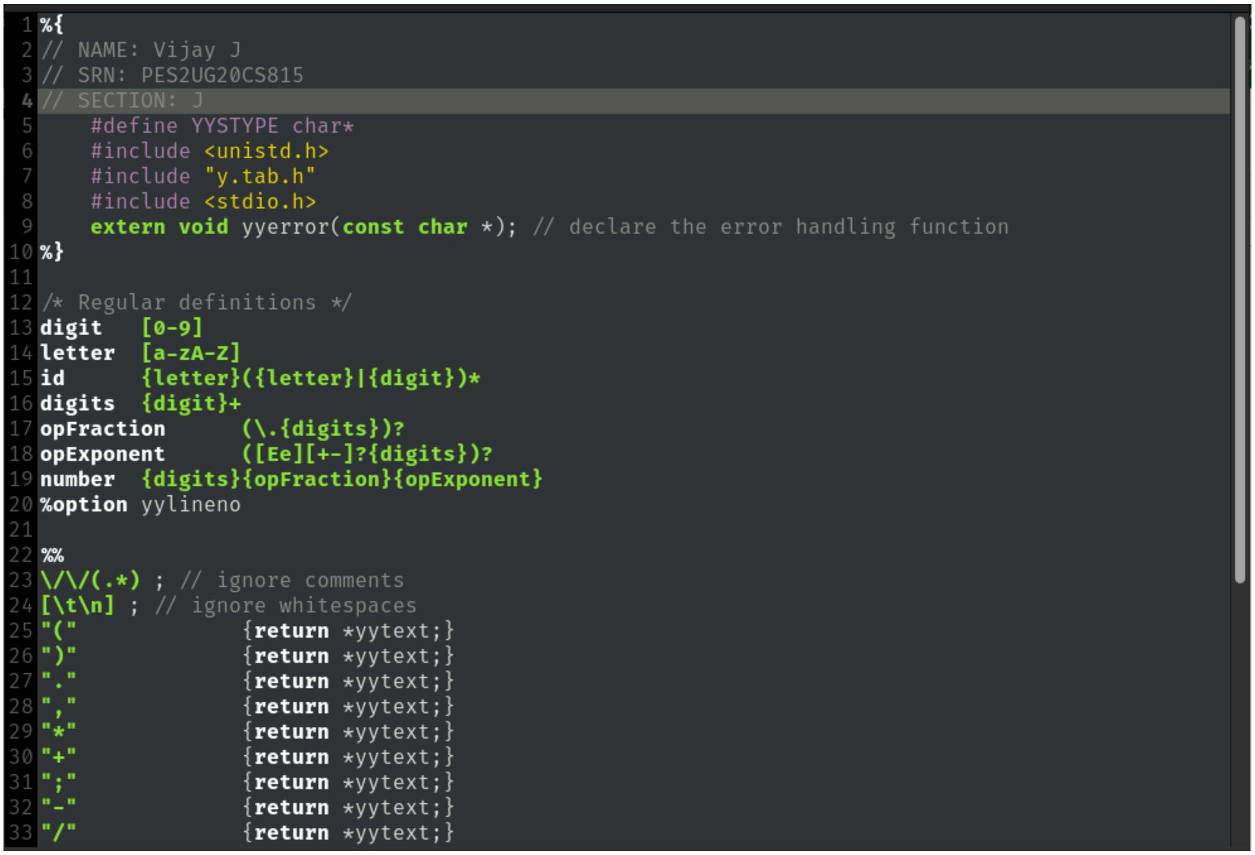
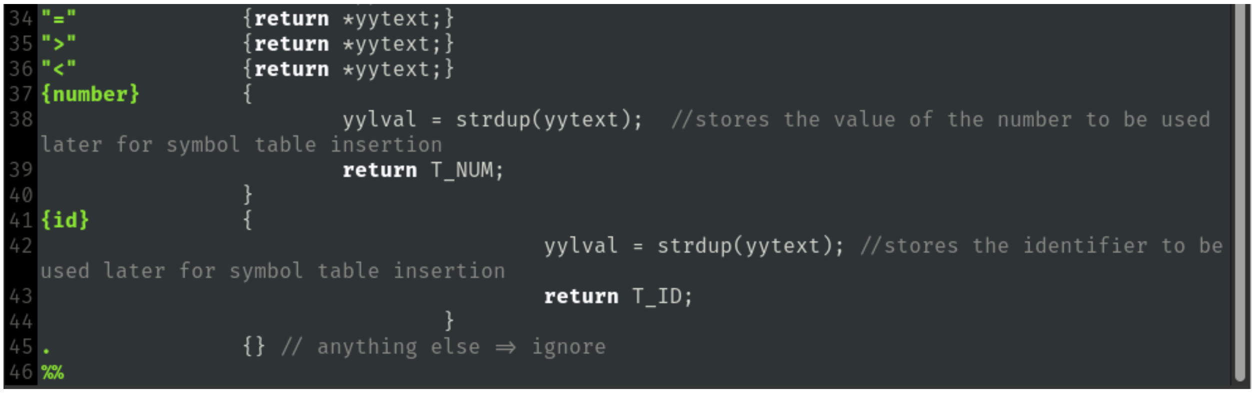
UE20CS353-CD LAB-4

