Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

In a coding competition, you are assigned a task to create a program that simulates a stack using a linked list.

The program should feature a menu-driven interface for pushing an integer to stack, popping, and displaying stack elements, with robust error handling for stack underflow situations. This challenge tests your data structure skills.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the integer value onto the stack. If the choice is 1, the following input is a space-separated integer, representing the element to be pushed onto the stack.

Choice 2: Pop the integer from the stack.

Choice 3: Display the elements in the stack.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

If the choice is 1, push the given integer to the stack and display the following: "Pushed element: " followed by the value pushed.

If the choice is 2, pop the integer from the stack and display the following: "Popped element: " followed by the value popped.

If the choice is 2, and if the stack is empty without any elements, print "Stack is empty. Cannot pop."

If the choice is 3, print the elements in the stack: "Stack elements (top to bottom): " followed by the space-separated values.

If the choice is 3, and there are no elements in the stack, print "Stack is empty".

If the choice is 4, exit the program and display the following: "Exiting program".

If any other choice is entered, print "Invalid choice".

Refer to the sample input and output for the exact format.

Sample Test Case

Input: 1 3
1 4
3
2
3
4
Output: Pushed element: 3
Pushed element: 4
Stack elements (top to bottom): 4 3
Popped element: 4
Stack elements (top to bottom): 3
Exiting program

Answer

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

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

struct Node* top = NULL;

void push(int value)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = top;
   top = newNode;
   printf("Pushed element: %d\n", value);
}
```

```
void pop()
{
   if (top == NULL)
   {
      printf("Stack is empty. Cannot pop.\n");
      return;
   }
   struct Node* temp = top;
   printf("Popped element: %d\n", top->data);
   top = top->next;
   free(temp);
}

void displayStack()
{
   if (top == NULL)
   {
      printf("Stack is empty\n");
      return;
   }
   struct Node* temp = top;
   printf("Stack elements (top to bottom): ");
   while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
   }
   printf("\n");
}



int main() {
   int choice, value;
   do {
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            scanf("%d", &value);
            push(value);
            break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                printf("Exiting program\n");
                return 0;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 4);

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Sanjeev is in charge of managing a library's book storage, and he wants to create a program that simplifies this task. His goal is to implement a program that simulates a stack using an array.

Help him in writing a program that provides the following functionality:

Add Book ID to the Stack (Push): You can add a book ID to the top of the book stack. Remove Book ID from the Stack (Pop): You can remove the top book ID from the stack and display its details. If the stack is empty, you cannot remove any more book IDs.Display Books ID in the Stack (Display): You can view the books ID currently on the stack.Exit the Library: You can choose to exit the program.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the book onto the stack. If the choice is 1, the following input is a space-separated integer, representing the ID of the book to be pushed onto the stack.

Choice 2: Pop the book ID from the stack.

Choice 3: Display the book ID in the stack.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

1. If the choice is 1, push the given book ID to the stack and display the corresponding message.
2. If the choice is 2, pop the book ID from the stack and display the corresponding message.
3. If the choice is 2, and if the stack is empty without any book ID, print "Stack Underflow"
4. If the choice is 3, print the book IDs in the stack.
5. If the choice is 3, and there are book IDs in the stack, print "Stack is empty"
6. If the choice is 4, exit the program and display the corresponding message.
7. If any other choice is entered, print "Invalid choice"

Refer to the sample output for the exact text and format.

Sample Test Case

Input: 1 19
1 28
2
3
2
4
Output: Book ID 19 is pushed onto the stack
Book ID 28 is pushed onto the stack

```
Book ID 28 is popped from the stack
Book ID in the stack: 19
Book ID 19 is popped from the stack
Exiting the program
```

