**Implementation Of SLR Parser**

**Code:-**

#include<stdio.h>

#include<ctype.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

#include<iostream.h>

#define epsilon '^'

char prod[20][20],T[20],NT[20],c[10][10],foll[10][10],fir[10][10];

int tt,tnt,tp,a;

int follow[20][20],first[20][20];

void first\_of(char);

int count(int j);

void rhs(int j);

void read\_tnt();

int rhs(int j);

void read\_tnt()

{ cout<<"For SLR parser: ";

 cout<<"\nEnter number of terminals: ";

 cin>>tt;

 cout<<"\nEnter terminals: ";

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

 T[i]=getche();

 getch();

 cout<<"\nEnter number of Non-terminals: ";

 cin>>tnt;

 cout<<"\nEnter Non-terminals: ";

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

 NT[i]=getche();

 getch(); }

void read\_prod()

{ int j;

 char x=0;

 cout<<"\n\nEnter number of productions: ";

 cin>>tp;

 cout<<"\n Enter productions: ";

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

 { j=x=0;

 while(x!='\r')

 { prod[i][j]=x=getche();

 j++; }

 cout<<"\n"; }

 getch(); }

 return(i);

if(t=='$')

 return(tt);

return(-1); }

int terminal(char x)

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

 if(T[i]==x)

 return(1);

 return(0); }

int nonterminal(char x)

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

 if(NT[i]==x)

 return(1);

 return(0); }

int in\_rhs(char \*s,char x)

{ for(int i=0;i<=strlen(s);i++)

 if(\*(s+i)==x)

 return(i);

 return(-1); }

void find\_first()

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

 first\_of(NT[i]); }

void first\_of(char n)

{ int t1,t2,p1,cnt=0,i,j;

 char x;

 static int over[20];

 p1=t\_no(epsilon);

 if(terminal(n))

 return;

 t1=nt\_no(n);

 if(over[t1])

 return;

 over[t1]=1;

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

 { t1=nt\_no(prod[i][0]);

 if(prod[i][0]==n)

 { int k=0;

 cnt=count(1);

 rhs(i);

 while(k<cnt)

 { x=c[i][k];

 if(terminal(x))

 { t2=t\_no(x);

 first[t1][t2]=1;

 break; }

 else

 { t2=nt\_no(x);

 first\_of(x);

 for(int j=0;j<tt;j++)

 first[t1][p1]=1; } } }

void follow\_of(char n)

{ int f,t1,t2,p1,t,cnt=0;

 char x,beta;

 static int over[20];

 p1=t\_no(epsilon);

 t1=nt\_no(n);

 if(over[t1])

 return;

 over[t1]=1;

 if(NT[0]==n)

 follow[nt\_no(NT[0])][tt]=1;

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

 { rhs(i);

 cnt=count(i);

 t=in\_rhs(c[i],n);

 int bno;

 for(int j=0;j<tt;j++)

 {

 bno=nt\_no(beta);

 if((first[bno][j]) && (j!=p1))

 follow[t1][j]=1; }

 if((p1!=-1) && (first[bno][p1]==1))

 continue;

 else if((t==(cnt-1)||(k>=cnt)))

 { follow\_of(prod[i][0]);

 t1=nt\_no(prod[i][0]);

 for(int l=0;l<=tt+1;l++)

 if(follow[t][l])

 follow[t1][l]=1; } } } }

 int count(int j)

 { int c1=0;

 for(int q=3;prod[j][q]!='\r';q++)

 c1++;

 return(c1); }

void show\_follow()

{ int b=0;

 a=0;

 cout<<"\n\n Follow Table For Grammar: \n";

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

 {

 b=0;

 cout<<"\n FOLLOW ("<<NT[i]<<" )= { ";

 for(int j=0;j<tt+1;j++)

 if(follow[i][j] && j!=tt)

 { foll[a][b]=T[j];

 b++;

 cout<<T[j]<<" "; }

 else

 if(j==tt)

 { foll[a][b]='$';

 b++;

 cout<<'$'; }

 a++;

 cout<<" } "; }

 getch(); }

 void show\_first()

 { int b=0;

 a=0;

 cout<<"\n\n First Table For Grammar: \n";

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

 { b=0;

 cout<<"\n FIRST ("<<NT[i]<<" )= { ";

 for(int j=0;j<tt+1;j++)

 if(first[i][j] && j!=tt)

 { fir[a][b]=T[j];

 b++;

 cout<<T[j]<<" "; }

 a++;

 cout<<" } "; }

 getch()}}}}

To construct parse table:

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>

#include<stdlib.h>

#include<iostream.h>

#include"c:\tc\bin\SLR.h"

int S=0,i=0,j=0,state[20];

char TNT[15];

struct node

{ int pno,dpos; };

struct t

{ char s;

 int n; };

struct t1

{ struct t lr[10];

 int gr[5]; };

struct t1 action[15];

struct node closure[10][10];

int g[15][10];

int l;

void sclosure(int,int);

int added(int);

int t\_into(char);

void print\_table(int);

void parser(void);

void find\_closure(int,int);

void SLR(void);

void main()

{ clrscr();

 mainf();

 getch();

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

 TNT[i]=NT[i];

 for(int j=0;j<tt;j++)

 { TNT[i]=T[j];

 i++; }

 strcat(T,"$");

 i=j=0;

 SLR();

 print\_table(S);

 getch(); }

void SLR()

{ int clno,no=0,x,y,z,len,cnt=-1,d=0;

 closure[i][j].pno=0;

 closure[i][j++].dpos=3;

 find\_closure(no,3);

 sclosure(i,j);

 state[i]=j;

 S=0;

 do

 { cnt++;

 z=state[cnt];

 for(int k=0;k<tnt+tt;k++)

 { i++;

 j=0;d=0;

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

 { x=closure[cnt][1].pno;

 y=closure[cnt][1].dpos;

 if(prod[x][y]==TNT[k])

 { d=1;

 closure[i][j].pno=x;

 closure[i][j++].dpos=++y;

 if((y<strlen(prod[x])) && (isupper(prod[x][y])))

 find\_closure(x,y); } }

 if(d==0)

 { i--;

 continue; }

 sclosure(i,j);

 else

 { action[cnt].lr[k-tnt].s='S';

 action[cnt].lr[k-tnt].n=clno;

 }

 if(added(i-1)!=-1)

 i--;

 else

 { S++;

 for(l=0;l<state[i];l++)

 { if(closure[i][1].pno==0)

 { action[i].lr[tt].s='A';

 continue; }

 len=(strlen(prod[closure[i][l].pno])-1);

 if(len==closure[i][l].dpos)

 { char v=prod[closure[i][l].pno][0];

 int u=nt\_no(v);

 for(x=0;x<strlen(foll[u]);x++)

 { int w=t\_ino(foll[u][x]);

 action[i].lr[w].s='R';

 action[i].lr[w].n=closure[i][l].pno;}}}}}}

 while(cnt!=S); }

void print\_table(int states)

{ int lin=5;

 cout<<"\n\n Parser Table: \n";

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

 cout<<"\t"<<T[i];

 cout<<"\t$";

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

 cout<<"\t"<<NT[i];

 cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

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

 { gotoxy(l,lin);

 cout<<"I"<<i<<"\t";

 for(int j=0;j<=tt;j++)

 { if(action[i].lr[j].s!='\x0')

 { if(action[i].lr[j].s=='A')

 { cout<<"Acc";

 continue; }

 else

 cout<<"\t"; }

 for(j=0;j<tnt;j++)

 if(action[i].gr[j])

 { cout<<action[i].gr[j];

 cout<<"\t"; }

 else

 cout<<"\t";

 lin++;

 cout<<"\n"; }

 cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"; }

void sclosure(int clno,int prodno)

{ struct node temp;

 for(int i=0;i<prodno-1;i++)

 { for(int j=i+1;j<prodno;j++)

 { if(closure[clno][i].pno>closure[clno][j].pno)

 { temp=closure[clno][i];

 closure[clno][i]=closure[clno][j];

 closure[clno][j]=temp; }}}

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

 {for(j=i+1;j<prodno;j++)

 {if((closure[clno][i].dpos>closure[clno][j].dpos) &&

 (closure[clno][i].pno==closure[clno][j].pno))

