Laboratory-4

Question

Implement Intermediate code generation of assignment statements and expressions using Lex / Yacc.

Lex File (lex.l):

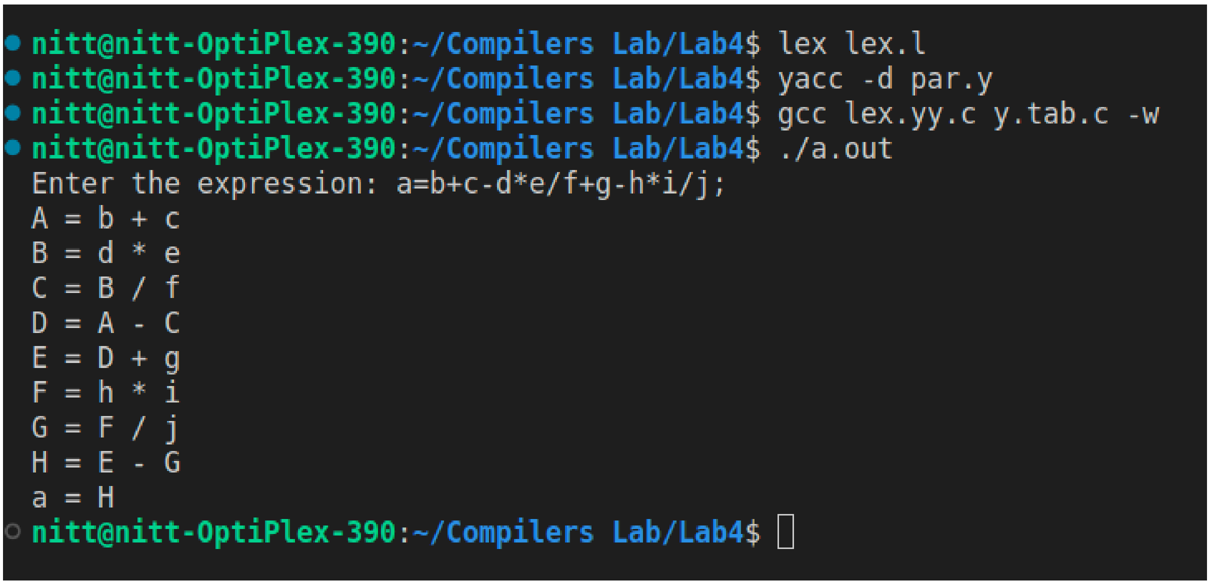
```
%{
#include"y.tab.h"
extern char yyval;
%}

%%

[0-9]+ {yylval.symbol=(char)(yytext[0]);return NUMBER;}
[a-z] {yylval.symbol= (char)(yytext[0]);return LETTER;}
. {return yytext[0];}
\n {return 0;}

%%
```

Yacc File (par.y):

```
%{
#include"y.tab.h"
#include<stdio.h>
char addtotable(char,char,char);


int index1=0;
char temp = 'A'-1;

struct expr{

char operand1;
char operand2;
char operator;
char result;
};

%}

%union{
char symbol;
}

%left '+' '-'
%left '/' '*'

%token <symbol> LETTER NUMBER
%type <symbol> exp
%%
```

```
statement: LETTER '=' exp ';' {addtotable((char)$1,(char)$3,'=');};
exp: exp '+' exp {$$ = addtotable((char)$1,(char)$3,'+');}
    |exp '-' exp {$$ = addtotable((char)$1,(char)$3,'-');}
    |exp '/' exp {$$ = addtotable((char)$1,(char)$3,'/');}
    |exp '*' exp {$$ = addtotable((char)$1,(char)$3,'*');}
    |'(' exp ')' {$$= (char)$2;}
    |NUMBER {$$ = (char)$1;}
    |LETTER {(char)$1;};

%%

struct expr arr[20];

void yyerror(char *s){
    printf("Errror %s",s);
}

char addtotable(char a, char b, char o){
    temp++;
    arr[index1].operand1 =a;
    arr[index1].operand2 = b;
    arr[index1].operator = o;
    arr[index1].result=temp;
    index1++;
    return temp;
}

void threeAdd(){

    int i=0;
    char temp='A';
    while(i<index1){
        if (arr[i].operator != '=')
            printf("%c = ",arr[i].result);
        printf("%c ",arr[i].operand1);
        printf("%c ",arr[i].operator);
        printf("%c ",arr[i].operand2);
        i++;
        temp++;
        printf("\n");
    }
}


int yywrap(){
    return 1;
}

int main(){
    printf("Enter the expression: ");
    yyparse();
    threeAdd();
    return 0;
}
```

Output:



```
● nitt@nitt-OptiPlex-390:~/Compilers Lab/Lab4$ lex lex.l
● nitt@nitt-OptiPlex-390:~/Compilers Lab/Lab4$ yacc -d par.y
● nitt@nitt-OptiPlex-390:~/Compilers Lab/Lab4$ gcc lex.yy.c y.tab.c -w
● nitt@nitt-OptiPlex-390:~/Compilers Lab/Lab4$ ./a.out
  Enter the expression: a=b+c-d*e/f+g-h*i/j;
  A = b + c
  B = d * e
  C = B / f
  D = A - C
  E = D + g
  F = h * i
  G = F / j
  H = E - G
  a = H
○ nitt@nitt-OptiPlex-390:~/Compilers Lab/Lab4$ 
```

Result:

Code for syntax analysis of conditional constructs was implemented successfully.