NAME	SRN	CLASS & SECTION
VIJAY J	PES2UG20CS815	6 - J

Lexer.l

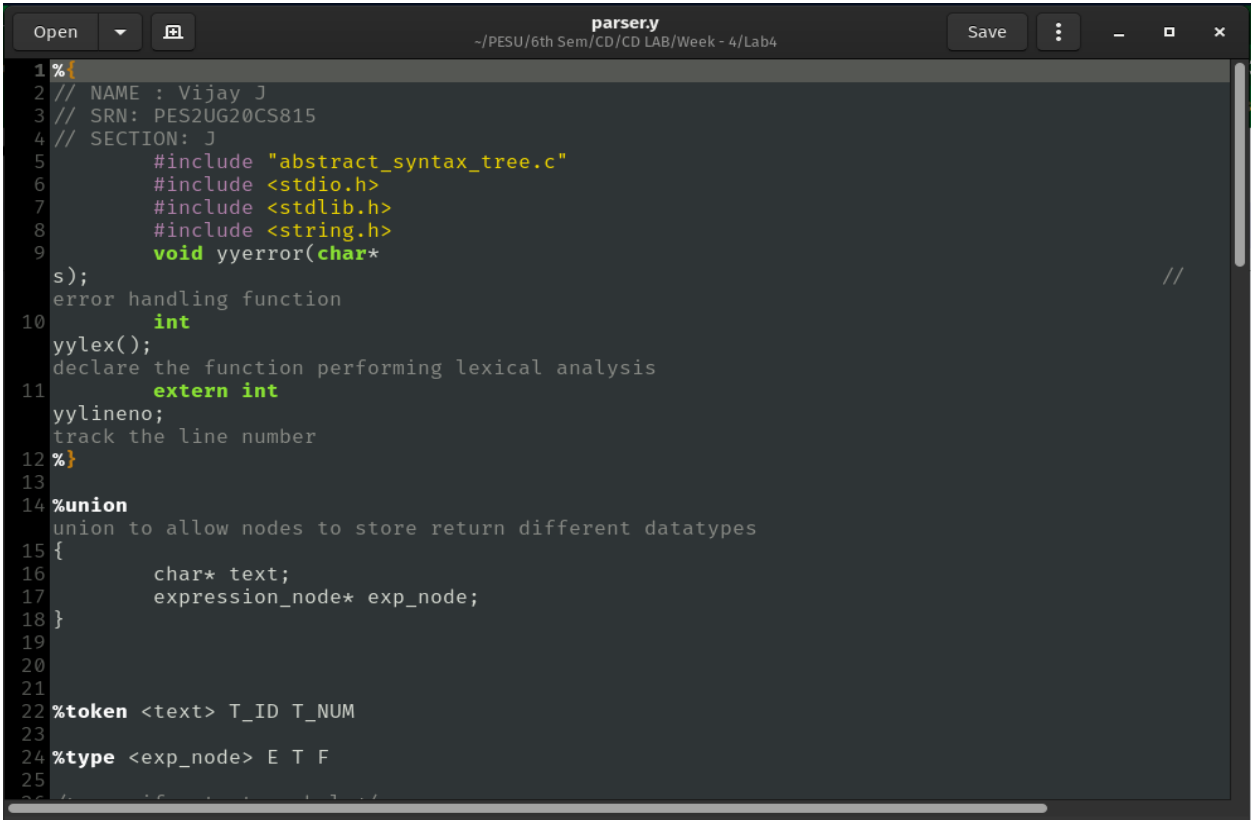


```
%{
// NAME: Vijay J
// SRN: PES2UG20CS815
// SECTION: J
    #define YYSTYPE char*
    #include <unistd.h>
    #include "y.tab.h"
    #include <stdio.h>
    extern void yyerror(const char *); // declare the error handling function
%}

/* 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;}
```

