SLR PARSER

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

using namespace std;

char terminals[100]={};
int no_t;
char non_terminals[100]={};
int no_nt;
char goto_table[100][100];
char follow[20][20];
char first[20][20];

struct state
{
	int prod_count;
	char prod[100][100];
};

void add_dots(struct state *I)
{
	int i,j;
	for(i=0;i<I->prod_count;i++)
	{
		for (j=99;j>3;j--)
			I->prod[i][j] = I->prod[i][j-1];
		I->prod[i][3]='.';
	}
}

void augument(struct state *S,struct state *I)
{
	if(I->prod[0][0]=='S')
		strcpy(S->prod[0],"Z->.S");
	else
	{
		strcpy(S->prod[0],"S->.");
		S->prod[0][4]=I->prod[0][0];
	}
	S->prod_count++;
}


bool is_non_terminal(char a)
{
	if (a >= 'A' && a <= 'Z')
		return true;
	else
		return false;
}
```

```
void get_prods(struct state *I)
{
	int i;
	cout<<"Enter the number of productions:\n";
	cin>>I->prod_count;
	cout<<"Enter the number of non terminals:"<<endl;
	cin>>no_nt;
	cout<<"Enter the non terminals one by one:"<<endl;
	for(i=0;i<no_nt;i++)
		cin>>non_terminals[i];
	cout<<"Enter the number of terminals:"<<endl;
	cin>>no_t;
	cout<<"Enter the terminals (single lettered) one by one:"<<endl;
	for(i=0;i<no_t;i++)
		cin>>terminals[i];
	cout<<"Enter the productions one by one in form (S->ABc):\n";
	for(i=0;i<I->prod_count;i++)
	{
		cin>>I->prod[i];
	}
}

bool in_state(struct state *I,char *a)
{
	int i;
	for(i=0;i<I->prod_count;i++)
	{
		if(!strcmp(I->prod[i],a))
			return true;
	}
	return false;
}

char char_after_dot(char a[100])
{
	int i;
	for(i=0;i<strlen(a);i++)
		if(a[i]=='.')
			return a[i+1];
}

char* move_dot(char b[100],int len)
{
	char a[100]={};
	int i;
	strcpy(a,b);
	for(i=0;i<len;i++)
	{
		if(a[i]=='.')
		{
			swap(a[i],a[i+1]);
			break;
		}
	}
	return &a[0];
}
```

```
bool same_state(struct state *I0,struct state *I)
{
	int i,j;
	if (I0->prod_count != I->prod_count)
		return false;
	for (i=0; i<I0->prod_count; i++)
	{
		int flag = 0;
		for (j=0; j<I->prod_count; j++)
			if (strcmp(I0->prod[i], I->prod[j]) == 0)
				flag = 1;
		if (flag == 0)
			return false;
	}
	return true;
}

void closure(struct state *I,struct state *I0)
{
	char a={};
	int i,j;
	for(i=0;i<I0->prod_count;i++)
	{
		a=char_after_dot(I0->prod[i]);
		if(is_non_terminal(a))
			for(j=0;j<I->prod_count;j++)
				if(I->prod[j][0]==a&&(!in_state(I0,I->prod[j])))
				{
					strcpy(I0->prod[I0->prod_count],I->prod[j]);
					I0->prod_count++;
				}
	}
}

void goto_state(struct state *I,struct state *S,char a)
{
	int i;
	for(i=0;i<I->prod_count;i++)
	{
		if(char_after_dot(I->prod[i])==a)
		{
			strcpy(S->prod[S->prod_count],move_dot(I->prod[i],strlen(I->prod[i])));
			S->prod_count++;
		}
	}
}

void print_prods(struct state *I)
{
	int i;
	for(i=0;i<I->prod_count;i++)
		printf("%s\n",I->prod[i]);
	cout<<endl;
}
```

```
bool in_array(char a[20],char b)
{
	int i;
	for(i=0;i<strlen(a);i++)
		if(a[i]==b)
			return true;
	return false;
}

void cleanup_prods(struct state * I)
{
	char a[100]={};
	int i;
	for(i=0;i<I->prod_count;i++)
		strcpy(I->prod[i],a);
	I->prod_count=0;
}

int return_index(char a)
{
	int i;
	for(i=0;i<no_t;i++)
		if(terminals[i]==a)
			return i;
	for(i=0;i<no_nt;i++)
		if(non_terminals[i]==a)
			return no_t+i;
}

void add_to_first(int n,char b)
{
	int i;
	for(i=0;i<strlen(first[n]);i++)
		if(first[n][i]==b)
			return;
	first[n][strlen(first[n])]=b;
}

void add_to_first(int m,int n)
{
	int i,j;
	for(i=0;i<strlen(first[n]);i++)
	{
		int flag=0;
		for(j=0;j<strlen(first[m]);j++)
			if(first[n][i]==first[m][j])
				flag=1;
		if(flag==0)
			add_to_first(m,first[n][i]);
	}
}
```

