Rajalakshmi Engineering College

Name: KEERTHI PRIYA T
Email: 240701258@rajalakshmi.edu.in
Roll no: 2116240701258
Phone: 7397397221
Branch: REC
Department: I CSE FC
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

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

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

// Top of stack
struct Node* top = NULL;

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

```
// Pop operation
void pop() {
    if (top != NULL) {
        struct Node* temp = top;
        top = top->next;
        free(temp);
    }
}

// Peek operation
int peek() {
    if (top != NULL) {
        return top->data;
    }
    return -1; // Assuming -1 indicates an empty stack
}

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

int main() {
    int a, b, c, d;
    scanf("%d %d %d %d", &a, &b, &c, &d);

    // Push values to stack
    push(a);
    push(b);
    push(c);
    push(d);

    // Display after all pushes
    display();
```

```
    // Pop top element
    pop();

    // Blank line
    printf("\n");

    // Display after pop
    display();

    // Peek top element
    printf("%d\n", peek());

    return 0;
}
```

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

2. 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

// Stack implementation
char stack[MAX];
int top = -1;

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

// Pop operation
char pop() {
```

```
    if (top >= 0) {
        return stack[top--];
    }
    return '\0'; // Empty stack
}

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

// Function to check for matching pairs
int isMatchingPair(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '[' && close == ']') ||
           (open == '{' && close == '}');
}

// Function to check if the string is valid
int isValid(char *str) {
    for (int i = 0; i < strlen(str); i++) {
        char ch = str[i];
        if (ch == '(' || ch == '[' || ch == '{') {
            push(ch);
        } else if (ch == ')' || ch == ']' || ch == '}') {
            if (top == -1 || !isMatchingPair(pop(), ch)) {
                return 0; // Invalid
            }
        }
    }
    return top == -1; // Valid if stack is empty
}

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

    if (isValid(input)) {
        printf("Valid string\n");
```

```
    } else {
        printf("Invalid string\n");
    }

    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

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

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

#define MAX 100

// Stack structure for operators
char stack[MAX];
int top = -1;

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

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

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

// Function to check if character is operator
int isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}

// Function to return precedence of operators
int precedence(char c) {
    if (c == '+' || c == '-') return 1;
    if (c == '*' || c == '/') return 2;
    return 0;
}

// Function to convert infix to postfix
```

```
void infixToPostfix(char* infix) {
    int i = 0;
    while (infix[i]) {
        char c = infix[i];

        if (isdigit(c)) {
            // Print number (operand) directly
            printf("%c", c);
        } else if (c == '(') {
            push(c);
        } else if (c == ')') {
            while (top != -1 && peek() != '(')
                printf("%c", pop());
            pop(); // Remove '(' from stack
        } else if (isOperator(c)) {
            while (top != -1 && precedence(peek()) >= precedence(c))
                printf("%c", pop());
            push(c);
        }
        i++;
    }

    // Pop remaining operators
    while (top != -1)
        printf("%c", pop());

    printf("\n");
}

// Main function
int main() {
    char infix[31];
    scanf("%s", infix);
    infixToPostfix(infix);
    return 0;
}
```

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