Name – Bhargav Shamuvel Gurav	**PRN** – 2041009
Class – L.Y. B-Tech (Computer)	**Batch** – B1
Course Code – CO406U	**Course Name -** CDL

Practical no. B (5)

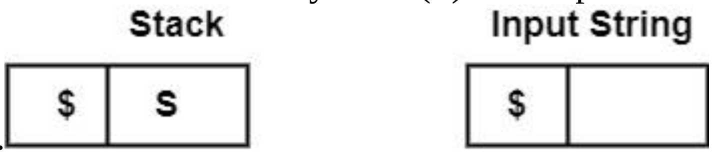
Aim: Implementation of shift reduce parsing algorithm.

Theory :
Shift Reduce Parser is a type of Bottom-Up Parser. It generates the Parse Tree from Leaves to the Root. In Shift Reduce Parser, the input string will be reduced to the starting symbol. This reduction can be produced by handling the rightmost derivation in reverse, i.e., from starting symbol to the input string.
Shift Reduce Parser requires two Data Structures

- Input Buffer
- Stack

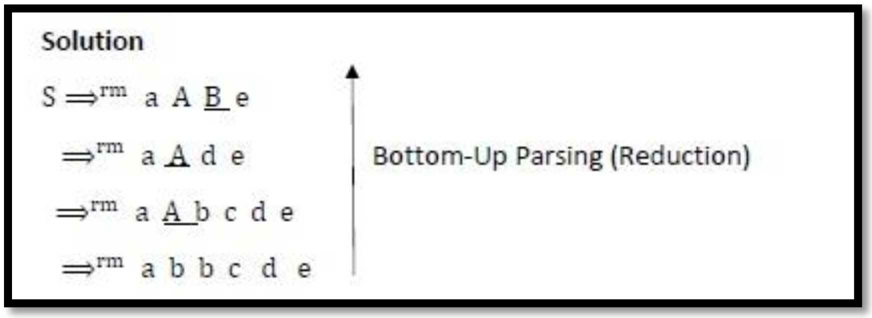
There are the various steps of Shift Reduce Parsing which are as follows −
There are the various steps of Shift Reduce Parsing which are as follows −

- It uses a stack and an input buffer.
- Insert $ at the bottom of the stack and the right end of the input string in Input Buffer.

Stack
$
Input String
w
$
- Shift − Parser shifts zero or more input symbols onto the stack until the handle is on top of the stack.
- Reduce − Parser reduce or replace the handle on top of the stack to the left side of production, i.e., R.H.S. of production is popped, and L.H.S is pushed.
- Accept − Step 3 and Step 4 will be repeated until it has detected an error or until the stack includes start symbol (S) and input Buffer is empty, i.e., it contains $.


Example1 − Perform the Bottom-Up Parsing for the given string on the Grammar, i.e., shows the reduction for string abbcde on the following Grammar
S → a A B e
A → A b c | b

B → d



Solution

$S \Rightarrow^{rm} a\ A\ \underline{B}\ e$

$\Rightarrow^{rm} a\ \underline{A}\ d\ e$

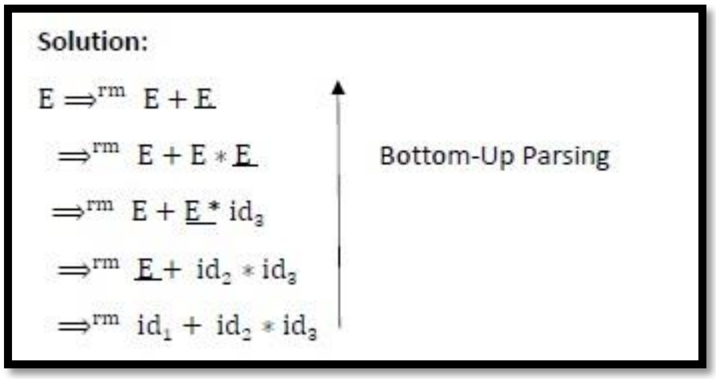
$\Rightarrow^{rm} a\ \underline{A\ b}\ c\ d\ e$

$\Rightarrow^{rm} a\ b\ b\ c\ d\ e$

Bottom-Up Parsing (Reduction)

It can reduce the string abbcde to the starting symbol S by applying the rightmost derivation in reverse at each step.
Handle − Each replacement of the Right side of production by the left side in the process above is known as "Reduction" and each replacement is called "Handle."

