JAVA CALCULATOR

Project report submitted to the Amrita Vishwa Vidyapeetham in partial fulfillment of the requirement for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY

R Aravind – AM.EN.U4AIE21151

Vishnu Shaji – AM.EN.U4AIE21167

Karthik Nair – AM.EN.U4AIE21138

Akhilesh – AM.EN.U4AIE21108

Mamidi Sowji Krishna -AM.EN.U4AIE21181

JULY 2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING AMRITA VISHWA VIDYAPEETHAM

(Est. U/S 3 of the UGC Act 1956)

Amritapuri Campus

Kollam -690525

BONAFIDE CERTIFICATE

Reviewer

Akshara P Byju

Chairperson

Prema Nedungadi

Dept. of Computer Science & Engineering

Place: Amritapuri
Date: 12 July 2022

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING AMRITA VISHWA VIDYAPEETHAM
(Est. U/S 3 of the UGC Act 1956)
Amritapuri Campus
Kollam -690525

DECLARATION

We**, R Aravind, Vishnu Shaji, Karthik Nair, Akhilesh, Mamidi Sowji Krishna** hereby declare that this project entitled is a record of the original work done by us under the guidance of **Akshara P Byju** Dept. of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, that this work has not formed the basis for any degree/diploma/associations/fellowship or similar awards to any candidate in any university to the best of our knowledge.

Place: Amritapuri
Date: 12 July 2022

Signature of the student   Signature of the Project

Acknowledgment

We take this occasion to Thank God for his blessings. We would like to express our special thanks of gratitude to our teacher Akshara P Byju who gave us the opportunity to do this project. We would also like to express gratitude to our class advisors for their cordial support, and valuable suggestions. At last, we would also like to thank our friends and family for the support and encouragement they have given us during the course.

Abstract

This work was centered on the Design and implementation of a simple calculator. The application can perform different functions like addition, subtraction, multiplication, division, Clear function, Backspace function, Memory functions, History functions, and Exception Handling. Also, we have created a special negative sign button. This application is very useful to people who want to calculate values using the above functions. We have used different Data Structures to implement this project in Java-like LinkedList, Array List, Queue, and Stack which increases the efficiency of the code. This project allows us to explore different Data Structures and their implementations.

List of Contents

Introduction

We have created the application based on our understanding of the object-oriented concepts of inheritance, polymorphism, aggregation. We have also used Data Structures like Stack, Queue, Array and Linked List.

The calculator application performs basic mathematical calculations and is presented to the user in a friendly and easy-to-use way and provides appropriate messages when wrong input is given by the user.

The application also stores it in the memory and there are functions that help to clear the memory or add value to the memory.

Problem Definition

Our main objective in this project is to create a Calculator where people can calculate different values using the given operators and provide output.

People should be able add the value into the calculator memory to use it later and remove it from the memory.

Challenges:

Consider an expression below:

20 + 14 * 15

Difficulty : Deciding which operator to be evaluated first.

Comparison :
Addition First : 20 + 14 * 15 = 510
Multiplication First : 20 + 14 * 15 = 230

Which is correct ?

There is a need to define priority to the computer and evaluate using an effective way while handling brackets and wrong inputs.

Applied Methodology

Priorities of the operators are predefined

Operators	Priority
^ (Exponentiation)	1 (highest)
* (Multiplication) and / (Division)	2
+ (Addition) and – (Subtraction)	3 (lowest)

We can't use infix expression to handle arithmetic operations are low-level. So we convert infix expression to postfix expression and evaluate postfix expression using the stack.

Infix Notation - an operator is written in between two operands.

a + b

Postfix Notation – the operators appear after operands. ab+.

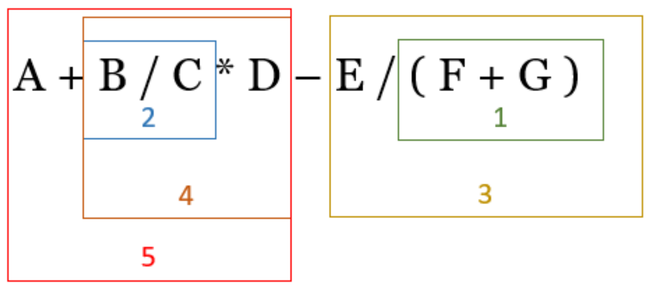
Expression Evaluation

Infix expression requires extra information to make an order of evaluation clear.

