Department of Computer Science & Engineering

Course Title – Compiler Design Lab

Course Code – CSE 430

Section – A1

Project

A calculator which can parse an arithmetic expressions and calculate the result of the input expressions

Submitted by:

Md. Hasibur Rahman (19101009)
Tanmoy Mazumder (19101013)
Shawan Das (19101020)

Submitted To:

Baivab Das, Lecturer
Department of Computer Science and Engineering,
University of Asia Pacific

Objective

Create a calculator which will be able to parse an arithmetic expression and calculate the result of the input expressions. We have to build a parser which will be able to parse certain types of grammar.

Project Description

The primary objective of this project is to create a calculator that can parse an arithmetic expression and compute the input expression's outcome. Essentially,it will be able to solve arithmetic operations such as summation, subtraction, multiplication, division and square-root. Our project is also able to operate parenthesis.

Also our project is able to resolve some operational problems such as missing operator between ending-starting parenthesis or number-starting parentheses. for example,

for a equation: 2(3+2) or (1+2)4, the solved equation will be 2*(3+2) and (1+2)*4. also the equation (2+4)(3-6) will be taken as (2+4)*(3-6).

Therefore, our calculator would be able to parse the arithmetic expressions which a real calculator can. We've used a recursive descent parser for the parsing part.

Tools & Languages

- Google Colab
- VS Code
- Python

Code:

- Basic Operation

```
def operate(data):
    if len(data) == 1:
        return data[0]

    # Root Operation
    while('sqrt' in data):  # Checking of Root operation and validation
        temp_data = []
        i = 0
        while(i < len(data)):
            if data[i] == 'sqrt':
                try:
                    temp_data.append(math.sqrt(data[i+1]))
                    i += 2
                except:  # Negative Root operation
                    print("sqrt ERROR")
                    return 'Math Error'
            else:
                temp_data.append(data[i])
                i += 1
        data = temp_data
        # print(data)
    # Multiplication operation:
    while('*' in data or '/' in data):
        temp_data = []
        for i in range(len(data)):  # Single operation and recheck for Multi and Div
            if data[i] == '*':
                temp_data.pop()
                temp_data.append(data[i-1]*data[i+1])  # Multiplication
                i += 1
                break
            elif data[i] == '/':
                temp_data.pop()
                temp_data.append(data[i-1]/data[i+1])  # Division
                i += 1
                break
            else:
                temp_data.append(data[i])
        data = temp_data + data[i+1:]  # update main data after iteration
        # print(data)
    ### Addition / Subtraction
    sum = 0
    temp = []
    i = 0
    while(i < len(data)):
        if data[i] == '-':
            temp.append(-1*data[i+1])
            i += 2
        elif data[i] == '+':
            temp.append(data[i+1])
            i += 2
        else:
            temp.append(data[i])
            i += 1
    data = temp
    #print("Check: ",data)
    for i in data:
        sum += i
    return sum
```

- Bracket Handle

```python
def manage_bracket(dt):
    while ('(' in dt):  # Brackets are in data
        cap = 0  # Check for Active parenthesis
        temp = []  # Temporary data to store
        stack = []  # Store '( )' data
        for i in dt:
            if i == '(' and cap == 1:  # Create new stack and insert previous stack to temp
                temp = temp+stack
                stack = []
                stack.append(i)   # New Stack
            # print('1')
            elif i == '(' and cap == 0:  # No previous Stack, Create new
                stack.append(i)
                cap = 1
                # print('2')
            elif i == ')' and cap == 1:  # Stack= Active, stop Stacking
                cap = 0
                stack.pop(0)        # Remove '(' from stack
                print('operate: ', stack)
                # stack operation and replce main data
                temp.append(operate(stack))
                stack = []           # empty Stack
                # print('3')
            # Stack= inactive, normal insert data to temp
            elif cap == 0 and i != '(':
                temp.append(i)
            # print('4')
            elif cap == 1:  # Stack=Active, Insert data to Stack
                stack.append(i)
                # print('5')
            elif cap == 0 and i == ')':  # Stack= Inactive, ')'- direct insert
                temp.append(i)
        # replace main data with temp data and check for more '(' ')'
        dt = temp
        print('Updated Stack: ', dt)
    return operate(dt)  # all bracket removed, Simple operation
```