```
void shift_table(int state_count,char shift_reduce_table[100][100][3])
{
	int i,j;
	for(i=0;i<state_count;i++)
	{
		int arr[no_nt+no_t]={-1};
		for(j=0;j<state_count;j++)
			if(goto_table[i][j]!='~')
				arr[return_index(goto_table[i][j])]= j;
		for(j=0;j<no_nt+no_t;j++)
		{
			if(i==1&&j==no_t-1)
			{
				shift_reduce_table[i][j][0]='A';
				shift_reduce_table[i][j][1]='C';
				shift_reduce_table[i][j][2]='C';
			}
			if(!(arr[j]==-1||arr[j]==0))
			{
				if(j<no_t)
					{
					shift_reduce_table[i][j][0]='S';
					shift_reduce_table[i][j][1]=arr[j]/10+'0';
					shift_reduce_table[i][j][2]=arr[j]%10+'0';
					}
				else
					{
						shift_reduce_table[i][j][1]=arr[j]/10+'0';
						shift_reduce_table[i][j][2]=arr[j]%10+'0';
					}
			}
		}
	}
}

void add_dot_at_end(struct state* I)
{
	int i;
	for(i=0;i<I->prod_count;i++)
		strcat(I->prod[i],".");
}

void add_to_follow(int n,char b)
{
	int i;
	for(i=0;i<strlen(follow[n]);i++)
		if(follow[n][i]==b)
			return;
	follow[n][strlen(follow[n])]=b;
}

void add_to_follow(int m,int n)
{
	int i,j;
	for(i=0;i<strlen(follow[n]);i++)
	{
		int flag=0;
```

```
		for(j=0;j<strlen(follow[m]);j++)
			if(follow[n][i]==follow[m][j])
				flag=1;
		if(flag==0)
			add_to_follow(m,follow[n][i]);
	}
}

void add_to_follow_first(int m,int n)
{
	int i,j;
	for(i=0;i<strlen(first[n]);i++)
	{
		int flag=0;
		for(j=0;j<strlen(follow[m]);j++)
			if(first[n][i]==follow[m][j])
				flag=1;
		if(flag==0)
			add_to_follow(m,first[n][i]);
	}
}

void find_first(struct state *I)
{
	int i,j;
	for(i=0;i<no_nt;i++)
		for(j=0;j<I->prod_count;j++)
			if(I->prod[j][0]==non_terminals[i]&&(!is_non_terminal(I->prod[j][3])))
				add_to_first(i,I->prod[j][3]);
}

void find_follow(struct state *I)
{
	int i,j,k;
	for(i=0;i<no_nt;i++)
	  for(j=0;j<I->prod_count;j++)
	      for(k=3;k<strlen(I->prod[j]);k++)
		if((I->prod[j][k]==non_terminals[i])&&(I->prod[j][k+1]!='\0')&&(!is_non_terminal(I->prod[j][k+1])))
			add_to_follow(i,I->prod[j][k+1]);
}

void reduce_table(int state_count,int *no_re,struct state *temp1,char shift_reduce_table[100][100][3])
{
	int i,j,k;
	int arr[temp1->prod_count][no_t]={-1};
	for(i=0;i<temp1->prod_count;i++)
	{
		int n=no_re[i];
		for(j=0;j<strlen(follow[return_index(temp1->prod[i][0])-no_t]);j++)
			for(k=0;k<no_t;k++)
				if(follow[return_index(temp1->prod[i][0])-no_t][j]==terminals[k])
					arr[i][k]=i+1;
		for(j=0;j<no_t;j++)
		{
			if(arr[i][j]!=-1&&arr[i][j]!=0&&arr[i][j]<state_count)
			{
				shift_reduce_table[n][j][0]='R';
```

