D24DCS155

Practical-2

**Aim:**

You are designing a simple text editor that uses an appropriate data structure to manage undo operations. Every time a user makes a change to the text, the state of the text before the change is inserted onto a data strucure. If the user decides to undo a change, the editor will delete the top state from the data structure and revert the text to that state. Additionally, the user should be able to see the list of recent changes that can be undone.

**1) Program (in Python):**

**def create\_text\_editor():**

**text = ""**

**undo\_stack = []**

**return text, undo\_stack**

**def insert\_text(text, undo\_stack, new\_text):**

**undo\_stack.append(text)**

**text += new\_text**

**return text, undo\_stack**

**def delete\_text(text, undo\_stack, num\_chars):**

**undo\_stack.append(text)**

**text = text[:-num\_chars]**

**return text, undo\_stack**

**def undo(text, undo\_stack):**

**if undo\_stack:**

**text = undo\_stack.pop()**

**return text, undo\_stack**

**def get\_undo\_list(undo\_stack):**

**return undo\_stack**

**def get\_current\_text(text):**

**return text**

**text, undo\_stack = create\_text\_editor()**

**text, undo\_stack = insert\_text(text, undo\_stack, "Hello, ")**

**print(get\_current\_text(text)) # Output: Hello,**

**text, undo\_stack = insert\_text(text, undo\_stack, "world!")**

**print(get\_current\_text(text)) # Output: Hello, world!**

**text, undo\_stack = undo(text, undo\_stack)**

**print(get\_current\_text(text)) # Output: Hello,**

**text, undo\_stack = undo(text, undo\_stack)**

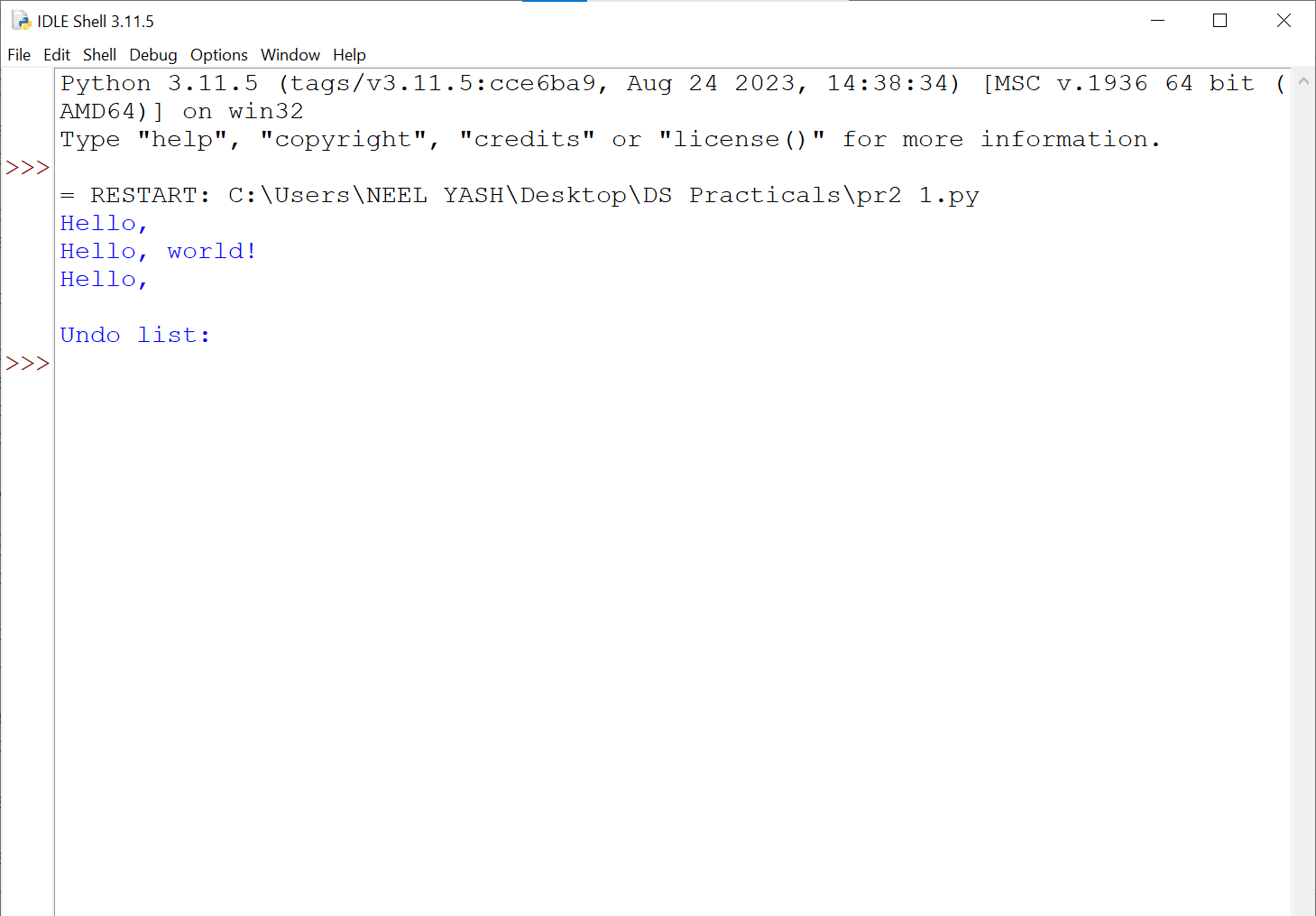
**print(get\_current\_text(text)) # Output: (empty string)**

**print("Undo list:")**

**for i, txt in enumerate(get\_undo\_list(undo\_stack)):**

**print(f"{i+1}. {txt}")**

**Output:**

****

**Fig.1: Text Editor**

**Conclusion:**

In conclusion, the code implements a simple text editor with undo functionality using a stack data structure. It demonstrates a basic understanding of stack operations and their application in text editing. Overall, the code is well-structured and effectively demonstrates the concept of undo functionality.

