Rajalakshmi Engineering College

Name: Ritesh Sivakumar
Email: 240701427@rajalakshmi.edu.in
Roll no: 240701427
Phone: 9342061449
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

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

#define MAX_SIZE 100

struct Stack {
    int data[MAX_SIZE];
    int top;
};

void push(struct Stack* stack, int value) {
    if (stack->top == MAX_SIZE - 1)
        return;
    stack->data[++stack->top] = value;
}

void pop(struct Stack* stack) {
    if (stack->top == -1)
        return;
    stack->top--;
```

```
}
int peek(struct Stack* stack) {
    if (stack->top == -1)
        return -1;
    return stack->data[stack->top];
}

void display(struct Stack* stack) {
    for (int i = stack->top; i >= 0; i--)
        printf("%d ", stack->data[i]);
    printf("\n");
}

int main() {
    struct Stack stack = {{0}, -1};
    int numbers[4];

    for (int i = 0; i < 4; i++)
        scanf("%d", &numbers[i]);

    for (int i = 0; i < 4; i++)
        push(&stack, numbers[i]);

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

    return 0;
}
```

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

2. Problem Statement

Buvi is working on a project that requires implementing an array-stack data structure with an additional feature to find the minimum element.

Buvi needs to implement a program that simulates a stack with the following functionalities:

Push: Adds an element onto the stack.Pop: Removes the top element from the stack.Find Minimum: Finds the minimum element in the stack.

Buvi's implementation should efficiently handle these operations with a maximum stack size of 20.

Input Format

The first line of input consists of an integer N, representing the number of elements to push onto the stack.

The second line consists of N space-separated integer values, representing the elements to be pushed onto the stack.

Output Format

The first line of output displays "Minimum element in the stack: " followed by the minimum element in the stack after pushing all elements.

The second line displays "Popped element: " followed by the popped element.

The third line displays "Minimum element in the stack after popping: " followed by the minimum element in the stack after popping one element.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 4
5 2 8 1
Output: Minimum element in the stack: 1
Popped element: 1
Minimum element in the stack after popping: 2

Answer

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

```
#define MAX_SIZE 20

int stack[MAX_SIZE], min_stack[MAX_SIZE];
int top = -1, min_top = -1;

void push(int value) {
    if (top == MAX_SIZE - 1) return;
    stack[++top] = value;
    if (min_top == -1 || value <= min_stack[min_top]) {
        min_stack[++min_top] = value;
    }
}

int pop() {
    if (top == -1) return -1;
    int popped = stack[top--];
    if (popped == min_stack[min_top]) {
        min_top--;
    }
    return popped;
}

int find_min() {
    return (min_top != -1) ? min_stack[min_top] : -1;
}

int main() {
    int N, value;
    scanf("%d", &N);

    for (int i = 0; i < N; i++) {
        scanf("%d", &value);
        push(value);
    }

    printf("Minimum element in the stack: %d\n", find_min());
    printf("Popped element: %d\n", pop());
    printf("Minimum element in the stack after popping: %d\n", find_min());

    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

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

#define MAX_SIZE 100

typedef struct {
    char data[MAX_SIZE];
    int top;
} Stack;

void push(Stack *stack, char ch) {
    if (stack->top == MAX_SIZE - 1) return;
    stack->data[++stack->top] = ch;
}

char pop(Stack *stack) {
    if (stack->top == -1) return '\0';
    return stack->data[stack->top--];
}

char peek(Stack *stack) {
    if (stack->top == -1) return '\0';
    return stack->data[stack->top];
}

int is_valid(char *str) {
    Stack stack;
    stack.top = -1;

    for (int i = 0; i < strlen(str); i++) {
        if (str[i] == '(' || str[i] == '[' || str[i] == '{') {
            push(&stack, str[i]);
        } else if (str[i] == ')' || str[i] == ']' || str[i] == '}') {
            char top = peek(&stack);
```

```
            if ((str[i] == ')' && top == '(') ||
                (str[i] == ']' && top == '[') ||
                (str[i] == '}' && top == '{')) {
                pop(&stack);
            } else {
                return 0;
            }
        }
    }

    return stack.top == -1;
}

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

    if (is_valid(str)) {
        printf("Valid string\n");
    } else {
        printf("Invalid string\n");
    }

    return 0;
}
```

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