```
                                        shift_reduce_table[n][j][1]=arr[i][j]/10+'0';
                                        shift_reduce_table[n][j][2]=arr[i][j]%10+'0';
                                }
                        }
                }
}

void print_shift_reduce_table(char shift_reduce_table[100][100][3],int state_count)
{
        int i,j,k;
        cout<<"\t";
        for(i=0;i<no_t;i++)
                cout<<terminals[i]<<"\t";
        for(i=0;i<no_nt;i++)
                cout<<non_terminals[i]<<"\t";
        cout<<endl;
        for(i=0;i<state_count;i++)
        {
                cout<<"I"<<i<<"\t";
                for(j=0;j<no_t+no_nt;j++)
                {
                        for(k=0;k<3;k++)
                        cout<<shift_reduce_table[i][j][k];
                        cout<<"\t";
                }
                cout<<"\n";
        }
}

int main()
{
        struct state init,temp,temp1,I[50];
        int state_count=1,i,j,k,z,l;
        get_prods(&init);
        temp1=temp=init;
        add_dots(&init);

        for(i=0;i<100;i++)
                for(j=0;j<100;j++)
                        goto_table[i][j]='~';

        augument(&I[0],&init);
        closure(&init,&I[0]);
        cout<<"\nI0:\n";
        print_prods(&I[0]);

        char characters[20]={};
        for(i=0;i<state_count;i++)
        {
                char characters[20]={};
                for(z=0;z<I[i].prod_count;z++)
                        if(!in_array(characters,char_after_dot(I[i].prod[z])))
                                characters[strlen(characters)]=char_after_dot(I[i].prod[z]);
                for(j=0;j<strlen(characters);j++)
                {
                        goto_state(&I[i],&I[state_count],characters[j]);
                        closure(&init,&I[state_count]);
```

```
                int flag=0;
                for(k=0;k<state_count-1;k++)
                {
                        if(same_state(&I[k],&I[state_count]))
                        {
                                cleanup_prods(&I[state_count]);flag=1;
                                cout<<"I"<<i<<" --- "<<characters[j]<<" ---> I"<<k<<"\n";
                                goto_table[i][k]=characters[j];
                                break;
                        }
                }
                if(flag==0)
                {
                        state_count++;
                        cout<<"I"<<i<<" --- "<<characters[j]<<" ---> I";
                        cout<<state_count-1<<"\n\nI";
                        cout<<state_count-1<<":\n";
                        goto_table[i][state_count-1]=characters[j];
                        print_prods(&I[state_count-1]);
                }
        }
}

int no_re[temp.prod_count]={-1};
terminals[no_t]='$';
no_t++;

add_dot_at_end(&temp1);
for(i=0;i<state_count;i++)
{
        for(j=0;j<I[i].prod_count;j++)
                for(k=0;k<temp1.prod_count;k++)
                        if(in_state(&I[i],temp1.prod[k]))
                                no_re[k]=i;
}

find_first(&temp);
for(l=0;l<no_nt;l++)
        for(i=0;i<temp.prod_count;i++)
                if(is_non_terminal(temp.prod[i][3]))
                    add_to_first(return_index(temp.prod[i][0])-no_t,return_index(temp.prod[i][3])-no_t);

find_follow(&temp);
add_to_follow(0,'$');
for(l=0;l<no_nt;l++)
{
        for(i=0;i<temp.prod_count;i++)
        {
                for(k=3;k<strlen(temp.prod[i]);k++)
                {
                        if(temp.prod[i][k]==non_terminals[l])
                        {
                                if(is_non_terminal(temp.prod[i][k+1]))
                                        add_to_follow_first(l,return_index(temp.prod[i][k+1])-no_t);
                                if(temp.prod[i][k+1]=='\0')
                                        add_to_follow(l,return_index(temp.prod[i][0])-no_t);
                        }
```

