**PROJECT SUBMISSION**

**BACHELOR OF TECHNOLOGY**

**IN**

**CSE**

**Session 2023-2024**

**Submitted To: Dept. of Submitted By: Harsh Agarwal**

**CSE Roll No. – 23EJICS055**

**(II Year / III Sem)**

**Department of CSE**

**JIET Institute of Engineering and Technology**

**Near NH-62, New Pali Road, Mogra, Jodhpur (Raj.)**

**Introduction**

Mathematical expressions are typically written in infix notation, where operators are placed between operands. For instance, an arithmetic expression like **A + B \*** C is an example of infix notation. While this form is intuitive for humans, computers process expressions more efficiently when they are converted into alternative formats like postfix (Reverse Polish Notation) or prefix (Polish Notation). These notations eliminate the need for parentheses and follow a strict order of operations, making them ideal for computational purposes such as in compilers, calculators, and stack-based machine architectures.

Infix notation requires the human to check the precedence of operations (such as multiplication over addition) and handle parentheses accordingly. In contrast, postfix and prefix notations follow a structure where the precedence and associativity are inherently encoded in the order of operators and operands. This makes them more suitable for algorithmic processing and evaluation using stacks.

**Postfix Notation**

Postfix notation places the operator after its operands. For example, the infix expression **A + B** is written as **A B +** in postfix. This notation is particularly useful in scenarios where the order of operations is vital, as postfix eliminates the need for parentheses entirely. Evaluation of postfix expressions can be easily accomplished using a stack, where operands are pushed onto the stack and operators pop two operands, apply the operation, and push the result back onto the stack.

**Prefix Notation**

Prefix notation places the operator before its operands. For example, the infix expression **A + B** is written as **+ A B** in prefix notation. Like postfix, prefix also removes the need for parentheses and is evaluated using a stack-based approach but in reverse order, processing the expression from right to left.

**Project Objective**

The objective of this project is to design and implement a graphical user interface (GUI) that allows users to input infix expressions and convert them into their corresponding postfix and prefix forms. The application, built using Python's Tkinter library, provides an interactive and educational tool for understanding how infix expressions are processed and converted. By offering a visual, step-by-step breakdown of the conversion process, the tool enhances learning and comprehension, particularly in domains like data structures, algorithms, and compiler design.

**Minimum System Requirement**

To run this project, the following minimum system requirements are recommended:

* **Operating System**: Windows, macOS, or any Linux distribution
* **RAM**: 2 GB or higher
* **Processor**: 1 GHz or faster
* **Python Version**: Python 3.6 or higher
* **Required Libraries**:
  + Tkinter (Pre-installed with Python)
  + Any text editor/IDE like Visual Studio Code or PyCharm for running Python code

**Problem Statement**

Design and develop a graphical user interface (GUI) application that accepts an infix expression from the user and converts it into either a postfix or prefix expression. The program should:

1. Convert a valid infix expression into its corresponding postfix and prefix forms.
2. Display a step-by-step breakdown of the conversion process.
3. Allow the user to choose between postfix and prefix conversion modes using a toggle option.
4. Provide a clear visual representation of the current processing character, operators, and the evolving output expression.

**Algorithm**

#### 1. ****Infix to Postfix Conversion Algorithm:****

* Initialize two stacks: symbol for operators and expression for the output.
* For each character in the infix string:
  + If it’s an operand, push it to the expression stack.
  + If it’s an operator, pop from the symbol stack to the expression stack based on precedence rules, then push the operator onto the symbol stack.
  + If it’s an opening parenthesis, push it onto the symbol stack.
  + If it’s a closing parenthesis, pop from the symbol stack until an opening parenthesis is encountered.
* After processing the string, pop any remaining operators from the symbol stack to the expression stack.
* The final expression stack represents the postfix expression.

#### 2. ****Infix to Prefix Conversion Algorithm:****

* Reverse the infix string and swap parentheses (i.e., ( becomes ) and vice versa).
* Apply the infix-to-postfix algorithm on the reversed string.
* Once the postfix expression is obtained, reverse it to get the prefix expression.

