Midterm

1. Multiple Choice Questions: select *all* the correct answers for the following questions.

   (a) (3 pt) Given the following grammar, select the strings that it accepts.
   $real - number \rightarrow signpart.part|signpart$
   $sign \rightarrow +|-$
   $part \rightarrow digit|digitpart$
   $digit \rightarrow 0|1|2|3|4|5|6|7|8|9$

   i. `+32.52`
   ii. `3`
   iii. `-.5`
   iv. `-000.`

   Sol i

   (b) (3 pt) What types of programming paradigms available?

   i. logic programming
   ii. functional programming
   iii. object-oriented programming
   iv. imperative programming

   Sol i,ii,iii,iv

   (c) (3 pt) Which of the following is/are true?

   i. terminals are tokens
   ii. a grammar always has a start symbol
   iii. the nodes in a parse tree are either terminals or non-terminals
   iv. even for an ambiguous grammar, left-most and right most derivations can generate the same parse trees

   Sol i,ii,iii,iv

2. (6 pt) What are the three common parts of programming languages? Provide examples to explain each part.

   Sol Atomic computation, composition, abstraction.

   For C programming language

   (a) the built-in `+,-,*,/` are atomic operators.
   (b) the conditional branch (`if` statement), loops (`for, while, do-while`) are composition
   (c) functions are abstractions

3. (15 pt) Consider the following grammar:

   $S \to S + A|S - A|A$
   $A \to A * B|A/B|B$
   $B \to 1|2|3|4|5|6|7|8|9$

   (a) (2 pt) What is the terminals and non-terminals?

   (b) (3 pt) Construct the parse tree for the string `1/3+8-2*4`

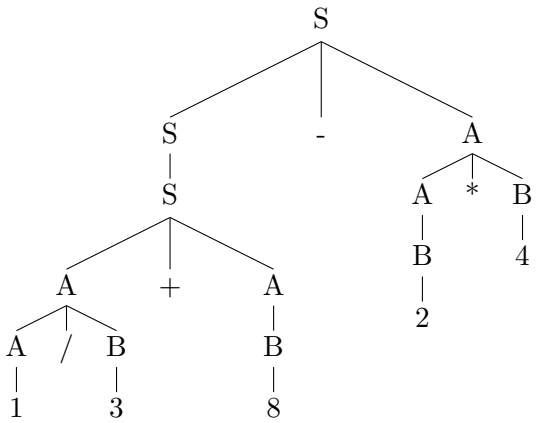
   (c) (5 pt) Is the grammar ambiguous or not? Justify your answer.

   (d) (5 pt) Extend the grammar to support braces, e.g. `1*(2+3)`, where the expression in braces should have the highest priority among all the operators.

   Sol

   (a)
   - terminals: `+,-,*,/`
   - non-terminals: `S,A,B`

   (b)



   (c) Not ambiguous. There's no string that has different parse tree for this grammar. The grammar used associativity and precedence of the operators, which are common techniques to remove ambiguous.

   (d) $S \to S + A|S - A|A$
   $A \to A * B|A/B|B$
   $B \to 1|2|3|4|5|6|7|8|9|(S)$

4. (5 pt) Identify free and bound variables in the following expression:
   (let ((addConst (lambda (x y) (+ x y c)))) (addConst 342 z))

   Sol

   - free variables: `c,z`
   - bound variables: `x,y`

5. (5 pt) Write an Arithlang expression that uses +,-,* and is equal to 10.

   Sol `(* (/ (- (+ 5 10) 11) 2) 5)`

6. (5 pt) Write a Varlang expression that calculates the area of a triangle($\frac{1}{2} * base * height$) given the base is 10 and the height is 6. You are also restricted to defining a single variable per let expression.

   Sol

```
(let ((base 10))
  (let ((height 6))
    (* (/ 1 2) base height)))
```

7. (5 pt) Write a Funclang program `reverse(lst)` that takes a list as an argument and reverses it.

```
(define reverse
  (lambda (lst)
    (if (list? lst)
        (append (reverse (cdr lst)) (list car lst))
      (list lst))))
```

8. (10 pt) Write Funclang programs to accomplish the following tasks.

   (a) (3 pt) Construct a global variable `mylist` that holds a list of three pairs, (1,3) (4,2) (5,6).

   (b) (7 pt) Write a function `nth` that takes two arguments `l` and `n`, where `l` is a list of pairs, and `n` is an integer. Returns the larger number of the two numbers in the nth pair. You can assume that `n` always within the range of the length of the list. Some examples of using `nth` (with the above `mylist` variable): If the n is out of range of the list, return -1. You can use the predicate `list?` which accepts an expression and return `#t` if it is a list and `#f` otherwise. E.g. `(list? (list 1 2))` returns `#t` and `(list? 3)` returns `#f`.

```
$ (nth mylist 1)
$ 3
$ (nth mylist 2)
$ 4
$ (nth mylist 4)
$ -1
$ (nth mylist 0)
$ -1
$ (nth mylist -4)
$ -1
```

   Sol

   (a) a

```
(define mylist (list (cons 1 3) (cons 4 2) (cons 5 6)))
```

```
(define nth
  (lambda (l n)
    (if (< n 1) -1
      (if (list? l)
          (if (= n 1) (car l)
            (nth (cdr l) (- n 1)))
        (if (= n 1) l -1)))))
```

(19) (15 pt) Extend the language to support modular operator (%). E.g. `(% 8 3)` returns 2, and `(%8 3 2)` returns 0. Write the grammar for the new operator, and complete the given evaluator class. To save the time, you don't need to write down the class `ModExp` in AST.java. But you can assume that The `ModExp` class is extended from the `CompoundArithExp` class and has the method `public List<Exp> all();` to extract operands.

(a) (10 pt) Evaluator

```
class Evaluator {
  public Value visit(ModExp e) {
    // write the evaluation of e here









  }
}
```

(b) (5pt) grammar. You only need to complete the production rule of `modexp`.

```
modexp returns [ModExp ast]
locals [ArrayList<Exp> list]
@init {list = new ArrayList<Exp>(); } :
// complete the grammar here
```

Sol

(a)

```
class Evaluator {
  public Value visit(ModExp e) {
    // write the evaluation of e here
    List<Exp> operands = e.all();
    Exp first = operands.get(0);
    NumVal first_val = (NumVal)first.accept(this);
    double result = first_val.v();
    for (int i=1;i<operands.size();i++) {
      NumVal v = (NumVal)operands.get(i).accept(this);
      result = result % v.v();
    }
    return new NumVal(result);
  }
}
```

(b)

```
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); }
```

10. (15 pt) Let expression is useful to define multiple variables at the same time. However, one might want to refer to the previous defined variables in the same let expression when defining later variables. For example, in the evaluation of `(let ((a 3) (b a) (c (+ a b))) c)`, we created three variables, a, b and c, where a=3, b=a, c=a+b. Currently the Varlang intepreter will report "No binding found for name: a" This problem asks you to modify existing let evaluator (code shown below) to eliminate such errors and support this behavior. In the above example, *b* will get the value 3

```
public Value visit(LetExp e, Env env) {
  List<String> names = e.names();
  List<Exp> value_exps = e.value_exps();
  List<Value> values = new ArrayList<Value>(value_exps.size());

  for(Exp exp : value_exps)
  values.add((Value)exp.accept(this, env));

  Env new_env = env;
  for (int i = 0; i < names.size(); i++)
  new_env = new ExtendEnv(new_env, names.get(i), values.get(i));

  return (Value) e.body().accept(this, new_env);
}
```

Sol

```
public Value visit(LetExp e, Env env) {
  List<String> names = e.names();
  List<Exp> value_exps = e.value_exps();
  List<Value> values = new ArrayList<Value>(value_exps.size());
  Env new_env = env;

  for (int i = 0; i < names.size(); i++) {
    Exp exp = value_exps.get(i);
    Value v = ((Value)exp.accept(this, env));
    new_env = new ExtendEnv(new_env, names.get(i), v);
  }

  return (Value) e.body().accept(this, new_env);
}
```

11. (Extra Credit: 8 pt) For Question 8: write a function that returns a new list, each element is the multiplication of the two elements in the pair located in a given list, e.g., for the `mylist` defined above, the output is (3, 6, 30)

Sol

```
(define mul-list
  (lambda (lst)
```

```
(if (list? lst)
    (cons (*
           (car (car lst))
           (cdr (car lst)))
          (mul-list (cdr lst)))
  ()))
```

12. (Extra Credit: 2 pt) Write a poem on the topic of programming languages