Additional information: precedence and associativity.

Example 1 : A + B / C

Example 2 :



Consider an expression A + B / C * D – E / (F + G)

Scan 1: Found an expression within parentheses.
Scan 2: B / C has the second-best priority.
Scan 3: Result of B / C must be multiplied by D.

Infix Notation requires multiple scans to evaluate an expression and therefore is inefficient in terms of time complexity.

Postfix Notation: AB+
: ABC/+
The order of evaluation of operators is left to right with no brackets in the expression to change the order.

In the example, division comes before addition, and therefore, the division must be performed before addition.

Infix to Postfix Conversion

Infix Expression:

A + B / C * D – E / (F + G)

Postfix Expression:

A + B / C * D – E / (F G +)

A + B C / * D – E / (F G +)

A + B C / D * – E / (F G +)

A + B C / D * – E (F G +) /

A B C / D * + – E (F G +) /

Postfix expression Evaluation

Consider,

Infix Expression: 3 + 5 * (5 / 5) – 2 ^ 2

Postfix Expression: 3 5 5 5 / * + 2 2 ^-

The expression will be scanned from left to right and as soon as we will encounter an operator, we will apply it to the last two operands.

3 5 5 5 / * + 2 2 ^ -

3 5 5 5 / * + 2 2 ^ -

3 5 5 5 / * + 2 2 ^ -

3 5 5 5 / * + 2 2 ^ -

3 5 5 5 / * + 2 2 ^ - -> Apply '/' to 5 and 5

3 5 5 5 / * + 2 2 ^- - > Apply '/' to 5 and 5

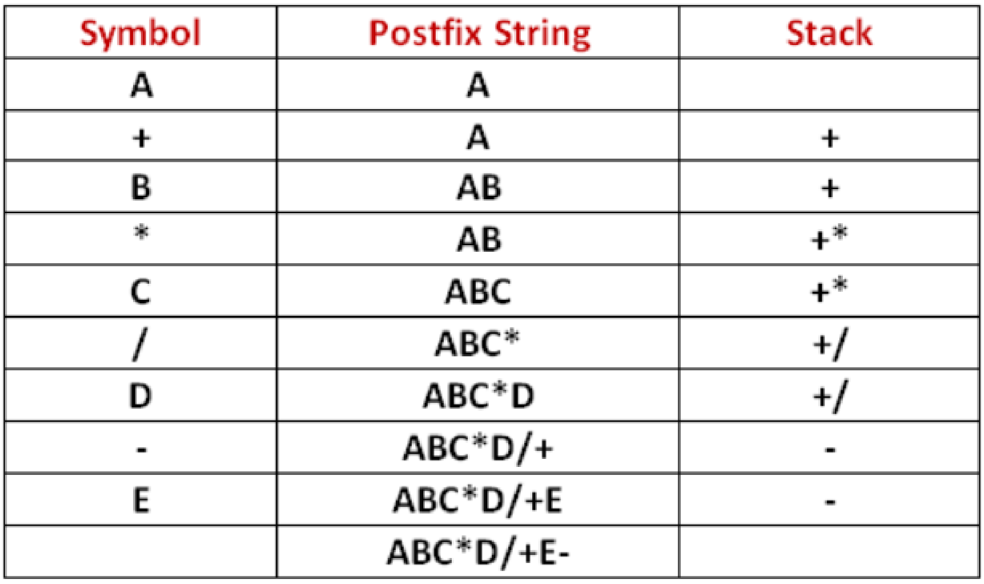
3 5 1 * + 2 2 ^ - -> Apply '*' to 5 and 1

3 5 + 2 2 ^ - -> Apply '+' to 3 and 5

3 5 + 2 2 ^ -	-> Apply '+' to 3 and 5
8 2 2 ^ -	-> Apply '^' to 2 and 2
8 4 -	-> Apply '-' to 8 and 4

12

Applying the idea using stack



Symbol	Postfix String	Stack
A	A	
+	A	+
B	AB	+
*	AB	+*
C	ABC	+*
/	ABC*	+/
D	ABC*D	+/
-	ABC*D/+	-
E	ABC*D/+E	-
	ABC*D/+E-	