**Aim:** Implement a program to convert infix notation to postfix notation using appropriate data structure.

**2) Program (in Python):**

**precedence = {'+': 1, '-': 1, '\*': 2, '/': 2}**

**operator\_stack = []**

**def infix\_to\_postfix(infix\_expression):**

**postfix\_expression = []**

**for i in infix\_expression.split():**

**if i.isalnum():**

**postfix\_expression.append(i)**

**elif i == '(':**

**operator\_stack.append(i)**

**elif i == ')':**

**while operator\_stack[-1] != '(':**

**postfix\_expression.append(operator\_stack.pop())**

**operator\_stack.pop() # Remove the '('**

**else:**

**while (operator\_stack and**

**operator\_stack[-1] != '(' and**

**precedence[operator\_stack[-1]] >= precedence[i]):**

**postfix\_expression.append(operator\_stack.pop())**

**operator\_stack.append(i)**

**while operator\_stack:**

**postfix\_expression.append(operator\_stack.pop())**

**return ' '.join(postfix\_expression)**

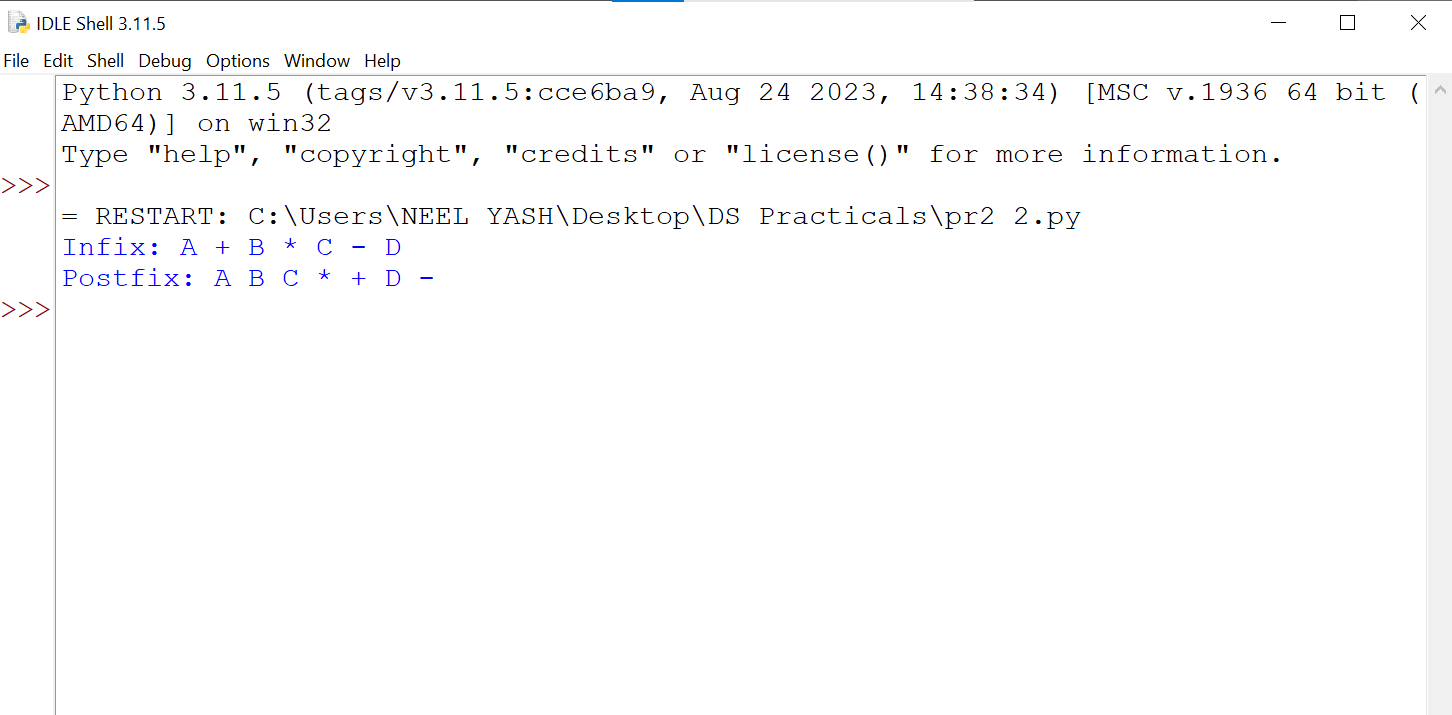
**infix\_expression = "A + B \* C - D"**

**postfix\_expression = infix\_to\_postfix(infix\_expression)**

**print(f"Infix: {infix\_expression}")**

**print(f"Postfix: {postfix\_expression}")**

**Output :**

****

**Fig 1 : Expression Conversion**

**Conclusion :**

In conclusion, the program successfully implements the conversion of infix notation to postfix notation using a stack data structure. The algorithm correctly parses the infix expression, applies the operator precedence rules, and constructs the equivalent postfix expression. This implementation provides a reliable and efficient solution for converting infix notation to postfix notation.