- Error Check

```python
def error_check(check):
    # Bracket Error
    flag = 0
    for i in check:  # Checking of missing Parenthesis
        if i == '(':
            flag += 1
        if i == ')':
            flag -= 1
        if flag < 0:     # ')' came before '('
            return "Incorrect Formation"
    if flag != 0:      # incorrect brackets
        return 'incomplete ( or )'

    # invalid Operation
    if check[0] == '+' or check[0] == '*' or check[0] == '/':
        return "Invalid Operation | Equation can't start with +,/,*"

    # Unwanted operation
    if check[-1] in operation:
        return "Invalid Operation | Equation can't end with any operant"
    # SQRT operation Check:
    for i in range(len(check)):
        if check[i] == 'sqrt' and check[i+2] == '-':  # Negative sqrt
            return "Negative Square Root Error"
        if check[i] == 'sqrt' and check[i+1] != '(':
            # Missing of sqrt parenthesis
            return "Undefined sqrt, Suggetion: sqrt(equation)"
    # print(check)
    # Arithmetic operation check
    for i in range(len(check)):
        if check[i] in operation[:4] and check[i+1] in operation[:4]:  # Operational Error
            return "Syntax Error "
    print(check)
    return manage_bracket(check)  # good to go for operation
```

- Clean Data

```
def clean_data(value):
    # Remove Extra Spaces
    value = re.sub('sqrt', '$', value, flags=re.IGNORECASE)
    val = ''
    for i in value:
        if i != ' ':
            val += i
    value = val

    # Create list of data
    string = ''
    temp = []
    stack = []
    for i in value:
        if (i in operation or i in priority) and string != '':  # Manage multi digit numbers
            if '.' in string:
                temp.append(float(string))  # Floating Numbers
                string = ''
            else:
                temp.append(int(string))  # Integer Numbers
                string = ''
            temp.append(i)
        elif i in numbers:
            string += i
        elif i == '$':
            temp.append('sqrt')
        else:
            temp.append(i)

    if string != '':  # insert remaining data
        if '.' in string:
            temp.append(float(string))
        else:
            temp.append(int(string))
    stack = temp
    # Manage "no operation" error between ')' and '('
    temp_data = []
    for i in range(len(stack)-1):
        if stack[i] == '(' and stack[i-1] not in operation and stack[i-1] != '(':
            temp_data.append('*')
            temp_data.append(stack[i])
        elif i != 0 and stack[i-1] == ')' and stack[i] not in operation and stack[i] != ')':
            temp_data.append('*')
            temp_data.append(stack[i])
        elif stack[i] == ')' and stack[i+1] == '(':
            temp_data.append(stack[i])
            temp_data.append('*')
        else:
            temp_data.append(stack[i])
    temp_data.append(stack[-1])
    stack = temp_data
    return error_check(stack)
```

