Assignment 4

// Question 1: Write a menu driven program with 4 options (Push, Pop, Display, and Exit) to demonstrate the working of stacks using arrays.

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

int stack[MAX\_SIZE];

int top = -1;

void push(int value) {

    if (top >= MAX\_SIZE - 1) {

        printf("Stack Overflow! Cannot push more elements.\n");

    } else {

        stack[++top] = value;

        printf("Pushed %d to the stack.\n", value);

    }

}

void pop() {

    if (top < 0) {

        printf("Stack Underflow! Stack is empty.\n");

    } else {

        printf("Popped %d from the stack.\n", stack[top--]);

    }

}

void display() {

    if (top < 0) {

        printf("Stack is empty.\n");

    } else {

        printf("Stack elements: ");

        for (int i = top; i >= 0; i--) {

            printf("%d ", stack[i]);

        }

        printf("\n");

    }

}

int main() {

    int choice, value;

    do {

        printf("\nStack Operations Menu:\n");

        printf("1. Push\n");

        printf("2. Pop\n");

        printf("3. Display\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value to push: ");

                scanf("%d", &value);

                push(value);

                break;

            case 2:

                pop();

                break;

            case 3:

                display();

                break;

            case 4:

                printf("Exiting program.\n");

                break;

            default:

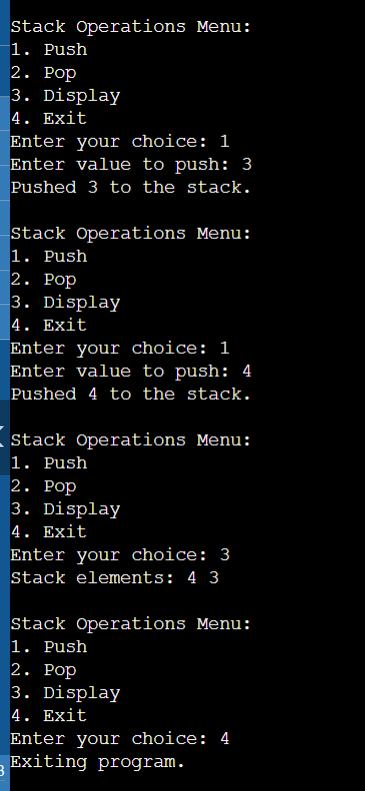
                printf("Invalid choice. Please try again.\n");

        }

    } while (choice != 4);

    return 0;

}



// Question 2: Write a menu driven program with 4 options (Push, Pop, Display, and Exit) to demonstrate the working of stacks using linked-list.

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node \*next;

};

struct Node \*top\_ll = NULL;

void push\_ll(int value) {

    struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

    if (newNode == NULL) {

        printf("Memory allocation failed.\n");

        return;

    }

    newNode->data = value;

    newNode->next = top\_ll;

    top\_ll = newNode;

    printf("Pushed %d to the stack.\n", value);

}

void pop\_ll() {

    if (top\_ll == NULL) {

        printf("Stack Underflow! Stack is empty.\n");

        return;

    }

    struct Node \*temp = top\_ll;

    top\_ll = top\_ll->next;

    printf("Popped %d from the stack.\n", temp->data);

    free(temp);

}

void display\_ll() {

    if (top\_ll == NULL) {

        printf("Stack is empty.\n");

        return;

    }

    struct Node \*current = top\_ll;

    printf("Stack elements: ");

    while (current != NULL) {

        printf("%d ", current->data);

        current = current->next;

    }

    printf("\n");

}

int main() {

    int choice, value;

    do {

        printf("\nLinked List Stack Operations Menu:\n");

        printf("1. Push\n");

        printf("2. Pop\n");

        printf("3. Display\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter value to push: ");

                scanf("%d", &value);

                push\_ll(value);

                break;

            case 2:

                pop\_ll();

                break;

            case 3:

                display\_ll();

                break;

            case 4:

                printf("Exiting program.\n");

                break;

            default:

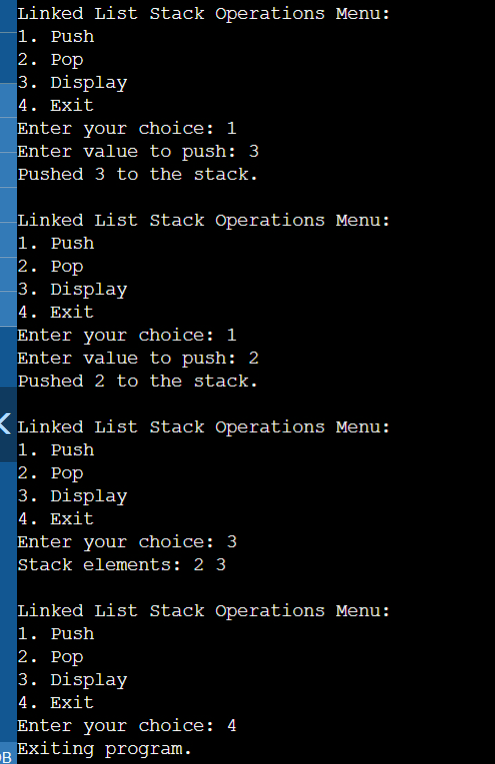
                printf("Invalid choice. Please try again.\n");

        }

    } while (choice != 4);

    return 0;

}



// Question 3: Write a program to convert infix expression into postfix expression using stack.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_SIZE 100

char stack\_infix[MAX\_SIZE];

int top\_infix = -1;

void push\_infix(char op) {

    if (top\_infix >= MAX\_SIZE - 1) {

        printf("Stack Overflow\n");

        return;

    }

    stack\_infix[++top\_infix] = op;

}

char pop\_infix() {

    if (top\_infix < 0) {

        return '\0';

    }

    return stack\_infix[top\_infix--];

}

int precedence(char op) {

    if (op == '+' || op == '-') {

        return 1;

    }

    if (op == '\*' || op == '/') {

        return 2;

    }

    return 0;

}

void infixToPostfix(char\* infix, char\* postfix) {

    int i, j;

    char op;

    for (i = 0, j = 0; infix[i] != '\0'; i++) {

        if (isalnum(infix[i])) {

            postfix[j++] = infix[i];

        } else if (infix[i] == '(') {

            push\_infix(infix[i]);

        } else if (infix[i] == ')') {

            while (top\_infix != -1 && stack\_infix[top\_infix] != '(') {

                postfix[j++] = pop\_infix();

            }

            if (top\_infix != -1 && stack\_infix[top\_infix] != '(') {

                printf("Invalid Expression\n");

                return;

            } else {

                pop\_infix();

            }

        } else {

            while (top\_infix != -1 && precedence(stack\_infix[top\_infix]) >= precedence(infix[i])) {

                postfix[j++] = pop\_infix();

            }

            push\_infix(infix[i]);

        }

    }

    while (top\_infix != -1) {

        if (stack\_infix[top\_infix] == '(') {

            printf("Invalid Expression\n");

            return;

        }

        postfix[j++] = pop\_infix();

    }

    postfix[j] = '\0';

}

int main() {

    char infix[MAX\_SIZE];

    char postfix[MAX\_SIZE];

    printf("Enter an infix expression: ");

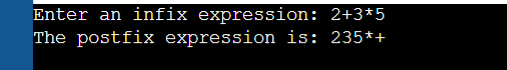
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("The postfix expression is: %s\n", postfix);

    return 0;

}



// Question 4: Write a program to convert infix expression into prefix expression using stack.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_SIZE 100

char stack\_prefix[MAX\_SIZE];

int top\_prefix = -1;

void push\_prefix(char op) {

    if (top\_prefix >= MAX\_SIZE - 1) {

        printf("Stack Overflow\n");

        return;

