# Ex:8 Leading And Trailing Computation

**RA1911030010138**
**Dipali Singh**
**25/03/2022**

**AIM:** Implementation of Shift Reduce Parser.

**ALGORITHM:**

1. For Leading, check for the first non-terminal.

2. If found, print it.

3. Look for the next production for the same non-terminal.

4. If not found, recursively call the procedure for the single non-terminal present before the comma or End Of Production String.

5. Include its results in the result of this non-terminal.

6. For trailing, we compute the same as leading but we start from the end of the production to the beginning.

7. Stop

**CODE:**

```
#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
using namespace std;

int vars,terms,i,j,k,m,rep,count,temp=-1;
char var[10],term[10],lead[10][10],trail[10][10];
struct grammar
{
        int prodno;
        char lhs,rhs[20][20];
}gram[50];
void get()
{
        cout<<"\nLEADING AND TRAILING\n";
        cout<<"\nEnter the no. of variables : ";
```

```
	cin>>vars;
	cout<<"\nEnter the variables : \n";
	for(i=0;i<vars;i++)
	{
		cin>>gram[i].lhs;
		var[i]=gram[i].lhs;
	}
	cout<<"\nEnter the no. of terminals : ";
	cin>>terms;
	cout<<"\nEnter the terminals : ";
	for(j=0;j<terms;j++)
		cin>>term[j];
	cout<<"\nPRODUCTION DETAILS\n";
	for(i=0;i<vars;i++)
	{
		cout<<"\nEnter the no. of production of "<<gram[i].lhs<<":";
		cin>>gram[i].prodno;
		for(j=0;j<gram[i].prodno;j++)
		{
			cout<<gram[i].lhs<<"->";
			cin>>gram[i].rhs[j];
		}
	}
}
void leading()
{
	for(i=0;i<vars;i++)
	{
		for(j=0;j<gram[i].prodno;j++)
		{
			for(k=0;k<terms;k++)
			{
				if(gram[i].rhs[j][0]==term[k])
					lead[i][k]=1;
				else
				{
					if(gram[i].rhs[j][1]==term[k])
						lead[i][k]=1;
				}
			}
		}
	}
	for(rep=0;rep<vars;rep++)
	{
```

```
		for(i=0;i<vars;i++)
		{
			for(j=0;j<gram[i].prodno;j++)
			{
				for(m=1;m<vars;m++)
				{
					if(gram[i].rhs[j][0]==var[m])
					{
						temp=m;
						goto out;
					}
				}
				out:
				for(k=0;k<terms;k++)
				{
					if(lead[temp][k]==1)
						lead[i][k]=1;
				}
			}
		}
	}
}
void trailing()
{
	for(i=0;i<vars;i++)
	{
		for(j=0;j<gram[i].prodno;j++)
		{
			count=0;
			while(gram[i].rhs[j][count]!='\x0')
				count++;
			for(k=0;k<terms;k++)
			{
				if(gram[i].rhs[j][count-1]==term[k])
					trail[i][k]=1;
				else
				{
					if(gram[i].rhs[j][count-2]==term[k])
						trail[i][k]=1;
				}
			}
		}
	}
	for(rep=0;rep<vars;rep++)
```

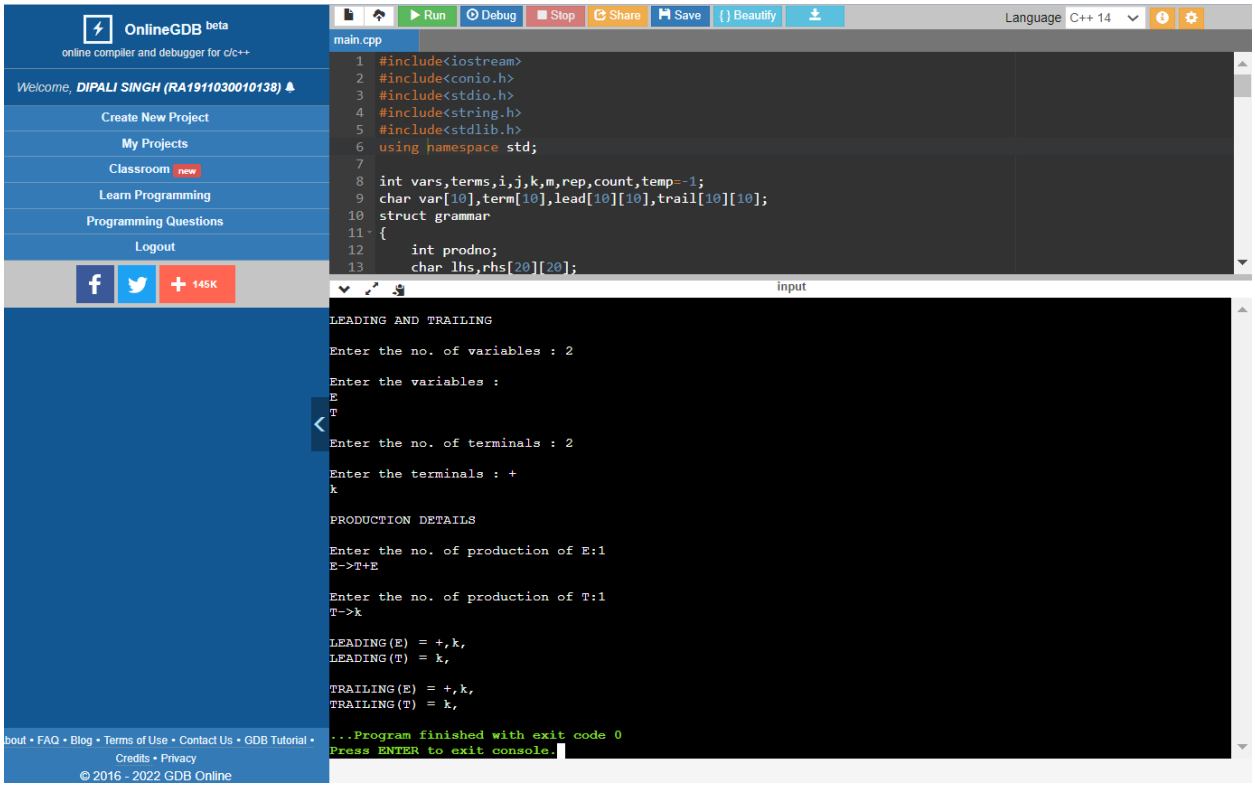
```
	{
		for(i=0;i<vars;i++)
		{
			for(j=0;j<gram[i].prodno;j++)
			{
				count=0;
				while(gram[i].rhs[j][count]!='\x0')
					count++;
				for(m=1;m<vars;m++)
				{
					if(gram[i].rhs[j][count-1]==var[m])
						temp=m;
				}
				for(k=0;k<terms;k++)
				{
					if(trail[temp][k]==1)
						trail[i][k]=1;
				}
			}
		}
	}
}
void display()
{
	for(i=0;i<vars;i++)
	{
		cout<<"\nLEADING("<<gram[i].lhs<<") = ";
		for(j=0;j<terms;j++)
		{
			if(lead[i][j]==1)
				cout<<term[j]<<",";
		}
	}
	cout<<endl;
	for(i=0;i<vars;i++)
	{
		cout<<"\nTRAILING("<<gram[i].lhs<<") = ";
		for(j=0;j<terms;j++)
		{
			if(trail[i][j]==1)
				cout<<term[j]<<",";
		}
	}
}
```

```
int main()
{

        get();
        leading();
        trailing();
        display();

}
```

**OUTPUT:**




**RESULT:**

Hence the trailing and leading were successfully computed.