Lexical analyser

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

int iskeyword(char buffer[]){

char keywords[][10]={"int","long","goto","return","extend","float","double","char","case",

"break","auto", "else","enum","if","void","static","switch","typedef","union","signed",

"unsigned","volatile","while","void", "sizeof","register","short","do","continue","const",

"default","struct"

};

int flag=0;

for(int i=0;i<32;i++){

if(strcmp(buffer,keywords[i])==0) {

flag=1;

break;

}

}

return flag;

}

int main()

{

char ch,buffer[15], operators[]="+-/\*%=";

FILE \*fp;

fp=fopen("program.txt","r");

int j=0;

if(fp==NULL){

printf("error opening the file");

exit(0);

}

while((ch=fgetc(fp))!= EOF){

for(int i=0;i<6;++i){

if(ch==operators[i]){

printf("%c is an operator",ch);

}

}

if(isalnum(ch)){

buffer[j++]=ch;

}

else if((ch==' ' || ch=='\n') && (j!=0)){

buffer[j]='\0';

j=0;

if(iskeyword(buffer)==1){

printf("%s is keyword\n", buffer);

}

else{

printf("%s is identifier\n", buffer);

}

}

}

fclose(fp);

return 0;

}

**Re to nfa**

#include <stdio.h>

#include <string.h>

int main(){

char reg[20];

int q[20][3],i,j,len,a,b;

scanf("%s",reg);

for(a=0;a<20;a++){

for(b=0;b<3;b++){

q[a][b]=0;

}

}

len=strlen(reg);

i=0;

j=1;

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+2;

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++;

}

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++;

}

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;

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;

j++;

}

if(reg[i]==')' && reg[i+1]=='\*'){

q[0][2]=((j+1)\*10)+1;

q[j][2]=((j+1)\*10)+1;

j++;

}

i++;

}

printf("transition function\n");

for(i=0;i<=j;i++){

if(q[i][0]!=0){

printf("q[%d][a]-->%d\n",i,q[i][0]);

}

if(q[i][1]!=0){

printf("q[%d][b]-->%d\n",i,q[i][1]);

}

if(q[i][2]!=0){

if(q[i][2]<10){

printf("q[%d][e]-->%d\n",i,q[i][2]);

}

else{

printf("q[%d][e]-->%d & %d\n",i,q[i][2]/10,q[i][2]%10);

}

}

}

return 0;

}

**Left recursion**

#include <stdio.h>

#include <string.h>

int main(){

char non\_terminal,alpha,beta;

printf("Enter no. of productions:");

int num;

scanf("%d",&num);

char productions[10][10];

int index=3;

for(int i=0;i<num;i++){

scanf("%s",productions[i]);

}

for(int i=0;i<num;i++){

printf("\nGRAMMAR : : %s",productions[i]);

non\_terminal=productions[i][0];

if(non\_terminal==productions[i][index]){

alpha=productions[i][index+1];

printf("is left recursive\n");

while(productions[i][index]!=0 && productions[i][index]!='|') index++;

if(productions[i][index]!=0){

beta=productions[i][index+1];

printf("GRAMMAR WITHOUT LEFT RECURSION:\n");

printf("%c->%c%c'\n",non\_terminal,beta,non\_terminal);

printf("%c'->%c%c' | E\n",non\_terminal,alpha,non\_terminal);

}

else{

printf(" can't be reduced\n");

}

}

else{

printf("is not left recursive\n");

}

}

}

**Left factoring**

#include<stdio.h>

#include<string.h>

int main () {

char gram[20],part1[20],part2[20],modified[20],newgram[20];

int i,j=0,k=0,pos;

printf("Enter the grammar: A->");

scanf("%s",gram);

for(i=0;gram[i]!='|';i++,j++){

part1[j]=gram[i];

}

part1[j]='\0';

for(j=++i,i=0;gram[j]!='\0';j++,i++){

part2[i]=gram[j];

}

part2[i]='\0';

for(i=0;i<strlen(part1) || i<strlen(part2);i++){

if(part1[i]==part2[i]){

modified[k]=part1[i];

k++;

pos=i+1;

}

}

for(i=pos,j=0;part1[i]!='\0';i++,j++){

newgram[j]=part1[i];

}

newgram[j++]='|';

for(i=pos;part2[i]!='\0';i++,j++){

newgram[j]=part2[i];

}

modified[k]='X';

modified[++k]='\0';

newgram[j]='\0';

printf("AFTER left factoring:\n");

printf("\n A->%s",modified);

printf("\nX->%s",newgram);

}

**FIRST**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

char gram[20][20],first[20];

int k=0,n;

void funcfirst(char g,int n){

if(!(isupper(g))){

first[k++]=g;

}

else{

for(int i=0;i<n;i++){

if(gram[i][0]==g){

if(islower(gram[i][3])){

first[k++]=gram[i][3];

}

else{

funcfirst(gram[i][3],n);

}

}

}

}

}

int main(){

printf("enter no of productions:");

scanf("%d",&n);

for(int i=0;i<n;i++){

scanf("%s",gram[i]);

}

for(int i=0;i<n;i++){

k=0;

funcfirst(gram[i][0],n);

printf("first of the %c is {",gram[i][0]);

for(int i=0;i<k;i++){

printf("%c",first[i]);

}

printf("}\n");

}

return 0;

}

**Shift reduce**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

//Global Variables

int z = 0, i = 0, j = 0, c = 0;

char a[16], ac[20], stk[15], act[10];

void check()

{

strcpy(ac,"REDUCE TO E -> ");

for(z = 0; z < c; z++)

{

if(stk[z] == '4')

{

printf("%s4", ac);

stk[z] = 'E';

stk[z + 1] = '\0';

printf("\n$%s\t%s$\t", stk, a);

}

}

for(z = 0; z < c - 2; z++)

{

if(stk[z] == '2' && stk[z + 1] == 'E' && stk[z + 2] == '2')

{

printf("%s2E2", ac);

stk[z] = 'E';

stk[z + 1] = '\0';

stk[z + 2] = '\0';

printf("\n$%s\t%s$\t", stk, a);

i = i - 2;

}

}

for(z=0; z<c-2; z++)

{

if(stk[z] == '3' && stk[z + 1] == 'E' && stk[z + 2] == '3')

{

printf("%s3E3", ac);

stk[z]='E';

stk[z + 1]='\0';

stk[z + 1]='\0';

printf("\n$%s\t%s$\t", stk, a);

i = i - 2;

}

}

return ;

}

//Driver Function

int main()

{

printf("GRAMMAR is - E->2E2 \n E->3E3\nE->4\n");

strcpy(a,"32423");

c=strlen(a);

strcpy(act,"SHIFT");

printf("\nstack \t input \t action");

printf("\n$\t%s$\t", a);

for(i = 0; j < c; i++, j++)

{

printf("%s", act);

stk[i] = a[j];

stk[i + 1] = '\0';

a[j]=' ';

printf("\n$%s\t%s$\t", stk, a);

check();

}

check();

if(stk[0] == 'E' && stk[1] == '\0')

printf("Accept");

else

printf("Reject");

}

**SLR**

#include <iostream>

#include <conio.h>

#include <string.h>

using namespace std;

char prod[20][20],listofvar[26]="ABCDEFGHIJKLMNOPQR"; int novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];

int noitem=0;

struct Grammar

{

char lhs;

char rhs[8];

}g[20],item[20],clos[20][10];

int isvariable(char variable)

{

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

if(g[i].lhs==variable)

return i+1;

return 0;

}

void findclosure(int z, char a)

{

int n=0,i=0,j=0,k=0,l=0;

for(i=0;i<arr[z];i++)

{

for(j=0;j<strlen(clos[z][i].rhs);j++)

{

if(clos[z][i].rhs[j]=='.' && clos[z][i].rhs[j+1]==a) {

clos[noitem][n].lhs=clos[z][i].lhs;

strcpy(clos[noitem][n].rhs,clos[z][i].rhs);

char temp=clos[noitem][n].rhs[j];

clos[noitem][n].rhs[j]=clos[noitem][n].rhs[j+1];

clos[noitem][n].rhs[j+1]=temp;

n=n+1;

}

}

}

for(i=0;i<n;i++)

{

for(j=0;j<strlen(clos[noitem][i].rhs);j++)

{

if(clos[noitem][i].rhs[j]=='.' &&

isvariable(clos[noitem][i].rhs[j+1])>0)

{

for(k=0;k<novar;k++)

{

if(clos[noitem][i].rhs[j+1]==clos[0][k].lhs)

{

for(l=0;l<n;l++)

if(clos[noitem][l].lhs==clos[0][k].lhs &&

strcmp(clos[noitem][l].rhs,clos[0][k].rhs)==0)

break;

if(l==n)

{

clos[noitem][n].lhs=clos[0][k].lhs;

strcpy(clos[noitem][n].rhs,clos[0][k].rhs);

n=n+1;

}

}

}

}

}

}

arr[noitem]=n;

int flag=0;

for(i=0;i<noitem;i++)

{

if(arr[i]==n)

{

for(j=0;j<arr[i];j++)

{

int c=0;

for(k=0;k<arr[i];k++)

if(clos[noitem][k].lhs==clos[i][k].lhs &&

strcmp(clos[noitem][k].rhs,clos[i][k].rhs)==0)

c=c+1;

if(c==arr[i])

{

flag=1;

goto exit;

}

}

}

}

exit:;

if(flag==0)

arr[noitem++]=n;

}

int main()

{

//clrscr();

cout<<"ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :\n"; do

{

cin>>prod[i++];

}while(strcmp(prod[i-1],"0")!=0);

for(n=0;n<i-1;n++)

{

m=0;

j=novar;

g[novar++].lhs=prod[n][0];

for(k=3;k<strlen(prod[n]);k++)

{

if(prod[n][k] != '|')

g[j].rhs[m++]=prod[n][k];

if(prod[n][k]=='|')

{

g[j].rhs[m]='\0';

m=0;

j=novar;

g[novar++].lhs=prod[n][0];

}

}

}

for(i=0;i<26;i++)

if(!isvariable(listofvar[i]))

break;

g[0].lhs=listofvar[i];

char temp[2]={g[1].lhs,'\0'};

strcat(g[0].rhs,temp);

cout<<"\nAugumented Grammar \n";

for(i=0;i<novar;i++)

cout<<endl<<g[i].lhs<<"->"<<g[i].rhs<<" ";

getch();

for(i=0;i<novar;i++)

{

clos[noitem][i].lhs=g[i].lhs;

strcpy(clos[noitem][i].rhs,g[i].rhs);

if(strcmp(clos[noitem][i].rhs,"ε")==0)

strcpy(clos[noitem][i].rhs,".");

else

{

for(int j=strlen(clos[noitem][i].rhs)+1;j>=0;j--) clos[noitem][i].rhs[j]=clos[noitem][i].rhs[j-1]; clos[noitem][i].rhs[0]='.';

}

}

arr[noitem++]=novar;

for(int z=0;z<noitem;z++)

{

char list[10];

int l=0;

for(j=0;j<arr[z];j++)

{

for(k=0;k<strlen(clos[z][j].rhs)-1;k++)

{

if(clos[z][j].rhs[k]=='.')

