Data Structure and Algorithms Lab

Code: PMDS605P

Digital Assignment 2

Name: Soumyadeep Ganguly

Reg. No.: 24MDT0082

Course: M.Sc. in Data Science

Convert infix to prefix

```c
1   // infix to prefix
2
3   #include <stdio.h>
4   #include <stdlib.h>
5   #include <string.h>
6   #include <ctype.h>
7
8   struct stack {
9       char data;
10      struct stack *bottom;
11  };
12
13  struct stack *createNode() {
14      return (struct stack *)malloc(sizeof(struct stack));
15  }
16
17  int isEmpty(struct stack *top) {
18      return top == NULL;
19  }
20
21  struct stack *push(struct stack *top, char data) {
22      struct stack *ptr = createNode();
23      if (!ptr) {
24          printf("Stack Overflowed!");
25          return top;
26      }
27      ptr->data = data;
28      ptr->bottom = top;
29      return ptr;
30  }
31
32  char pop(struct stack **top) {
33      if (isEmpty(*top)) {
34          printf("Stack is empty!");
35          return -1;
36      }
37      struct stack *ptr = *top;
38      char data = ptr->data;
39      *top = (*top)->bottom;
40      free(ptr);
41      return data;
42  }
43
44  char peek(struct stack *top) {
45      return isEmpty(top) ? -1 : top->data;
46  }
47
48  int precedence(char op) {
49      if (op == '+' || op == '-') return 1;
50      if (op == '*' || op == '/') return 2;
51      return 0;
52  }
53
54  int isOperator(char ch) {
55      return ch == '+' || ch == '-' || ch == '*' || ch == '/';
56  }
```

```c
1   void reverseString(char *str) {
2       int n = strlen(str);
3       for (int i = 0; i < n / 2; i++) {
4           char temp = str[i];
5           str[i] = str[n - i - 1];
6           str[n - i - 1] = temp;
7       }
8   }
9
10  void infixToPrefix(char *infix) {
11      reverseString(infix);
12      struct stack *opStack = NULL;
13      char prefix[strlen(infix) + 1];
14      int j = 0;
15
16      for (int i = 0; i < strlen(infix); i++) {
17          char ch = infix[i];
18
19          if (isalnum(ch)) {
20              prefix[j++] = ch;
21          } else if (ch == ')') {
22              opStack = push(opStack, ch);
23          } else if (ch == '(') {
24              while (!isEmpty(opStack) && peek(opStack) != ')') {
25                  prefix[j++] = pop(&opStack);
26              }
27              pop(&opStack);
28          } else if (isOperator(ch)) {
29              while (!isEmpty(opStack) && precedence(peek(opStack)) > precedence(ch)) {
30                  prefix[j++] = pop(&opStack);
31              }
32              opStack = push(opStack, ch);
33          }
34      }
35
36      while (!isEmpty(opStack)) {
37          prefix[j++] = pop(&opStack);
38      }
39      prefix[j] = '\0';
40
41      reverseString(prefix);
42      printf("Prefix Expression: %s\n", prefix);
43  }
44
45  int main() {
46      char infix[] = "a+b*c-d";
47      printf("Infix Expression: %s\n", infix);
48      infixToPrefix(infix);
49      return 0;
50  }
```

OUTPUT:

```
Infix Expression: a+b*c-d
Prefix Expression: -+a*bcd
PS E:\VIT Study Materials\SEM 2\DSA\LAB>
```