Answer

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

#define max 10

int stack[max], top = -1;

void push(int data)
{
   if (top == max - 1)
   {
      printf("Stack overflow\n");
      return;
   }
   top = top + 1;
   stack[top] = data;
   printf("Book ID %d is pushed onto the stack\n", data);
}

void pop()
{
   if (top == -1)
   {
      printf("Stack underflow\n");
   }
   else
   {
      int d = stack[top];
      top = top - 1;
      printf("Book ID %d is popped from the stack\n", d);
   }
}

void display()
{
    if (top == -1)
    {
        printf("Stack is empty\n");
    }
    else
    {
        int temp = top;
        printf("Book ID in the stack:\n");
        while (temp != -1) {
            printf("%d\n", stack[temp]);
            temp--;
        }
    }
}

int main()
{
    int n, data;
    do
    {
        scanf("%d", &n);
        switch (n)
        {
            case 1:
                scanf("%d", &data);
                push(data);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting the program\n");
                break;
            default:
                printf("Invalid choice\n");
                break;
        }
    }
    while (n != 4);
```

```
return 0;
}
```

Status : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Sharon is developing a programming challenge for a coding competition. The challenge revolves around implementing a character-based stack data structure using an array.

Sharon's project involves a stack that can perform the following operations:

Push a Character: Users can push a character onto the stack.Pop a Character: Users can pop a character from the stack, removing and displaying the top character.Display Stack: Users can view the current elements in the stack.Exit: Users can exit the stack operations application.

Write a program to help Sharon to implement a program that performs the given operations.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the character onto the stack. If the choice is 1, the following input is a space-separated character, representing the character to be pushed onto the stack.

Choice 2: Pop the character from the stack.

Choice 3: Display the characters in the stack.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

1. If the choice is 1, push the given character to the stack and display the pushed character having the prefix "Pushed: ".
2. If the choice is 2, undo the character from the stack and display the character that is popped having the prefix "Popped: ".
3. If the choice is 2, and if the stack is empty without any characters, print "Stack is empty. Nothing to pop."
4. If the choice is 3, print the elements in the stack having the prefix "Stack elements: ".
5. If the choice is 3, and there are no characters in the stack, print "Stack is empty."
6. If the choice is 4, exit the program.
7. If any other choice is entered, print "Invalid choice"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 2
4
Output: Stack is empty. Nothing to pop.

Answer

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

```
#include <stdbool.h>

#define MAX_SIZE 100

char items[MAX_SIZE];
int top = -1;

void initialize() {
   top = -1;
}
bool isFull() {
   return top == MAX_SIZE - 1;
}

bool isEmpty() {
   return top == -1;
}

void push(char value)
{
   if (isFull()) return;
   items[++top] = value;
   printf("Pushed: %c\n", value);
}

void pop()
{
   if (isEmpty())
   {
      printf("Stack is empty. Nothing to pop.\n");
   }
   else
   {
      printf("Popped: %c\n", items[top--]);
   }
}

void display()
{
   if (isEmpty())
   {
      printf("Stack is empty.\n");
   }
```

```
    else
    {
        printf("Stack elements: ");
        for (int i = top; i >= 0; i--)
        {
            printf("%c ", items[i]);
        }
        printf("\n");
    }
}

int main() {
    initialize();
    int choice;
    char value;

    while (true) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                scanf(" %c", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

You are a software developer tasked with building a module for a scientific calculator application. The primary function of this module is to convert infix mathematical expressions, which are easier for users to read and write, into postfix notation (also known as Reverse Polish Notation). Postfix notation is more straightforward for the application to evaluate because it removes the need for parentheses and operator precedence rules.

The scientific calculator needs to handle various mathematical expressions with different operators and ensure the conversion is correct. Your task is to implement this infix-to-postfix conversion algorithm using a stack-based approach.

Example

Input:

a+b

Output:

ab+

Explanation:

The postfix representation of (a+b) is ab+.

Input Format

The input is a string, representing the infix expression.

Output Format

The output displays the postfix representation of the given infix expression.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: a+(b*e)
Output: abe*+

