Data Structures Project 2

Jakub Podmokly
Nhi Pham Le Yen
Lab Section: Wednesday 1:15pm - 3:15pm

April 2019

1 Project

In this project, we design the program mycalc to calculate the value of the expressions given in the input file. We create two classes:

- **class El**: stores type (variable, digit, unary operator, binary operator, parenthesis), value, and symbol.
- **class Expression**: stores the expression, boolean solved to keep track of solvable expressions, a vector of El *postfix* to store the postfix expression, vector of El *arr* to store the splitted expressions.

When we scan the input file, if the input line of expression does not end with **";"**, we consider the expression **invalid**.
The functions we use to evaluate and output the expression - value are:

- **bool tokenize**: split the given expression based on different types (variable, digit, unary operator, binary operator, parenthesis), and then store in the vector *arr*. The function also return true/false to indicate if the expression is **valid** or **invalid** (in this case, we check if the expression contains some characters that do not fall into 5 types, such as special characters, and check for invalid numbers such as 12.34.5).
- **checkValid**: If we can tokenize the expression, we move on to check the number of open and closed parentheses, check if the operators are put in the right positions, and return the boolean value into the bool *valid* of the class Expression.

- **infixToPostfix**: After we tokenize and checkValid of the expression, we convert the given infix expression to postfix expression and store it into the vector postfix. In this function, we use the algorithm presented in the lab to convert infix to postfix.

- **evaluate**: From the postfix expression, we design the function to calculate the value of the expression and store the value, if possible to solve, into the vector *exList* and the map *expressionMap* (of string and double).

In general, the following cases would be considered ***INVALID***:

- The expression does not end with **";"**.

- Invalid identifiers, such as 1var or containing special characters. Variables should starts from a char/ (underscore) followed by char/digit/ (underscore).

- Invalid numbers, such as 12.34.5

- Invalid operator combination, including:

  - Binary operator:

    * Is preceded by a binary operator or unary operator or an opening parenthesis.
    * Is followed by a binary operator or a closing parenthesis.
    * Is at the position 0 or expression.length-1.

  - Unary operator:

    * Is preceded by a digit or variable or a closing parenthesis or a unary operator.
    * Is followed by a binary or unary operator or a closing parenthesis.
    * Is at the position expression.length-1.

- Invalid paretheses: invalid if the number of left parentheses is different from the number of right parentheses.

2 Algorithm

In this project, we use the algorithm to convert infix to postfix as presented below:

- Initialize the stack of El, call operators. We will store the postfix expression in the vector *postfix*.
- If a left parenthesis, store in the stack.
- If a right parenthesis, pop from the stack and store in the vector *postfix* until we meet the left parenthesis. We pop the left parenthesis and do not store the left parenthesis in the vector *postfix*.
- If an operator, we pop from the stack and store in the vector *postfix* if the stack.top() is the operator that has higher or equivalent precedence.
- In the end, if the stack is not empty, we repeatedly pop and store in the vector *postfix*.

We also use the algorithm to evaluate the postfix and get the value if the expression is solvable.

- Initialize a stack.
- If a number, push it into the stack.
- If a solvable variable, push it into the stack. Else if a not solvable variable, stop the function.
- If an operator, pop 2 elements from the stack. Depending on which type of operator, we apply proper calculations and calculate the value and push the value into the stack again.
- If scan throughout the expression successfully, return the result.