 { temp=closure[clno][i];

 closure[clno][i]=closure[clno][j];

 closure[clno][j]=temp;}}}}

int added(int n)

{ int d=1;

 for(int k=0;k<=n;k++)

 {if(state[k]==state[n+1])

 { d=0;

return(k); } }

 return(-1); }

void find\_closure(int no,int dp)

{ int k;

 char temp[5];

 if(isupper(prod[no][dp]))

 {for(k=0;k<tp;k++)

 {if(prod[k][0]==prod[no][dp])

 { int t\_ino(char t)

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

 if(T[i]==t)

 return(i);

 return(-1); }

char pops2;

struct node1

{ char s2;int s1; };

struct node1 stack[10];

int pops1,top=0;

void parser(void)

{ int r,c;

 struct t lr[10];

 char t,acc='f',str[10];

 cout<<"Enter I/p String To Parse: ";

 cin>>str;

 strcat(str,"$");

 stack[0].s1=0;

 stack[0].s2='\n';

 cout<<"\n\n STACK";

 cout<<"\t\t INPUT";

 cout<<"\t\t ACTION";

 for(int j=0;j<strlen(str);j++)

 cout<<str[j];

 do

 {r=stack[top].s1;

 c=find\_index(str[i]);

 if(c==-1)

 cout<<"\n Error! Invalid String!";

 return; }

 while(top!=0);

 switch(action[r],lr[c].s)

 {case 'S': { push(str[i],action[r].lr[c].n);

 i++;

 cout<<"\t\t\t Shift";

 break; }

 case 'R': { t=prod[action[r].lr[c].n][3];

 do { pop(); }

 while(pops2!=t);

 t=prod[action[r].lr[c].n][0];

 r=stack[top].s1;

 c=find\_index(t);

 push(t,action[r].gr[c-tt-1]);

 cout<<"\t\t\t Reduce";

 break;}

 case 'A':{ cout<<"\t\t\t Accept";

 cout<<"\n\n\n String accepted";

 acc='t';

 getch();

 return; }

 default: { cout<<"\n\n\n Error! String not accepted!";

 getch();

 exit(0);}}

for(j=0;j<=top;j++)

 cout<<stack[j].s2<<stack[j].s1;

if(top<4)

 cout<<"\t\t\t";

else

 cout<<"\t\t";

for(j=i;j<strlen(str);j++)

 cout<<str[j];

if(acc=='t')

 return; }

int find\_index(char temp)

{for(int i=0;i<=tt+tnt;i++)

 {if(i<=tt)

 { if(T[i]==temp)

 return(i);}

 else

 if(NT[i-tt-1]==temp)

 return(i); }

return(-1); }

void push(char t2,int t1)

{++top;

stack[top].s1=t1;

stack[top].s2=t2;

return; }

void pop(void)

{pops1=stack[top].s1;

pops2=stack[top].s2;

--top; getch(); }

**Output:-**

****

Enter number of terminals: 5

Enter terminals:+\*()i

Enter number of non-terminals:3

Enter non-terminals:ETF

Enter number of productions:6

Enter productions:

E->E+T

E->T

T->T\*F

T->F

F->(E)

F->i

Follow table:

FOLLOW(E)={+ ) $}

FOLLOW(F)={+ \* ) $}

FOLLOW(T)={ + \* ) $}

First Table :

FIRST(E)={ ( i }

FIRST(E)={ ( i }

FIRST(E)={ ( i }

Expected parse table:

 + \* ( ) i $ E T F

I0 S4 S5 1 2 3

I1 S6 ACC

I2 R1 S7 R1 R1

I3 R3 R3 R3 R3

I4 S4 S5 ACC 8 2 3

I5 R5 R5 R5 R5

I6 ACC

I7 S4 S5 9

I8 S10 S11 ACC

I9 R2 R2 R2 R2

I10 ACC

I11 R4 R4 R4 R4

Enter i/p string: i+i\*i

STACK INPUT ACTION

0 i+i\*i$ Shift

0i5 +i\*i$ Reduce

0F3 +i\*i$ Reduce

0T2 +i\*i$ Reduce

0E1 +i\*i$ Shift

0E1+6 i\*i$

ERROR! STRING NOT ACCEPTED!

**Experiment No.- 7**