Data Structure and Algorithms Lab

Code: PMDS605P

Digital Assignment 1

Name: Soumyadeep Ganguly

Reg. No.: 24MDT0082

Course: M.Sc. in Data Science

Convert infix to postfix

```c
// infix to prefix

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

struct stack {
    char data;
    struct stack *bottom;
};

struct stack *createNode() {
    return (struct stack *)malloc(sizeof(struct stack));
}

int isEmpty(struct stack *top) {
    return top == NULL;
}

struct stack *push(struct stack *top, char data) {
    struct stack *ptr = createNode();
    if (!ptr) {
        printf("Stack Overflowed!");
        return top;
    }
    ptr->data = data;
    ptr->bottom = top;
    return ptr;
}

char pop(struct stack **top) {
    if (isEmpty(*top)) {
        printf("Stack is empty!");
        return -1;
    }
    struct stack *ptr = *top;
    char data = ptr->data;
    *top = (*top)->bottom;
    free(ptr);
    return data;
}

char peek(struct stack *top) {
    return isEmpty(top) ? -1 : top->data;
}

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

int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}
```

```c
void reverseString(char *str) {
    int n = strlen(str);
    for (int i = 0; i < n / 2; i++) {
        char temp = str[i];
        str[i] = str[n - i - 1];
        str[n - i - 1] = temp;
    }
}

void infixToPrefix(char *infix) {
    reverseString(infix);
    struct stack *opStack = NULL;
    char prefix[strlen(infix) + 1];
    int j = 0;

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

        if (isalnum(ch)) {
            prefix[j++] = ch;
        } else if (ch == ')') {
            opStack = push(opStack, ch);
        } else if (ch == '(') {
            while (!isEmpty(opStack) && peek(opStack) != ')') {
                prefix[j++] = pop(&opStack);
            }
            pop(&opStack);
        } else if (isOperator(ch)) {
            while (!isEmpty(opStack) && precedence(peek(opStack)) > precedence(ch)) {
                prefix[j++] = pop(&opStack);
            }
            opStack = push(opStack, ch);
        }
    }

    while (!isEmpty(opStack)) {
        prefix[j++] = pop(&opStack);
    }
    prefix[j] = '\0';

    reverseString(prefix);
    printf("Prefix Expression: %s\n", prefix);
}

int main() {
    char infix[] = "a+b*c-d";
    printf("Infix Expression: %s\n", infix);
    infixToPrefix(infix);
    return 0;
}
```

OUTPUT: