Saket Kumar Baranwal – RA1911003010414 – G1

**COMPILER DESIGN LAB**

Aim:- To study the implementation of SLR Parsing.

**SLR Parser :**  
SLR is simple LR. It is the smallest class of grammar having few number of states. SLR is very easy to construct and is similar to LR parsing. The only difference between SLR parser and LR(0) parser is that in LR(0) parsing table, there’s a chance of ‘shift reduced’ conflict because we are entering ‘reduce’ corresponding to all terminal states. We can solve this problem by entering ‘reduce’ corresponding to FOLLOW of LHS of production in the terminating state. This is called SLR(1) collection of items

**Steps for constructing the SLR parsing table :**

1. Writing augmented grammar
2. LR(0) collection of items to be found
3. Find FOLLOW of LHS of production
4. Defining 2 functions:goto[list of terminals] and action[list of non-terminals] in the parsing table

**STEP1 –** Find augmented grammar

**STEP2 –**Find LR(0) collection of items

**STEP3 –**Find FOLLOW of LHS of production

**STEP 4 ­­-**Defining 2 functions:goto[list of non-terminals] and action[list of terminals] in the parsing table. Below is the SLR parsing table.

**Code:-**

#include<iostream>

#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()

{

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<<"\n\n augumented grammar \n";

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

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

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 THE SET OF ITEMS ARE \n\n";

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

{

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

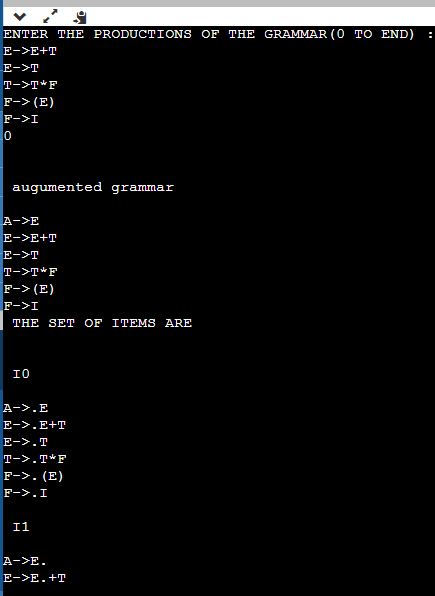
for(j=0;j<arr[z];j++)

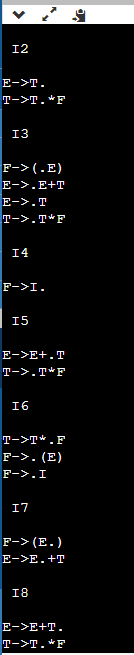
cout<<clos[z][j].lhs<<"->"<<clos[z][j].rhs<<"\n";

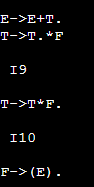
}

}

**Output Screenshot:-**

****

****

****