Source Code:

```
/*
You are tasked with implementing a program that can evaluate mathematical
expressions
entered by the user. Your program should use a stack-based solution to
correctly handle the
order of operations (parentheses, multiplication, division, addition, and
subtraction).

This code was written by: Kyla Ronquillo for DSA
 ---- I used Linked Lists to implement stacks.
References:
    Lecture #04 - Stacks PPT
    Supplemental Materials Stacks
    Digital Ocean - Understanding Order of Operations in Programming
*/

#include <iostream>
#include <string>
#include <cctype>
#include <cstdlib>
#include <sstream>

using namespace std;

// node structure for the linked list
struct Node {
    double data;
    Node* next;
};

Node* operatorExp = nullptr;
Node* operandExp = nullptr;

// function to check if stack is empty
bool isStackEmpty(Node* topStack) {
    return topStack == nullptr;
}

// function to push a new stuff onto a stack
```

```
void push(Node* &top, double digit) {
    Node* newNode = new Node;
    newNode->data = digit;
    newNode->next = top;
    top = newNode;
}

// function to peek at the top element of the stack
double peek(Node* top) {
    if (!isStackEmpty(top)) {
        return top->data;
    }
    else {
        exit(1);
    }
}

// function to pop the top element from the stack
double pop(Node*& top) {
    if (!isStackEmpty(top)) {
        double item = top->data;
        Node* temp = top;
        top = top->next;
        delete temp;
        return item;
    }
    else {
        exit(1);
    }
}

// function to check precedence of operations
int precedence(char c) {
    if (c == '*' || c == '/') {
        return 2;
    }
    else if (c == '+' || c == '-') {
        return 1;
    }
    return 0;
```

```
}

// function to check if parentheses are balanced in the expression
bool isBalanced(const string& input){
    int count1 = 0, count2 = 0;

    for (char ch : input) {
        if (ch == '(') count1++;
        else if (ch == ')') count2++;
    }

    if (count1 == count2) return true;
    else return false;
}

// function to check if the expression contains valid characters
bool isValidExpression(const string& input) {
    for (char c : input) {
        if (isspace(c)) {
            continue;
        }
        if (!isdigit(c) && c != '+' && c != '-' && c != '*' && c != '/' &&
c != '(' && c != ')' && c != '.') {
            return false;
        }
    }
    return true;
}

// function that checks if input is empty or not
bool isExpEmpty(const string& input){
    return input.empty();
}

// function that checks the operator used by the user for the expression
void ansMath(char arithmeticOp, double operand1, double operand2){
    if (arithmeticOp == '+') {
        push(operandExp, operand1 + operand2);
    }
    else if (arithmeticOp == '-') {
```

```
        push(operandExp, operand1 - operand2);
    }
    else if (arithmeticOp == '*') {
        push(operandExp, operand1 * operand2);
    }
    else if (arithmeticOp == '/') {
        if (operand2 == 0) { //checks if the divisor / operand2 is  0
            cerr << "ERROR: Undefined. Cannot divide by zero.\n";
            exit(1);
        }
        push(operandExp, operand1 / operand2);
    }
}

// function to evaluate a mathematical expression
double evaluateMathExpression(const string& input) {
    stringstream onebyone(input);
    bool inNumber = false;
    double currentNumber = 0.0;
    bool isNegative = false;
    char c;

    while (onebyone >> c) {
        if (isspace(c)) { // checks if a character of the input string is
a space
            continue;
        }
        else if (isdigit(c) || c == '.') { // checks if a character of the
input string is a digit
            inNumber = true;
            if (isdigit(c)) {
                onebyone.putback(c);
                onebyone >> currentNumber;
            } else { // it's a decimal point
                // Read the fractional part
                double fraction = 0.1;