Result

Here are some sample answers that have been tested as output by our calculator.

```
D:\Documents\4-2\430 compiler lab\codes>python -u "d:\Documents\4-2\430 compiler lab\codes\FinalProject\calc.py"
_____________________________________________

Test 1: -2.5+( 2 * 10 / 5 + ( sqrt(20+16) / (3*2) + 10)* 2 * 3 / 6)

['-', 2.5, '+', '(', 2, '*', 10, '/', 5, '+', '(', 'sqrt', '(', 20, '+', 16, ')', '/', '(', 3, '*', 2, ')', '+', 10, ')', '*', 2, '*', 3, '/', 6, ')']
operate:  [20, '+', 16]
operate:  [3, '*', 2]
Updated Stack:  ['-', 2.5, '+', '(', 2, '*', 10, '/', 5, '+', '(', 'sqrt', 36, '/', 6, '+', 10, ')', '*', 2, '*', 3, '/', 6, ')']
operate:  ['sqrt', 36, '/', 6, '+', 10]
Updated Stack:  ['-', 2.5, '+', '(', 2, '*', 10, '/', 5, '+', 11.0, '*', 2, '*', 3, '/', 6, ')']
operate:  [2, '*', 10, '/', 5, '+', 11.0, '*', 2, '*', 3, '/', 6]
Updated Stack:  ['-', 2.5, '+', 15.0]
=  12.5
_____________________________________________

_____________________________________________

Test 2: 18* 32 - 69 * 13 /5 - (8+9/2)

[18, '*', 32, '-', 69, '*', 13, '/', 5, '-', '(', 8, '+', 9, '/', 2, ')']
operate:  [8, '+', 9, '/', 2]
Updated Stack:  [18, '*', 32, '-', 69, '*', 13, '/', 5, '-', 12.5]
=  384.1
_____________________________________________
```

```
_____________________________________________

Test 3: 18* 32 - 69 * 13 /5 - 2(8+9/2)

[18, '*', 32, '-', 69, '*', 13, '/', 5, '-', 2, '*', '(', 8, '+', 9, '/', 2, ')']
operate:  [8, '+', 9, '/', 2]
Updated Stack:  [18, '*', 32, '-', 69, '*', 13, '/', 5, '-', 2, '*', 12.5]
=  371.6
_____________________________________________

_____________________________________________

Test 4: 2(5+5)3+3

[2, '*', '(', 5, '+', 5, ')', '*', 3, '+', 3]
operate:  [5, '+', 5]
Updated Stack:  [2, '*', 10, '*', 3, '+', 3]
=  63
_____________________________________________

_____________________________________________

Test 5: 22(3-3/(sqrt(55)*(3*10(5-6(7+2)))))

operate:  [55]
operate:  [7, '+', 2]
Updated Stack:  [22, '*', '(', 3, '-', 3, '/', '(', 'sqrt', 55, '*', '(', 3, '*', 10, '*', '(', 5, '-', 6, '*', 9, ')', ')', ')', ')']
operate:  [5, '-', 6, '*', 9]
Updated Stack:  [22, '*', '(', 3, '-', 3, '/', '(', 'sqrt', 55, '*', '(', 3, '*', 10, '*', -49, ')', ')', ')']
operate:  [3, '*', 10, '*', -49]
Updated Stack:  [22, '*', '(', 3, '-', 3, '/', '(', 'sqrt', 55, '*', -1470, ')', ')']
operate:  ['sqrt', 55, '*', -1470]
Updated Stack:  [22, '*', '(', 3, '-', 3, '/', -10901.811776030625, ')']
operate:  [3, '-', 3, '/', -10901.811776030625]
Updated Stack:  [22, '*', 3.000275183617332]
=  66.0060540395813
_____________________________________________
```

```
_____________________________________________

Test 6: 18* 32 - 69 * 13 /5 - 2(8+9/2))

=  Incorrect Formation
_____________________________________________

_____________________________________________

Test 7: sqrt(3-5)

['sqrt', '(', 3, '-', 5, ')']
operate:  [3, '-', 5]
Updated Stack:  ['sqrt', -2]
sqrt ERROR
=  Math Error
_____________________________________________
```

Challenges

We faced several challenges during our compiler project, including managing brackets for operation, finding and solving possible errors, and managing multi-digit float or integer type data. Specifically, we had to ensure that the compiler correctly identified and evaluated expressions containing brackets in the correct order. Additionally, we had to identify and report errors in the code, such as syntax errors, math errors etc. Finally, we had to correctly identify and parse numeric data types, such as integers and floating-point numbers, that may contain multiple digits or decimal points.