**Source Code**

import tkinter as tk

# Define colors

BACKGROUND\_COLOR = '#fff4f4'

TEXT\_COLOR = '#000000'

BUTTON\_COLOR = '#d8e1cb'

ENTRY\_COLOR = '#fce5cd'

class Stack:

    def \_\_init\_\_(self, size):

        self.size = size

        self.arr1 = [None] \* size

        self.top = -1

    def top\_ele(self):

        if self.top == -1:

            return None

        return self.arr1[self.top]

    def push(self, ele):

        if self.top >= self.size - 1:

            print("Stack Overflow!")

        else:

            self.top += 1

            self.arr1[self.top] = ele

    def pop(self):

        if self.top < 0:

            print("Stack Underflow!")

            return None

        else:

            popped\_element = self.arr1[self.top]

            self.top -= 1

            return popped\_element

    def traverse(self):

        elements = []

        for i in range(0, self.top + 1):

            elements.append(self.arr1[i])

        return elements

def infix\_to\_postfix(string):

    symbol = Stack(len(string))

    expression = Stack(len(string))

    steps = []

    for i in string:

        if i == '(':

            symbol.push(i)

        elif i in ('^', '\*', '/', '+', '-'):

            if i == '^':

                while symbol.top\_ele() == '^':

                    expression.push(symbol.pop())

                symbol.push(i)

            elif i in ('\*', '/'):

                while symbol.top\_ele() in ('^', '\*', '/'):

                    expression.push(symbol.pop())

                symbol.push(i)

            elif i in ('+', '-'):

                while symbol.top\_ele() in ('+', '-', '\*', '/', '^'):

                    expression.push(symbol.pop())

                symbol.push(i)

        elif i == ')':

            while symbol.top\_ele() != '(':

                expression.push(symbol.pop())

            symbol.pop()

        else:

            expression.push(i)

        steps.append((list(symbol.traverse()), list(expression.traverse())))

    while symbol.top != -1:

        expression.push(symbol.pop())

        steps.append((list(symbol.traverse()), list(expression.traverse())))

    return steps

def infix\_to\_prefix(string):

    # Reverse the string and convert parentheses

    rev\_string = string[::-1]

    rev\_string = ''.join('(' if ch == ')' else ')' if ch == '(' else ch for ch in rev\_string)

    symbol = Stack(len(rev\_string))

    expression = Stack(len(rev\_string))

    steps = []

    for i in rev\_string:

        current\_char\_text.delete(1.0, tk.END)  # Clear the text widget

        current\_char\_text.insert(tk.END, i)  # Insert the current character

        if i == '(':

            symbol.push(i)

        elif i == ')':

            while symbol.top\_ele() is not None and symbol.top\_ele() != '(':

                expression.push(symbol.pop())

            if symbol.top\_ele() == '(':

                symbol.pop()

        elif i in '^\*/+-':

            while (symbol.top\_ele() is not None and

                   symbol.top\_ele() != '(' and

                   ((i in '+-' and symbol.top\_ele() in '\*/^') or

                    (i in '\*/' and symbol.top\_ele() in '^') or

                    (i in '^' and symbol.top\_ele() in '\*/'))):

                expression.push(symbol.pop())

            symbol.push(i)

        else:

            expression.push(i)

        steps.append((list(symbol.traverse()), list(expression.traverse())))

    while symbol.top != -1:

        expression.push(symbol.pop())

        steps.append((list(symbol.traverse()), list(expression.traverse())))

