```
// This program evaluates expressions after first converting them from infix to postfix format. It
//supports addition, subtraction, division, multiplication operators and sin, cos, absolute value
//and square root.

INCLUDE Evaluator, Parser, ParseErr, iostream and string headers

int main()
    INIT expression string
    INIT done bool and SET to false
    INIT postfix string
    INIT lhsVar string
    INIT eval Evaluator
    INIT prsr Parser
    INIT answer char
    INIT result double

    PRINT [welcome message and instructions]

    LOOP while NOT done
        PRINT “Enter an expression: “
        INPUT expression

        TRY
            SET postfix TO prsr.infixToPostfix(expression)
            PRINT “Postfix: ” + postfix

            SET result TO eval.evaluate(expression)
            SET lhsVal TO expression.substr(0, ‘=’)  // string method
            PRINT lhsVar + “= ” result
        CATCH (std::runtime_error& re)
            PRINT “Evaluation Error - ” + re.what()
        CATCH (ParseErr& pe)
            PRINT “Parse Error - ” + pe.what()
        END TRY-CATCH BLOCK

        PRINT “Would you like to enter another expression? Enter q to quit or another
        letter to continue: ”
        INPUT answer
        IF answer EQUALS ‘q’ OR answer EQUALS ‘Q’
            SET done TO true
    END LOOP

    PRINT “Ending Expression Evaluator Program”
end main function
```

Scratchwork - Class Lists

ListItem Class
List Class
Stack Class

Evaluator Class
contains>>Parser Class
+ Evaluator()
+ Evaluator(string)
+ Evaluator(const Evaluator&)
+ evaluate(string) : double ?
+ setExpression(string) : void
- Parser tknr (for tokenizing in evaluate method)

Parser Class
contains>>Symbol Table Class

Symbol Table
typedef of>>Hashtable Class
- Only has lowercase version
- Hash actual string KEYWORD (check length!) And store type as data
- hash KEYWORD as KEY then store VALUE as DATA (key-value pair!)
- Retrieve data by looking up key, return data
- Data type string as default, but can convert later into double

Hashtable Class
contains>>Bucket Class

Bucket Class
contains>>Slot Class

Slot Class
Has key and data pair

Expression Evaluator Algorithm

This **Evaluator** class method takes a string in postfix format (ex. y a b c / d * + =) and calculates then returns the result as a double. Assumes symbols are separated by spaces.
Assumes expression is in postfix and symbols are separated by whitespaces.

```
double Evaluator::evaluate(string expression)
    INIT nums as Stack<double>          // temp holds operands, then final result
    INIT token string
    INIT doubles op1 and op2 for operands for calculation
    INIT resultKey string               // for variable which final result will be assigned to
    INIT result double
    INIT firstVarRead bool AND SET TO true
    CALL tknr.setStr(expression)        // to extract tokens from expression

    LOOP while there are still tokens to get from string
        SET token to next token

        IF NOT firstVarRead AND NOT valid identifier name
            THROW runtime_error("invalid identifier on LHS")
        ELSE IF token is a number constant (check for negative too)
            CONVERT token to double
            CALL nums.push(token)
        ELSE IF token is a valid variable name AND not a unary operator
            IF NOT firstVarRead
                SET resultKey TO token
                SET firstVarRead TO true
            ELSE IF variable is predefined          // get value and push to stack
                GET value matching key from SymbolTable
                CONVERT token to double
                CALL nums.push(token)
            ELSE
                THROW runtime_error("undefined identifier on RHS")
            ENDIF
        ELSE IF binary operator
            SET op2 TO nums.pop()
            SET op1 TO nums.pop()

            // perform calculation based on operator, push result back to stack
            IF token EQUAL TO "*"
                CALL nums.push(op1 * op2)
            ELSE IF token EQUAL TO "/"
                CALL nums.push(op1 / op2)
            ELSE IF token EQUAL TO "+"
                CALL nums.push(op1 + op2)
            ELSE IF token EQUAL TO "-"
                CALL **nums.push**(op1 - op2)
        ELSE IF unary operator (sin, cos, sqrt or abs)          // only needs one operand
            SET op1 TO **nums.pop()**

            IF **lowercase**(token) EQUAL TO “sin”
                CALL **nums.push**(**sin**(token))              // uses math.h functions
            ELSE IF **lowercase**(token) EQUAL TO “cos”
                CALL **nums.push**(**cos**(token))
            ELSE IF **lowercase**(token) EQUAL TO “sqrt”
                CALL **nums.push**(**sqrt**(token))
            ELSE IF **lowercase**(token) EQUAL TO “abs”
                CALL **nums.push**(**abs**(token))
        ELSE IF token EQUAL TO “=”          // final and only item in stack is result
            SET result to **nums.pop()**
            INSERT resultKey (key) and result (data) into SymbolTable
        ENDIF
    END LOOP

    RETURN result
end evaluate method
```

