LEXICAL ANALYZER

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>

int main()
{
    FILE *input, *output;
    int l = 1;
    int t = 0;
    int j = 0;
    int i, flag;
    char ch, str[100];
    input = fopen("lexical_input.txt", "r");
    output = fopen("lexical_output.txt", "w");
    char keyword[30][30] = { "int", "main", "if", "else", "do", "while","printf" };
    fprintf(output, "Line no. \t Token no. \t\t Token \t\t Lexeme\n\n");

    while (!feof(input))
    {
        i = 0;
        flag = 0;
        ch = fgetc(input);

        if (ch == '"')
        {
            ch = fgetc(input);

            while (ch != '"' && ch != EOF)
            {
                str[i++] = ch;
                ch = fgetc(input);
            }

            str[i] = '\0';
            fprintf(output, "%7d\t\t %7d\t\t Literal\t %7s\n", l, t, str);
            t++;
        }
        else if (ch == '+' || ch == '-' || ch == '*' || ch == '/'|| ch == '%'|| ch == '>'|| ch
== '<'|| ch == '=')
        {
            fprintf(output, "%7d\t\t %7d\t\t Operator\t %7c\n", l, t, ch);
            t++;
        }
        else if (ch == ';' || ch == '{' || ch == '}' || ch == '(' || ch == ')' || ch == '?' || ch == '@' || ch == '!' || ch == ',')
        {
            fprintf(output, "%7d\t\t %7d\t\t Special symbol\t %7c\n", l, t, ch);
            t++;
        }
        else if (isdigit(ch))
        {
            str[i++] = ch;
            ch = fgetc(input);

            while (isdigit(ch))
            {
                str[i++] = ch;
                ch = fgetc(input);
            }

            str[i] = '\0';
            fprintf(output, "%7d\t\t %7d\t\t Digit\t\t %7s\n", l, t, str);
            t++;

            if (!isspace(ch) && !isalnum(ch))
            {
                ungetc(ch, input);
            }
        }
        else if (isalpha(ch))
        {
            str[i++] = ch;
            ch = fgetc(input);

            while (isalnum(ch))
            {
                str[i++] = ch;
                ch = fgetc(input);
            }

            str[i] = '\0';

            for (j = 0; j < 30; j++)
            {
                if (strcmp(str, keyword[j]) == 0)
                {
                    flag = 1;
                    break;
                }
            }
            if (flag == 1)
            {
                fprintf(output, "%7d\t\t %7d\t\t Keyword\t %7s\n", l, t, str);
                t++;
            }
            else
            {
                fprintf(output, "%7d\t\t %7d\t\t Identifier\t %7s\n", l, t, str);
                t++;
            }

            if (!isspace(ch) && !isalnum(ch))
            {
                ungetc(ch, input);
            }
        }
        else if (ch == '\n')
        {
            l++;
        }
    }
    fclose(input);
    fclose(output);
    return 0;
}
```

Input:

lexical_input.txt :

```
int a,b,c;
d=a+b;
printf("Sum is",a);
```

Output:

lexical_output.txt :