    # Reverse the result for prefix

    intermediate = []

    while expression.top != -1:

        intermediate.append(expression.pop())

    prefix = "".join(intermediate)

    return steps, prefix[::-1]  # Reverse the result for prefix

def on\_button\_click():

    input\_expression = input\_field.get()

    if var.get() == 1:  # Checkbox is selected, perform postfix conversion

        steps = infix\_to\_postfix(input\_expression)

        output1.delete(1.0, tk.END)

        output2.delete(1.0, tk.END)

        current\_char\_text.delete(1.0, tk.END)

        horizontal\_output.delete(1.0, tk.END)

        for i in input\_field.get():

            current\_char\_text.insert(tk.END,i+'\n')

        for step in steps:

            symbols, expressions = step

            output1.insert(tk.END, " ".join(map(str, symbols)) + '\n')

            output2.insert(tk.END, " ".join(map(str, expressions)) + '\n')

        horizontal\_output.insert(tk.END, " ".join(map(str, steps[-1][1])))

    else:  # Checkbox is not selected, perform prefix conversion

        steps, prefix = infix\_to\_prefix(input\_expression)

        prefix\_rev=prefix[::-1]

        output1.delete(1.0, tk.END)

        output2.delete(1.0, tk.END)

        current\_char\_text.delete(1.0, tk.END)

        horizontal\_output.delete(1.0, tk.END)

        for i in input\_field.get()[::-1]:

            current\_char\_text.insert(tk.END,i+'\n')

        for step in steps:

            symbols, expressions = step

            output1.insert(tk.END, " ".join(map(str, symbols)) + '\n')

            output2.insert(tk.END, " ".join(map(str, expressions)) + '\n')

        horizontal\_output.insert(tk.END, prefix\_rev)

# Initialize the main window

root = tk.Tk()

root.title("Infix to Postfix/Prefix")

root.geometry('650x600')

root.resizable(0, 0)

root.configure(bg=BACKGROUND\_COLOR)

# Create input field and button at the top

input\_frame = tk.Frame(root, bg=BACKGROUND\_COLOR)

input\_frame.pack(pady=5)

input\_field = tk.Entry(input\_frame, width=30, bg=ENTRY\_COLOR, fg=TEXT\_COLOR)

input\_field.pack(side=tk.LEFT, padx=(20, 5), ipadx=30)

button = tk.Button(input\_frame, text="Submit", command=on\_button\_click, bg=BUTTON\_COLOR, fg='#ffffff')

button.pack(side=tk.LEFT, padx=(30, 10))

# Create a frame for the checkboxes

checkbox\_frame = tk.Frame(root, bg=BACKGROUND\_COLOR)

checkbox\_frame.pack(pady=5)

# Checkbox for postfix/prefix

var = tk.IntVar(value=1)

checkbox\_postfix = tk.Checkbutton(checkbox\_frame, text="Postfix", variable=var, onvalue=1, offvalue=0, bg=BACKGROUND\_COLOR, fg=TEXT\_COLOR)

checkbox\_postfix.pack(side=tk.LEFT, padx=(20, 5))

checkbox\_prefix = tk.Checkbutton(checkbox\_frame, text="Prefix", variable=var, onvalue=0, offvalue=1, bg=BACKGROUND\_COLOR, fg=TEXT\_COLOR)

checkbox\_prefix.pack(side=tk.LEFT, padx=(20, 5))

# Create a frame for the output fields

output\_frame = tk.Frame(root, bg=BACKGROUND\_COLOR)

output\_frame.pack(pady=10)

# Frame for current character and vertical outputs

current\_char\_and\_outputs\_frame = tk.Frame(output\_frame, bg=BACKGROUND\_COLOR)

current\_char\_and\_outputs\_frame.pack(side=tk.LEFT, padx=10)

# Frame for current character display

output\_fields\_frame = tk.Frame(current\_char\_and\_outputs\_frame, bg=BACKGROUND\_COLOR)

output\_fields\_frame.pack(side=tk.LEFT, padx=10)

current\_char\_text = tk.Text(output\_fields\_frame, height=25, width=20, bg=ENTRY\_COLOR, fg=TEXT\_COLOR, wrap='none')

current\_char\_text.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

current\_char\_horizontal\_scrollbar = tk.Scrollbar(output\_fields\_frame, orient=tk.HORIZONTAL, command=current\_char\_text.xview)

current\_char\_horizontal\_scrollbar.pack(side=tk.BOTTOM, fill=tk.X)

current\_char\_text.config(xscrollcommand=current\_char\_horizontal\_scrollbar.set)

current\_char\_vertical\_scrollbar = tk.Scrollbar(output\_fields\_frame, orient=tk.VERTICAL, command=current\_char\_text.yview)

current\_char\_vertical\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

current\_char\_text.config(yscrollcommand=current\_char\_vertical\_scrollbar.set)