Answer

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

struct Stack {
   int top;
   unsigned capacity;
   char* array;
};

struct Stack* createStack(unsigned capacity) {
   struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));

   if (!stack)
```

```
        return NULL;

    stack->top = -1;
    stack->capacity = capacity;
    stack->array = (char*)malloc(stack->capacity * sizeof(char));

    return stack;
}

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

char peek(struct Stack* stack) {
    return stack->array[stack->top];
}

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

void push(struct Stack* stack, char op) {
    stack->array[++stack->top] = op;
}
int isOperand(char ch)
{
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0' && ch <= '9');
}
int Prec(char ch)
{
    switch (ch)
    {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
    }
    return -1;
}
void infixToPostfix(char* exp)
{
    int i, k;
    struct Stack* stack = createStack(strlen(exp));
    if (!stack) return;

    for (i = 0, k = -1; exp[i]; ++i)
    {
        if (isOperand(exp[i]))
            printf("%c", exp[i]);

        else if (exp[i] == '(')
            push(stack, exp[i]);
        else if (exp[i] == ')') {
            while (!isEmpty(stack) && peek(stack) != '(')
                printf("%c", pop(stack));
            if (!isEmpty(stack) && peek(stack) != '(')
                return;
            else
                pop(stack);
        }
        else
        {
            while (!isEmpty(stack) && Prec(exp[i]) <= Prec(peek(stack)))
                printf("%c", pop(stack));
            push(stack, exp[i]);
        }
    }
    while (!isEmpty(stack))
        printf("%c", pop(stack));

   printf("\n");
}


int main() {
    char exp[100];
    scanf("%s", exp);

    infixToPostfix(exp);
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 3_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Milton is a diligent clerk at a school who has been assigned the task of managing class schedules. The school has various sections, and Milton needs to keep track of the class schedules for each section using a stack-based system.

He uses a program that allows him to push, pop, and display class schedules for each section. Milton's program uses a stack data structure, and each class schedule is represented as a character. Help him write a program using a linked list.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Push the character onto the stack. If the choice is 1, the following input is a space-separated character, representing the class schedule to be pushed onto the stack.

Choice 2: Pop class schedule from the stack

Choice 3: Display the class schedules in the stack.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the stack:

- If the choice is 1, push the given class schedule to the stack and display the following: "Adding Section: [class schedule]"
- If the choice is 2, pop the class schedule from the stack and display the following: "Removing Section: [class schedule]"
- If the choice is 2, and if the stack is empty without any class schedules, print "Stack is empty. Cannot pop."
- If the choice is 3, print the class schedules in the stack in the following: "Enrolled Sections: " followed by the class schedules separated by space.
- If the choice is 3, and there are no class schedules in the stack, print "Stack is empty"
- If the choice is 4, exit the program and display the following: "Exiting the program"
- If any other choice is entered, print "Invalid choice"

Refer to the sample output for the exact format.

Sample Test Case

Input: 1 d
1 h
3
2

3
4
Output: Adding Section: d
Adding Section: h
Enrolled Sections: h d
Removing Section: h
Enrolled Sections: d
Exiting program

Answer

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

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

struct Node* top = NULL;

void push(char value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (!newNode) {
      printf("Memory error\n");
      return;
   }
   newNode->data = value;
   newNode->next = top;
   top = newNode;
   printf("Adding Section: %c\n", value);
}

void pop() {
   if (top == NULL) {
      printf("Stack is empty. Cannot pop.\n");
      return;
   }
   struct Node* temp = top;
   printf("Removing Section: %c\n", top->data);
   top = top->next;
   free(temp);
}
```

```
void displayStack() {
    if (top == NULL) {
        printf("Stack is empty\n");
        return;
    }
    struct Node* temp = top;
    printf("Enrolled Sections: ");
    while (temp != NULL) {
        printf("%c ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}


int main() {
    int choice;
    char value;
    do {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                scanf(" %c", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                displayStack();
                break;
            case 4:
                printf("Exiting program\n");
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 4);

    return 0;
}
```

Status : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

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;
};
void push(struct Node** top, int value)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (newNode == NULL)
   {
      printf("Stack overflow\n");
      exit(1);
   }
   newNode->data = value;
   newNode->next = *top;
   *top = newNode;
}
int pop(struct Node** top)
```

```
{
    if (*top == NULL)
    {
        printf("Stack underflow\n");
        return -1;
    }
    struct Node* temp = *top;
    int poppedValue = temp->data;
    *top = temp->next;
    free(temp);
    return poppedValue;
}
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)
    {
        printf("Stack is empty\n");
        return -1;
    }
    return top->data;
}

int main()
{
    struct Node* top = NULL;
    int n1, n2, n3, n4;
    scanf("%d %d %d %d", &n1, &n2, &n3, &n4);
    push(&top, n1);
    push(&top, n2);
    push(&top, n3);
    push(&top, n4);
    display(top);
    pop(&top);
    display(top);
    int topElement = peek(top);
    if (topElement != -1)
    {
        printf("%d\n", topElement);
    }
    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>
#include <stdlib.h>
#include <limits.h>
#define MAX_SIZE 20
typedef struct
{
   int data[MAX_SIZE];
   int top;
   int min[MAX_SIZE];
   int minTop;
} Stack;
void initialize(Stack *stack)
{
   stack->top = -1;
   stack->minTop = -1;
}
int isEmpty(Stack *stack)
{
   return stack->top == -1;
}
int isFull(Stack *stack)
{
   return stack->top == MAX_SIZE - 1;
}
void push(Stack *stack, int value)
{
   if (isFull(stack))
```

```
    {
        printf("Stack is full. Cannot push %d\n", value);
        return;
    }
    stack->top++;
    stack->data[stack->top] = value;
    if (stack->minTop == -1 || value <= stack->min[stack->minTop])
    {
        stack->minTop++;
        stack->min[stack->minTop] = value;
    }
}
int pop(Stack *stack)
{
    if (isEmpty(stack))
    {
        printf("Stack is empty. Cannot pop.\n");
        return INT_MIN;
    }
    int poppedValue = stack->data[stack->top];
    stack->top--;
    if (poppedValue == stack->min[stack->minTop])
    {
        stack->minTop--;
    }
    return poppedValue;
}

int findMinimum(Stack *stack)
{
    if (isEmpty(stack))
    {
        return INT_MAX;
    }
    return stack->min[stack->minTop];
}
int main()
{
    int n, element;
    Stack stack;
    initialize(&stack);
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &element);
        push(&stack, element);
    }
    printf("Minimum element in the stack: %d\n", findMinimum(&stack));
    int poppedElement = pop(&stack);
    printf("Popped element: %d\n", poppedElement);
    printf("Minimum element in the stack after popping: %d\n",
findMinimum(&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 <stdbool.h>
#include <string.h>
#define MAX_SIZE 100
typedef struct
{
   char data[MAX_SIZE];
   int top;
} Stack;
void initialize(Stack *stack)
{
   stack->top = -1;
}
bool isEmpty(Stack *stack)
{
   return stack->top == -1;
}
bool isFull(Stack *stack)
{
   return stack->top == MAX_SIZE - 1;
}
void push(Stack *stack, char value)
```

```
{
   if (isFull(stack))
   {
      printf("Stack is full. Cannot push %c\n", value);
      return;
   }
   stack->top++;
   stack->data[stack->top] = value;
}
char pop(Stack *stack)
{
   if (isEmpty(stack))
   {
      printf("Stack is empty. Cannot pop.\n");
      return '\0';
   }
   char poppedValue = stack->data[stack->top];
   stack->top--;
   return poppedValue;
}
char peek(Stack *stack) {
   if (isEmpty(stack)) {
      return '\0';
   }
   return stack->data[stack->top];
}
bool isMatchingPair(char open, char close)
{
   if (open == '(' && close == ')') return true;
   if (open == '[' && close == ']') return true;
   if (open == '{' && close == '}') return true;
   return false;
}
bool isValidBracketSequence(char *str)
{
   Stack stack;
   initialize(&stack);
   int len = strlen(str);
   for (int i = 0; i < len; i++)
   {
      char currentChar = str[i];
      if (currentChar == '(' || currentChar == '[' || currentChar == '{')
        {
            push(&stack, currentChar);
        }
        else if (currentChar == ')' || currentChar == ']' || currentChar == '}')
        {
            if (isEmpty(&stack))
            {
                return false;
            }
            char top = peek(&stack);
            pop(&stack);
            if (!isMatchingPair(top, currentChar))
            {
                return false;
            }
        }
    }
    return isEmpty(&stack);
}
int main()
{
    char str[101];
    scanf("%s", str);
    if (isValidBracketSequence(str))
    {
        printf("Valid string\n");
    }
    else
    {
        printf("Invalid string\n");
    }
return 0;
}
```

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