Example2 − Consider the Grammar
E → E + E
E → E * E
E → (E)
E → id
Perform Rightmost Derivation string id1 + id2 * id3. Find Handles at each step.



Solution:

$E \Rightarrow^{rm} E + \underline{E}$

$\Rightarrow^{rm} E + E * \underline{E}$

$\Rightarrow^{rm} E + \underline{E *} id_3$

$\Rightarrow^{rm} \underline{E} + id_2 * id_3$

$\Rightarrow^{rm} id_1 + id_2 * id_3$

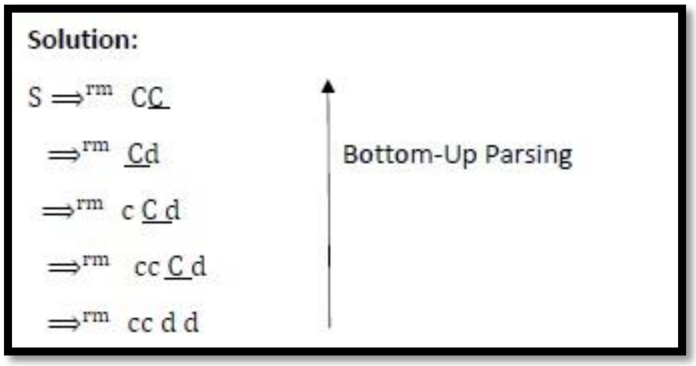
Bottom-Up Parsing

Handles at each step

Right Sentenial Form	HANDLE	Production Used
id1 + id2 * id3	id1	E → id1
E + id2 * id3	id2	E → id2
E + E * id3	id3	E → id3
E + E * E	E * E	E → E ∗ E

Right Sentienal Form	HANDLE	Production Used
E + E	E + E	E → E + E
E		

Example3 − Consider the following Grammar
S → CC
C → cC
C → d
Check whether input string "ccdd" is accepted or not accepted using Shift-Reduce parsing.


Solution:

$S \overset{rm}{\Rightarrow} C\underline{C}$

$\overset{rm}{\Rightarrow} \underline{Cd}$

$\overset{rm}{\Rightarrow} c\,\underline{C\,d}$

$\overset{rm}{\Rightarrow} cc\,\underline{C}\,d$

$\overset{rm}{\Rightarrow} cc\,d\,d$

Bottom-Up Parsing

Stack	Input String	Action
$	ccdd$	Shift
$ c	cdd$	Shift
$ cc	dd$	Shift
$ ccd	d$	Reduce by C → id
$ ccC	d$	Reduce by C → cC
$cC	d$	Reduce by C → cC
$C	d$	Shift
$Cd	$	Reduce by C → d
$CC	$	Reduce by S → CC

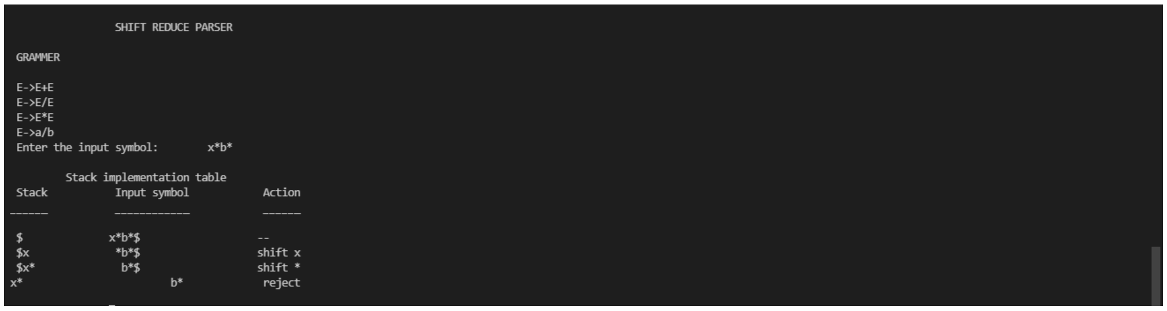
Stack	Input String	Action
$S	$	Accept

∴ At last, Stack contains Starting Symbol S, and input Buffer is empty. It will accept the string.

