

Name: AH-J

For this project we will be using Prolog to tokenize, generate a parse tree, and interpret.

Parsing Techniques (Subject to change):

- Three Address Code
- Common Subexpression Elimination
- Constant folding and propagation

Interpreter:

- Prolog

Data Structure Used:

- Lists Representation of (due to usage of Prolog):
 - Concrete Syntax Tree(s)
 - Abstract Syntax Tree(s)
-

Design & Grammar:

PROGRAM ::= *'Salutations Xiangyu,'*, LIST, *'Sincerely, Ajay Bansal'*

LIST ::= *'Would you mind doing the following:'*, DECLARATION, COMMAND, *'Thank you.'*

| *'Would you mind doing the following:'*, DECLARATION, BLOCK_COMMAND, *'Thank you.'*

| *'Would you mind doing the following:'*, COMMAND, *'Thank you.'*

| *'Would you mind doing the following:'*, BLOCK_COMMAND, *'Thank you.'*

DECLARATION ::= *'Create the variable'*, IDENTIFIER, [*.'*]

| DECLARATION, *','*, DECLARATION

COMMAND ::= COMMAND, *','*, COMMAND

| *'Assign the boolean'*, IDENTIFIER, *' to the value of'*, BOOLEAN

| *'Assign the integer'*, IDENTIFIER, *' to the value of'*, NUMBER

BLOCK_COMMAND ::=

| IF_COMMAND

| WHILE_COMMAND

WHILE_COMMAND ::= *'So long as'*, BOOLEAN, *'please'*, LIST, *'thank you for your iterations'*

IF_COMMAND ::= *'Should it be the case'*, BOOLEAN, *','*, *'please'*, LIST, *'otherwise'*, LIST, *'that is all.'*

BOOLEAN ::= *'TRUE'*

| *'FALSE'*

| EXP, *'EQUALS'*, EXP

| EXP, *'AND'*, EXP

| EXP, *'OR'*, EXP

| *'NOT'*, BOOLEAN

EXP ::= MULT_EXP

| DIV_EXP

| ADD_EXP

| SUB_EXP

| NUMBER

| IDENTIFIER

| BOOLEAN

MULT_EXP ::= EXP, *'*'*, EXP

DIV_EXP ::= EXP, *'/'*, EXP

ADD_EXP ::= EXP, *'+'*, EXP

SUB_EXP ::= EXP, *'-'*, EXP

IDENTIFIER ::= IDENTIFIER LETTER | LETTER

LETTER ::= *'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'*

NUMBER ::= NUMBER DIGIT | DIGIT

DIGIT ::= *'0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'*