                while (isdigit(onebyone.peek())) {
                    onebyone >> c;
                    currentNumber += (c - '0') * fraction;
                    fraction *= 0.1;
```

```
                }
            }
            if (isNegative) {
                currentNumber = -currentNumber;
                isNegative = false;
            }
            push(operandExp, currentNumber);
            inNumber = false;
            currentNumber = 0.0;
        }
        else { // if character is not a space or digit
            if (c == '(') {
                push(operatorExp, c);
            }
            else if (c == ')') { // means end of parentheses expression,
time to evaluate this first!
                while (!isStackEmpty(operatorExp) && peek(operatorExp) !=
'(') {
                    char arithmeticOp = pop(operatorExp);
                    double operand2 = pop(operandExp);
                    double operand1 = pop(operandExp);
                    ansMath(arithmeticOp, operand1, operand2);
                }
                if (!isStackEmpty(operatorExp) && peek(operatorExp) ==
'(') {
                    pop(operatorExp);
                }
            }
            else if (c == '-' && isStackEmpty(operandExp)){ // for numbers
who have negative signs at the beginning
                isNegative = true;
            }
            else if (c == '+' || c == '-' || c == '*' || c == '/') {
                if (c == '-' && isStackEmpty(operatorExp) &&
isStackEmpty(operandExp)){
                    cerr << "ERROR: Invalid Math Expression.\n";
                    exit(1);
                }
                while (!isStackEmpty(operatorExp) && peek(operatorExp) !=
'(' && precedence(c) <= precedence(peek(operatorExp))) {
```

```
                    char arithmeticOp = pop(operatorExp);
                    double operand2 = pop(operandExp);
                    double operand1 = pop(operandExp);
                    ansMath(arithmeticOp, operand1, operand2);
                }
                push(operatorExp, c);
            }
            else {
                cerr << "ERROR: Invalid expression. Please try again.\n";
                exit(1);
            }
        }
    }

    // process any remaining numbers and operators
    if (inNumber) {
        push(operandExp, currentNumber);
    }
    while (!isStackEmpty(operatorExp)) { // this part handles the
remaining operators and values in the stacks
        char arithmeticOp = pop(operatorExp);
        double operand2 = pop(operandExp);
        double operand1 = pop(operandExp);
        ansMath(arithmeticOp, operand1, operand2);
    }

    if (isStackEmpty(operandExp)) {
        cerr << "ERROR: Your math expression is invalid.\n"; //requires
this statement in case of errors
        exit(1);
    }
    else {
        return pop(operandExp); //return the answer
    }
}
int main() {
    string mathExpression;

    do {
        cout << "\nEnter a mathematical expression: ";
```

```
        getline(cin, mathExpression);

        bool balanced = isBalanced(mathExpression); // if parentheses are
balanced
        bool valid = isValidExpression(mathExpression); // if there are
wrong characters
        bool empty = isExpEmpty(mathExpression); // if the input string is
empty

        if (!balanced) {
            cerr << "ERROR: The parentheses are not balanced. Please try
again.\n";
        }
        if (!valid) {
            cerr << "ERROR: Invalid character in the math expression.
Please try again.\n";
        }
        if (empty) {
            cerr << "ERROR: You did not input a mathematical expression.
Please try again.\n";
        }
        if (balanced && valid && !empty) {
            double answerExp = evaluateMathExpression(mathExpression);
            cout << "Result: " << answerExp << endl;
        }

    } while (!isBalanced(mathExpression) ||
!isValidExpression(mathExpression) || isExpEmpty(mathExpression));

    return 0;
}
```