ALGORITHMS USED

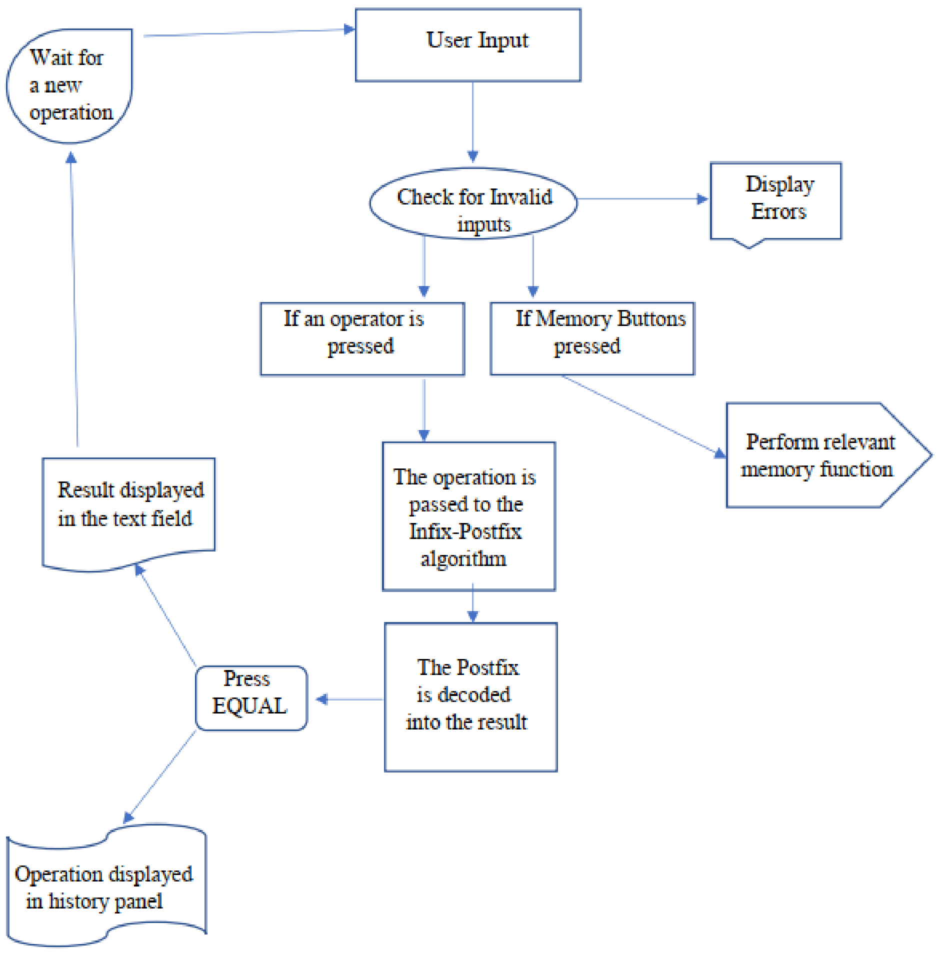
Infix to Postfix Conversion:

```
Begin
  initially push some special character say # into the stack
  for each character ch from infix expression, do
    if ch is alphanumeric character, then
      add ch to postfix expression
    else if ch = opening parenthesis (, then
      push ( into stack
    else if ch = ^, then          //exponential operator of higher precedence
      push ^ into the stack
    else if ch = closing parenthesis ), then
      while stack is not empty and stack top ≠ (,
        do pop and add item from stack to postfix expression
      done

      pop ( also from the stack
    else
      while stack is not empty AND precedence of ch <= precedence of stack top
element, do
        pop and add into postfix expression
      done

      push the newly coming character.
  done

  while the stack contains some remaining characters, do
    pop and add to the postfix expression
  done
  return postfix
End
```

Postfix evaluation:

```
Begin
  for each character ch in the postfix expression, do
    if ch is an operator ⊙ , then
      a := pop first element from stack
      b := pop second element from the stack
      res := b ⊙ a
      push res into the stack
    else if ch is an operand, then
      add ch into the stack
  done
  return element of stack top
End
```

Flowchart


User Input
Wait for a new operation
Check for Invalid inputs
Display Errors
If an operator is pressed
If Memory Buttons pressed
Perform relevant memory function
The operation is passed to the Infix-Postfix algorithm
The Postfix is decoded into the result
Press EQUAL
Result displayed in the text field
Operation displayed in history panel

Data Structures Used

Queue: queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence.

