# 

**COMPILER CONSTRUCTION LAB 9**

## Code:

#include<conio.h>

using namespace std;

#include <iostream>

#include<string.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

struct grammer {

char p[20];

char prod[20];

}g[10];

void main()

{

int i, stpos, j, k, l, m, o, p, f, r;

int np, tspos, cr;

cout << "\nEnter Number of productions:";

cin >> np;

char sc, ts[10];

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

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

{

cin >> ts;

strncpy\_s(g[i].p, ts, 1);

strcpy\_s(g[i].prod, &ts[3]);

}

char ip[10];

cout << "\nEnter Input:";

cin >> ip;

int lip = strlen(ip);

char stack[10];

stpos = 0;

i = 0;

//moving input

sc = ip[i];

stack[stpos] = sc;

i++; stpos++;

cout << "\n\nStack\tInput\tAction";

do

{

r = 1;

while (r != 0)

{

cout << "\n";

for (p = 0; p < stpos; p++)

{

cout << stack[p];

}

cout << "\t";

for (p = i; p < lip; p++)

{

cout << ip[p];

}

if (r == 2)

{

cout << "\tReduced";

}

else

{

cout << "\tShifted";

}

r = 0;

//try reducing

getchar();

for (k = 0; k < stpos; k++)

{

f = 0;

for (l = 0; l < 10; l++)

{

ts[l] = '\0';

}

tspos = 0;

for (l = k; l < stpos; l++) //removing first caharcter

{

ts[tspos] = stack[l];

tspos++;

}

//now compare each possibility with production

for (m = 0; m < np; m++)

{

cr = strcmp(ts, g[m].prod);

//if cr is zero then match is found

if (cr == 0)

{

for (l = k; l < 10; l++) //removing matched part from stack

{

stack[l] = '\0';

stpos--;

}

stpos = k;

//concatinate the string

strcat\_s(stack, g[m].p);

stpos++;

r = 2;

}

}

}

}

//moving input

sc = ip[i];

stack[stpos] = sc;

i++; stpos++;

} while (strlen(stack) != 1 && stpos != lip);

if (strlen(stack) == 1)

{

cout << "\n String Accepted";

}

getchar();

}

## Outputs:

# 1: Implement Bottom-Up (Shift-Reduce) Parsing for following grammar

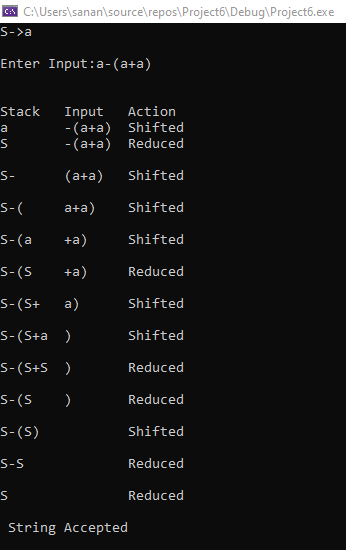
S → S+S

S → S-S

S → (S)

S → a

# Input : a1-(a2+a3)



**2: Implement Bottom-Up (Shift-Reduce) Parsing for following grammar**

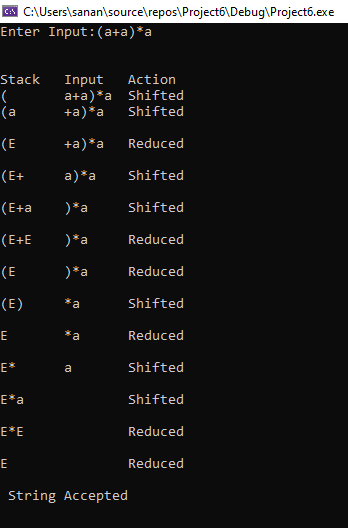
E -> E +E

E -> E\*E

E -> (E)

E -> a

Input is (a+a)\*a



**3: Implement Bottom-Up (Shift-Reduce) Parsing for following grammar**

S → ( L ) | a

L → L , S | S

Input : ( a , ( a , a ) )

