// C code to convert infix to postfix expression

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_EXPR\_SIZE 100

// Function to return precedence of operators

int precedence(char operator)

{

switch (operator) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

default:

return -1;

}

}

// Function to check if the scanned character

// is an operator

int isOperator(char ch)

{

return (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^');

}

// Main functio to convert infix expression

// to postfix expression

char\* infixToPostfix(char\* infix)

{

int i, j;

int len = strlen(infix);

char\* postfix = (char\*)malloc(sizeof(char) \* (len + 2));

char stack[MAX\_EXPR\_SIZE];

int top = -1;

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

if (infix[i] == ' ' || infix[i] == '\t')

continue;

// If the scanned character is operand

// add it to the postfix expression

if (isalnum(infix[i])) {

postfix[j++] = infix[i];

}

// if the scanned character is '('

// push it in the stack

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

stack[++top] = infix[i];

}

// if the scanned character is ')'

// pop the stack and add it to the

// output string until empty or '(' found

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

while (top > -1 && stack[top] != '(')

postfix[j++] = stack[top--];

if (top > -1 && stack[top] != '(')

return "Invalid Expression";

else

top--;

}

// If the scanned character is an operator

// push it in the stack

else if (isOperator(infix[i])) {

while (top > -1

&& precedence(stack[top])

>= precedence(infix[i]))

postfix[j++] = stack[top--];

stack[++top] = infix[i];

}

}

// Pop all remaining elements from the stack

while (top > -1) {

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

return "Invalid Expression";

}

postfix[j++] = stack[top--];

}

postfix[j] = '\0';

return postfix;

}

// Driver code

int main()

{

char infix[MAX\_EXPR\_SIZE] = "a+b\*(c^d-e)^(f+g\*h)-i";

// Function call

char\* postfix = infixToPostfix(infix);

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

free(postfix);

return 0;

}

// C program to evaluate value of a postfix expression

#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Stack type

struct Stack {

int top;

unsigned capacity;

int\* array;

};

// Stack Operations

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

= (int\*)malloc(stack->capacity \* sizeof(int));

if (!stack->array)

return NULL;

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;

}

// The main function that returns value

// of a given postfix expression

int evaluatePostfix(char\* exp)

{

// Create a stack of capacity equal to expression size

struct Stack\* stack = createStack(strlen(exp));

int i;

// See if stack was created successfully

if (!stack)

return -1;

// Scan all characters one by one

for (i = 0; exp[i]; ++i) {

// If the scanned character is an operand

// (number here), push it to the stack.

if (isdigit(exp[i]))

push(stack, exp[i] - '0');

// If the scanned character is an operator,

// pop two elements from stack apply the operator

else {

int val1 = pop(stack);

int val2 = pop(stack);

switch (exp[i]) {

case '+':

push(stack, val2 + val1);

break;

case '-':

push(stack, val2 - val1);

break;

case '\*':

push(stack, val2 \* val1);

break;

case '/':

push(stack, val2 / val1);

break;

}

}

}

return pop(stack);

}

// Driver code

int main()

{

char exp[] = "231\*+9-";

// Function call

printf("postfix evaluation: %d", evaluatePostfix(exp));

return 0;

}

**#include<stdio.h>**

**int stk[100]; // stack**

**int size = 100; // size of stack**

**int ptr = -1; // store the index of top element of the stack**

**// push x to stack**

**void push(int x){**

**if(ptr==size-1){**

**printf("OverFlow \n");**

**}**

**else{**

**++ptr;**

**stk[ptr] = x;**

**}**

**}**

**// return top element of the stack**

**int top(){**

**if(ptr==-1){**

**printf("UnderFlow \n");**

**return -1;**

**}**

**else{**

**return stk[ptr];**

**}**

**}**

**// remove top element from the stack**

**void pop(){**

**if(ptr==-1){**

**printf("UnderFlow \n");**

**}**

**else{**

**--ptr;**

**}**

**}**

**// check if stack is empty or not**

**int isempty(){**

**if(ptr==-1)**

**return 1;**

**else**

**return 0;**

**}**

**int main() {**

**int i, n;**

**printf("Enter a number: ");**

**scanf("%d", &n);**

**push(1);**

**for(i=2;i<=n;++i){**

**push(top() \* i);**

**}**

**printf("Factorial: %d", top());**

**return 0;**

**}**