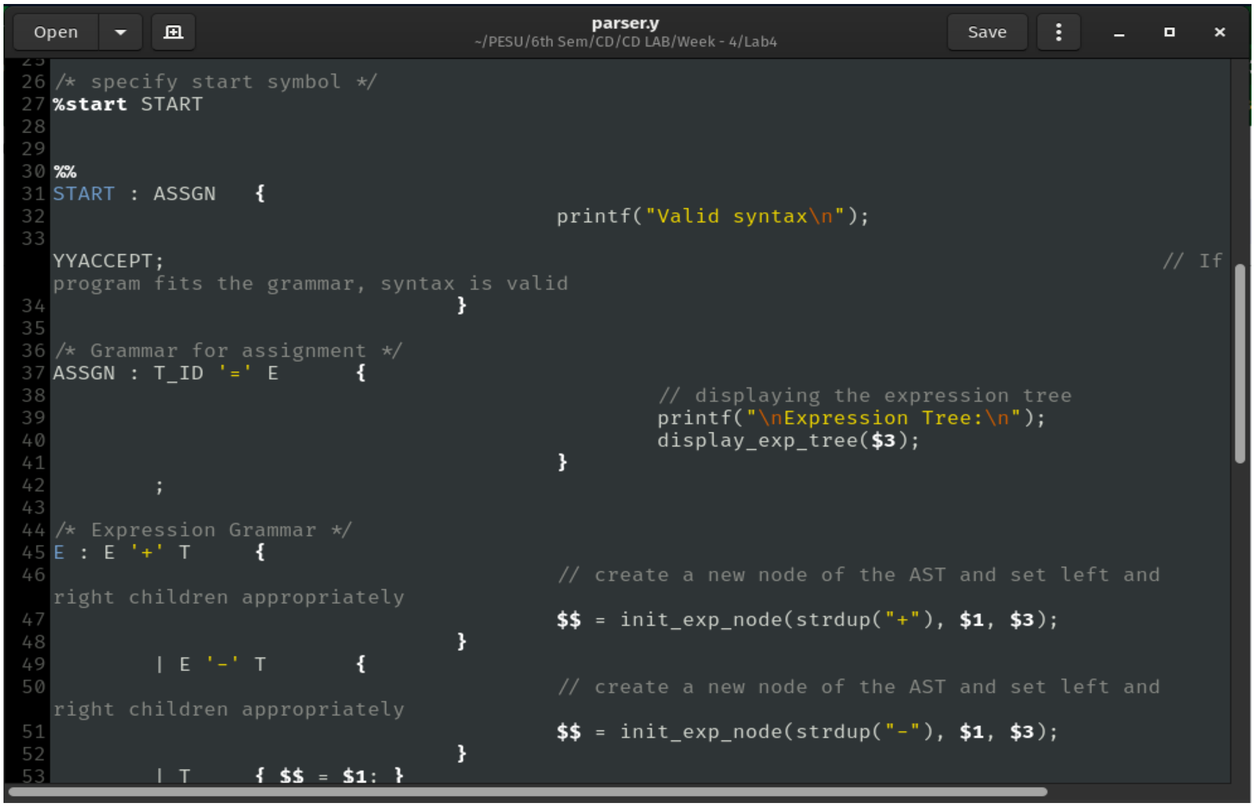


```
"="             {return *yytext;}
">"             {return *yytext;}
"<"             {return *yytext;}
{number}        {
                        yylval = strdup(yytext);  //stores the value of the number to be used later for symbol table insertion
                        return T_NUM;
                }
{id}            {
                                        yylval = strdup(yytext); //stores the identifier to be used later for symbol table insertion
                                        return T_ID;
                                }
.               {} // anything else => ignore
%%
```

