Name: Jack Travis

Student ID: ---

Username: ---

Assessment Title: Assignment 1

Module Name: Data Structures & Algorithms

CRN: Unknown

Summary of submission

This document consists of several sections:

* [Acknowledgements](#Acknowledgements) – This section gives credit to all individuals who’s work/research was used to create my program.
* [ADT Justification](#ADT_Justification) –Detailed reasoning as to why I chose a specific abstract data type, over others.
* [Class Diagram](#Class_Diagram) – A image of the class diagram, showing the relationships between all classes that are present in my program.
* [Proof of functionality](#Proof_of_functionality) - A written explanation of what my program does, including six examples, proving that the program functions correctly

Zip file where this text document was located also contains/contained:

* ‘Java Files’ – This is a folder containing all the source code for the program.
* ‘JAR executable’ – This a folder containing the program as an executable JAR file. It is recommended to execute this file using a command line which allows for the execution of jar files. Execution is done using the command ‘ java -jar [file location]’

Acknowledgements

Information regarding building a linked stack was provided by Dr Tarek Gaber, via PowerPoint slides uploaded to blackboard. More specifically:

* Section 3: Part 2 – The Abstract Data Type: Linear List – Slide 3.12 – A linked node class definition - ©2019-20
* Section 5 – Abstract Data Type: Stack – Slide 5.28 – 5.31 - ©2019-20

ADT Justification

A queue operates on a first in, first out basis, whereas a stack operates on a first in, last out basis. The task set requires for an infix expression to be turned into a postfix expression. The process of this requires variables and operators to be added to data structures until a closed parenthesis is located. This closed parenthesis relates to the last two variables added to the variable data structure, and the last operator added to the operator data structure. Therefore, a stack is the best choice because stacks pop the last added item first, which is what the program is interested in, leaving the data not yet of interest alone. If a queue was chosen, the data structure would require to be emptied to get to the data that is wanted.

A stack can either be implemented as an array or as a linked list, each having various advantages and disadvantages. Certain operations of these data structures are irrelevant for the given task, such as searching, or finding, because at no point does the program require data that it can’t already directly access (by peeking or popping). Both options are similar in complexity to add data items in this program, because adding new data doesn’t affect any data already being stored. For example, in a linear stack, new data is added to the back (not requiring moving the positions of other items), and the pointer is incremented. With a linked list, a data node adds the new data to the front, and points to the original top.

One main task that is required of the stacks is the ability to remove/pop the data they hold. This process is easier to do with a linked stack, rather than a linear stack, because the program is only interested in the last data item. For a linked stack, this only requires accessing the data in top and setting top to top.next. For a linear stack, this would require accessing the data where the pointer is pointing to, setting the value of that position to null, and then decrementing the pointer.

The main difference between a linked stack and a linear stack is a linear stack is static, requiring the size to be known in advanced, whereas a linked stack is dynamic, as it is only as big as it requires. The downside of being dynamic is the extra overhead; Each data node requires a pointer to the next data node, which isn’t the case for a linear stack. The exact size required is unknown but can be estimated based on the maximum input size of the infix expression. At most, the user can enter 20 characters, which allows for a maximum of 6 variables and 5 operators. Therefore, a linear stack could set the required size to 6 and 5 respectively, and the data will be guaranteed to fit in. This is an insignificant amount of data loss for modern systems and is potentially as efficient on average as a linked stack when the overhead of a data node pointer per data node is considered.

Instead, I have considered the future potential of this system. If the input limit was raised or removed completely, the size of the required stack would change. This would require setting a default size for a linear stack, and creating a new, bigger stack when required. This would result in more allocated space being unused, and therefore wasted. Whereas, a stack only uses the amount of memory it requires, which is potentially significantly less wasteful and faster to execute.

Therefore, due to linked stacks being able to allocate memory dynamically, and its quicker execution speed of removing the top data node, I decided that a linked stack would be the best choice for the given task.

Class Diagram

A screenshot of a computer

Description automatically generated with medium confidence

Proof of functionality

My program asks the user to input a fully parenthesized infix expression. This is scanned in, using a scanner, as a String.

The validity of the input is checked by iterating though this String, checking for what it expects to appear next, based off the previous character(s). This is done by having an integer variable, the value of which is dependent on the last character iterated. The meaning of each number can be found commented next to the initialisation of the variable (perviousCharType). This value dictates which section of the switch-case statement to use, which in turn checks the current character against what character is expected to make the inputted String a valid infix expression.

If the expression is valid, two stacks are created: One to hold values (varStack) and one to hold operators (opStack).

The input is iterated again. If the current character is a:

* Open parenthesis, nothing happens
* Variable (an alphabetical character (not case sensitive)), an object is created, holding the value of said variable, and is pushed onto varStack
* Operator, an object is created holding the value of said operator, and pushed onto opStack
* Close parenthesis, three objects are created: var1, var2, op1. The varStack is popped, and the value is saved to var1. varStack is then popped again, with the value saved to var2. opStack is then popped, with the value saved to op1. An object is then created, containing the value of var2 + var1 +op1, and is pushed onto the varStack.

Once the end of the String is reached:

* Three objects are created: var1, var2, op1. The varStack is popped, and the value is saved to var1. varStack is then popped again, with the value saved to var2. opStack is then popped, with the value saved to op1. A String output, containing the value of var2 + var1 +op1 and is outputted to the terminal. This is the completed postfix expression

Below are 6 examples of an infix expression being inputted, followed by the program outputting the postfix expression.

Expression 1

Infix Expression (P\*Q)+(R/(S\*(T/U)))

Screenshot of program:

A screenshot of a computer

Description automatically generated with medium confidence

Expression 2

Infix Expression (P\*Q)+(R/(S\*(T\*U)))

Screenshot of program:

A screenshot of a computer

Description automatically generated with medium confidence

Expression 3

Infix Expression (P\*(R-(S/T)\*U))

Screenshot of program:

A screenshot of a computer

Description automatically generated with medium confidence

Expression 4

Infix Expression (A\*(B-C))+(D\*E)

Screenshot of program:

A screenshot of a computer

Description automatically generated with medium confidence

Expression 5

Infix Expression (A-(B+C))/(D\*(E-F))

Screenshot of program:

A screenshot of a computer

Description automatically generated with medium confidence

Expression 6

Infix Expression (Y\*F)-((C\*G)-B)

Screenshot of program:

A screenshot of a computer

Description automatically generated with medium confidence