We have used Queue to compare the strings.

Stack: Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO (Last In First Out) or FILO(First In Last Out).

We have used Stack to convert Infix to Postfix and for postfix evaluation.

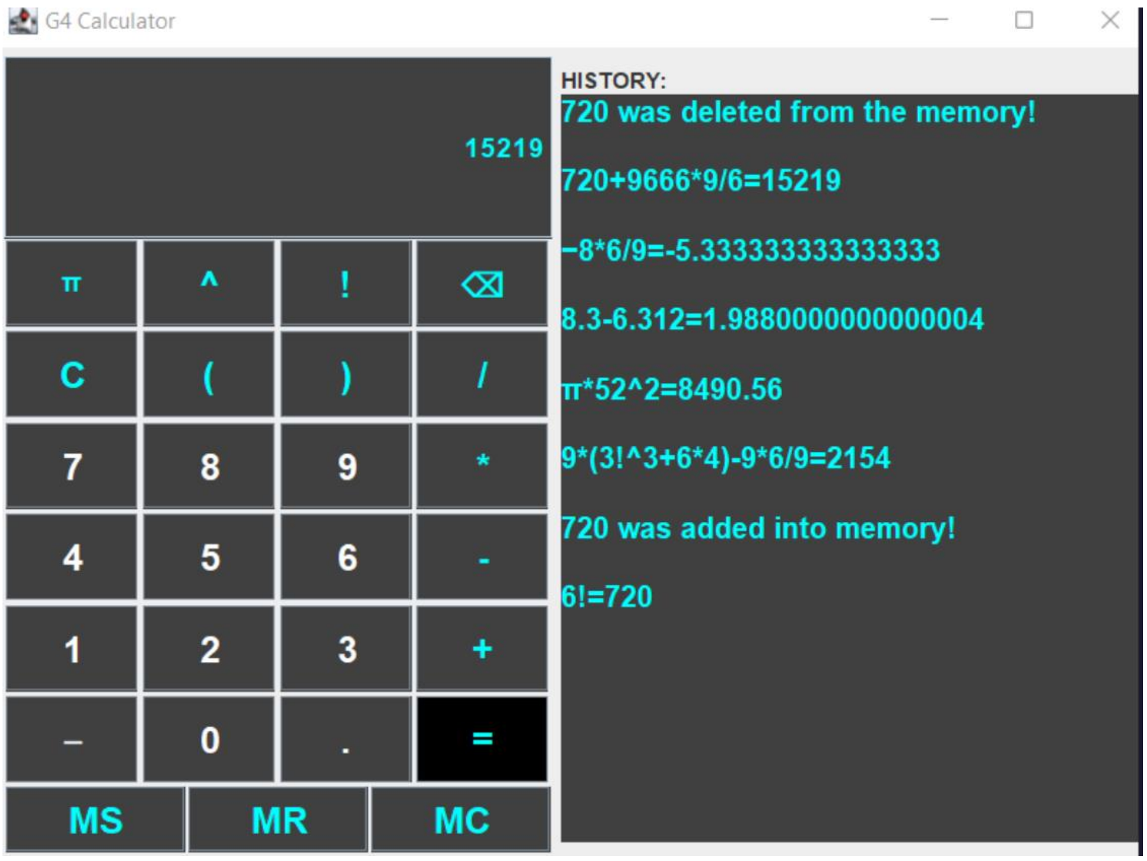
Array List: Array List is a resizable array implementation in java.

We have used Array List to handle postfix and infix expressions,

Linked List: Linked List is a sequence of Data Structures, which are connected via links.

We have used Single Linked List for implementing Stack and Queue in our Calculator.

Sample Input/Output



HISTORY:
720 was deleted from the memory!

720+9666*9/6=15219

−8*6/9=-5.333333333333333

8.3-6.312=1.9880000000000004

π*52^2=8490.56

9*(3!^3+6*4)-9*6/9=2154

720 was added into memory!

6!=720

Conclusion

We started this project to test our coding skills, but each step was a challenging step that taught us important lessons. This was the first project that we used UI and gave us the opportunity to work with different Data Structures and to use the algorithms we studied this year.

Developing this project made us feel more confident on our skills in programming and algorithms.

Source Code

https://github.com/ar-seven/JavaCalculator

\