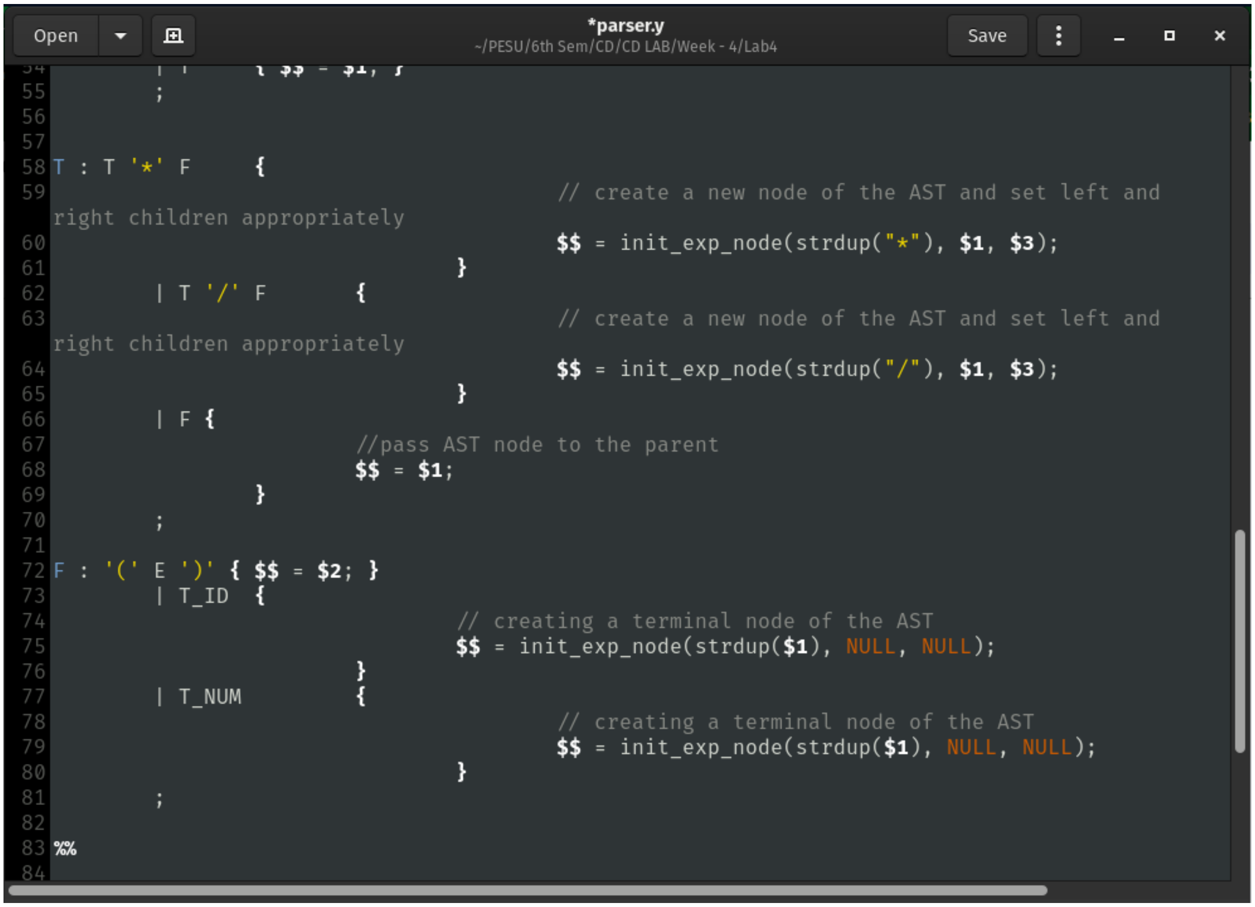
Parser.y



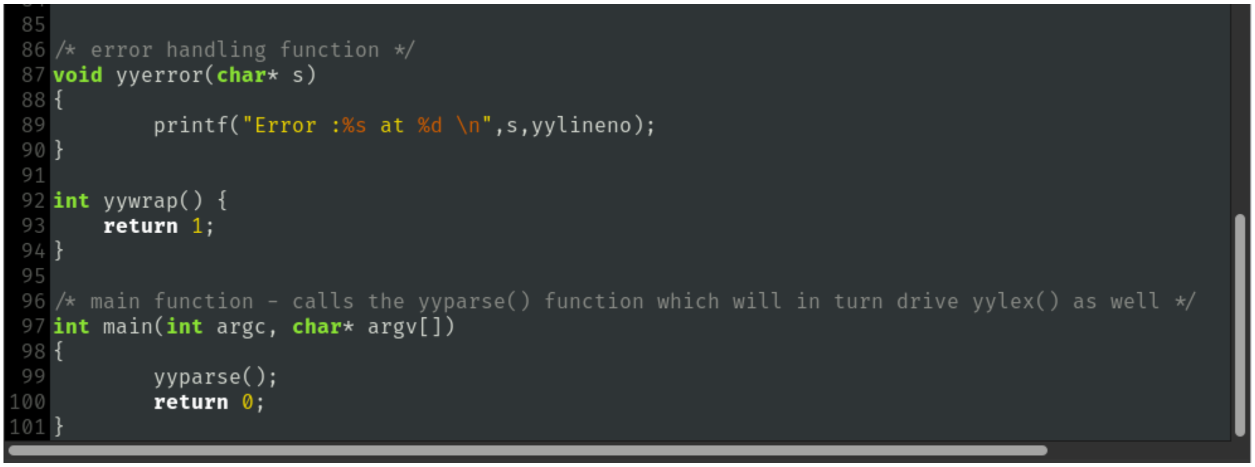
```
%{
// NAME : Vijay J
// SRN: PES2UG20CS815
// SECTION: J
        #include "abstract_syntax_tree.c"
        #include <stdio.h>
        #include <stdlib.h>
        #include <string.h>
        void yyerror(char* s);                                                                  // error handling function
        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

```



```
/* specify start symbol */
%start START


%%
START : ASSGN    {
                                        printf("Valid syntax\n");