# Output fields for steps

output1 = tk.Text(output\_fields\_frame, height=25, width=20, bg=ENTRY\_COLOR, fg=TEXT\_COLOR, wrap='none')

output1.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

output1\_horizontal\_scrollbar = tk.Scrollbar(output\_fields\_frame, orient=tk.HORIZONTAL, command=output1.xview)

output1\_horizontal\_scrollbar.pack(side=tk.BOTTOM, fill=tk.X)

output1.config(xscrollcommand=output1\_horizontal\_scrollbar.set)

output1\_vertical\_scrollbar = tk.Scrollbar(output\_fields\_frame, orient=tk.VERTICAL, command=output1.yview)

output1\_vertical\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

output1.config(yscrollcommand=output1\_vertical\_scrollbar.set)

# Second output field for expressions

output2 = tk.Text(output\_fields\_frame, height=25, width=20, bg=ENTRY\_COLOR, fg=TEXT\_COLOR, wrap='none')

output2.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

output2\_horizontal\_scrollbar = tk.Scrollbar(output\_fields\_frame, orient=tk.HORIZONTAL, command=output2.xview)

output2\_horizontal\_scrollbar.pack(side=tk.BOTTOM, fill=tk.X)

output2.config(xscrollcommand=output2\_horizontal\_scrollbar.set)

output2\_vertical\_scrollbar = tk.Scrollbar(output\_fields\_frame, orient=tk.VERTICAL, command=output2.yview)

output2\_vertical\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

output2.config(yscrollcommand=output2\_vertical\_scrollbar.set)

# Output field for final result

horizontal\_output\_frame = tk.Frame(root, bg=BACKGROUND\_COLOR)

horizontal\_output\_frame.pack(pady=10)

horizontal\_output = tk.Text(horizontal\_output\_frame, height=2, width=60, bg=ENTRY\_COLOR, fg=TEXT\_COLOR, wrap='none')

horizontal\_output.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

horizontal\_output\_horizontal\_scrollbar = tk.Scrollbar(horizontal\_output\_frame, orient=tk.HORIZONTAL, command=horizontal\_output.xview)

horizontal\_output\_horizontal\_scrollbar.pack(side=tk.BOTTOM, fill=tk.X)

horizontal\_output.config(xscrollcommand=horizontal\_output\_horizontal\_scrollbar.set)