```
                        }
                    }
                }
                char shift_reduce_table[100][100][3];
                shift_table(state_count,shift_reduce_table);
                reduce_table(state_count,&no_re[0],&temp1,shift_reduce_table);
                cout<<"\n\n";
                print_shift_reduce_table(shift_reduce_table,state_count);
                return 0;
}
```

LALR PARSER

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

using namespace std;

char terminals[100]={};
int no_t;
char non_terminals[100]={};
int no_nt;
char goto_table[100][100];
char reduce[20][20];
char fo_co[20][20];
char first[20][20];
char *lookahead(char a[100]);
struct state
{
	int prod_count;
	char prod[100][100];
};

void add_dots(struct state *I)
{
	int i,j;
	for(i=0;i<I->prod_count;i++)
	{
		for (j=99;j>3;j--)
			I->prod[i][j] = I->prod[i][j-1];
		I->prod[i][3]='.';
	}
}

void augument(struct state *S,struct state *I)
{
	if(I->prod[0][0]=='S')
		strcpy(S->prod[0],"Z->.S $");
	else
	{
		strcpy(S->prod[0],"S->.");
		S->prod[0][4]=I->prod[0][0];
		S->prod[0][5]= ' ';
		S->prod[0][6]= '$';
	}
	S->prod_count++;
}
```

```
void get_prods(struct state *I)
{
	int i;
	cout<<"Enter the number of productions:\n";
	cin>>I->prod_count;
	cout<<"Enter the number of non terminals:"<<endl;
	cin>>no_nt;
	cout<<"Enter the n on terminals one by one:"<<endl;
	for(i=0;i<no_nt;i++)
		cin>>non_terminals[i];
	cout<<"Enter the number of terminals:"<<endl;
	cin>>no_t;
	cout<<"Enter the terminals (single lettered) one by one:"<<endl;
	for(i=0;i<no_t;i++)
		cin>>terminals[i];
	cout<<"Enter the productions one by one in form (S->ABc):\n";
	for(i=0;i<I->prod_count;i++)
	{
		cin>>I->prod[i];
	}
}

bool is_non_terminal(char a)
{
	if (a >= 'A' && a <= 'Z')
		return true;
	else
		return false;
}

bool in_state(struct state *I,char *a)
{
	int i,j;
	for(i=0;i<I->prod_count;i++)
	{
		for(j=0;a[j]!=' '&&j<strlen(I->prod[i]);j++)
			if(I->prod[i][j]!=a[j])
				break;
		if(a[j]==' '&&j==strlen(I->prod[i]))
		return true;
	}
	return false;
}

bool term_state(struct state *I,char *a)
{
	int i,j;
	for(i=0;i<I->prod_count;i++)
	{
		for(j=0;a[j]!='.'&&j<strlen(I->prod[i]);j++)
		{
			if(I->prod[i][j]!=a[j])
				break;
		}
		if(a[j]=='.'&&I->prod[i][j]=='.')
		{
			return true;
```

```
                }
        }
        return false;
}

char char_after_dot(char a[100])
{
        int i;
        for(i=0;i<strlen(a);i++)
                if(a[i]=='.')
                        return a[i+1];
}

char* move_dot(char b[100],int len)
{
        char a[100]={};
        int i;
        strcpy(a,b);
        for(i=0;i<len;i++)
        {
                if(a[i]=='.')
                {
                        swap(a[i],a[i+1]);
                        break;
                }
        }
        return &a[0];
}

bool same_state(struct state *I0,struct state *I)
{
        int i,j,k;
        if (I0->prod_count != I->prod_count)
                return false;
        for (i=0; i<I0->prod_count; i++)
        {
                int flag = 0;
                        for(k=0;I0->prod[i][k]!=' ';k++)
                        {
                                if(I0->prod[i][k]!=I->prod[i][k])
                                {
                                        flag=1;
                                        break;
                                }
                        }
                if (flag == 1||I->prod[i][k]!=' ')
                        return false;
        }
        return true;
}

void merge_state(struct state *I0,struct state *I)
{
        int i,j,k,l,m,flag;
        for (i=0; i<I0->prod_count; i++)
        {
                for(j=0;I0->prod[i][j]!=' ';j++);
```

```
                for(k=0;I->prod[i][k]!=' ';k++);
                for(l=k+1;l<strlen(I->prod[i]);l++)
                {
                        flag=0;
                        for(m=j+1;m<strlen(I0->prod[i]);m++)
                        {
                                if(I0->prod[i][m]==I->prod[i][l])
                                flag=1;
                        }
                        if(flag==0)
                        {
                                I0->prod[i][m]=I->prod[i][l];
                        }
                }
        }
}

void closure(struct state *I,struct state *I0)
{
        char a={};
        int i,j,k,l;
        for(i=0;i<I0->prod_count;i++)