                                        YYACCEPT;                                               // If program fits the grammar, syntax is valid
                                }

/* Grammar for assignment */
ASSGN : T_ID '=' E      {
                                                // displaying the expression tree
                                                printf("\nExpression Tree:\n");
                                                display_exp_tree($3);
                                        }

        ;

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

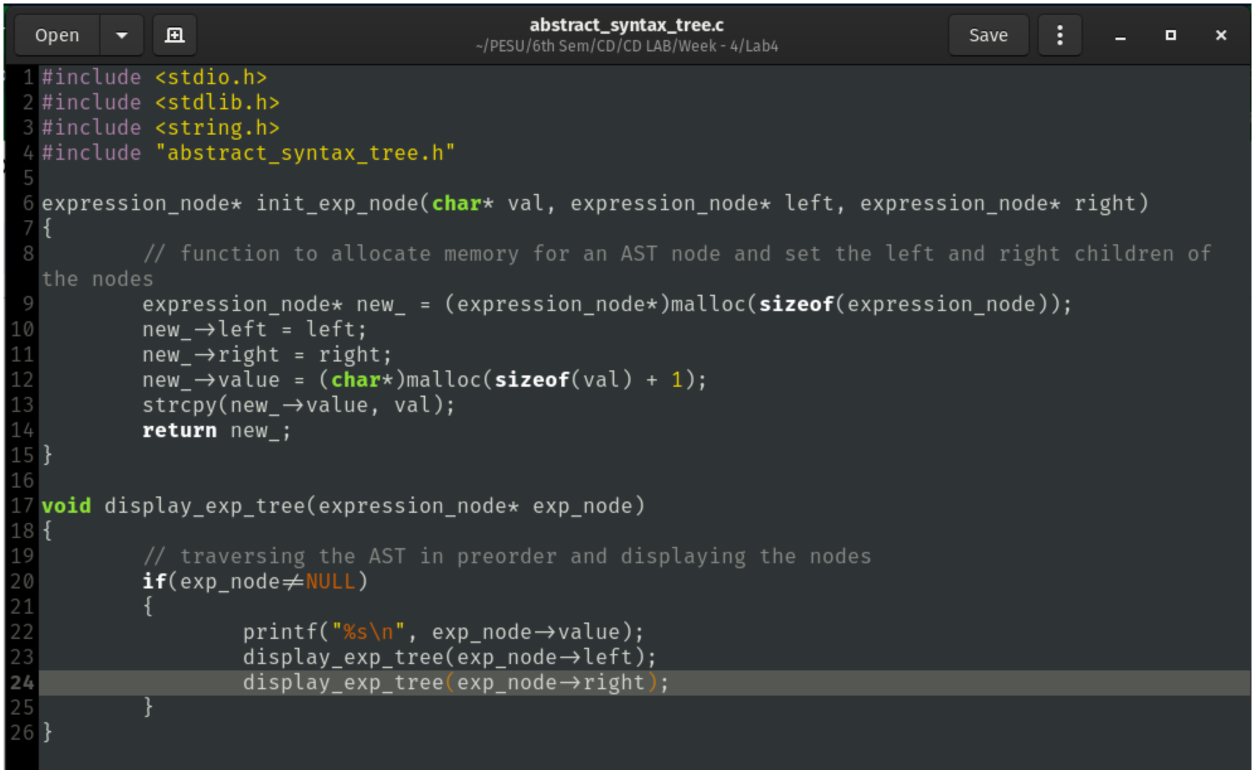


