CD LAB WEEK 8

AP19110010369

1. Implementation of Shift Reduce parser using C for the following grammar and illustrate the
parser’s actions for a valid and an invalid string.

E→E+E
E→E*E
E→(E)
E→d

CODE:

```
#include<stdio.h>
#include<stdlib.h>
void pop(),push(char),display();
char stack[100]="\0", input[100], *ip;
int top=-1;
void push(char c)
{
top++;
stack[top]=c;
}
void pop()
{
stack[top]='\0';
top--;
}
void display()

{
printf("\n%s\t%s\t",stack,ip);
}
```

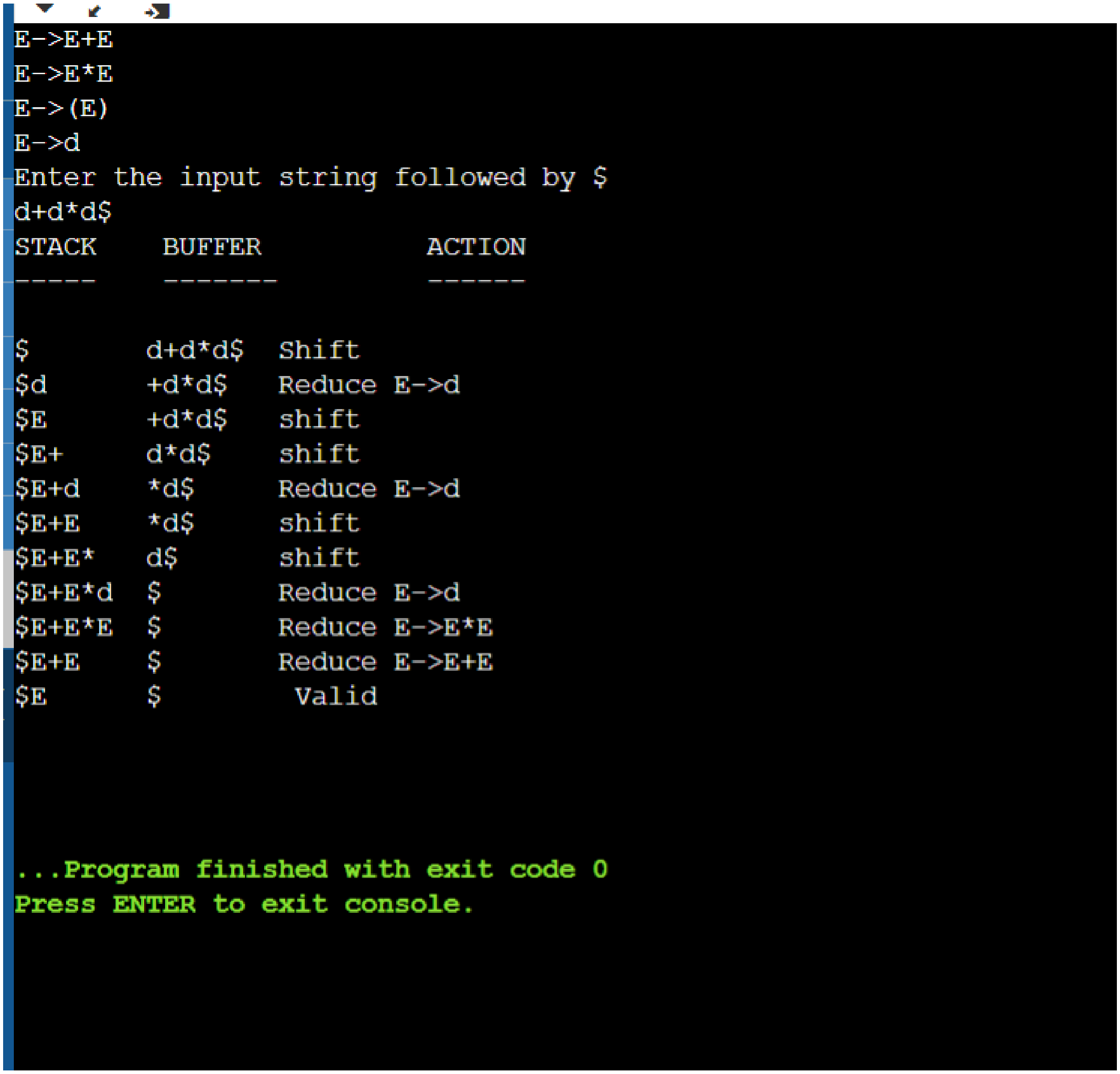
```
void main()
{
printf("E->E+E\n");
printf("E->E*E\n");
printf("E->(E)\n");
printf("E->d\n");
printf("Enter the input string followed by $ \n");
scanf("%s",input);
ip=input;
push('$');
printf("STACK\t BUFFER \t ACTION\n");
printf("-----\t ------- \t ------\n");
display();
if(stack[top]=='$' && *ip=='$'){
printf("Null Input");
exit(0);
}
do
{
if((stack[top]=='E' && stack[top-1]=='$') && (*(ip)=='$'))
{
display();
printf(" Valid\n\n\n");
break;
}
if(stack[top]=='$')
{
push(*ip);
ip++;
printf("Shift");
}
else if(stack[top]=='d')
{
display();
pop();
push('E');
printf("Reduce E->d");
```

```
}
else if(stack[top]=='E' && stack[top-1]=='+' && stack[top-2]=='E'&& *ip!='*')
{
display();
pop();
pop();
pop();
push('E');
printf("Reduce E->E+E");
}
else if(stack[top]=='E' && stack[top-1]=='*' && stack[top-2]=='E')
{
display();
pop();
pop();
pop();
push('E');
printf("Reduce E->E*E");
}
else if(stack[top]==')' && stack[top-1]=='E' && stack[top-2]=='(')
{
display();

pop();
pop();
pop();
push('E');
printf("Reduce E->(E)");
}
else if(*ip=='$')
{ printf(" Invalid\n\n\n");
break;
}
else
{
display();
push(*ip);
```

```
ip++;
printf("shift");
}
}while(1);
}
```