Infix to Postfix Conversion Algorithm

// This method takes an expression in infix form and returns it as a postfix.

```
string Evaluator::infixToPostfix(string expression)
    INIT postfix string        // to hold final expression
    INIT token string
    INIT action ParseAction as action code corresponding to token read
    INIT compareAgain bool and SET to false   (for action U1, compare token again)
    CALL tknr.setStr(expression)

    LOOP while there are still tokens to read
        IF NOT compareAgain
            SET token to tknr.getNextToken()
        SET action to getAction(token, s2.showTop(), s2.isEmpty())
        SET compareAgain TO false

        IF action EQUALS ParseAction::S1
            DoS1(token)
        ELSE IF action EQUALS ParseAction::S2
            DoS2(token)
        ELSE IF action EQUALS ParseAction::ERR
            THROW ParseErr()
        ELSE IF action EQUALS ParseAction::UC
            DoUC()
        ELSE IF action EQUALS ParseAction::U1
            DoU1()
            SET compareAgain TO true
        ELSE IF action EQUALS ParseAction::U2
            DoU2()
        ENDIF
    ENDLOOP

    // unstack s2 to s1 until s2 is empty
    LOOP while NOT s2.isEmpty()
        s1.enqueue(s2.pop())
    ENDLOOP

    // then pop contents to get postfix in correct order
    LOOP while (!s1.isEmpty())
        postfix += " " + s1.dequeue();
    END LOOP

    RETURN postfix
end of infixToPostfix method
```

Determining Parse Action Algorithm

This helper function helps to convert an infix to a postfix expression (to be used in **infixToPostfix()**). It returns a code corresponding to an action based on the current state of member Stack s2. Arguments passed include a token from **evaluate()**, the top of stack S2 (operators) and a bool determining if S2 is empty. **ParseAction** is an enum in **Evaluator** class.

```
ParseAction Evaluator::getAction(string token, string stackTop, bool stackIsEmpty)
    INIT ParseAction nextAction           // to return action code

    IF token is a unary operator
        IF stackIsEmpty is true
           SET nextAction to ParseAction::ERR
        ELSE
           SET nextAction = ParseAction::S2
    ELSE IF token is operand (identifier) OR numeric constant
        SET nextAction TO ParseAction::S1
    ELSE IF token EQUALS “=”
        IF stackIsEmpty is true
            SET nextAction TO ParseAction::S2
        ELSE
            SET nextAction TO ParseAction::ERR
    ELSE IF token EQUALS “+” OR token EQUALS “-”
        IF stackIsEmpty is true
            SET nextAction to ParseAction::ERR
        ELSE IF stackTop EQUALS "=" OR stackTop EQUALS "("
            SET nextAction to ParseAction::S2
        ELSE IF stackTop EQUALS "+" OR stackTop EQUALS "-"
              OR stackTop EQUALS "*" OR stackTop EQUALS "/"
              OR isUnaryOp(stackTop)
            // have to do another comparison
            SET nextAction to ParseAction::U1
    ELSE IF token EQUALS  "*" OR token EQUALS "/"
        IF stackIsEmpty
            SET nextAction to ParseAction::ERR
        ELSE IF stackTop EQUALS "=" OR stackTop EQUALS "+"
              OR stackTop EQUALS "-" OR stackTop EQUALS "("
            SET nextAction to ParseAction::S2
        ELSE IF stackTop EQUALS "*" OR stackTop EQUALS "/"
            // have to do another comparison
            SET nextAction to ParseAction::U1
    ELSE IF token EQUALS "("
        IF stackIsEmpty is true
            SET nextAction to ParseAction::ERR
        else
```

```
            SET nextAction to ParseAction::S2
    ELSE IF token EQUALS ")"
        IF stackIsEmpty is true OR stackTop EQUALS "="
            SET nextAction TO ParseAction::ERR
        ELSE
            SET nextAction TO ParseAction::UC
    ELSE IF token EQUALS "\0" OR token EQUALS ""
        SET nextAction to ParseAction::U2
    ENDIF

    RETURN nextAction
end of getAction method
```