```
        | T     { $$ = $1; }
        ;


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

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

%%

```



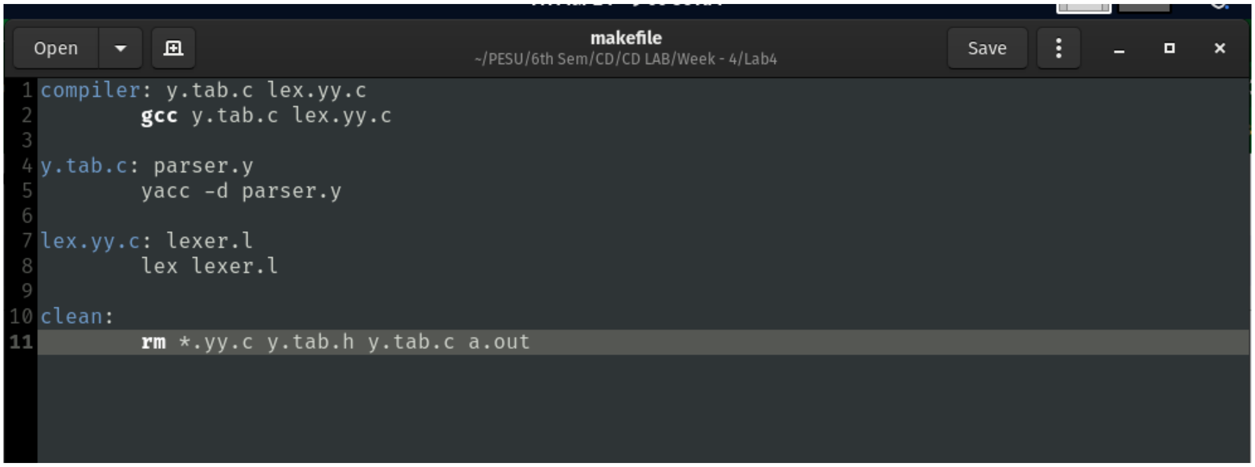
```

/* 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_syntax_tree.c



```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)
{
	// function to allocate memory for an AST node and set the left and right children of the nodes
	expression_node* new_ = (expression_node*)malloc(sizeof(expression_node));
	new_->left = left;
	new_->right = right;
	new_->value = (char*)malloc(sizeof(val) + 1);
	strcpy(new_->value, val);
	return new_;
}

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

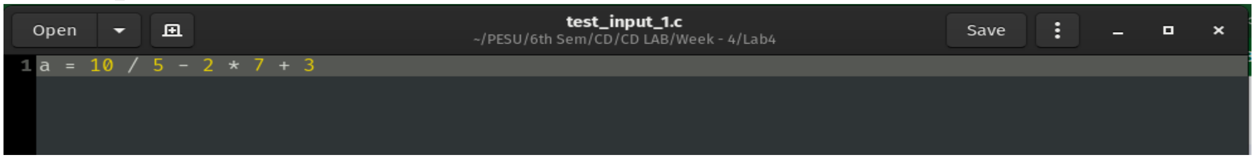
Makefile