{

for(m=0;m<l;m++)

if(list[m]==clos[z][j].rhs[k+1])

break;

if(m==l)

list[l++]=clos[z][j].rhs[k+1];

}

}

}

for(int x=0;x<l;x++)

findclosure(z,list[x]);

}

cout<<"\n\nTHE SET OF ITEMS ARE\n";

for(int z=0;z<noitem;z++)

{

cout<<"\n I"<<z<<"\n";

cout<<"-------- \n";

for(j=0;j<arr[z];j++)

cout<<clos[z][j].lhs<<"->"<<clos[z][j].rhs<<"\n"; getch();

}

getch();

}

**PREDICTIVE PARSING**

#include <bits/stdc++.h>

using namespace std;

int main()

{

char fin[10][20],st[10][20],ft[20][20],fol[20][20];

int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;

cout << ("enter the no. of nonterminals\n");

scanf("%d",&n);

cout << ("enter the productions in a grammar\n"); for(i=0;i<n;i++)

scanf("%s",st[i]);

for(i=0;i<n;i++)

fol[i][0]='\0';

for(s=0;s<n;s++)

{

for(i=0;i<n;i++)

{

j=3;

l=0;

a=0;

l1:if(!((st[i][j]>64)&&(st[i][j]<91)))

{

for(m=0;m<l;m++)

{

if(ft[i][m]==st[i][j])

goto s1;

}

ft[i][l]=st[i][j];

l=l+1;

s1:j=j+1;

}

else

{

if(s>0)

{

while(st[i][j]!=st[a][0])

{

a++;

}

b=0;

while(ft[a][b]!='\0')

{

for(m=0;m<l;m++)

{

if(ft[i][m]==ft[a][b])

goto s2;

}

ft[i][l]=ft[a][b];

l=l+1;

s2:b=b+1;

}

}

}

while(st[i][j]!='\0')

{

if(st[i][j]=='|')

{

j=j+1;

goto l1;

}

j=j+1;

}

ft[i][l]='\0';

}

}

cout << ("first \n");

for(i=0;i<n;i++)

cout << ("FIRS[%c]=%s\n",st[i][0],ft[i]); fol[0][0]='$';

for(i=0;i<n;i++)

{

k=0;

j=3;

if(i==0)

l=1;

else

l=0;

k1:while((st[i][0]!=st[k][j])&&(k<n))

{

if(st[k][j]=='\0')

{

k++;

j=2;

}

j++;

}

j=j+1;

if(st[i][0]==st[k][j-1])

{

if((st[k][j]!='|')&&(st[k][j]!='\0'))

{

a=0;

if(!((st[k][j]>64)&&(st[k][j]<91)))

{

for(m=0;m<l;m++)

{

if(fol[i][m]==st[k][j])

goto q3;

}

fol[i][l]=st[k][j];

l++;

q3:;

}

else

{

while(st[k][j]!=st[a][0])

{

a++;

}

p=0;

while(ft[a][p]!='\0')

{

if(ft[a][p]!='@')

{

for(m=0;m<l;m++)

{

if(fol[i][m]==ft[a][p])

goto q2;

}

fol[i][l]=ft[a][p];

l=l+1;

}

else

e=1;

q2:p++;

}

if(e==1)

{

e=0;

goto a1;

}

}

}

else

{

a1:c=0;

a=0;

while(st[k][0]!=st[a][0])

{

a++;

}

while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0]))

{

for(m=0;m<l;m++)

{

if(fol[i][m]==fol[a][c])

goto q1;

}

fol[i][l]=fol[a][c];

l++;

q1:c++;

}

}

goto k1;

}

fol[i][l]='\0';

}

cout << ("follow \n");

for(i=0;i<n;i++)

cout << ("FOLLOW[%c]=%s\n",st[i][0],fol[i]); cout << ("\n");

s=0;

for(i=0;i<n;i++)

{

j=3;

while(st[i][j]!='\0')

{

if((st[i][j-1]=='|')||(j==3))

{

for(p=0;p<=2;p++)

{

fin[s][p]=st[i][p];

}

t=j;

for(p=3;((st[i][j]!='|')&&(st[i][j]!='\0'));p++)

{

fin[s][p]=st[i][j];

j++;

