Name:Kushagra Agarwal
SRN:PES2UG22CS275

lexer.l

```
%{
    #define YYSTYPE char*
    #include <unistd.h>
    #include "parser.tab.h"
    #include <stdio.h>
     #include <string.h>
    extern void yyerror(const char *); // declare the error handling
function
%}

/* Regular definitions */
digit   [0-9]
letter  [a-zA-Z]
id  {letter}({letter}|{digit})*
digits  {digit}+
opFraction  (\.{digits})?
opExponent  ([Ee][+-]?{digits})?
number  {digits}{opFraction}{opExponent}
%option yylineno

%%
\/\/(.*) ; // ignore comments
[\t\n] ; // ignore whitespaces
"("     {return *yytext;}
")"     {return *yytext;}
"."         {return *yytext;}
","         {return *yytext;}
"*"         {return *yytext;}
"+"         {return *yytext;}
";"         {return *yytext;}
"-"         {return *yytext;}
"/"         {return *yytext;}
"="         {return *yytext;}
">"         {return *yytext;}
"<"         {return *yytext;}
{number}{
            yylval = strdup(yytext);  //stores the value of the number
to be used later for symbol table insertion
            return T_NUM;
        }
{id}        {
                    yylval = strdup(yytext); //stores the identifier to
be used later for symbol table insertion
                    return T_ID;
                }
.       {} // anything else => ignore
%%
```

Parser.y

```
%{
    #include "abstract_syntax_tree.c"
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    void yyerror(char* s);        // error handling function
    int yylex();              // declare the function performing lexical
analysis
    extern int yylineno;      // track the line number
%}

%union  // union to allow nodes to store return different datatypes
{
    char* text;
    expression_node* exp_node;
}

%token <text> T_ID T_NUM

%type <exp_node> E T F ASSGN START

/* specify start symbol */
%start START


%%
START : ASSGN { display_exp_tree($1); printf("Valid syntax\n");
YYACCEPT; }
;




/* Grammar for assignment */
ASSGN : T_ID '=' E   {
    //displaying the expression tree
    $$ = init_exp_node(strdup(""), init_exp_node($1, NULL, NULL), $3);
};

/* Expression Grammar */
E : E '+' T  {
            //create a new node of the AST and set left and right
children appropriately
            $$=init_exp_node(strdup("+"), $1, $3);
}
  | E '-' T {
            $$=init_exp_node(strdup("-"), $1, $3);
            //create a new node of the AST and set left and right
children appropriately
  }
  | T {$$ = $1;}
  ;
```

```
T : T '*' F {
            $$=init_exp_node(strdup("*"), $1, $3);
            //create a new node of the AST and set left and right
children appropriately
}
  | T '/' F {
            $$=init_exp_node(strdup("/"), $1, $3);
            //create a new node of the AST and set left and right
children appropriately
  }
  | F {$$ = $1;}
        //pass ast node to the parent
  ;

F : '(' E ')' {
    $$ = $2;
}
  | T_ID {
    // Create a leaf node for identifier
    $$ = init_exp_node($1, NULL, NULL);
}
  | T_NUM {
    // Create a leaf node for number
    $$ = init_exp_node($1, NULL, NULL);
};


%%


/* error handling function */
void yyerror(char* s)
{
    printf("Error :%s at %d \n",s,yylineno);
}

int yywrap() {
    return 1;
}

/* main function - calls the yyparse() function which will in turn drive
yylex() as well */
int main(int argc, char* argv[])
{
    yyparse();
    return 0;
}%
```

abstract_sytax_tree.h

```
typedef struct expression_node
{
	/*needs 3 members
		i) pointer to the left child
		ii) pointer to the right child
		iii) string to store the value of the node
	*/
	struct expression_node* left;
	char* val;
	struct expression_node* right;
}expression_node;

expression_node* init_exp_node(char* val, expression_node* left,
expression_node* right);
void display_exp_tree(expression_node* exp_node);%
```

abstract_sytax_tree.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "abstract_syntax_tree.h"

expression_node* init_exp_node(char* val, expression_node* left,
expression_node* right)
{
	expression_node*
node=(expression_node*)malloc(sizeof(expression_node));
	node->left=left;
	node->val = val;
	node->right=right;
	return node;
	// function to allocate memory for an AST node and set the left
and right children of the nodes
}

void display_exp_tree(expression_node* exp_node)
{
	// traversing the AST in preorder and displaying the nodes
```

```
    if(exp_node==NULL)
        return;

    printf("%s\n", exp_node->val);
    display_exp_tree(exp_node->left);
    display_exp_tree(exp_node->right);
}%
```