```make
compiler: y.tab.c lex.yy.c
	gcc y.tab.c lex.yy.c

y.tab.c: parser.y
	yacc -d parser.y

lex.yy.c: lexer.l
	lex lexer.l

clean:
	rm *.yy.c y.tab.h y.tab.c a.out
```

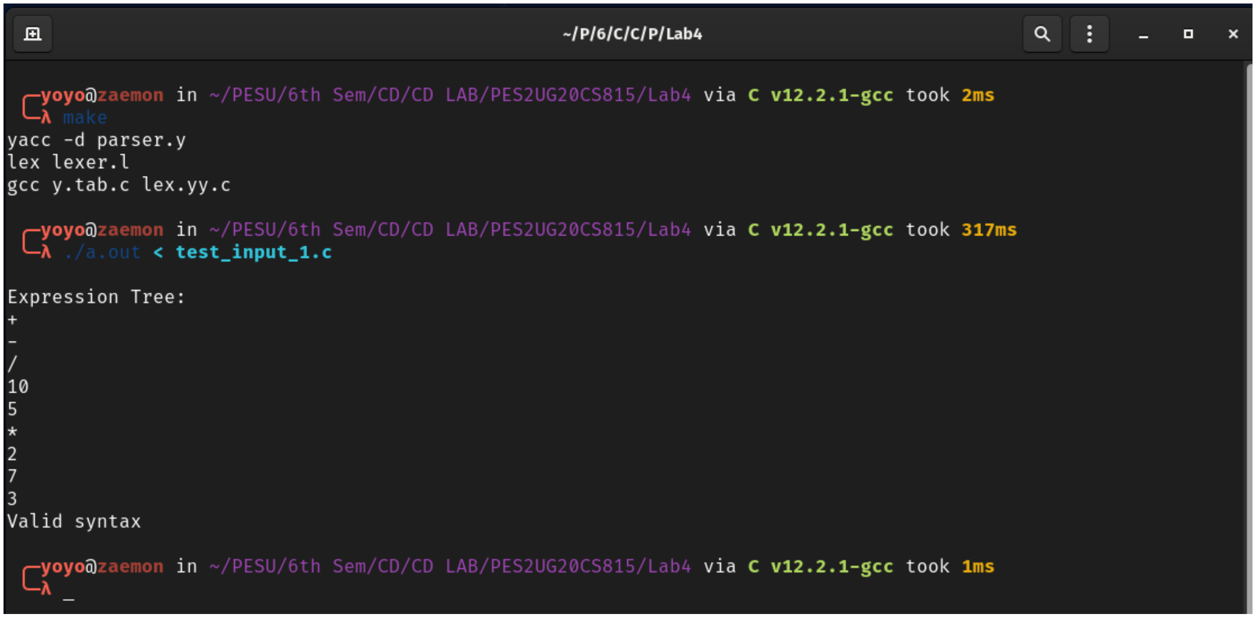
Input And Output

Test_input_1.c



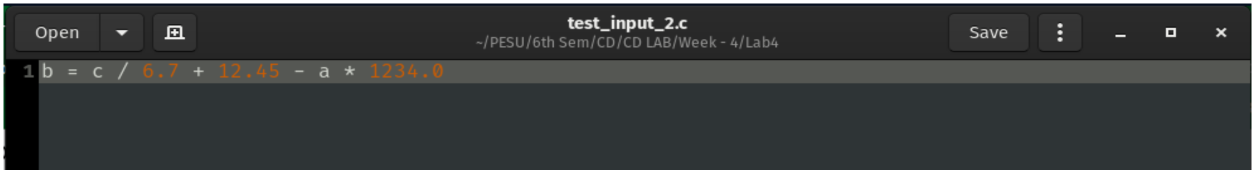
```
a = 10 / 5 - 2 * 7 + 3
```

Output_1



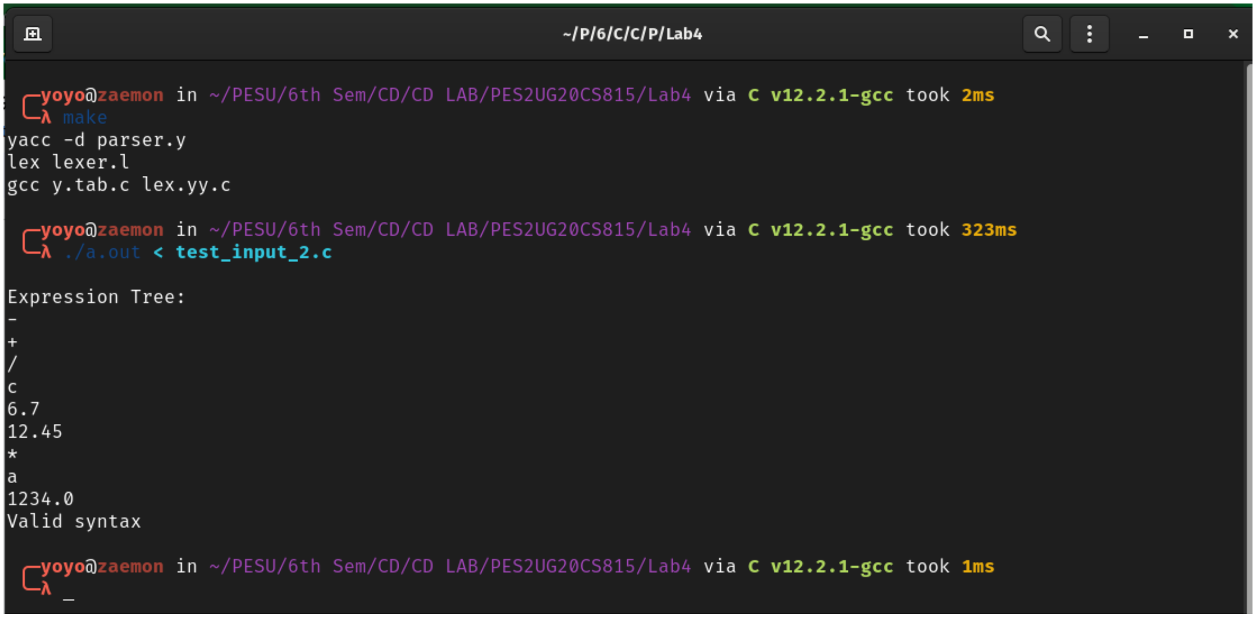
```
┌yoyo@zaemon in ~/PESU/6th Sem/CD/CD LAB/PES2UG20CS815/Lab4 via C v12.2.1-gcc took 2ms
└λ make
yacc -d parser.y
lex lexer.l
gcc y.tab.c lex.yy.c