        {
                a=char_after_dot(I0->prod[i]);
                if(is_non_terminal(a))
                {
                        char *look=lookahead(I0->prod[i]);
                        for(j=0;j<I->prod_count;j++)
                                if(I->prod[j][0]==a&&(!in_state(I0,I->prod[j])))
                                {
                                        strcpy(I0->prod[I0->prod_count],I->prod[j]);
                                        l=strlen(I0->prod[I0->prod_count]);
                                        I0->prod[I0->prod_count][l]=' ';
                                        for(k=0;k<strlen(look);k++)
                                        {
                                        I0->prod[I0->prod_count][l+k+1]=look[k];
                                        }
                                        I0->prod_count++;
                                }
                }
        }
}

void goto_state(struct state *I,struct state *S,char a)
{
        int i;
        for(i=0;i<I->prod_count;i++)
        {
                if(char_after_dot(I->prod[i])==a)
                {
                        strcpy(S->prod[S->prod_count],move_dot(I->prod[i],strlen(I->prod[i])));
                        S->prod_count++;
                }
        }
}
```

```
void print_prods(struct state *I)
{
	int i;
	for(i=0;i<I->prod_count;i++)
		printf("%s\n",I->prod[i]);
	cout<<endl;
}

bool in_array(char a[20],char b)
{
	int i;
	for(i=0;i<strlen(a);i++)
		if(a[i]==b)
			return true;
	return false;
}

void cleanup_prods(struct state * I)
{
	char a[100]={};
	int i;
	for(i=0;i<I->prod_count;i++)
		strcpy(I->prod[i],a);
	I->prod_count=0;
}

int return_index(char a)
{
	int i;
	for(i=0;i<no_t;i++)
		if(terminals[i]==a)
			return i;
	for(i=0;i<no_nt;i++)
		if(non_terminals[i]==a)
			return no_t+i;
}

char* symbols_after_space(char a[100])
{
	int i,j;
	for(i=0;i<strlen(a);i++)
		if(a[i]==' ')
			break;
	char look[20];
	for(j=i+1;j<strlen(a);j++)
		look[j-i-1]=a[j];
	return look;
}

char char_after_after_dot(char a[100])
{
	int i;
	for(i=0;i<strlen(a);i++)
		if(a[i]=='.')
			return a[i+2];
}
```

```
char *lookahead(char a[100])
{
	int i,j;
	char c=char_after_after_dot(a);
	if(c==' ')
	{
		return symbols_after_space(a);
	}
	else if(!is_non_terminal(c))
	{
		return &c;
	}
	else
	{
		int k=return_index(c)-no_nt;
		return first[k];
	}
}

void shift_table(int state_count,char shift_reduce_table[100][100][3])
{
	int i,j;
	for(i=0;i<state_count;i++)
	{
		int arr[no_nt+no_t]={-1};
		for(j=0;j<state_count;j++)
			if(goto_table[i][j]!='~')
				arr[return_index(goto_table[i][j])]= j;
		for(j=0;j<no_nt+no_t;j++)
		{
			if(i==1&&j==no_t-1)
			{
				shift_reduce_table[i][j][0]='A';
				shift_reduce_table[i][j][1]='C';
				shift_reduce_table[i][j][2]='C';
			}
			if(!(arr[j]==-1||arr[j]==0))
			{
				if(j<no_t)
					{
					shift_reduce_table[i][j][0]='S';
					shift_reduce_table[i][j][1]=arr[j]/10+'0';
					shift_reduce_table[i][j][2]=arr[j]%10+'0';
					}
				else
					{
						shift_reduce_table[i][j][1]=arr[j]/10+'0';
						shift_reduce_table[i][j][2]=arr[j]%10+'0';
					}
			}
		}
	}
}
```

```
void add_dot_at_end(struct state* I)
{
	int i;
	for(i=0;i<I->prod_count;i++)
		strcat(I->prod[i],".");
}

void add_to_first(int n,char b)
{
	int i;
	for(i=0;i<strlen(first[n]);i++)
		if(first[n][i]==b)
			return;
	first[n][strlen(first[n])]=b;
}

void add_to_first(int m,int n)
{
	int i,j;
	for(i=0;i<strlen(first[n]);i++)
	{
		int flag=0;
		for(j=0;j<strlen(first[m]);j++)
			if(first[n][i]==first[m][j])
				flag=1;
		if(flag==0)
			add_to_first(m,first[n][i]);
	}
}

void find_first(struct state *I)
{
	int i,j;
	for(i=0;i<no_nt;i++)
		for(j=0;j<I->prod_count;j++)
			if(I->prod[j][0]==non_terminals[i]&&(!is_non_terminal(I->prod[j][3])))
				add_to_first(i,I->prod[j][3]);
}

void reduce_table(int state_count,int *no_re,struct state *temp1,struct state I[50], char
shift_reduce_table[100][100][3])
{
	int i,j,k;
	int arr[temp1->prod_count][no_t]={-1};
	for(i=0;i<temp1->prod_count;i++)
	{
		int n=no_re[i];
		for(j=0;j<I[n].prod_count;j++)
		{
			for(k=0;k<strlen(temp1->prod[i])&&k<strlen(I[n].prod[j]);k++)
			{
				if(I[n].prod[j][k]!=temp1->prod[i][k])
				break;
			}
			if(k==strlen(temp1->prod[i])&&I[n].prod[j][k]==' ')
			{
			break;
```

```
                        }
                }
