Rajalakshmi Engineering College

Name: Soameshwaran D
Email: 240701520@rajalakshmi.edu.in
Roll no: 240701520
Phone: 7358671540
Branch: REC
Department: I CSE FE
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 20

Section 1 : Coding

1. Problem Statement

Rithi is building a simple text editor that allows users to type characters, undo their typing, and view the current text. She has implemented this text editor using an array-based stack data structure.

She has to develop a basic text editor with the following features:

Type a Character (Push): Users can type a character and add it to the text editor.Undo Typing (Pop): Users can undo their typing by removing the last character they entered from the editor.View Current Text (Display): Users can view the current text in the editor, which is the sequence of characters in the buffer.Exit: Users can exit the text editor application.

Write a program that simulates this text editor's undo feature using a character stack and implements the push, pop and display operations accordingly.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the character onto the stack. If the choice is 1, the following input is a space-separated character, representing the character to be pushed onto the stack.

Choice 2: Pop the character from the stack.

Choice 3: Display the characters in the stack.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

1. If the choice is 1, print: "Typed character: <character>" where <character> is the character that was pushed to the stack.
2. If the choice is 2, print: "Undo: Removed character <character>" where <character> is the character that was removed from the stack.
3. If the choice is 2, and if the stack is empty without any characters, print "Text editor buffer is empty. Nothing to undo."
4. If the choice is 3, print: "Current text: <character1> <character2> ... <characterN>" where <character1>, <character2>, ... are the characters in the stack, starting from the last pushed character.
5. If the choice is 3, and there are no characters in the stack, print "Text editor buffer is empty."
6. If the choice is 4, exit the program.
7. If any other choice is entered, print "Invalid choice"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1 H
1 A
3

4
Output: Typed character: H
Typed character: A
Current text: A H

Answer

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

char stack[MAX];
int top = -1;

// Function to push a character onto the stack
void push(char ch) {
    if (top >= MAX - 1) {
        printf("Text editor buffer is full. Cannot type more characters.\n");
        return;
    }
    stack[++top] = ch;
    printf("Typed character: %c\n", ch);
}

// Function to pop a character from the stack
void pop() {
    if (top == -1) {
        printf("Text editor buffer is empty. Nothing to undo.\n");
    } else {
        printf("Undo: Removed character %c\n", stack[top--]);
    }
}

// Function to display current text
void display() {
    if (top == -1) {
        printf("Text editor buffer is empty.\n");
    } else {
        printf("Current text: ");
        for (int i = top; i >= 0; i--) {
            printf("%c ", stack[i]);
```

```
        }
        printf("\n");
    }
}

int main() {
    int choice;
    char ch;

    while (1) {
        if (scanf("%d", &choice) != 1) {
            // Clear invalid input
            while (getchar() != '\n');
            printf("Invalid choice\n");
            continue;
        }

        switch (choice) {
            case 1:
                if (scanf(" %c", &ch) == 1) {
                    push(ch);
                } else {
                    printf("Invalid input for typing a character.\n");
                }
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

Status : Correct *Marks : 10/10*

2. Problem Statement

Latha is taking a computer science course and has recently learned about infix and postfix expressions. She is fascinated by the idea of converting infix expressions into postfix notation. To practice this concept, she wants to implement a program that can perform the conversion for her.

Help Latha by designing a program that takes an infix expression as input and outputs its equivalent postfix notation.

Example

Input:

(3+4)5

Output:

34+5

Input Format

The input consists of a string, the infix expression to be converted to postfix notation.

