**DEPARTMENT OF INFORMATION TECHNOLOGY**

**Course Name and Code:** Data Structures Lab **(**ITL302)

**Semester:** III (SYIT)

**Academic Year:** 2024-25 (Odd Semester)

**Experiment No. 03**

**Aim:** I**mplementations of Infix to Postfix Expression for real-world application**

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct Stack

{

int N; // Max Capacity of Stack

char S[1000];

int Top;

};

int isOperator(char symbol);

int getPrecedence(char s);

void Push(struct Stack \* Sptr, char ele);

char Pop(struct Stack\* Sptr);

int PopAllInto(struct Stack \* Sptr, char \* str, int start);//Pops all operators into str(string) starting from index 'start'

int PushWithPrecedence(struct Stack \* Sptr, char \* str, int start, char currentOp);

/\* Checks precedence before pushing, if precedence of currentOp(Operator to push) is lower keeps popping the top

into str starting from index start \*/

int main()

{

int len;

printf("Enter length of the string: ");

scanf("%d", &len);

char infix[1000], postfix[1000];

struct Stack stk;

stk.N = len;

stk.Top = -1;

printf("Enter the infix string: ");

scanf("%s", &infix);

int k = 0;

for(int i = 0; i < len; i++)

{

if(isOperator(infix[i]) == 1)

{

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

Push(&stk, infix[i]);

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

k = PopAllInto(&stk, &postfix[0], k);

else

k = PushWithPrecedence(&stk, &postfix[0], k, infix[i]);

}

else //Alphabets or numbers are assumed

{

postfix[k] = infix[i];

k++;

}

}

k = PopAllInto(&stk, &postfix[0], k);//Pops the remaining STACK into postfix expression

postfix[k] = '\0'; //Adds a null byte to remove garbage value

printf("Postfix String: %s", postfix);

return 0;

}

int isOperator(char symbol)

{

if(symbol == '+' || symbol == '-' || symbol == '\*' || symbol == '/' || symbol == '^' || symbol == '(' || symbol == ')')

return 1;

return 0;

}

int getPrecedence(char s)

{

if(s == '-' || s == '+')

return 0;

if(s == '\*' || s == '/')

return 1;

if(s == '^')

return 2;

}

void Push(struct Stack \* Sptr, char ele)

{

if(Sptr->Top >= Sptr->N - 1)

{

printf("Stack Overflow Error!");

exit(-1);

}

Sptr->Top += 1;

Sptr->S[Sptr->Top] = ele;

}

char Pop(struct Stack \* Sptr)

{

if(Sptr->Top < 0)

{

printf("Stack Underflow Error!");

exit(-1);

}

Sptr->Top -= 1;

return Sptr->S[Sptr->Top + 1];

}

int PopAllInto(struct Stack \* Sptr, char \* str, int start)

{

while(Sptr->Top > 0 && Sptr->S[Sptr->Top] != '(' ) //Pop till you find ( or till stack is emptied

{

\*(str + start) = Pop(Sptr);

start++;

}

char ele = Pop(Sptr);

if(ele != '(')//Add the last element if it isnt an open bracket

{

\*(str + start) = ele;

start++;

}

return start;

}

int PushWithPrecedence(struct Stack \* Sptr, char \* str, int start, char currentOp)

{

while(Sptr->Top != -1 )

{

char TopOp = Sptr->S[Sptr->Top];

if(TopOp == '(' || getPrecedence(currentOp) > getPrecedence(TopOp) )

break;

\*(str + start) = Pop(Sptr);

start++;

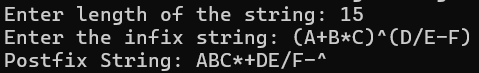
}

Push(Sptr, currentOp);

return start;

}

**Output:**

****

**Submitter Details:-**

**Name:** Faizan Dodiya

**Roll No:** 24

**Div/Batch :** A/ S-2