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

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

#define MAX 30

// Stack structure
struct Stack {
    char arr[MAX];
    int top;
};

// Function to initialize stack
void initStack(struct Stack* stack) {
    stack->top = -1;
}

// Function to check if stack is empty
int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

// Function to push an element onto the stack
void push(struct Stack* stack, char value) {
    if (stack->top < MAX - 1) {
        stack->arr[++stack->top] = value;
    }
}

// Function to pop an element from the stack
```

```
char pop(struct Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->arr[stack->top--];
    }
    return '\0';
}

// Function to get top element of stack
char peek(struct Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->arr[stack->top];
    }
    return '\0';
}

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

// Function to convert infix to postfix
void infixToPostfix(char* infix, char* postfix) {
    struct Stack stack;
    initStack(&stack);
    int i = 0, j = 0;

    while (infix[i] != '\0') {
        if (isdigit(infix[i])) {
            postfix[j++] = infix[i];
        } else if (infix[i] == '(') {
            push(&stack, infix[i]);
        } else if (infix[i] == ')') {
            while (!isEmpty(&stack) && peek(&stack) != '(') {
                postfix[j++] = pop(&stack);
            }
            pop(&stack); // Remove '('
        } else {
            while (!isEmpty(&stack) && precedence(peek(&stack)) >=
precedence(infix[i])) {
                postfix[j++] = pop(&stack);
```

```
            }
            push(&stack, infix[i]);
        }
        i++;
    }

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

int main() {
    char infix[MAX], postfix[MAX];
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("%s\n", postfix);
    return 0;
}
```

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

2. 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;
};

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

void pop(struct Node** top) {
    if (*top == NULL) return;

    struct Node* temp = *top;
    *top = (*top)->next;
```

```
    free(temp);
}

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

int peek(struct Node* top) {
    if (top != NULL) return top->data;
    return -1;
}

int main() {
    struct Node* stack = NULL;
    int values[4];

    for (int i = 0; i < 4; i++) {
        scanf("%d", &values[i]);
        push(&stack, values[i]);
    }

    display(stack);

    pop(&stack);
    printf("\n");

    display(stack);

    printf("%d\n", peek(stack));

    return 0;
}
```

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

3. Problem Statement

Raj is a software developer, and his team is building an application that processes user inputs in the form of strings containing brackets. One of the essential features of the application is to validate whether the input string meets specific criteria.

During testing, Raj inputs the string "(([])){}". The application correctly returns "Valid string" because the input satisfies the criteria: every opening bracket (, [, and { has a corresponding closing bracket),], and }, arranged in the correct order.

Next, Raj tests the application with the string "([)]". This time, the application correctly returns "Invalid string" because the opening bracket [is incorrectly closed by the bracket), which violates the validation rules.

Finally, Raj enters the string "{[()]}". The application correctly identifies it as a "Valid string" since all opening brackets are matched with the corresponding closing brackets in the correct order.

As a software developer, Raj's responsibility is to ensure that the application works reliably and produces accurate results for all input strings, following the validation rules. He accomplishes this by using a method for solving such problems.

Input Format

The input comprises a string representing a sequence of brackets that need to be validated.

Output Format

The output prints "Valid string" if the string is valid. Otherwise, it prints "Invalid string".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: (([])){}
Output: Valid string

Answer

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

#define MAX 100

struct Stack {
    char arr[MAX];
    int top;
};

void initStack(struct Stack* stack) {
    stack->top = -1;
}

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

void push(struct Stack* stack, char value) {
    if (stack->top < MAX - 1) {
        stack->arr[++stack->top] = value;
    }
}

char pop(struct Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->arr[stack->top--];
    }
    return '\0';
}

int isMatchingPair(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '[' && close == ']') ||
           (open == '{' && close == '}');
}

int isValidString(char* str) {
    struct Stack stack;
    initStack(&stack);
```

```
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == '(' || str[i] == '[' || str[i] == '{') {
            push(&stack, str[i]);
        } else if (str[i] == ')' || str[i] == ']' || str[i] == '}') {
            if (isEmpty(&stack) || !isMatchingPair(pop(&stack), str[i])) {
                return 0;
            }
        }
    }

    return isEmpty(&stack);
}

int main() {
    char str[MAX];
    scanf("%s", str);

    if (isValidString(str))
        printf("Valid string\n");
    else
        printf("Invalid string\n");

    return 0;
}
```

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