Output Format

The output displays a string, the postfix expression equivalent of the input infix expression.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: A+B*C-D/E

Output: ABC*+DE/-

Answer

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <ctype.h>
#include <string.h>

#define MAX 100

char stack[MAX];
int top = -1;

// Push to stack
void push(char ch) {
    if (top < MAX - 1) {
        stack[++top] = ch;
    }
}

// Pop from stack
char pop() {
    if (top >= 0) {
        return stack[top--];
    }
    return '\0';
}

// Peek top of stack
char peek() {
    if (top >= 0) return stack[top];
    return '\0';
}

// Check if character is operator
int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

// Get precedence of operator
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}

// Main conversion function
```

```
void infixToPostfix(char* infix) {
    char postfix[MAX] = "";
    int j = 0;

    for (int i = 0; i < strlen(infix); i++) {
        char ch = infix[i];

        // If operand, add to output
        if (isalnum(ch)) {
            postfix[j++] = ch;

            // Handle implicit multiplication e.g., (3+4)5 or 5(3+4)
            if (i + 1 < strlen(infix) && (infix[i + 1] == '(')) {
                while (top != -1 && precedence('*') <= precedence(peek())) {
                    postfix[j++] = pop();
                }
                push('*');
            }
        }

        // If '(', push to stack
        else if (ch == '(') {
            if (i > 0 && isalnum(infix[i - 1])) {
                // Implicit multiplication before '('
                while (top != -1 && precedence('*') <= precedence(peek())) {
                    postfix[j++] = pop();
                }
                push('*');
            }
            push(ch);
        }

        // If ')', pop till '('
        else if (ch == ')') {
            while (top != -1 && peek() != '(') {
                postfix[j++] = pop();
            }
            if (peek() == '(') {
                pop(); // remove '('
            }
        }
```

```
        // If operator
        else if (isOperator(ch)) {
            while (top != -1 && precedence(ch) <= precedence(peek())) {
                postfix[j++] = pop();
            }
            push(ch);
        }
    }

    // Pop remaining operators
    while (top != -1) {
        postfix[j++] = pop();
    }

    postfix[j] = '\0';
    printf("%s\n", postfix);
}

int main() {
    char infix[MAX];
    scanf("%s", infix);
    infixToPostfix(infix);
    return 0;
}
```

Status : Correct **Marks : 10/10**

3. Problem Statement

Suppose you are building a calculator application that allows users to enter mathematical expressions in infix notation. One of the key features of your calculator is the ability to convert the entered expression to postfix notation using a Stack data structure.

Write a function to convert infix notation to postfix notation using a Stack.

Input Format

The input consists of a string, an infix expression that includes only digits(0-9), and operators(+, -, *, /).

Output Format

The output displays the equivalent postfix expression of the given infix expression.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1+2*3/4-5

Output: 123*4/+5-

Answer

```
// You are using GCC
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return not self.items

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        return None

    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        return None

def precedence(operator):
    if operator in ('+', '-'):
        return 1
    elif operator in ('*', '/'):
        return 2
    return 0
```

```
def infix_to_postfix(infix_expression):
    output = []
    operators = Stack()
    tokens = []
    current_number = ""
    for char in infix_expression:
        if char.isdigit():
            current_number += char
        else:
            if current_number:
                tokens.append(current_number)
                current_number = ""
            tokens.append(char)
    if current_number:
        tokens.append(current_number)

    for token in tokens:
        if token.isdigit():
            output.append(token)
        elif token == '(':
            operators.push(token)
        elif token == ')':
            while not operators.is_empty() and operators.peek() != '(':
                output.append(operators.pop())
            operators.pop()  # Pop the '('
        else:
            while not operators.is_empty() and precedence(operators.peek()) >=
precedence(token):
                output.append(operators.pop())
            operators.push(token)

    while not operators.is_empty():
        output.append(operators.pop())

    return "".join(output)

if __name__ == "__main__":
    infix1 = "1+2*3/4-5"
    postfix1 = infix_to_postfix(infix1)
    print(f"Input 1 : \n\n{infix1}\nOutput 1 : \n\n{postfix1}")
```

```
infix2 = "5+6-4*8/2"
postfix2 = infix_to_postfix(infix2)
print(f"\nInput 2 : \n\n{infix2}\nOutput 2 : \n\n{postfix2}")
```

Status : Wrong ***Marks : 0/10***