```
Line no.        Token no.            Token          Lexeme

      1               0              Keyword            int
      1               1              Identifier           a
      1               2              Special symbol       ,
      1               3              Identifier           b
      1               4              Special symbol       ,
      1               5              Identifier           c
      1               6              Special symbol       ;
      2               7              Identifier           d
      2               8              Operator             =
      2               9              Identifier           a
      2              10              Operator             +
      2              11              Identifier           b
      2              12              Special symbol       ;
      3              13              Keyword            printf
      3              14              Special symbol       (
      3              15              Literal            Sum is
      3              16              Special symbol       ,
      3              17              Identifier           a
      3              18              Special symbol       )
      3              19              Special symbol       ;
```

E-CLOSURE OF NFA

Program:

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

char result[20][20],copy[3],states[20][20];

void add_state(char a[3],int i)
{
        strcpy(result[i],a);
}

void display(int n)
{
        int k=0;
        printf("\n Epsilon closure of %s = { ",copy);
        while(k < n)
        {
                printf(" %s",result[k]);
                k++;
        }
        printf(" } \n");
}

int main()
{
   FILE *INPUT;
   INPUT=fopen("closure_input.txt","r");
   char state[3];
   int end,i=0,n,k=0;
   char state1[3],input[3],state2[3];
   printf("\n Enter the no of states: ");
   scanf("%d",&n);
   printf("\n Enter the states \n");
   for(k=0;k<3;k++)
        {
                scanf("%s",states[k]);

        }

        for( k=0;k<n;k++)
        {
                i=0;
                strcpy(state,states[k]);
                strcpy(copy,state);
        add_state(state,i++);
        while(1)
        {
            end = fscanf(INPUT,"%s%s%s",state1,input,state2);
            if (end == EOF )
            {
                break;
            }

            if( strcmp(state,state1) == 0 )
            {
                if( strcmp(input,"e") == 0 )
                {
                    add_state(state2,i++);
                    strcpy(state, state2);
                }
            }
        }
        display(i);
        rewind(INPUT);
    }
  return 0;
}
```

Input:

Closure_input.txt :

```
q0 0 q0
q0 1 q1
q0 e q1
q1 1 q2
q1 e q2
```

Output:

```
Enter the no of states: 3

Enter the states
q0
q1
q2

Epsilon closure of q0 = {  q0 q1 q2 }

Epsilon closure of q1 = {  q1 q2 }

Epsilon closure of q2 = {  q2 }
```

E-NFA TO NFA

Program:

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

char enfa[20][3];
char final[30];
int ntrans;

int isin(char c, char str[])
{
   for (int i = 0; i < strlen(str); i++)
   {
      if (str[i] == c)
         return 1;
   }
   return 0;
}

void add(char str[], char c)
{
   if (!isin(c, str))
   {
      int len = strlen(str);
      str[len] = c;
      str[len + 1] = '\0';
   }
}

void addstate(char c1, char c2)
{
   for (int i = 0; i < ntrans; i++)
   {
      if (enfa[i][0] == c2 && enfa[i][1] != 'e')
      {
         printf("%c%c%c\n", c1, enfa[i][1], enfa[i][2]);
      }
      else if (enfa[i][0] == c2 && enfa[i][1] == 'e' && enfa[i][2] != c1)
      {
         addstate(c1, enfa[i][2]);
      }
   }
}
```

```
int main()
{
   int i;
   printf("Enter number of transitions: ");
   scanf("%d", &ntrans);
   printf("Enter transitions in format state symbol state:\n");
   for (i = 0; i < ntrans; i++)
   {
      scanf(" %c %c %c", &enfa[i][0], &enfa[i][1], &enfa[i][2]);
   }

   printf("Enter final states: ");
   scanf("%s", final);

   printf("NFA transitions:\n");
   for (i = 0; i < ntrans; i++)
   {
      if (enfa[i][1] != 'e')
      {
         printf("%c%c%c\n", enfa[i][0], enfa[i][1], enfa[i][2]);
      }
      else
      {
         addstate(enfa[i][0], enfa[i][2]);
      }
   }

   for (i = ntrans - 1; i >= 0; i--)
   {
      if (isin(enfa[i][2], final) && enfa[i][1] == 'e')
      {
         add(final, enfa[i][0]);
      }
   }
   printf("Final states: {%s}\n", final);
   return 0;
}
```

Output:

```
Enter number of transitions: 7
Enter transitions in format state symbol state:
0 b 1
0 e 2
1 b 0
2 a 3
2 b 4
3 a 2
4 a 2
Final states: 2
NFA transitions
0 b 1
0 a 3
0 b 4
1 b 0
2 a 3
2 b 4
3 a 2
4 a 2
Final states: {02}
```

FIRST AND FOLLOW

Program:

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int n, m , p, i , j ;
char production[10][10], f[10];
void follow(char c);
void first(char c);

int main()
{
   int z;
   char c;
   printf("\nNo of productions : ");
   scanf("%d", &n);
   printf("\nEnter the productions :\n");
   for (i = 0; i < n; i++)
   {
      scanf("%s", production[i]);
      getchar();
   }
   do
   {
      printf("Enter the element whose first & follow is to be found : ");
      scanf("%c", &c);

      m = 0;
      first(c);
      printf("First(%c)={", c);
      for (i = 0; i < m; i++)
         printf("%c ", f[i]);
      printf("}\n");
      strcpy(f, " ");

      m = 0;
      follow(c);
      printf("Follow(%c)={", c);
      for (i = 0; i < m; i++)
         printf("%c ", f[i]);
      printf("}\n");

      printf("\nContinue(0/1) : ");
```

```
        scanf("%d", &z);
        getchar();
    } while (z == 1);

  return (0);
}

void first(char c)
{
    int k;
    if (!isupper(c))
        f[m++] = c;
    for (k = 0; k < n; k++)
    {
        if (production[k][0] == c)
        {
            if (islower(production[k][2]))
                f[m++] = production[k][2];
            else
                first(production[k][2]);
        }
    }
}

void follow(char c)
{
    if (production[0][0] == c)
        f[m++] = '$';
    for (i = 0; i < n; i++)
    {
        for (j = 2; j < strlen(production[i]); j++)
        {
            if (production[i][j] == c)
            {
                if (production[i][j + 1] != '\0')
                    first(production[i][j + 1]);
                if (production[i][j + 1] == '\0' && c != production[i][0])
                    follow(production[i][0]);
            }
        }
    }
}
```

<u>Output:</u>

```
No of productions : 4

Enter the productions :
S=AB
A=a
A=e
B=b
Enter the element whose first & follow is to be found : S
First(S)={a e }
Follow(S)={$ }

Continue(0/1) : 1
Enter the element whose first & follow is to be found : A
First(A)={a e }
Follow(A)={b }

Continue(0/1) : 0
```

RECURSIVE DESCENT PARSER

Program:

```
/* Recursive descent parser, Grammer:
 E -> TE'
 T -> FT'
 E' -> +TE' | -TE' |ep
 T' -> *FT' | /FT' |ep
 F -> (E) | alnum
 */

#include <stdio.h>
#include <string.h>
#include <ctype.h>
char input[10];
int i, error;
void E();
void T();
void Eprime();
void Tprime();
void F();

int main()
{
   i = 0;
   error = 0;
   printf("\nEnter an arithmetic expression   :  ");
   scanf("%s", input);
   E();
   if (strlen(input) == i && error == 0)
      printf("\nString accepted..!!!\n");
   else
      printf("\nString rejected..!!!\n");
}

void E()
{
   T();
   Eprime();
}

void Eprime()
{
   if (input[i] == '+' || input[i] == '-')
   {
      i++;
      T();
      Eprime();
   }
}

void T()
{
   F();
   Tprime();
}

void Tprime()
{
   if (input[i] == '*' || input[i] == '/')
   {
      i++;
      F();
      Tprime();
   }
}

void F()
{
   if (isalnum(input[i]))
      i++;
   else if (input[i] == '(')
   {
      i++;
      E();
      if (input[i] == ')')
         i++;

      else
         error = 1;
   }
   else
      error = 1;
}
```

Output:

```
Enter an arithmetic expression   :  a+b*c-(d/e)

String accepted..!!!
```