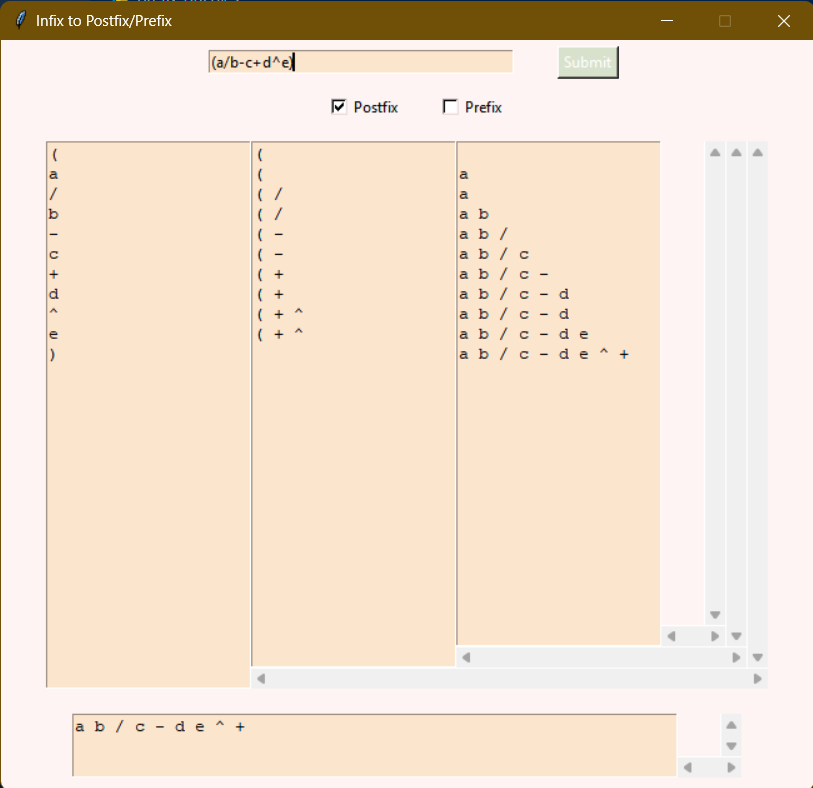
horizontal\_output\_vertical\_scrollbar = tk.Scrollbar(horizontal\_output\_frame, orient=tk.VERTICAL, command=horizontal\_output.yview)

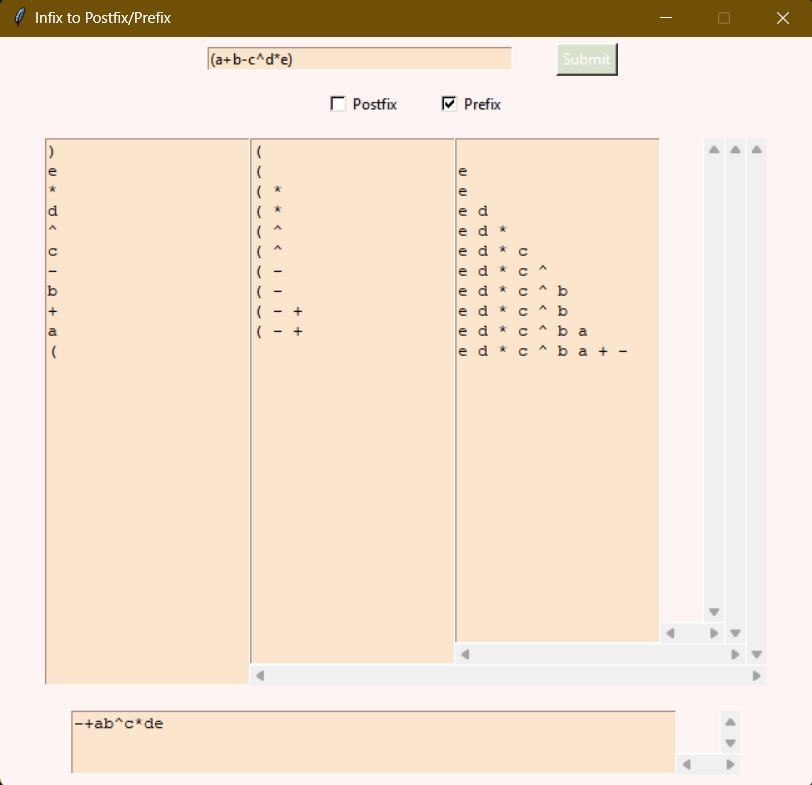
horizontal\_output\_vertical\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

horizontal\_output.config(yscrollcommand=horizontal\_output\_vertical\_scrollbar.set)

root.mainloop()

**OUTPUT**

****

****

**References**

 Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

 Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in Python. Wiley.

 Python Tkinter GUI Programming by Example by David Love (2018).

 GeeksforGeeks. (n.d.). Conversion of Infix to Postfix Expression. Retrieved from <https://www.geeksforgeeks.org/>

 Python Documentation: Tkinter, <https://docs.python.org/3/library/tkinter.html>