```
// Subject Code: COMP20007
// Assignment 1
// Name: Edward Marozzi
// Student ID: 910193


// Problem 1) String calculator in pseudocode

// Useful set up functions
function isOperator(c)

    if c is '*' or c is '/' or c is '+' or c is '-' do
        return true
    return false

// Maths priority
function getPriority(c) do

    if c is '-' or c is '+' do
        return 1
    else if c is '*' or c is '/' do
        return 2
    return 0

// Error handling
function checkInitErrors(infix)

    bracketsBal <- 0
    totalBrackets <- 0
    totalOperators <- 0

    // No string given
    if infix.length() is 0 do
        return "NotWellFormed"

    // If steing is only one char must be a digit
    if infix.length() is 1 and !isDigit(infix[0]) do
        return "NotWellFormed"

    // If it is one char its already in the right form
    if infix.length() is 1 do
        return infix

    // Must open with bracket if string is more than one char
    if infix.length() is greater than 1 and infix[0] is not '(' do
        return "NotWellFormed"

    // Can't be three chars long and valid e.g "(8)" is not valid
    if infix.length() is 3 do
        return "NotWellFormed"

    // Count brackets
    for i <- 0 to i is less than infix.length() do i++ and
```

```
        if infix[i] is '(' do
            bracketsBal++
            totalBrackets++
        else if infix[i] is ')' do
            bracketsBal--
            totalBrackets++

        if isOperator(infix[i]) do
            totalOperators++

    // Check for balanced brackets
    if bracketsBal is not 0 do
        return "NotWellFormed"

    // Ensure every operation is wrapped in the correct amount of brackets
    if totalOperators*2 is not totalBrackets do
        return "NotWellFormed"

    return "NoInitialErrors"



// Evaluates the infix string given and returns the answer
function evaluatePrefix(prefix)

    Stack <- createStack()

    for j <- prefix.size() - 1 to j greater than or equal to 0 do j-- and

        if isDigit(prefix[j]) do
            Stack.push(prefix[j] - '0')
        else do

            // Operator encountered to be operatored on by operator
            operand1 <- Stack.top()
            Stack.pop()
            operand2 <- Stack.top()
            Stack.pop()

            // Determine operator and perform operation
            if prefix[j] is '+' do
                Stack.push(operand1 + operand2)
            else if prefix[j] is '-' do
                Stack.push(operand1 - operand2)
            else if prefix[j] is '*' do
                Stack.push(operand1 * operand2)
            else if prefix[j] is '/' do
                Stack.push(operand1 / operand2)

    return Stack.top()


// Function that converts infix to prefix for calculation
function infixToPrefix(infix)
```

```
    // Checks some of the validity rules for the infix string before
    //  begining parsing
    initErrors = checkInitErrors(infix)

    if initErrors is "NotWellFormed" do
        return initErrors

    // stack for operators
    operators <- createStack()

    // stack for operands
    operands <- createStack()

    // Go through every character in infix string and perform actions
based
    //  on the character.
    for i is 0 to i is less than infix.length() do i++ and

        // If the character is an opening bracket, then we push into the
        // operators stack.
        if infix[i] is '(' do
            // Perform check to ensure that string doesn't end in a '('
            //  or empty bracket or '( followed by operator
            if i is infix.length() - 1 or infix[i+1] is ')' or
                isOperator(infix[i+1]) do

                return "NotWellFormed"

            operators.push(infix[i])
        // If current character is a closing bracket, then pop from both
        // stacks and push result in operands stack until matching opening
        // bracket is found.
        else if infix[i] is ')' do
            // Checking that the next character is valid
            if infix[i+1] is '(' or isOperand(infix[i+1]) do
                return "NotWellFormed"

            while operators.top() is not '(' do
                // operand 1
                string operand1 <- operands.top()
                operands.pop()

                // operand 2
                string operand2 <- operands.top()
                operands.pop()

                // operator
                char op <- operators.top()
                operators.pop()

                // Add operands and operator in order
                string tmp <- op + operand2 + operand1
                operands.push(tmp)
```

```
            // Pop opening bracket from stack.
            operators.pop()
        // If current character is an operand then push it into  the
        //      operands
        else if isOperand(infix[i]) do
            // Check validity of next character again
            if i is infix.length() - 1 or isOperand(infix[i+1]) do
                return "NotWellFormed"

            operands.push(string(1, infix[i]))


        // Charcter must be an operator so push it into operators
        //  after removing higher priority operators then pushing result
in
        //  operands stack.
        else
            // Validity check
            if i is infix.length() - 1 or isOperator(infix[i+1]) or
infix[i+1] is ')' do
                return "NotWellFormed"

            while not operators.empty() and
                    getPriority(infix[i]) is less than or equal to
                    getPriority(operators.top()) do

                operand1 <- operands.top()
                operands.pop()

                operand2 <- operands.top() do
                operands.pop()

                op <- operators.top()
                operators.pop()

                tmp <- op + operand2 + operand1
                operands.push(tmp)


            operators.push(infix[i])

    // Remove operators from operators stack until it is empty and add
    //  result of each pop operation in operands stack.
    while not operators.empty()
        operand1 <- operands.top()
        operands.pop()

        operand2 <- operands.top()
        operands.pop()

        op <- operators.top()
        operators.pop()

        tmp <- op + operand2 + operand1
```

```
        operands.push(tmp)

    // Now operands is just left with final prefix expression
    return operands.top()


// Driver code
function main do
    // Get input infix string
    inputString <- getUserInput()
    prefixString <- infixToPrefix(s)
    // Evaluate as long as its valid
    if prefix is not "NotWellFormed" do
        output(evaluatePrefix(infixToPrefix(s)))
```