Details :

- Name : P K Navin Shrinivas
- Section : D
- SRN : PES2UG20CS237

Lexer :

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

/* 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
            return T_NUM;
        }
{id}            {
                    yylval = strdup(yytext); //stores th
                    return T_ID;
                }
.       {} // anything else => ignore
%%
```

Yacc :

```
%{
    #include "abstract_syntax_tree.c"
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    void yyerror(char* s);
    int yylex();
    extern int yylineno;
%}

%union
{
    char* text;
    expression_node* exp_node;

}
```

```
%token <text> T_ID T_NUM

%type <exp_node> E T F

/* specify start symbol */
%start START


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

/* Grammar for assignment */
ASSGN : T_ID '=' E  {
                        // displaying the expression tre
                    display_exp_tree($3);
                    }
    ;

/* Expression Grammar */
E : E '+' T     {
                        // create a new node of the AST and
                expression_node* node = init_exp_node("+",
                $$ = node;
                    }
    | E '-' T   {
                        // create a new node of the AST and
                expression_node* node = init_exp_node("-",
                $$ = node;
                    }
    | T     { $$ = $1; }
    ;
```

```
T : T '*' F     {
                    // create a new node of the AST and
                expression_node* node = init_exp_node("*",
                $$ = node;
                  }
    | T '/' F   {
                    // create a new node of the AST and
                expression_node* node = init_exp_node("/",
                $$ = node;
                  }
    | F {
            //pass AST node to the parent
         $$ =  $1;
        }
    ;

F : '(' E ')' { $$ = $2; }
    | T_ID  {
                // creating a terminal node of the AST
              expression_node* node = init_exp_node($1,N
              $$ = node;
            }
    | T_NUM     {
                    // creating a terminal node of the A
              expression_node* node = init_exp_node($1,N
              $$ = node;
                }
    ;

%%


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

```
}


/* main function - calls the yyparse() function which wi
int main(int argc, char* argv[])
{
    yyparse();
    return 0;
}
```

C file :

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

expression_node* init_exp_node(char* val, expression_nod
{
    // function to allocate memory for an AST node and s
   expression_node* head = (expression_node*)malloc(size
   head -> value = val;
   head -> left = left;
   head -> right = right;
   return head;
}

void display_exp_tree(expression_node* exp_node)
{
    // traversing the AST in preorder and displaying the
   if(exp_node == NULL){
      return;
   }
   printf("%s \n", exp_node->value);
   display_exp_tree(exp_node->left);
   display_exp_tree(exp_node->right);

   return;
```

```
}
```

H file :

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

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

Outputs :

```
➜ Student Codebase git:(main) ✗nvim lexer.l
➜ Student Codebase git:(main) ✗lex lexer.l
➜ Student Codebase git:(main) ✗yacc parser.y -d
➜ Student Codebase git:(main) ✗gcc  y.tab.c lex.yy.c -lfl
➜ Student Codebase git:(main) ✗./a.out < test_input_1.c
+
-
/
10
5
*
2
7
3
Valid syntax
➜ Student Codebase git:(main) ✗./a.out < test_input_2.c
-
+
/
c
6.7
12.45
*
a
1234.0
Valid syntax
➜ Student Codebase git:(main) ✗
[CD] 0:zshM 1:zsh- 2:zsh*
```