SHIFT REDUCE PARSER

Program:

```
#include <stdio.h>
#include <string.h>
int z = 0, i = 0, j = 0, len = 0;
char buffer[16], ac[20], stk[15], act[10];
void check();

int main()
{
   printf("\nGRAMMAR is \n E->E+E \n E->E*E \n E->(E) \n E->id \n");
   printf("\nEnter input string : ");
   scanf("%s", buffer);
   len = strlen(buffer);
   printf("stack \t input \t action\n");
   for (i = 0; j < len; i++, j++)
   {
      if (buffer[j] == 'i' && buffer[j + 1] == 'd')
      {
         stk[i] = buffer[j];
         stk[i + 1] = buffer[j + 1];
         stk[i + 2] = '\0';
         buffer[j] = ' ';
         buffer[j + 1] = ' ';
         printf("\n$%s\t%s$\tSHIFT->id", stk, buffer);
         check();
      }
      else
      {
         stk[i] = buffer[j];
         stk[i + 1] = '\0';
         buffer[j] = ' ';
         printf("\n$%s\t%s$\tSHIFT->symbol %c", stk, buffer,stk[i]);
         check();
      }
   }

          if ( stk[1]=='\0')
          {
             printf("\nSTRING ACCEDPTED\n");
          }
          else
          {
             printf("\nSTRING REJECTED\n");
          }
}
void check()
{
   for (z = 0; z <= len; z++)
   {
      if (stk[z] == 'i' && stk[z + 1] == 'd')
      {
         stk[z] = 'E';
         stk[z + 1] = '\0';
         printf("\n$%s\t%s$\tREDUCE TO E", stk, buffer);
         j++;
      }
   }

   for (z = 0; z <= len; z++)
   {
      if (stk[z] == 'E' && stk[z + 1] == '+' && stk[z + 2] == 'E')
      {
         stk[z] = 'E';
         stk[z + 1] = '\0';
         printf("\n$%s\t%s$\tREDUCE TO E", stk, buffer);
         i = i - 2;
      }
   }

   for (z = 0; z <= len; z++)
   {
      if (stk[z] == 'E' && stk[z + 1] == '*' && stk[z + 2] == 'E')
      {
         stk[z] = 'E';
         stk[z + 1] = '\0';
         printf("\n$%s\t%s$\tREDUCE TO E", stk, buffer);
         i = i - 2;
      }
   }

   for (z = 0; z <= len; z++)
   {
      if (stk[z] == '(' && stk[z + 1] == 'E' && stk[z + 2] == ')')
      {
         stk[z] = 'E';
         stk[z + 1] = '\0';
         printf("\n$%s\t%s$\tREDUCE TO E", stk, buffer);
         i = i - 2;
      }
   }
}
```

Output:

```
GRAMMAR is
 E->E+E
 E->E*E
 E->(E)
 E->id

Enter input string : id+id*id+id
stack  input        action

$id     +id*id+id$ SHIFT->id
$E      +id*id+id$ REDUCE TO E
$E+      id*id+id$ SHIFT->symbol +
$E+id      *id+id$ SHIFT->id
$E+E       *id+id$ REDUCE TO E
$E         *id+id$ REDUCE TO E
$E*         id+id$ SHIFT->symbol *
$E*id         +id$ SHIFT->id
$E*E          +id$ REDUCE TO E
$E            +id$ REDUCE TO E
$E+            id$ SHIFT->symbol +
$E+id            $ SHIFT->id
$E+E             $ REDUCE TO E
$E               $ REDUCE TO E
STRING ACCEPTED
```

INTERMEDIATE CODE GENERATION

Program:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#define MAX 100

char stack[MAX];
int top = -1;

void push(char c)
{
   stack[++top] = c;
}

char pop()
{
   return stack[top--];
}

int priority(char c)
{
   if (c == '^')
      return 3;
   else if (c == '*' || c == '/')
      return 2;
   else if (c == '+' || c == '-')
      return 1;
   else
      return 0;
}

void infixToPostfix(char infix[], char postfix[])
{
   int i, j = 0;
   for (i = 0; infix[i]; i++)
   {
      if (isalpha(infix[i]))
         postfix[j++] = infix[i];
      else if (infix[i] == '(')
         push(infix[i]);
      else if (infix[i] == ')')
      {
         while (stack[top] != '(')
            postfix[j++] = pop();
            pop();
        }
        else
        {
            while (priority(stack[top]) >= priority(infix[i]))
                postfix[j++] = pop();
            push(infix[i]);
        }
    }

    while (top >= 0)
        postfix[j++] = pop();
    postfix[j] = '\0';
}

void threeadd(char *str)
{
    top=-1;
    int t1=90;
    char t2,t3;
    for(int i=0;i<strlen(str);i++)
    {
        if(isalpha(str[i]))
        {
            push(str[i]);
        }
        else
        {
            t3=pop();
            t2=pop();
            printf("%c := %c %c %c\n",t1,t2,str[i],t3);
            push(t1--);
        }
    }
}

int main()
{
    char infix[MAX], postfix[MAX];
    printf("Enter an simple expression: ");
    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    threeadd(postfix);
    return 0;
}
```

Output:

```
Enter an simple expression: (a+b)*(c+d)
Z := a + b
Y := c + d
X := Z * Y
```

BACKEND OF COMPILER

Program:

```
#include <stdio.h>
#include <string.h>
int main()
{
  FILE *f = fopen("backend_input.txt", "r+");
  char res[3], op[2], op1[2], op2[2], eq[2];
  while (fscanf(f, "%s %s %s %s %s", res, eq, op1,op, op2) != EOF)
  {
    printf("MOV AX,[%s]\n", op1);
    switch (op[0])
    {
    case '+':
      printf("ADD AX,[%s]\n", op2);
      break; // AX=AX+memory
    case '-':
      printf("SUB AX,[%s]\n", op2);
      break; // AX=AX-memory
    case '*':
      printf("MOV BX,[%s]\nMUL BX\n", op2);
      break; // AX=AX*BX
    case '/':
      printf("MOV BX,[%s]\nDIV BX\n", op2);
      break; // Quotient in AX reminder in DX
    }
    printf("MOV [%s],AX\n", res);
  }
}
```

<u>Input:</u>

backend_input.txt :

```
X = a + b
Y = X * c
Z = Y / e
```

<u>Output:</u>

```
MOV AX,[a]
ADD AX,[b]
MOV [X],AX
MOV AX,[X]
MOV BX,[c]
MUL BX
MOV [Y],AX
MOV AX,[Y]
MOV BX,[e]
DIV BX
MOV [Z],AX
```

CONSTANT PROPOGATION

Program:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
struct expr
{
   char op[2], op1[5], op2[5], res[5];
   int flag;
} arr[10];
int n;

void input()
{
   int i;
   printf("\n\nEnter the maximum number of expressions : ");
   scanf("%d", &n);
   printf("\nEnter the input : \n");
   for (i = 0; i < n; i++)
   {
      scanf("%s %s %s %s", arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);
      arr[i].flag = 0;
   }
}

void output()
{
   int i = 0;
   printf("\nOptimized code is : ");
   for (i = 0; i < n; i++)
   {
      if (!arr[i].flag)
      {
         printf("\n%s %s %s %s", arr[i].op, arr[i].op1, arr[i].op2, arr[i].res);
      }
   }
}
void change(int p, char *res)
{
   int i;
   for (i = p + 1; i < n; i++)
   {
      if (strcmp(arr[p].res, arr[i].op1) == 0)
         strcpy(arr[i].op1, res);
```

```
        if (strcmp(arr[p].res, arr[i].op2) == 0)
            strcpy(arr[i].op2, res);
    }
}
void constant()
{
    int i;
    int op1, op2, res;
    char op, res1[5];
    for (i = 0; i < n; i++)
    {
        if (isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) || strcmp(arr[i].op, "=") == 0)
        /*if both digits, store them in variables*/
        {
            op1 = atoi(arr[i].op1);
            op2 = atoi(arr[i].op2);
            op = arr[i].op[0];
            switch (op)
            {
            case '+':
                res = op1 + op2;
                break;
            case '-':
                res = op1 - op2;
                break;
            case '*':
                res = op1 * op2;
                break;
            case '/':
                res = op1 / op2;
                break;
            case '=':
                res = op1;
                break;
            }
            sprintf(res1, "%d", res);
            arr[i].flag = 1;
            change(i, res1);
        }
    }
}
void main()
{
    input();
    constant();
    output();
}
```

Output:

```
Enter the maximum number of expressions : 4

Enter the input :
= 5 _ a
+ a a b
* a b c
- c d x

Optimized code is :
- 50 d x
```