***Week-12***

***4.Design, develop and implement YACC/C program to demonstrate Shift ReduceParsing technique for the grammar rules: E →E+T \| T, T →T*F \| F, F →(E) \| id andparse the sentence: id + id * id.***

*Lex Part:*

gedit lab4.l

```
%{
#include "y.tab.h"
%}
%%
"id"    {return id;}
"+"     {return plus;}
"*"     {return star;}
"("     {return opar;}
")"     {return cpar;}
.       return yytext[0];
\n      return 0;
%%
```

*Yacc Part :*

*gedit lab4.y*

```
%{
        #include<stdio.h>
        #include<string.h>
        extern FILE *yyin;
        char inp[30],stack[30];
         int inpCount,sCount;
%}
%token id
%token plus
%token star
%token opar
```

```
%token cpar


%%
E : E P T   {
                printf("$%s\t%s$\tREDUCE E -> E+T\n",stack,inp);
                sCount -= 3;
                stack[sCount++] = 'E';
                stack[sCount] = '\0';
            }

      | T   {
                printf("$%s\t%s$\tREDUCE E -> T\n",stack,inp);
                sCount -= 1;
                stack[sCount++] = 'E';
                stack[sCount] = '\0';
            }

T : T S F   {
                printf("$%s\t%s$\tREDUCE T -> T*F \n",stack,inp);
                sCount -= 3;
                stack[sCount++] = 'T';
                stack[sCount] = '\0';
            }

      | F   {
                printf("$%s\t%s$\tREDUCE T -> F \n",stack,inp);
                sCount -= 1;
                stack[sCount++] = 'T';
                stack[sCount] = '\0';
            }

F : O E C   {
                printf("$%s\t%s$\tREDUCE F -> (E)  \n",stack,inp);
                sCount -= 3;
                stack[sCount++] = 'F';
                stack[sCount] = '\0';
```

```
        }


F : id      {
                printf("$%s\t%s$\tSHIFT id\n",stack,inp);
                inp[inpCount++] = ' ';
                inp[inpCount++] = ' ';
                stack[sCount++] = 'i';
                stack[sCount++] = 'd';
                stack[sCount] = '\0';

                printf("$%s\t%s$\tREDUCE F-> id\n",stack,inp);
                sCount -= 2;
                stack[sCount++] = 'F';
                stack[sCount] = '\0';
        }

O : opar    {
                printf("$%s\t%s$\tSHIFT (\n",stack,inp);
                inp[inpCount++] = ' ';
                stack[sCount++] = '(';
                stack[sCount] = '\0';
        }

C : cpar    {
                printf("$%s\t%s$\tSHIFT )\n",stack,inp);
                inp[inpCount++] = ' ';
                stack[sCount++] = ')';
                stack[sCount] = '\0';
        }

P : plus    {
                printf("$%s\t%s$\tSHIFT +\n",stack,inp);
                inp[inpCount++] = ' ';
                stack[sCount++] = '+'
                stack[sCount] = '\0';
```

```
            }


S : star    {
                printf("$%s\t%s$\tSHIFT *\n",stack,inp);
                inp[inpCount++] = ' ';
                stack[sCount++] = '*';
                stack[sCount] = '\0';
            }
;
%%

void main(){
    printf("Enter the input : \n");
    scanf("%s",inp);
    FILE *fp = fopen("temp.txt","w");
    fprintf(fp,"%s",inp);
    fclose(fp);
    yyin = fopen("temp.txt","r");
    printf("Stack\tInput\tAction\n");
    yyparse();

    if(sCount == 1 && stack[sCount-1] == 'E'
&&inp[inpCount]=='\0')
    {
        printf("$%s\t%s$\tSuccess\n",stack,inp);
    }
}


int yyerror(){
    printf("$%s\t%s$\tError\n",stack,inp);
}
```

*Output:*

```
    $yacc lab4.y -ld
    $ lex lab4.l
    $ cc y.tab.c lex.yy.c -ll
```

```
bharath23@bharathgowda:~$ ./a.out
Enter the input :
id+id*id
Stack   Input   Action
$       id+id*id$       SHIFT id
$id       +id*id$       REDUCE F-> id
$F        +id*id$       REDUCE T -> F
$T        +id*id$       REDUCE E -> T
$E        +id*id$       SHIFT +
$E+        id*id$       SHIFT id
$E+id        *id$       REDUCE F-> id
$E+F         *id$       REDUCE T -> F
$E+T         *id$       SHIFT *
$E+T*         id$       SHIFT id
$E+T*id         $       REDUCE F-> id
$E+T*F          $       REDUCE T -> T * F
$E+T            $       REDUCE E -> E + T
$E              $       Success
bharath23@bharathgowda:~$ ./a.out
Enter the input :
id++id
Stack   Input   Action
$       id++id$ SHIFT id
$id       ++id$ REDUCE F-> id
$F        ++id$ REDUCE T -> F
$T        ++id$ REDUCE E -> T
$E        ++id$ SHIFT +
$E+        +id$ Error
```

```
bharath23@bharathgowda:~$ ./a.out
Enter the input :
(id+id)*id
Stack   Input   Action
$       (id+id)*id$     SHIFT (
$(       id+id)*id$     SHIFT id
$(id       +id)*id$     REDUCE F-> id
$(F        +id)*id$     REDUCE T -> F
$(T        +id)*id$     REDUCE E -> T
$(E        +id)*id$     SHIFT +
$(E+        id)*id$     SHIFT id
$(E+id        )*id$     REDUCE F-> id
$(E+F         )*id$     REDUCE T -> F
$(E+T         )*id$     REDUCE E -> E + T
$(E           )*id$     SHIFT )
$(E)           *id$     REDUCE F -> (E)
$F             *id$     REDUCE T -> F
$T             *id$     SHIFT *
$T*             id$     SHIFT id
$T*id             $     REDUCE F-> id
$T*F              $     REDUCE T -> T * F
$T                $     REDUCE E -> T
$E                $     Success
bharath23@bharathgowda:~$
```