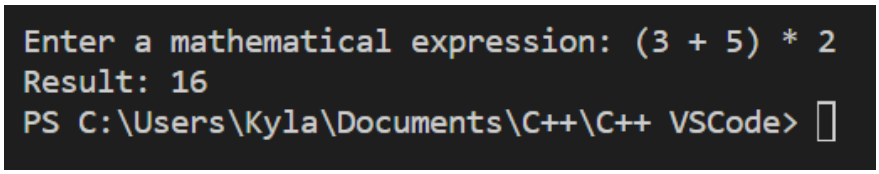
Brief Explanation and Challenges:

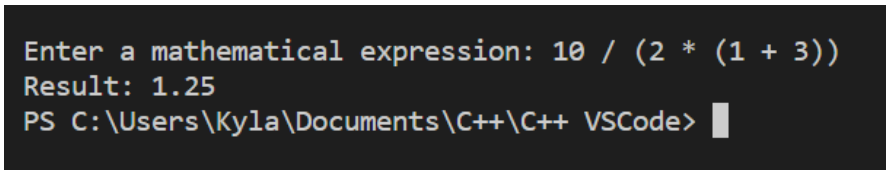
My approach was to use two stacks and implement it using linked lists. I ironically found it a bit difficult to visualize and implement an array probably because 1) linked lists were the one used in the PPT file, (2) I do not understand how pop() will work in arrays, and 3() I found linked lists more interesting.

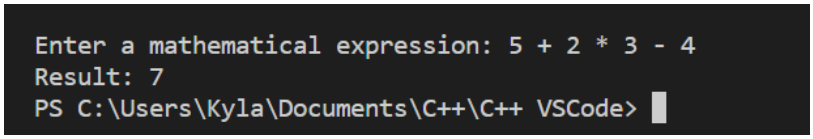
Now, as for using stacks, it was a bit difficult for me to track whether my if-else statements were making sense. I had to detail it down first on a piece of paper and draw an imaginary stack in order for me to understand what I was supposed to do. I also got confused sometimes because I used operandExp and operatorExp as the variable names of pointers to implement lists and it would often confuse me because of how similar they sound.

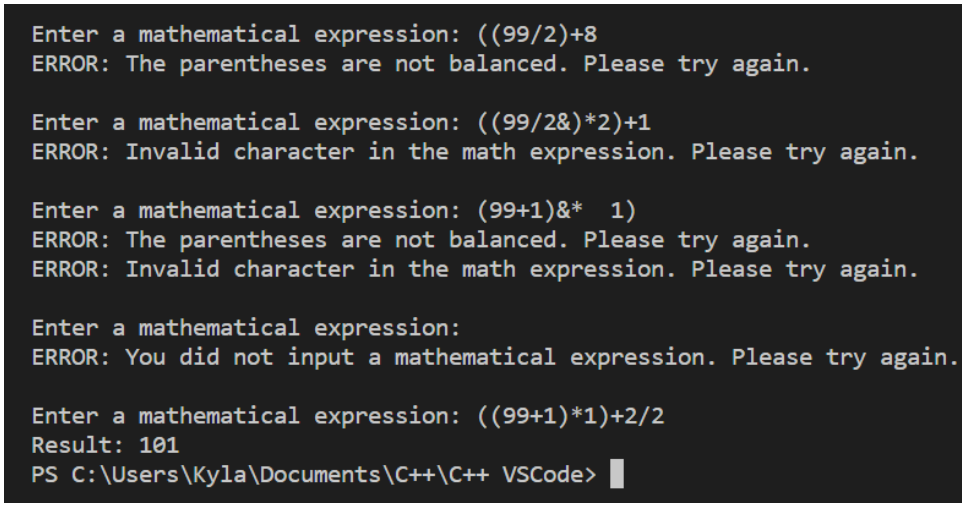
It also took me four submission before settling to what version of my code I am really going to submit. I think in coding, I need very precise instruction to what is needed and what is not, which, I believe is not really that much of a good thing. I conclude that aside fron the aforementioned, I need to improve more on error handling and spotting what things I might have missed.

Sample Runs:









```
Enter a mathematical expression: 99/0
ERROR: Undefined. Cannot divide by zero.
PS C:\Users\Kyla\Documents\C++\C++ VSCode>
```

```
Enter a mathematical expression: 99*1 * 2/3 +1/2
Result: 66.5
PS C:\Users\Kyla\Documents\C++\C++ VSCode>
```

```
Enter a mathematical expression: (-9+2)*2
Result: -14
PS C:\Users\Kyla\Documents\C++\C++ VSCode>
```

```
Enter a mathematical expression: (-10.5+30)* 5
Result: 97.5
PS C:\Users\Kyla\Documents\C++\C++ VSCode>
```