Lab :-

Assignment - 6

- Objective :- Our objective is to design a Shift Reduce Parser using C programming language.

- Resources :- We took help from Geeks-for-Geeks, educatech.in and Javapoint.

- Procedure :-
  (i) Get the input expression and store it in the input buffer.
  (ii) Read the data from the input buffer one at the time.
  (iii) Using stack and push & pop operation shift and reduce symbols w.r.t production rules available.
  (iv) Continue the process till symbol shift and production rule reduce reaches the start symbol.
  (v) Display the stack implementation table with corresponding stack action with input symbols.

- 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 GRAMMAR\n");
    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 ____");
    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((!strcmpi(temp2,"a")) || (!strcmpi(temp2,"b"))){
        stack[st_ptr] = 'E';
        if(!strcmpi(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", stack, ip_sym);
        flag=1;
    }
    if((!strcmpi(stack,"E+E")) || (!strcmpi(stack,"E\E")) ||
    (!strcmpi(stack,"E*E"))){
        strcpy(stack,"E");
        st_ptr=0;
        if(!strcmpi(stack,"E+E"))
            printf("\n$%s\t\t%s$\t\t\tE->E+E", stack, ip_sym);
        else if(!strcmpi(stack,"E\E"))
            printf("\n$%s\t\t%s$\t\t\tE->E\E", stack, ip_sym);
        else if(!strcmpi(stack,"E*E"))
            printf("\n$%s\t\t%s$\t\t\tE->E*E", stack, ip_sym);
        flag=1;
    }
    if(!strcmpi(stack,"E") && ip_ptr==len){
        printf("\n$%s\t\t%s$\t\t\tACCEPT", stack, ip_sym);
        getch();
        exit(0);
    }
```

```
if (flag==0){
    printf("\n%s\t\t\t%s\t\t\t REJECT", stack, ip_sym);
    exit(0);
}
return;
}
```

• Output:

SHIFT REDUCE PARSER

GRAMMAR

E→ E+E
E→ E/E
E→ E*E
E→ a/b

Enter the Input symbol: a+b.

Stack Implementation Table:

	Stack	Input Symbol	Action
--	$	a+b$	
	$a	+b$	
Shift a	$E	+b$	E-
>a	$E+	b$	
Shift +	$E+b	$	
Shift b.	$E+E	$	E-
>b	$E	$	E-
>E+E	$E	$	

ACCEPT

- Discussion of Shift Reduce Parser attempts for the construction of parse in a similar manner as done in bottom-up parsing i.e. the parse tree is constructed from leaves (bottom) to the root (up). A more general form of the shift-reduce parser is the LR Parser.

26/4/22