                char* after_dot=symbols_after_space(I[n].prod[j]);
                for(j=0;j<strlen(after_dot);j++)
                        for(k=0;k<no_t;k++)
                                if(after_dot[j]==terminals[k])
                                        arr[i][k]=i+1;
                for(j=0;j<no_t;j++)
                {
                        if(arr[i][j]!=-1&&arr[i][j]!=0&&arr[i][j]<state_count)
                        {
                                shift_reduce_table[n][j][0]='R';
                                shift_reduce_table[n][j][1]=arr[i][j]/10+'0';
                                shift_reduce_table[n][j][2]=arr[i][j]%10+'0';
                        }
                }
        }
}

void print_shift_reduce_table(char shift_reduce_table[100][100][3],int state_count)
{
        int i,j,k;
        cout<<"\t";
        for(i=0;i<no_t;i++)
                cout<<terminals[i]<<"\t";
        for(i=0;i<no_nt;i++)
                cout<<non_terminals[i]<<"\t";
        cout<<endl;
        for(i=0;i<state_count;i++)
        {
                cout<<"I"<<i<<"\t";
                for(j=0;j<no_t+no_nt;j++)
                {
                        for(k=0;k<3;k++)
                        cout<<shift_reduce_table[i][j][k];
                        cout<<"\t";
                }
                cout<<"\n";
        }
}

int main()
{
        struct state init,temp,temp1,I[50];
        int state_count=1,i,j,k,z,l;
        get_prods(&init);
        temp1=temp=init;
        add_dots(&init);

        for(i=0;i<100;i++)
                for(j=0;j<100;j++)
                        goto_table[i][j]='~';

        find_first(&temp);
        for(l=0;l<no_nt;l++)
                for(i=0;i<temp.prod_count;i++)
                        if(is_non_terminal(temp.prod[i][3]))
```

```
                    add_to_first(return_index(temp.prod[i][0])-no_t,return_index(temp.prod[i][3])-no_t);

augument(&I[0],&init);
closure(&init,&I[0]);
cout<<"\nI0:\n";
print_prods(&I[0]);

char characters[20]={};
for(i=0;i<state_count;i++)
{
        char characters[20]={};
        for(z=0;z<I[i].prod_count;z++)
                if(!in_array(characters,char_after_dot(I[i].prod[z])))
                {
                        if(char_after_dot(I[i].prod[z])!=' ')
                        characters[strlen(characters)]=char_after_dot(I[i].prod[z]);
                }

        for(j=0;j<strlen(characters);j++)
        {
                goto_state(&I[i],&I[state_count],characters[j]);
                closure(&init,&I[state_count]);
                int flag=0;
                for(k=0;k<state_count-1;k++)
                {
                        if(same_state(&I[k],&I[state_count]))
                        {
                                merge_state(&I[k],&I[state_count]);
                                cleanup_prods(&I[state_count]);
                                flag=1;
                                cout<<"I"<<i<<" --- "<<characters[j]<<" ---> I"<<k<<"\n";
                                goto_table[i][k]=characters[j];
                                break;
                        }
                }
                if(flag==0)
                {
                        state_count++;
                        cout<<"I"<<i<<" --- "<<characters[j]<<" ---> I";
                        cout<<state_count-1<<"\n\nI";
                        cout<<state_count-1<<":\n";
                        goto_table[i][state_count-1]=characters[j];
                        print_prods(&I[state_count-1]);
                }
        }
}

int no_re[temp.prod_count]={-1};
terminals[no_t]='$';
no_t++;
add_dot_at_end(&temp1);
for(i=0;i<state_count;i++)
{
        for(j=0;j<I[i].prod_count;j++)
                for(k=0;k<temp1.prod_count;k++)
                {
                        if(term_state(&I[i],temp1.prod[k]))
```

```
                                        {
                                                no_re[k]=i;
                                                break;
                                        }
                                }

        }
        char shift_reduce_table[100][100][3];
        shift_table(state_count,shift_reduce_table);
        reduce_table(state_count,&no_re[0],&temp1,I,shift_reduce_table);
        cout<<"\n\n";
        print_shift_reduce_table(shift_reduce_table,state_count);
        return 0;
}
```