OUTPUT:

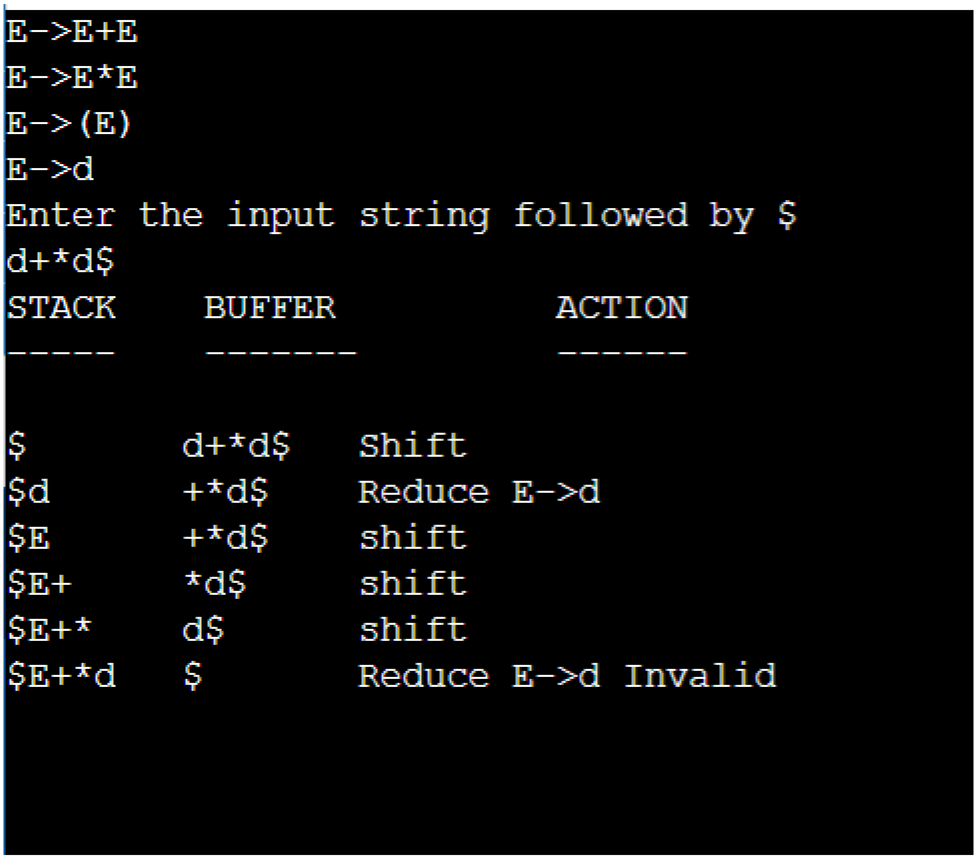


```
E->E+E
E->E*E
E->(E)
E->d
Enter the input string followed by $
d+d*d$
STACK    BUFFER          ACTION
-----    -------         ------

$       d+d*d$  Shift
$d      +d*d$   Reduce  E->d
$E      +d*d$   shift
$E+     d*d$    shift
$E+d    *d$     Reduce  E->d
$E+E    *d$     shift
$E+E*   d$      shift
$E+E*d  $       Reduce  E->d
$E+E*E  $       Reduce  E->E*E
$E+E    $       Reduce  E->E+E
$E      $        Valid




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



