|  |
| --- |
| CIS 350 Data Structures |
| Program 1 – Turn in 1 |
| Fall 2014 |

|  |
| --- |
| Daniel Frey  9-24-2014 |

Contents

[Problem Summary 2](#_Toc399349911)

[Requirements 2](#_Toc399349912)

[Decomposition Diagram 3](#_Toc399349913)

[Order 4](#_Toc399349914)

[Testing Strategy – Categories 5](#_Toc399349915)

[Test Plan – Version 1 5](#_Toc399349916)

[Initial Algorithm 6](#_Toc399349917)

[Test Plan – Version 2 7](#_Toc399349918)

# Problem Summary

The program will create and traverse expression trees in prefix, infix and postfix order. Print out traversal results and order of operations.

# Requirements

The input will be read from a text file named statements.dat and each line will contain one infix expression to be processed. The requirements for the infix expression are:

* single uppercase letter operands
* operators are +, -, \* and / with \* and / having higher precedence than + and –
* normal left to right precedence for operator at same level
* a minimum of one space on each side of am operand (including initial and final operand)
* a minimum of one space on each side of each operator
* a minimum of one space on each side of parenthesis (including initial and final parenthesis)

The program will need to be able to handle exceptions such as file errors (no input file exists, input file empty, etc.), invalid data (invalid operand, invalid operator, spacing), etc.

For each valid input line the program will build the corresponding expression tree then traverse the tree in prefix, infix and postfix order. Must also print the sequence of partial expressions evaluated.

The output for the program will be written to the screen and to a file named output.dat which will contain some or all of the following (e.g. if input file cannot be opened there is obviously only the program completed message after file open error message):

* program started message
* files open success or fail message
* echo print input line
* if valid input data write the traversals and sequence of operations otherwise error message including the piece of data causing the error
* program complete message

For infix traversal only relevant parenthesis will be displayed.

# Decomposition Diagram

# Order

* Check if file exists
* Check if file empty
* Read valid file
* Print/write input line
* Check spacing
* Check matching parentheses
* Check parentheses contain data
* Check valid operands/operators
* Check order of operands/operators
* Convert to postfix
* Build tree
* Traverse and print/write prefix
* Traverse and print/write infix
* Traverse and print/write postfix

# Testing Strategy – Categories

1. File Processing
2. Invalid Data
3. Valid Data

# Test Plan – Version 1

|  |  |  |
| --- | --- | --- |
| Test Case Number | Test Case Category | Test Case Description |
| 1.1 | File Processing | No file exists |
| 1.2 | File Processing | File exists, but is empty |
| 1.3 | File Processing | File exists and contains data |
| 1.4 | File Processing | Data written to file |
| 2.1 | Invalid Data | File contains incorrect spacing |
| 2.2 | Invalid Data | File contains mismatching parentheses |
| 2.3 | Invalid Data | File contains incorrect operands |
| 2.4 | Invalid Data | File contains incorrect operators |
| 3.1 | Valid Data | Input contains incorrect sequence of operators/operands |
| 3.2 | Valid Data | Correct conversion to post |
| 3.3 | Valid Data | Tree created |
| 3.4 | Valid Data | Correct output of traversals is shown/written |

# Initial Algorithm

Begin with “Program started” message. Then take in an input file. If no file of given name exists, then “File does not exist.” message is displayed. If a file does exist, then “File successfully opened.” message is displayed.

If the file is empty and no lines exist to read, then give the message “File is empty.” and program ends with “Program complete.” message. If file is not empty, then read one line at a time from file. Display and write to a file the line to ensure correctness of copy.

Go through the whole line checking that a space is at the beginning, on either side of any character, and at the end. If the spacing is incorrect, then “Incorrect spacing.” message is given for the current line and continue to the next line.

If the spacing is correct, check that the number of left parentheses equals the number of right parentheses. If there are mismatched parentheses, then give the message “Missing parentheses.”

If parentheses are correct, check that there is data inside. If there is none, then the message “Empty parentheses.” is given.

Go through the line checking that the operators are only +, -, \*, and /, and that the operands are only upper case letters. If some other symbol is encountered, then it results in “Invalid operator/operand.”

Once the valid characters are confirmed, then the order needs to be checked. If there is only a single operand then it is valid. The first character, other than parenthesis, must be an operand. The following must be an operator. Continue checking sequence until a final operand is found. If this sequence is not followed, then the message “Invalid sequence of operators/operands.” is given.

Next, the infix expression is converted to postfix for building the expression tree. Write operands to a string as they are encountered. While the stack is not empty and the top is not a left parenthesis, and if the top of stack operator is different precedence than operator being evaluated, pop stack operator to string and push other to stack. If left parenthesis encountered, push to stack. If right parenthesis encountered, and while stack is not empty, continue writing to string until left parenthesis encountered. Anything remaining is put into string.

Build tree using data and left and right pointers from postfix. Read through post fix, operands get pushed on a stack. If current node is operator, pop two operands and a new node and push sub tree into stack.

Traverse in pre-order by visiting first, go left, then once left done go right. In-order by going left, visiting, then right once all lefts are done. Post-order left, right, then visit. Each visit puts the character in a string to be written to file and displayed to screen.

# Test Plan – Version 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case Number | Test Case Category | Test Case Description | Test Case Input | Test Case Expected Result |
| 1.1 | File Processing | No file exists | Bad file name or nonexistent file | File does not exist |
| 1.2 | File Processing | File exists, but is empty | Empty file | File is empty |
| 1.3 | File Processing | File exists and contains data | Non-empty file | Input checks |
| 1.4 | File Processing | Data written to file | Valid file, passes checks | Output file contains data |
| 1.5 | File Processing | File opens regardless empty or not | Existing file | File opened successfully |
| 2.1 | Invalid Data | File contains incorrect spacing | Bad spaced input | Incorrect spacing |
| 2.2 | Invalid Data | File contains mismatching parentheses | Uneven parentheses in input | Missing parentheses |
| 2.3 | Invalid Data | File contains incorrect operands | Input contains lower case operands | Invalid operator/operand |
| 2.4 | Invalid Data | File contains incorrect operators | Input contains some other symbol (+,-,\*,/) | Invalid operator/operand |
| 3.1 | Valid Data | Input contains incorrect sequence of operators/operands | Input does not follow operand, operator, operand | Invalid sequence of operators/operands |
| 3.2 | Valid Data | Correct conversion to post | Infix expression | Postfix expression |
| 3.3 | Valid Data | Tree created | Postfix expression | Expression tree |
| 3.4 | Valid Data | Correct output of traversals is shown/written | Expression tree | Pre-, in-, and post-order expressions |
| 3.5 | Valid Data | Empty parentheses | Input with no operands or operators in parentheses | Empty parentheses |
| 3.6 | Valid Data | Single operator in parentheses | Input with only operand in parentheses | Continue other checks |
| 3.7 | Valid Data | Program completes | Valid data | Program complete |
| 3.8 | Valid Data | Data read correct | File data | Echo input |