Rajalakshmi Engineering College

Name: Pranav Narayanan
Email: 240701393@rajalakshmi.edu.in
Roll no:
Phone: 9500579427
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>

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

Node* top = NULL;

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

// Pop operation
int pop() {
    if (top == NULL) return -1; // stack empty
    int val = top->data;
    Node* temp = top;
    top = top->next;
    free(temp);
    return val;
}

// Peek operation
int peek() {
    if (top == NULL) return -1;
    return top->data;
}

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

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

    // Push elements in input order so that last input is on top
    for (int i = 0; i < 4; i++) {
        push(nums[i]);
    }

    // Display stack (top to bottom)
    display();

    // Pop operation (no output)
    pop();
    // Display stack 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

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

#define MAX 100

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

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

// Pop function
char pop() {
   if (top >= 0) {
      return stack[top--];
   }
   return '\0'; // Return null char if stack empty
}

// Check if stack is empty
int isEmpty() {
    return top == -1;
}

// Function to check if brackets match
int isMatchingPair(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '[' && close == ']') ||
           (open == '{' && close == '}');
}

int isValidBracketString(char *str) {
    int i, len = strlen(str);
    for (i = 0; i < len; i++) {
        char ch = str[i];

        if (ch == '(' || ch == '[' || ch == '{') {
            push(ch);
        }
        else if (ch == ')' || ch == ']' || ch == '}') {
            if (isEmpty()) {
                return 0; // Invalid: closing bracket without matching opening
            }
            char topChar = pop();
            if (!isMatchingPair(topChar, ch)) {
                return 0; // Invalid: mismatched brackets
            }
        }
        else {
            // Invalid character found
            return 0;
        }
    }
    // If stack empty, all matched
    return isEmpty();
}

int main() {
    char input[MAX + 1];
    scanf("%s", input);

    if (isValidBracketString(input)) {
        printf("Valid string\n");
    } else {
        printf("Invalid string\n");
    }

    return 0;
}
```

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

3. Problem Statement

In an educational setting, Professor Smith tasks Computer Science students with designing an algorithm to evaluate postfix expressions efficiently, fostering problem-solving skills and understanding of stack-based computations.

The program prompts users to input a postfix expression, evaluates it, and displays the result, aiding students in honing their coding abilities.

Input Format

The input consists of the postfix mathematical expression.

The expression will contain real numbers and mathematical operators (+, -, *, /), without any space.

Output Format

The output prints the result of evaluating the given postfix expression.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 82/
Output: 4

Answer

```
// You are using GCC
```

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

#define MAX 100

double stack[MAX];
int top = -1;

void push(double val) {
    if (top < MAX - 1) {
        stack[++top] = val;
    }
}

double pop() {
    if (top >= 0) {
        return stack[top--];
    }
    return 0; // Should not happen for valid input
}

int main() {
    char expr[MAX + 1];
    scanf("%s", expr);

    for (int i = 0; expr[i] != '\0'; i++) {
        char ch = expr[i];

        if (isdigit(ch)) {
            // Convert char digit to number and push
            push(ch - '0');
        } else {
            // Operator encountered, pop two operands
            double b = pop();
            double a = pop();
            double res = 0;

            switch (ch) {
                case '+': res = a + b; break;
                case '-': res = a - b; break;
                case '*': res = a * b; break;
                case '/': res = a / b; break;
                default:
                    printf("Invalid operator\n");
                    return 1;
            }
            push(res);
        }
    }

    // Result will be at the top of the stack
    double result = pop();

    // Print result without decimal if whole number, else with decimal
    if (result == (int)result) {
        printf("%d\n", (int)result);
    } else {
        printf("%lf\n", result);
    }

    return 0;
}
```

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