NumExp e);
    public T visit(AST.AddExp e);
    public T visit(AST.SubExp e);
    public T visit(AST.MultExp e);
   // public T visit(AST.DivExp e);
    public T visit(AST.Program p);
}
```

- (10 pt) Evaluator (Evaluator.java)

```java
@Override
public Value visit(AddExp e) {
    Exp lop = e.getLOp();
    Exp rop = e.getROp();

    Value resultLOp = (Value) lop.accept(this);
    Value resultROp = (Value) rop.accept(this);
    if (resultLOp instanceof PositiveVal) {
        if (resultROp instanceof PositiveVal || resultROp instanceof
            ZeroVal) {
            return new PositiveVal();
        } else {
            return new UnknownVal();
        }
    } else if (resultLOp instanceof NegativeVal) {
        if (resultROp instanceof PositiveVal || resultROp instanceof
            UnknownVal) {
             return new UnknownVal();
        } else {
             return new NegativeVal();
        }
    } else if (resultLOp instanceof ZeroVal) {
        return resultROp;
    }

    return new UnknownVal();

}

@Override
public Value visit(NumExp e) {
    if (e.v().equals("p")) {
        return new PositiveVal();
    } else if (e.v().equals("n")) {
        return new NegativeVal();
    } else if (e.v().equals("u")) {
        return new UnknownVal();
    }

    return new ZeroVal();
}

@Override
public Value visit(MultExp e) {
    Exp lop = e.getLOp();
    Exp rop = e.getROp();

    Value resultLOp = (Value) lop.accept(this);
    Value resultROp = (Value) rop.accept(this);
    if (resultLOp instanceof ZeroVal || resultROp instanceof ZeroVal) {
        return new ZeroVal();
    } else if (resultLOp instanceof PositiveVal) {
        if (resultROp instanceof NegativeVal) {
            return new NegativeVal();
        } else if (resultROp instanceof PositiveVal) {
            return new PositiveVal();
        } else {
            return new UnknownVal();
        }
    } else if (resultLOp instanceof NegativeVal) {
        if (resultROp instanceof NegativeVal) {
            return new PositiveVal();
        } else if (resultROp instanceof PositiveVal) {
            return new NegativeVal();
        } else {
            return new UnknownVal();
        }
```

```java
    }

    return new UnknownVal();
}

@Override
public Value visit(Program p) {
    return (Value) p.e().accept(this);
}

@Override
public Value visit(SubExp e) {
    Exp lop = e.getLOp();
    Exp rop = e.getROp();

    Value resultLOp = (Value) lop.accept(this);
    Value resultROp = (Value) rop.accept(this);
    if (resultLOp instanceof ZeroVal) {
        if (resultROp instanceof PositiveVal) {
            return new NegativeVal();
        } else if (resultROp instanceof NegativeVal) {
            return new PositiveVal();
        } else if (resultROp instanceof ZeroVal) {
            return new ZeroVal();
        }
        else {
            return new UnknownVal();
        }
    } else if (resultLOp instanceof PositiveVal) {
        if (resultROp instanceof PositiveVal || resultROp instanceof
            UnknownVal) {
            return new UnknownVal();
        } else {
            return new PositiveVal();
        }
    } else if (resultLOp instanceof NegativeVal) {
        if (resultROp instanceof PositiveVal || resultROp instanceof
            ZeroVal) {
            return new NegativeVal();
        } else {
            return new UnknownVal();
        }
    }

    return new UnknownVal();
}
```

- (3 pt) Printer file (Printer.java)

```java

public String visit(AddExp e) {
    String result = "(+";
    result += " " + e.getLOp() + " " + e.getROp();
    return result + ")";
}

public String visit(SubExp e) {
    String result = "(-";
    result += " " + e.getLOp() + " " + e.getROp();
    return result + ")";
}

public String visit(MultExp e) {
    String result = "(*";
    result += " " + e.getLOp() + " " + e.getROp();
    return result + ")";
}
```

- (3 pt) Value file (Value.java)

```
public interface Value {
    public String toString();
    static class ZeroVal implements Value {
        public String toString() {
            return "0";
        }
    }

    static class PositiveVal implements Value {
        public String toString() {
            return "p";
        }
    }

    static class NegativeVal implements Value {
        public String toString() {
            return "n";
        }
    }

    static class UnknownVal implements Value {
        public String toString() {
            return "u";
        }
    }
```