**Ds day7**

1. **Write a C program to implement infix, prefix and postfix notations for arithmetic expressions using stack:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char c) {

stack[++top] = c;

}

char pop() {

return stack[top--];

}

int precedence(char c) {

if (c == '+' || c == '-') return 1;

if (c == '\*' || c == '/') return 2;

return 0;

}

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

int i = 0, j = 0;

while (infix[i]) {

if (isalnum(infix[i])) postfix[j++] = infix[i];

else if (infix[i] == '(') push(infix[i]);

else if (infix[i] == ')') {

while (top != -1 && stack[top] != '(') postfix[j++] = pop();

pop();

} else {

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

postfix[j++] = pop();

push(infix[i]);

}

i++;

}

while (top != -1) postfix[j++] = pop();

postfix[j] = '\0';

}

void reverse(char\* exp) {

int n = strlen(exp);

for (int i = 0; i < n / 2; i++) {

char temp = exp[i];

exp[i] = exp[n - i - 1];

exp[n - i - 1] = temp;

}

}

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

reverse(infix);

for (int i = 0; infix[i]; i++) {

if (infix[i] == '(') infix[i] = ')';

else if (infix[i] == ')') infix[i] = '(';

}

infixToPostfix(infix, prefix);

reverse(prefix);

}

int main() {

char infix[MAX], postfix[MAX], prefix[MAX];

printf("Enter infix expression: ");

scanf("%s", infix);

infixToPostfix(infix, postfix);

infixToPrefix(infix, prefix);

printf("Postfix: %s\n", postfix);

printf("Prefix: %s\n", prefix);

return 0;

}

Output:



1. **Write a C program to check if the parentheses in an expression are balanced using a stack. Extend the program to handle multiple types of parentheses (e.g., {}, [], ()):**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char c) {

if (top < MAX - 1) {

stack[++top] = c;

}

}

char pop() {

if (top >= 0) {

return stack[top--];

}

return '\0';

}

bool isEmpty() {

return top == -1;

}

bool isMatchingPair(char opening, char closing) {

if (opening == '(' && closing == ')') return true;

if (opening == '{' && closing == '}') return true;

if (opening == '[' && closing == ']') return true;

return false;

}

bool areParenthesesBalanced(const char\* expression) {

for (int i = 0; expression[i]; i++) {

char current = expression[i];

if (current == '(' || current == '{' || current == '[') {

push(current);

} else if (current == ')' || current == '}' || current == ']') {

if (isEmpty() || !isMatchingPair(pop(), current)) {

return false;

}

}

}

return isEmpty();

}

int main() {

const char\* expression = "{[()]}";

if (areParenthesesBalanced(expression)) {

printf("The parentheses are balanced.\n");

} else {

printf("The parentheses are not balanced.\n");

}

return 0;

}

Output:



1. **Write a program to evaluate a postfix expression using a stack. The program should handle basic arithmetic operators (+, -, \*, /):**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX 100

typedef struct Stack {

int top;

int items[MAX];

} Stack;

void initStack(Stack\* s) {

s->top = -1;

}

int isFull(Stack\* s) {

return s->top == MAX - 1;

}

int isEmpty(Stack\* s) {

return s->top == -1;

}

void push(Stack\* s, int item) {

if (!isFull(s)) {

s->items[++(s->top)] = item;

}

}

int pop(Stack\* s) {

if (!isEmpty(s)) {

return s->items[(s->top)--];

}

return -1; // Error value

}

int evaluatePostfix(char\* expression) {

Stack s;

initStack(&s);

int i = 0;

while (expression[i]) {

if (isdigit(expression[i])) {

push(&s, expression[i] - '0');

} else {

int val2 = pop(&s);

int val1 = pop(&s);

switch (expression[i]) {

case '+':

push(&s, val1 + val2);

break;

case '-':

push(&s, val1 - val2);

break;

case '\*':

push(&s, val1 \* val2);

break;

case '/':

push(&s, val1 / val2);

break;

}

}

i++;

}

return pop(&s);

}

int main() {

char expression[MAX];

printf("Enter a postfix expression: ");

fgets(expression, sizeof(expression), stdin);

printf("Result: %d\n", evaluatePostfix(expression));

return 0;

}

output:



1. **Write a C program to solve the Tower of Hanoi problem using recursion:**

#include <stdio.h>

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod) {

if (n == 1) {

printf("Move disk 1 from rod %c to rod %c\n", from\_rod, to\_rod);

return;

}

towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

printf("Move disk %d from rod %c to rod %c\n", n, from\_rod, to\_rod);

towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

}

int main() {

int n = 3; // Number of disks

towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods

return 0;

}

Output:

