EXP 9

SLR PARSING

AIM: A program to find the implementation of **SLR PARSING**

ALGORITHM:

1. Start.
2. Create structure for production with LHS and RHS.
3. Open file and read input from file.
4. Build state 0 from extra grammar Law S' -> S $ that is all start symbol of grammar and one Dot (.) before S symbol.
5. If Dot symbol is before a non-terminal, add grammar laws that this non-terminal is in Left Hand Side of that Law and set Dot in before of first part of Right Hand Side.
6. If state exists (a state with this Laws and same Dot position), use that instead.
7. Now find set of terminals and non-terminals in which Dot exist in before.
8. If step 7 Set is non-empty go to 9, else go to 10.
9. For each terminal/non-terminal in set step 7 create new state by using all grammar law that Dot position is before of that terminal/non-terminal in reference state by increasing Dot point to next part in Right Hand Side of that laws.
10. Go to step 5.
11. End of state building.
12. Display the output.
13. End

CODE:

```
#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()
{
	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";

    }

}
```

PROBLEM:

Example 2 :

- Construct SLR parsing table for the grammar

  S → L = R | R

  L → * R | **id**

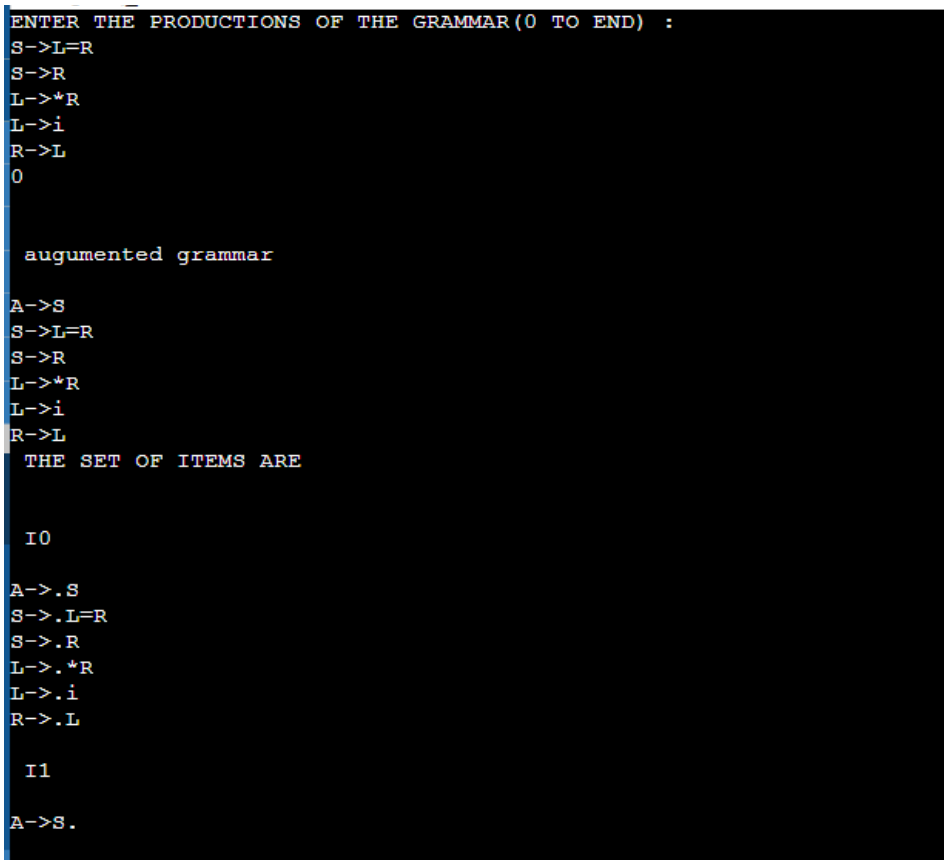
  R → L
- **Step 1** : Find FIRST and FOLLOW for all the non terminals

  FIRST(S) = { * , id } FOLLOW(S) = { $ }

  FIRST(L) = { * , id } FOLLOW(L) = { = , $ }

  FIRST(R) = { * , id } FOLLOW(R) = { = , $ }
- **Step 2** : Number all the productions
  1. S → L = R
  2. S → R
  3. L → * R
  4. L → **id**
  5. R → L

OUTPUT:



```
 I2

S->L.=R
R->L.

 I3

S->R.

 I4

L->*.R
R->.L
L->.*R
L->.i

 I5

L->i.
```

```
 I6

S->L=.R
R->.L
L->.*R
L->.i

 I7

L->*R.

 I8

R->L.

 I9

S->L=R.


...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT:

The Program for implementing SLR PARSING was successfully compiled and executed

RA1911003010387

CHITRALEKHA.CH