1. **Problem Statement**

Create a program that takes an infix equation as input, the processes it into an expression tree and prints the traversals in prefix, infix, and postfix.

1. **Requirements**
   1. **Assumptions**

* A minimum of one space will be on each side of a character in the file
* Only single uppercase characters are used as operands
  1. **Specifications**
* Print a welcome message when the program starts
* Must read inputs from a file
  + Echo all inputs back to the user
  + User should enter the file name
    - Verify that the file exists
    - Verify the file is of the correct type (.dat file extension)
  + Check that the file is in the correct format
    - Each line should contain only one expression
    - Characters should have one space between them
* Convert the expression to postfix and print the operations that would be performed to solve the expression
* Build an expression tree of the equation
  + Traverse the tree and print the results in infix, postfix, and prefix
  + Evaluate the expression using prefix notation
    - Print each step of the evaluation and the corresponding result
  + Print the tree structure to the screen
* Expression variable values
  + A = 1, B = 2, C = 3 …
* Print all errors and outputs to a file called output.dat
  + Verify that parentheses match up
  + Check for 2 operators for each operand
* Print a message stating that program has completed

1. **Decomposition Diagram**

Main

Input

Print all results and errors to the screen and to a file

Evaluate the expression using prefix notation

Checks the file for errors

Reads the expressions from the input file

Converts each expression to postfix and traverse it through an expression tree

User enters the filename for the input file

Output

Process

1. **Test Strategy**

* Valid
  + Test valid expressions and formats that should NOT throw any errors.
* Invalid
  + Test invalid expressions and formats that SHOULD throw errors.
* File Handling
  + Test file functions

1. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1.1 | Test expression with 1 operator and no parentheses or operands |  |  |  |  |
| Valid | 1.2 | Test expression with 2 values, + operator, no parentheses |  |  |  |  |
| Valid | 1.3 | Test expression with 2 values, - operator, no parentheses |  |  |  |  |
| Valid | 1.4 | Test expression with 2 values, \* operator, no parentheses |  |  |  |  |
| Valid | 1.5 | Test expression with 2 values, / operator, no parentheses |  |  |  |  |
| Valid | 1.6 | Test expression with 2 values, 1 operator, all enclosed in parentheses |  |  |  |  |
| Valid | 1.7 | Test expression with 3 values, operation with 2 values enclosed in parentheses, first operation added to 3rd operator |  |  |  |  |
| Valid | 1.8 | Test expression with 3 values, operation with 2 values enclosed in parentheses, first operation divided by 3rd operator |  |  |  |  |
| Valid | 1.9 | Test expression with 3 values, operation with 2 values enclosed in parentheses, first operation multiplied by 3rd operator |  |  |  |  |
| Valid | 1.10 | Test expression with 3 values, operation with 2 values enclosed in parentheses, first operation subtracted from 3rd operator |  |  |  |  |
| Valid | 1.11 | Test expression with a combination of operations inside and outside of the parentheses |  |  |  |  |
| Valid | 1.12 | Test order of operations at the same level: \*, /. +, - |  |  |  |  |
| Valid | 1.13 | Test order of operations at the same level: +, -, \*, / |  |  |  |  |
| Valid | 1.14 | Test order of operations at the same level: \*, +, /, - |  |  |  |  |
| Valid | 1.15 | Test order of operations at the same level: +, /. +, \* |  |  |  |  |
| Valid | 1.16 | Test order of operations between operations separated by parentheses (-)\*(+) |  |  |  |  |
| Valid | 1.17 | Test order of operations between operations separated by parentheses (\*)-(/) |  |  |  |  |
| Valid | 1.18 | Test order of operations between operations separated by parentheses (\*)/(+) |  |  |  |  |
| Invalid | 2.1 | Test an expression with 2 operands, 1 operator, and only an opening parenthesis at the begging and no closing parenthesis |  |  |  |  |
| Invalid | 2.2 | Test an expression with 2 operands, 1 operator, and only a closing parenthesis at the end |  |  |  |  |
| Invalid | 2.3 | Test an expression with 2 operands and no operators |  |  |  |  |
| Invalid | 2.4 | Test an expression with 1 operand and 1 operator |  |  |  |  |
| Invalid | 2.5 | Test an expression with no operands and 1 operator |  |  |  |  |
| Invalid | 2.6 | Test an expression with only a set of parentheses |  |  |  |  |
| Invalid | 2.7 | Test an expression with two operations and a missing closing parenthesis |  |  |  |  |
| Invalid | 2.8 | Test an expression with two operations and a missing opening parenthesis |  |  |  |  |
| Invalid | 2.9 | Test an expression with two operations enclosed in parentheses and no operator between them |  |  |  |  |
| File Handling | 3.1 | Input a file name that does not exist |  |  |  |  |
| File Handling | 3.2 | Input a file with a type other than “.dat” |  |  |  |  |
| File Handling | 3.3 | Input a file of type “.dat” that exists |  |  |  |  |
| File Handling | 3.4 | Input a file of type “.dat” that exists, but is empty |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Initial Algorithm**

* Main Method (called on program startup)
  + Open the output file “output.dat” for writing
    - All out put displayed to the screen will also be wrote to this file
  + Display a welcome message to the user
  + Ask the user to enter the location of the file they want to run through the program. (ex: [c://users//](../../../../c://users//)…//filename.dat)
  + If the file does not exist
    - Display an error message that it doesn’t exist and ask the user to enter another file name
  + If the file is empty
    - Display an error message that the file is empty and ask the user to enter another file name
  + If the file is of the wrong type (not “.dat”)
    - Display an error message that the file is of the wrong type and ask the user to enter another file name
  + Open the file and print a message that the file was opened successfully
  + For each line in the input file
    - read the line in and print it to the screen
    - create an empty variable for the postfix expression string
    - create an empty character stack
    - For each character in the input line
      * if character is a space
        + move to the next character

1. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Code**

A baseline for commenting is before any function add this:

//Description: What does the function do

//Pre-condition: What do input do you need for the function to work

//Post-condition: What is the end result of the function or what do you get out of the function

Also the beginning of your program should have these comments:

//Program Name:

//Programmer Name:

//Description:

//Date Created:

1. **Updated Algorithm**
2. **Test Plan Version 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Screenshots**
2. **Error Log**

|  |  |  |
| --- | --- | --- |
| Error Type | Cause of Error | Solution to Error |
|  |  |  |
|  |  |  |

1. **Status**