}

fin[s][p]='\0';

if(st[i][k]=='@')

{

b=0;

a=0;

while(st[a][0]!=st[i][0])

{

a++;

}

while(fol[a][b]!='\0')

{

cout << ("M[%c,%c]=%s\n",st[i][0],fol[a][b],fin[s]);

b++;

}

}

else if(!((st[i][t]>64)&&(st[i][t]<91)))

cout << ("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]);

else

{

b=0;

a=0;

while(st[a][0]!=st[i][3])

{

a++;

}

while(ft[a][b]!='\0')

{

cout << ("M[%c,%c]=%s\n",st[i][0],ft[a][b],fin[s]);

b++;

}

}

s++;

}

if(st[i][j]=='|')

j++;

}

}

}

**Postfix to prefix**

#include <bits/stdc++.h>

**using** **namespace** std;

**bool** isOperator(**char** x)

{

**switch** (x) {

**case** '+':

**case** '-':

**case** '/':

**case** '\*':

**return** **true**;

    }

**return** **false**;

}

string postToPre(string post\_exp)

{

    stack<string> s;

**int** length = post\_exp.size();

**for** (**int** i = 0; i < length; i++) {

**if** (isOperator(post\_exp[i])) {

             string op1 = s.top();

            s.pop();

            string op2 = s.top();

            s.pop();

            string temp = post\_exp[i] + op2 + op1;

            s.push(temp);

        }

**else** {

            s.push(string(1, post\_exp[i]));

        }

    }

    string ans = "";

**while** (!s.empty()) {

        ans += s.top();

        s.pop();

    }

**return** ans;

}

// Driver Code

**int** main()

{

    string post\_exp = "ABC/-AK/L-\*";

    // Function call

    cout << "Prefix : " << postToPre(post\_exp);

**return** 0;

}

**Three address code**

#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

#include<string.h>

void small();

void dove(int i);

int p[5]={0,1,2,3,4},c=1,i,k,l,m,pi;

char sw[5]={'=','-','+','/','\*'},j[20],a[5],b[5],ch[2]; void main()

{

 printf("Enter the expression : ");

 scanf("%s",j);

 printf("The Intermediate code is :\n");  small();

}

void dove(int i)

{

 a[0]=b[0]='\0';

 if(!isdigit(j[i+2])&&!isdigit(j[i-2]))  {

 a[0]=j[i-1];

 b[0]=j[i+1];

 }

 if(isdigit(j[i+2]))

 {

 a[0]=j[i-1];

 b[0]='t';

 b[1]=j[i+2];

 }

 if(isdigit(j[i-2]))

 {

 b[0]=j[i+1];

 a[0]='t';

 a[1]=j[i-2];

 b[1]='\0';

 }

 if(isdigit(j[i+2]) &&isdigit(j[i-2]))  {

 a[0]='t';

 b[0]='t';

 a[1]=j[i-2];

 b[1]=j[i+2];

 sprintf(ch,"%d",c);

 j[i+2]=j[i-2]=ch[0];

 }

 if(j[i]=='\*')

 printf("t%d=%s\*%s\n",c,a,b);  if(j[i]=='/')

 printf("t%d=%s/%s\n",c,a,b);

 if(j[i]=='+')

 printf("t%d=%s+%s\n",c,a,b);if(j[i]=='-')  printf("t%d=%s-%s\n",c,a,b);

 if(j[i]=='=')

 printf("%c=t%d",j[i-1],--c);

 sprintf(ch,"%d",c);

 j[i]=ch[0];

 c++;

 small();

}

void small()

{

 pi=0;l=0;

 for(i=0;i<strlen(j);i++)

 {

 for(m=0;m<5;m++)

 if(j[i]==sw[m])

 if(pi<=p[m])

 {

 pi=p[m];

 l=1;

 k=i;

 }

 }

 if(l==1)

 dove(k);

 else

 exit(0);

}

**Input:**

a=b+c-d