Rajalakshmi Engineering College

Name: Nitin Aakash
Email: 240701370@rajalakshmi.edu.in
Roll no: 240701370
Phone: 9498349045
Branch: REC
Department: I CSE FD
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 : 30

Section 1 : Coding

1. Problem Statement

You are required to implement a stack data structure using a singly linked list that follows the Last In, First Out (LIFO) principle.

The stack should support the following operations: push, pop, display, and peek.

Input Format

The input consists of four space-separated integers N, representing the elements to be pushed onto the stack.

Output Format

The first line of output displays all four elements in a single line separated by a space.

The second line of output is left blank to indicate the pop operation without displaying anything.

The third line of output displays the space separated stack elements in the same line after the pop operation.

The fourth line of output displays the top element of the stack using the peek operation.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 11 22 33 44
Output: 44 33 22 11

33 22 11
33

Answer

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int data;
   struct Node* next;
};

struct Stack {
   struct Node* top;
};

struct Stack* createStack() {
   struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
   stack->top = NULL;
   return stack;
}

int isEmpty(struct Stack* stack) {
   return stack->top == NULL;
```

```
}

void push(struct Stack* stack, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = stack->top;
   stack->top = newNode;
}

int pop(struct Stack* stack) {
   if (isEmpty(stack)) {
      return -1;
   }
   struct Node* temp = stack->top;
   int poppedData = temp->data;
   stack->top = stack->top->next;
   free(temp);
   return poppedData;
}

int peek(struct Stack* stack) {
   if (isEmpty(stack)) {
      return -1;
   }
   return stack->top->data;
}

void display(struct Stack* stack) {
   struct Node* current = stack->top;
   while (current != NULL) {
      printf("%d ", current->data);
      current = current->next;
   }
}

int main() {
   struct Stack* stack = createStack();
   int elements[4];

   printf("");
   for (int i = 0; i < 4; i++) {
      scanf("%d", &elements[i]);
```

```
        push(stack, elements[i]);
    }

    display(stack);
    printf("\n");

    pop(stack);

    display(stack);
    printf("\n");

    int topElement = peek(stack);
    if (topElement != -1) {
        printf("%d\n", topElement);
    } else {
        printf("Stack is empty\n");
    }

    while (!isEmpty(stack)) {
        pop(stack);
    }
    free(stack);

    return 0;
}
```

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

2. 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

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

#define MAX 100
typedef struct {
   int top;
   char items[MAX];
} Stack;
void initStack(Stack* stack) {
   stack->top = -1;
}
int isEmpty(Stack* stack) {
   return stack->top == -1;
}
void push(Stack* stack, char item) {
   if (stack->top < MAX - 1) {
      stack->items[++stack->top] = item;
   }
}
char pop(Stack* stack) {
   if (!isEmpty(stack)) {
      return stack->items[stack->top--];
   }
   return '\0';
}
```

```
char peek(Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->items[stack->top];
    }
    return '\0';
}
int precedence(char operators) {
    switch (operators) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return 0;
    }
}
void infixToPostfix(char* infix, char* postfix) {
    Stack stack;
    initStack(&stack);
    int j = 0;

    for (int i = 0; infix[i]; i++) {
        char c = infix[i];
        if (isalnum(c)) {
            postfix[j++] = c;
        }
        else if (c == '(') {
            push(&stack, c);
        }
        else if (c == ')') {
            while (!isEmpty(&stack) && peek(&stack) != '(') {
                postfix[j++] = pop(&stack);
            }
            pop(&stack);
        }
        else {
            while (!isEmpty(&stack) && precedence(peek(&stack)) >= precedence(c))
{
```

```
            postfix[j++] = pop(&stack);
        }
        push(&stack, c);
      }
   }
   while (!isEmpty(&stack)) {
      postfix[j++] = pop(&stack);
   }
   postfix[j] = '\0';
}
int main() {
   char infix[MAX];
   char postfix[MAX];

   printf("");
   fgets(infix, sizeof(infix), stdin);
   infix[strcspn(infix, "\n")] = 0;

   infixToPostfix(infix, postfix);
   printf("%s\n", postfix);

   return 0;
}
```

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

3. 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

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX 100
typedef struct {
   int top;
   char items[MAX];
} Stack;
void initStack(Stack* stack) {
   stack->top = -1;
}
int isEmpty(Stack* stack) {
   return stack->top == -1;
}
void push(Stack* stack, char item) {
   if (stack->top < MAX - 1) {
      stack->items[++stack->top] = item;
   }
```

```
}
char pop(Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->items[stack->top--];
    }
    return '\0';
}
char peek(Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->items[stack->top];
    }
    return '\0';
}
int precedence(char operators) {
    switch (operators) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return 0;
    }
}
void infixToPostfix(char* infix, char* postfix) {
    Stack stack;
    initStack(&stack);
    int j = 0;

    for (int i = 0; infix[i]; i++) {
        char c = infix[i];
        if (isalnum(c)) {
            postfix[j++] = c;
        }
        else if (c == '(') {
            push(&stack, c);
        }
        else if (c == ')') {
            while (!isEmpty(&stack) && peek(&stack) != '(') {
```

```
            postfix[j++] = pop(&stack);
         }
         pop(&stack);
      }
      else {
         while (!isEmpty(&stack) && precedence(peek(&stack)) >= precedence(c))
{
            postfix[j++] = pop(&stack);
         }
         push(&stack, c);
      }
   }
   while (!isEmpty(&stack)) {
      postfix[j++] = pop(&stack);
   }
   postfix[j] = '\0';
}
int main() {
   char infix[MAX];
   char postfix[MAX];

   printf("");
   fgets(infix, sizeof(infix), stdin);
   infix[strcspn(infix, "\n")] = 0;

   infixToPostfix(infix, postfix);
   printf("%s\n", postfix);

   return 0;
}
```

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