```
E->E+E
E->E*E
E->(E)
E->d
Enter the input string followed by $
d+*d$
STACK    BUFFER          ACTION
-----    -------         ------

$       d+*d$   Shift
$d      +*d$    Reduce E->d
$E      +*d$    shift
$E+     *d$     shift
$E+*    d$      shift
$E+*d   $       Reduce E->d Invalid
```

2. Implementation of Shift Reduce parser using C for the following grammar and illustrate the
parser’s actions for a valid and an invalid string.

S –> 0S0 | 1S1 | 2

CODE:

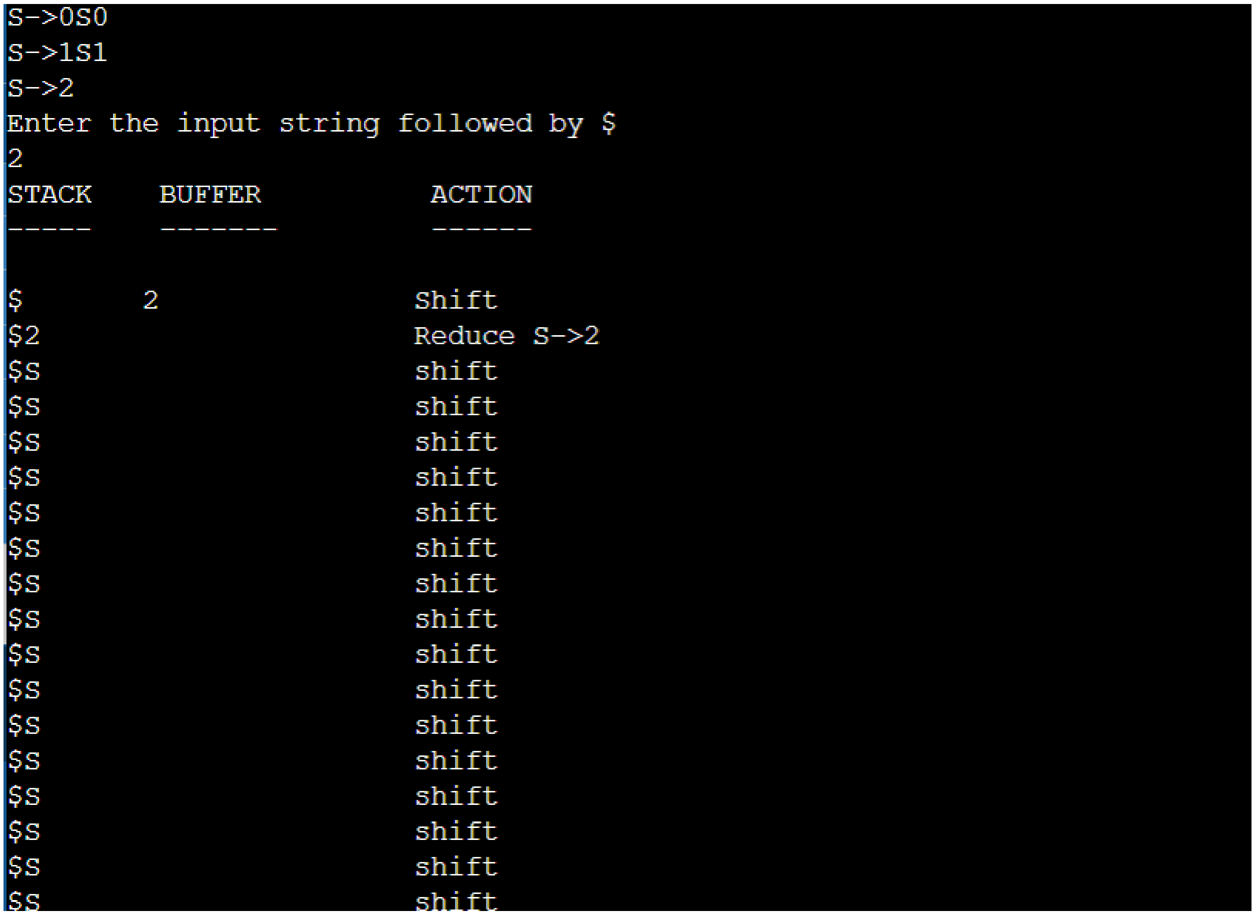
```
#include<stdio.h>
#include<stdlib.h>
void pop(),push(char),display();
```