┌yoyo@zaemon in ~/PESU/6th Sem/CD/CD LAB/PES2UG20CS815/Lab4 via C v12.2.1-gcc took 317ms
└λ ./a.out < test_input_1.c

Expression Tree:
+
-
/
10
5
*
2
7
3
Valid syntax

┌yoyo@zaemon in ~/PESU/6th Sem/CD/CD LAB/PES2UG20CS815/Lab4 via C v12.2.1-gcc took 1ms
└λ _
```

Test_input_2.c



```
b = c / 6.7 + 12.45 - a * 1234.0
```

Output_2



```
┌yoyo@zaemon in ~/PESU/6th Sem/CD/CD LAB/PES2UG20CS815/Lab4 via C v12.2.1-gcc took 2ms
└λ make
yacc -d parser.y
lex lexer.l
gcc y.tab.c lex.yy.c

┌yoyo@zaemon in ~/PESU/6th Sem/CD/CD LAB/PES2UG20CS815/Lab4 via C v12.2.1-gcc took 323ms
└λ ./a.out < test_input_2.c

Expression Tree:
-
+
/
c
6.7
12.45
*
a
1234.0
Valid syntax

┌yoyo@zaemon in ~/PESU/6th Sem/CD/CD LAB/PES2UG20CS815/Lab4 via C v12.2.1-gcc took 1ms
└λ _
```