EXP:04

DATE:16/07/2025

PATTERN MATCHING AND NFA CONSTRUCTION

1. Write a lex program to validate an expression and to categorize them into infix, prefix and postfix output files.

   Sample:
   1. Abc+*
   2. A+b-c
   3. +*abc

   Postfix file: Abc+*

   Prefix file: +*abc

   Infix file: +*abc

CODE:

```
%{
#include <stdio.h>
#include <string.h>
#include <ctype.h>
FILE *infix, *prefix, *postfix;
int isOperator(char c) { return c=='+'||c=='-'||c=='*'||c=='/'||c=='^'; }
int isOperand(char c) { return isalpha(c) || isdigit(c); }
int isPostfix(const char *exp) {
   int s=0; for(int i=0; exp[i]; i++)
      if(isOperand(exp[i])) s++;
      else if(isOperator(exp[i])) { if(s<2) return 0; s--; }
   return s==1;
}
int isPrefix(const char *exp) {
```

```
    int s=0, len=strlen(exp);
    for(int i=len-1; i>=0; i--)
        if(isOperand(exp[i])) s++;
        else if(isOperator(exp[i])) { if(s<2) return 0; s--; }
    return s==1;
}
int isInfix(const char *exp) {
    int len=strlen(exp), found=0;
    for(int i=1;i<len-1;i++)
        if(isOperand(exp[i-1])&&isOperator(exp[i])&&isOperand(exp[i+1])) found=1;
    return found;
}
%}
%%
[A-Za-z0-9+\-*/^()]+ {
    if(isPrefix(yytext))     fprintf(prefix,"%s\n",yytext);
    else if(isPostfix(yytext)) fprintf(postfix,"%s\n",yytext);
    else if(isInfix(yytext))   fprintf(infix,"%s\n",yytext);
}
[ \t\n]+ ; /* ignore whitespace and newlines */
%%
int main() {
    infix=fopen("infix.txt","w");
    prefix=fopen("prefix.txt","w");
    postfix=fopen("postfix.txt","w");
    yylex();
    fclose(infix); fclose(prefix); fclose(postfix);
    return 0;
}
```

OUTPUT:

Open

input3.txt
~/

```
1
2       A+B-C
3       X*Y/Z
4       (A+B)*C
5       P-Q+R
6       M*N+O/P
7
8       +AB
9       *+ABC
10      -A/BC
11      +*ABCD
12      -+AB*CD
13
14      AB+
15      AB+C*
16      ABCD+*-
17      XY/Z*
18      PQ-R+
```

Open

infix.txt
~/

```
1 A+B-C
2 X*Y/Z
3 (A+B)*C
4 P-Q+R
5 M*N+O/P
```

Open

postfix.txt
~/

```
1 AB+
2 AB+C*
3 ABCD+*-
4 XY/Z*
5 PQ-R+
```

Open

prefix.txt
~/

```
1 +AB
2 *+ABC
3 -A/BC
4 -+AB*CD
```

2. Write a program to construct NFA from regular expression (a|b)*.

CODE:

```
%{
#include <stdio.h>
#include <string.h>

int q[20][3], j=1, len;
char reg[20];

void print_transition_table() {
   int i;
   printf("\n\tTransition Table \n");
   printf("_____\n");
   printf("Current State |\tInput |\tNext State");
   printf("\n_____\n");
   for(i=0;i<=j;i++)
   {
      if(q[i][0]!=0) printf("\n  q[%d]\t    |  a  | q[%d]",i,q[i][0]);
      if(q[i][1]!=0) printf("\n  q[%d]\t    |  b  | q[%d]",i,q[i][1]);
      if(q[i][2]!=0)
      {
         if(q[i][2]<10) printf("\n  q[%d]\t    |  e  | q[%d]",i,q[i][2]);
         else printf("\n  q[%d]\t    |  e  | q[%d] , q[%d]",i,q[i][2]/10,q[i][2]%10);
      }
   }
   printf("\n_____\n");
}

%}

%%
[a-zA-Z|*e\(\)]+ {
   int i=0, a, b;
   for(a=0;a<20;a++)
      for(b=0;b<3;b++)
         q[a][b]=0;

   strcpy(reg, yytext);
   len=strlen(reg);

   j=1; // reset state counter
```

```
    while(i<len)
    {
       if(reg[i]=='a'&&reg[i+1]!='|'&&reg[i+1]!='*') { q[j][0]=j+1; j++; }
       if(reg[i]=='b'&&reg[i+1]!='|'&&reg[i+1]!='*') { q[j][1]=j+1; j++; }
       if(reg[i]=='e'&&reg[i+1]!='|'&&reg[i+1]!='*') { q[j][2]=j+1; j++; }
       if(reg[i]=='a'&&reg[i+1]=='|'&&reg[i+2]=='b')
       {
          q[j][2]=((j+1)*10)+(j+3); j++;
          q[j][0]=j+1; j++;
          q[j][2]=j+3; j++;
          q[j][1]=j+1; j++;
          q[j][2]=j+1; j++;
          i=i+2;
       }
       if(reg[i]=='b'&&reg[i+1]=='|'&&reg[i+2]=='a')
       {
          q[j][2]=((j+1)*10)+(j+3); j++;
          q[j][1]=j+1; j++;
          q[j][2]=j+3; j++;
          q[j][0]=j+1; j++;
          q[j][2]=j+1; j++;
          i=i+2;
       }
       if(reg[i]=='a'&&reg[i+1]=='*')
       {
          q[j][2]=((j+1)*10)+(j+3); j++;
          q[j][0]=j+1; j++;
          q[j][2]=((j+1)*10)+(j-1); j++;
       }
       if(reg[i]=='b'&&reg[i+1]=='*')
       {
          q[j][2]=((j+1)*10)+(j+3); j++;
          q[j][1]=j+1; j++;
          q[j][2]=((j+1)*10)+(j-1); j++;
       }
       if(reg[i]==')'&&reg[i+1]=='*')
       {
          q[0][2]=((j+1)*10)+1;
          q[j][2]=((j+1)*10)+1;
          j++;
       }
       i++;
    }
    printf("Given regular expression: %s\n", reg);
    print_transition_table();
}
```

```
\n   ;
.    ;
%%

int yywrap(void) { return 1; }

int main(void) {
   printf("Enter a regular expression (over a, b, e, |, *, (, )):\n");
   yylex();
   return 0;
}
```

OUTPUT:

```
adimax@LAPTOP-6RINASNI:/mnt/c/Users/adith/OneDrive/Documents$ flex adi.l
adimax@LAPTOP-6RINASNI:/mnt/c/Users/adith/OneDrive/Documents$ gcc lex.yy.c -lfl -o adi
adimax@LAPTOP-6RINASNI:/mnt/c/Users/adith/OneDrive/Documents$ ./adi
Enter a regular expression (over a, b, e, |, *, (, )):
(a|b)*
Given regular expression: (a|b)*

        Transition Table
_____________________________________
Current State | Input | Next State
_____________________________________

  q[0]        |   e   |  q[7] , q[1]
  q[1]        |   e   |  q[2] , q[4]
  q[2]        |   a   |  q[3]
  q[3]        |   e   |  q[6]
  q[4]        |   b   |  q[5]
  q[5]        |   e   |  q[6]
  q[6]        |   e   |  q[7] , q[1]
_____________________________________
```

RESULT:

The code and outputs for the LEX program and have been executed and verified.