```
char stack[100]="\0", input[100], *ip;
int top=-1;
void push(char c)
{
top++;
stack[top]=c;
}
void pop()
{
stack[top]='\0';
top--;
}
void display()

{
printf("\n%s\t%s\t",stack,ip);
}
void main()
{
printf("S->0S0\n");
printf("S->1S1\n");
printf("S->2\n");
printf("Enter the input string followed by $ \n");
scanf("%s",input);
ip=input;
push('$');
printf("STACK\t BUFFER \t ACTION\n");
printf("-----\t ------- \t ------\n");
display();
if(stack[top]=='$' && *ip=='$'){
printf("Null Input");
exit(0);
}
do
{
```

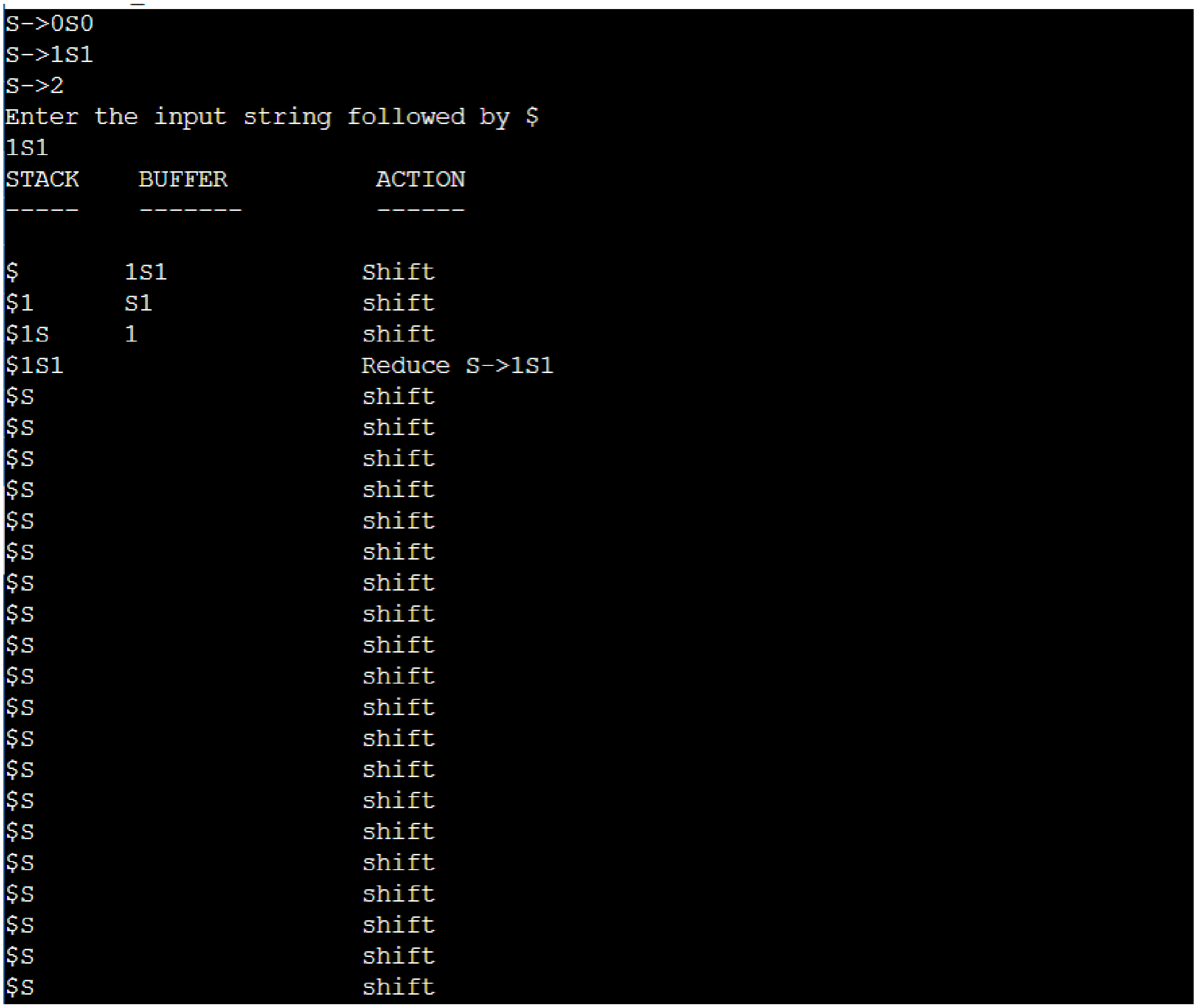
```
if((stack[top]=='S' && stack[top-1]=='$') && (*(ip)=='$'))
{
display();
printf("\t Valid\n\n\n");
break;
}
if(stack[top]=='$')
{
push(*ip);
ip++;
printf("\tShift");
}
else if(stack[top]=='2')
{
display();
pop();
push('S');
printf("\tReduce S->2");
}
else if(stack[top]=='0' && stack[top-1]=='S' && stack[top-2]=='0')
{
display();
pop();
pop();
pop();
push('S');
printf("\tReduce S->0S0");
}
else if(stack[top]=='1' && stack[top-1]=='S' && stack[top-2]=='1')
{
display();
pop();
pop();
pop();
push('S');
printf("\tReduce S->1S1");
}
else if(*ip=='$')
{ printf("\tInvalid\n\n\n");
break;
}
else
{
display();
push(*ip);
ip++;
printf("\tshift");
}
}while(1);
}
```

OUTPUT:



```
S->0S0
S->1S1
S->2
Enter the input string followed by $
2
STACK    BUFFER          ACTION
-----    -------         ------

$       2               Shift
$2                      Reduce S->2
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
```



```
S->0S0
S->1S1
S->2
Enter the input string followed by $
1S1
STACK    BUFFER          ACTION
-----    -------         ------

$       1S1             Shift
$1      S1              shift
$1S     1               shift
$1S1                    Reduce S->1S1
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
$S                      shift
```