NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL

Name – Rajat Goyal

Roll No. – 197267

Section – B

Branch – CSE(3rd year)

LP LAB Assignment - 2

Infix to Postfix by constructing expression tree

Question:-

2. Write flex specifications to convert an infix expression to postfix notation by constructing the corresponding expression tree. Use the binary operators +, -,*, /, @(exponentiation) and um (Unary Minus). Assume usual precedence and associativity rules for the operators. Note that some of the operator symbols are meta-characters in flex.

Code:-

```
%{
#include<stdio.h>
typedef struct tnode* tptr;
// Creating Expression Tree Structure
struct tnode{
char *data;
tptr left,right;
};
// Stack to store operators
struct Stack{
int data[100];
int top;
int size;
};
// Stack to create expression tree
struct StackTree{
int top;
int size;
tptr T[100];
};
// Function of Stack
struct Stack st;
void push(char);
char pop();
char Top();
int empty();
// Function of Tree Stack
```

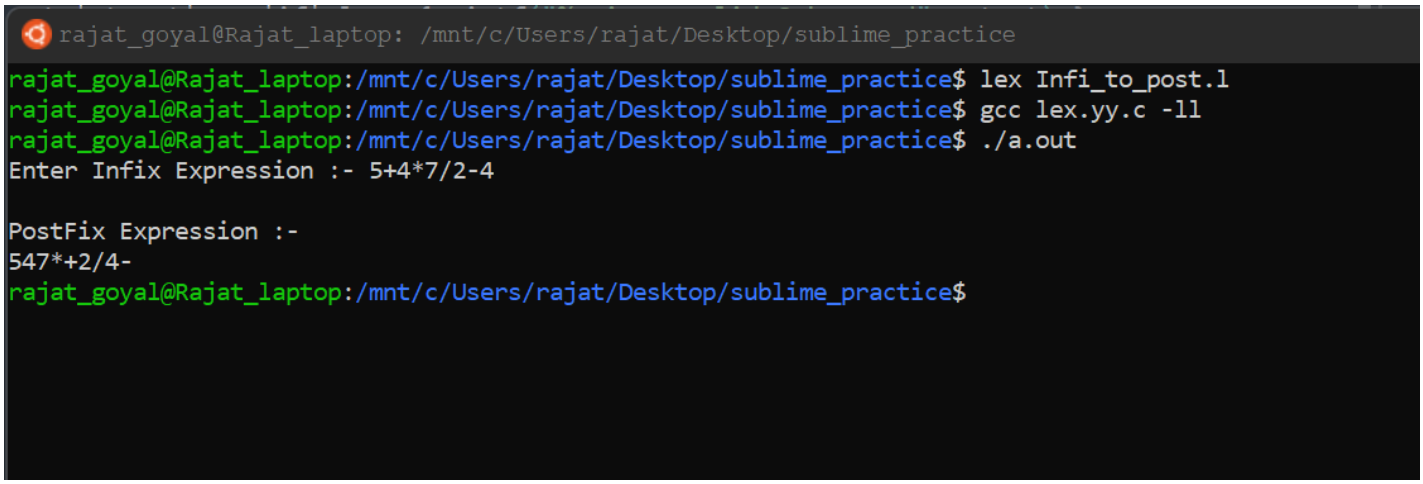
```
struct StackTree st_tree;
void pushTree(tptr);
tptr popTree();
int emptyTree();
int sizeTree();
void operator(char *);
void operand(char *);
int precedence(char);
void postorder(tptr);
%}
%%
um[0-9a-zA-Z]+ {operand(yytext);}
[0-9a-zA-Z]+ {operand(yytext);}
[+\-*/@] {operator(yytext);}
. {;}
%%
void push(char ch){
if(st.top >= (st.size - 1)){
printf("Stack Overflow");
return;
}
else{
st.data[++st.top] = ch;
return;
}
}
char pop(){
if(st.top < 0){
printf("Stack Underflow");
return 0;
}
else{
char ch = st.data[st.top--];
return ch;
}
}
char Top(){
if(st.top < 0){
printf("Stack Underflow");
```

```
return 0;
}
else{
char ch = st.data[st.top];
return ch;
}
}
int empty(){
return (st.top < 0);
}
void pushTree(tptr t){
if(st_tree.top >= (st_tree.size-1)){
printf("Stack Overflow");
return;
}
else{
st_tree.T[++st_tree.top] = t;
return;
}
}
tptr popTree(){
if(st_tree.top < 0){
printf("Stack Underflow");
return NULL;
}
else{
tptr T = st_tree.T[st_tree.top--];
return T;
}
}
int emptyTree(){
return st_tree.top < 0;
}
int sizeTree(){
return st_tree.top;
}
// To ckeck precedence of operators
int precedence(char op){
if(op == '+' || op == '-'){
```

```
return 1;
}
else if(op == '*' || op == '/'){
return 2;
}
else if(op == '@'){
return 3;
}
}
// To insert operators in exression tree
void operator(char *num){
if(empty() == 0){
int prec1 = precedence(Top());
int prec2 = precedence(num[0]);
while(prec2 <= prec1){
if(empty() == 0){
prec1 = precedence(Top());
if(prec1 == 3 && prec1 == prec2){
break;
}
char ch = pop();
char *num1 = malloc(sizeof(char));
num1[0] = ch;
// creating new Tree Node
tptr t = malloc(sizeof(struct tnode));
t->data = num1;
t->right = popTree();
t->left = popTree();
pushTree(t);
}
else{
break;
}
}
}
push(num[0]);
}
void operand(char *op){
tptr t = malloc(sizeof(struct tnode));
```

```
t->data = strdup(op);
t->left = t->right = NULL;
pushTree(t);
}
// To Print PostFix Expression
void postorder(tptr T){
if(T == NULL){
return;
}
postorder(T->left);
postorder(T->right);
printf("%s",T->data);
}
int main(){
st.top = -1;
st.size = 100;
st_tree.top = -1;
st_tree.size = 100;
printf("Enter Infix Expression :- ");
yylex();
while(sizeTree() > 0){
char ch = pop();
char *num = malloc(sizeof(char));
num[0] = ch;
tptr t = malloc(sizeof(struct tnode));
t->data = num;
t->right = popTree();
t->left = popTree();
pushTree(t);
}
printf("PostFix Expression :-\n");
postorder(popTree());
printf("\n");
}
```

Output:


rajat_goyal@Rajat_laptop: /mnt/c/Users/rajat/Desktop/sublime_practice
rajat_goyal@Rajat_laptop:/mnt/c/Users/rajat/Desktop/sublime_practice$ lex Infi_to_post.l
rajat_goyal@Rajat_laptop:/mnt/c/Users/rajat/Desktop/sublime_practice$ gcc lex.yy.c -ll
rajat_goyal@Rajat_laptop:/mnt/c/Users/rajat/Desktop/sublime_practice$ ./a.out
Enter Infix Expression :- 5+4*7/2-4
PostFix Expression :-
547*+2/4-
rajat_goyal@Rajat_laptop:/mnt/c/Users/rajat/Desktop/sublime_practice$