**Week 02**

Program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), \* (multiply) and / (divide).

**Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define SIZE 20

struct stack {

int top;

char data[SIZE];

};

typedef struct stack STACK;

void push(STACK \*S, char item) {

if (S->top < SIZE - 1) {

S->data[++(S->top)] = item;

} else {

printf("Stack overflow!\n");

}

}

char pop(STACK \*S) {

if (S->top != -1) {

return S->data[(S->top)--];

} else {

printf("Stack underflow!\n");

return '\0';

}

}

int preced(char symbol) {

switch (symbol) {

case '^': return 5;

case '\*':

case '/': return 3;

case '+':

case '-': return 1;

default: return 0;

}

}

void infixpostfix(char infix[20], STACK \*S) {

char postfix[20], symbol, temp;

int i, j = 0;

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

symbol = infix[i];

if (isalnum(symbol)) {

postfix[j++] = symbol;

} else {

switch (symbol) {

case '(':

push(S, symbol);

break;

case ')':

while (S->top != -1 && (temp = pop(S)) != '(') {

postfix[j++] = temp;

}

break;

case '+':

case '-':

case '\*':

case '/':

case '^':

while (S->top != -1 && preced(S->data[S->top]) >= preced(symbol)) {

postfix[j++] = pop(S);

}

push(S, symbol);

break;

}

}

}

while (S->top != -1) {

postfix[j++] = pop(S);

}

postfix[j] = '\0';

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

}

int main() {

char infix[20];

STACK S;

S.top = -1;

printf("Enter infix expression: ");

scanf("%s", infix);

infixpostfix(infix, &S);

printf("\nName: Abhay NY \nUSN: 24BECS404\n");

return 0;

}

**Output:**