    }

    stack\_prefix[++top\_prefix] = op;

}

char pop\_prefix() {

    if (top\_prefix < 0) {

        return '\0';

    }

    return stack\_prefix[top\_prefix--];

}

void reverseString(char\* str) {

    int len = strlen(str);

    int i, j;

    char temp;

    for (i = 0, j = len - 1; i < j; i++, j--) {

        temp = str[i];

        str[i] = str[j];

        str[j] = temp;

    }

}

int precedence\_prefix(char op) {

    if (op == '+' || op == '-') {

        return 1;

    }

    if (op == '\*' || op == '/') {

        return 2;

    }

    return 0;

}

void infixToPrefix(char\* infix, char\* prefix) {

    int i, j;

    char op;

    reverseString(infix);

    for (i = 0, j = 0; infix[i] != '\0'; i++) {

        if (infix[i] == '(') {

            infix[i] = ')';

        } else if (infix[i] == ')') {

            infix[i] = '(';

        }

    }

    for (i = 0; infix[i] != '\0'; i++) {

        if (isalnum(infix[i])) {

            prefix[j++] = infix[i];

        } else if (infix[i] == '(') {

            push\_prefix(infix[i]);

        } else if (infix[i] == ')') {

            while (top\_prefix != -1 && stack\_prefix[top\_prefix] != '(') {

                prefix[j++] = pop\_prefix();

            }

            if (top\_prefix != -1 && stack\_prefix[top\_prefix] != '(') {

                printf("Invalid Expression\n");

                return;

            } else {

                pop\_prefix();

            }

        } else {

            while (top\_prefix != -1 && precedence\_prefix(stack\_prefix[top\_prefix]) >= precedence\_prefix(infix[i])) {

                prefix[j++] = pop\_prefix();

            }

            push\_prefix(infix[i]);

        }

    }

    while (top\_prefix != -1) {

        if (stack\_prefix[top\_prefix] == '(') {

            printf("Invalid Expression\n");

            return;

        }

        prefix[j++] = pop\_prefix();

    }

    prefix[j] = '\0';

    reverseString(prefix);

}

int main() {

    char infix[MAX\_SIZE];

    char prefix[MAX\_SIZE];

    printf("Enter an infix expression: ");

    scanf("%s", infix);

    infixToPrefix(infix, prefix);

    printf("The prefix expression is: %s\n", prefix);

    return 0;

}



// Question 5: Write a program to evaluate a postfix expression using stack.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX\_SIZE 100

int stack\_eval[MAX\_SIZE];

int top\_eval = -1;

void push\_eval(int value) {

    if (top\_eval >= MAX\_SIZE - 1) {

        printf("Stack Overflow\n");

        return;

    }

    stack\_eval[++top\_eval] = value;

}

int pop\_eval() {

    if (top\_eval < 0) {

        return -1; // Indicates error

    }

    return stack\_eval[top\_eval--];

}

int evaluatePostfix(char\* postfix) {

    int i;

    for (i = 0; postfix[i] != '\0'; i++) {

        if (isdigit(postfix[i])) {

            push\_eval(postfix[i] - '0');

        } else {

            int op2 = pop\_eval();

            int op1 = pop\_eval();

            switch (postfix[i]) {

                case '+': push\_eval(op1 + op2); break;

                case '-': push\_eval(op1 - op2); break;

                case '\*': push\_eval(op1 \* op2); break;

                case '/': push\_eval(op1 / op2); break;

                default: printf("Invalid operator\n"); return -1;

            }

        }

    }

    return pop\_eval();

}

int main() {

    char postfix[MAX\_SIZE];

    printf("Enter a postfix expression (single-digit operands): ");

    scanf("%s", postfix);

    int result = evaluatePostfix(postfix);

    if (result != -1) {

        printf("Result of the postfix expression: %d\n", result);

    }

    return 0;

}