Program Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char ip_sym[15],stack[15];
int ip_ptr=0,st_ptr=0,len,i;
char temp[2],temp2[2];
char act[15];
void check();
void main()
{
 printf("\n\t\t SHIFT REDUCE PARSER\n");
 printf("\n GRAMMER\n");
 printf("\n E->E+E\n E->E/E");
 printf("\n E->E*E\n E->a/b");
 printf("\n Enter the input symbol:\t");
 gets(ip_sym);
 printf("\n\t Stack implementation table");
 printf("\n Stack\t\t Input symbol\t\t Action");
 printf("\n______\t\t ____________\t\t ______\n");
 printf("\n $\t\t%s$\t\t\t--",ip_sym);
 strcpy(act,"shift ");
 temp[0]=ip_sym[ip_ptr];
 temp[1]='\0';
 strcat(act,temp);
 len=strlen(ip_sym);
 for(i=0;i<=len-1;i++) //
 {
 stack[st_ptr]=ip_sym[ip_ptr];
 stack[st_ptr+1]='\0';
 ip_sym[ip_ptr]=' ';
 ip_ptr++;
 printf("\n $%s\t\t%s$\t\t\t%s",stack,ip_sym,act);
 strcpy(act,"shift ");
 temp[0]=ip_sym[ip_ptr];
 temp[1]='\0';
 strcat(act,temp);
 check();
 st_ptr++;
 }
 st_ptr++;
 check();
}
void check()
{
 int flag=0;
 temp2[0]=stack[st_ptr];
 temp2[1]='\0';
 if((!strcmp(temp2,"a"))||(!strcmp(temp2,"b")))
 {
 stack[st_ptr]='E';
 if(!strcmp(temp2,"a"))
 printf("\n $%s\t\t%s$\t\t\tE->a",stack, ip_sym);
 else
 printf("\n $%s\t\t%s$\t\t\tE->b\n",stack,ip_sym);
 flag=1;
 }
 if((!strcmp(temp2,"+"))||(strcmp(temp2,"*"))||(!strcmp(temp2,"/")
))
 {
 flag=1;
 }
 if((!strcmp(stack,"E+E"))||(!strcmp(stack,"E\E"))||(!strcmp(stack
,"E*E")))
 {
 strcpy(stack,"E");
 st_ptr=0;
 if(!strcmp(stack,"E+E"))
 printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);
 else if(!strcmp(stack,"E\E"))
 printf("\n $%s\t\t %s$\t\t\tE->E\E",stack,ip_sym);
 else
 printf("\n $%s\t\t%s$\t\t\tE->E*E",stack,ip_sym);
 flag=1;
 }
 if(!strcmp(stack,"E")&&ip_ptr==len)
 {
 printf("\n $%s\t\t%s$\t\t\tACCEPT\n",stack,ip_sym);
 exit(0);
 }
 if(flag==0)
 {
 printf("\n%s\t\t\t%s\t\t reject\n",stack,ip_sym);
 exit(0);
 }
 return;
}
```

Output:



```
                SHIFT REDUCE PARSER

 GRAMMER

 E->E+E
 E->E/E
 E->E*E
 E->a/b
 Enter the input symbol:       x*b*

        Stack implementation table
 Stack          Input symbol           Action
------          ------------           ------

 $             x*b*$                  --
 $x             *b*$                  shift x
 $x*             b*$                  shift *
x*                      b*             reject
```

Conclusion : In this practical we implemented shift reduce parser algorithm.