COMPILER_DESGIN_LAB-5

LEXER.L

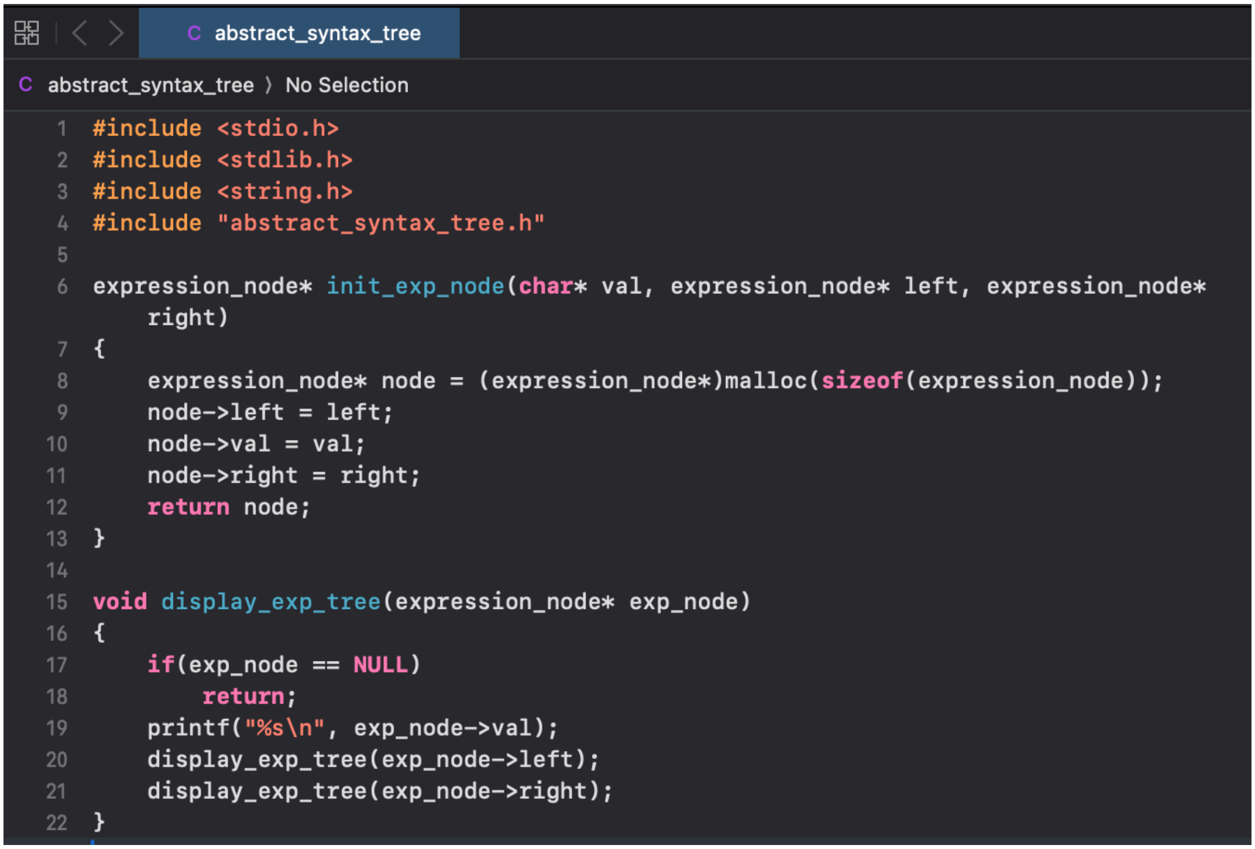
```
%{
   #define YYSTYPE char*
   #include <unistd.h>
   #include <string.h>
   #include "y.tab.h"
   #include <stdio.h>
extern void yyerror(const char *); // declare the error handling function
%}

/* 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
%%
int yywrap() { return 1; }
```

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();  // Function performing lexical analysis
   extern int yylineno;  // Track the line number
%}

%union  // Union to allow nodes to store different data types
{
   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 {
            display_exp_tree($3);  // Display the expression tree ($3)
         }
;

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

T : T '*' F {
```

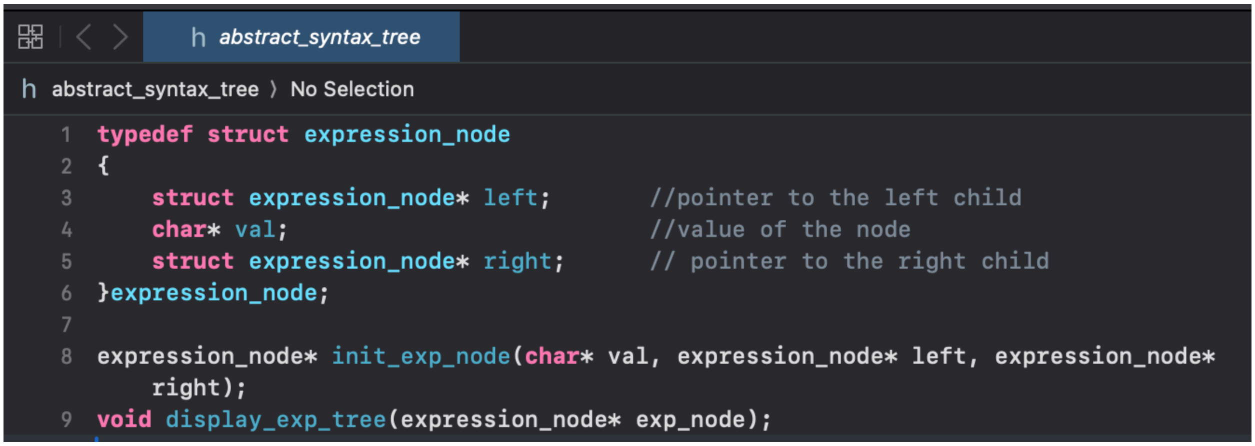
```
                $$ = init_exp_node("*", $1, $3);  // Create a new node of the AST and set left and
right children
            }
  | T '/' F {
                $$ = init_exp_node("/", $1, $3);  // Create a new node of the AST and set left and
right children
            }
  | F { $$ = $1; }  // Pass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

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

%%

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

/* 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)
{
    expression_node* node = (expression_node*)malloc(sizeof(expression_node));
    node->left = left;
    node->val = val;
    node->right = right;
    return node;
}

void display_exp_tree(expression_node* exp_node)
{
    if(exp_node == NULL)
        return;
    printf("%s\n", exp_node->val);
    display_exp_tree(exp_node->left);
    display_exp_tree(exp_node->right);
}
```

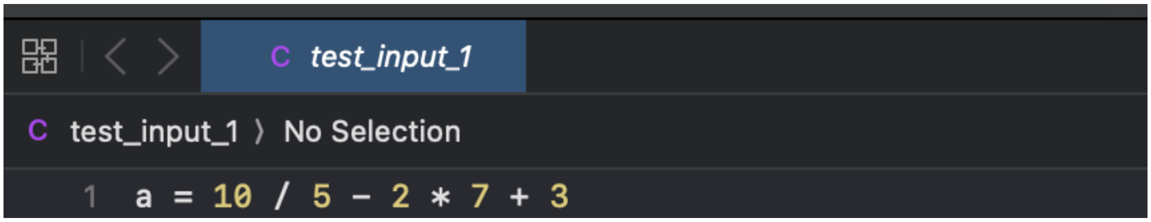
ABSTRACT_SYNTAX_TREE.H



```c
typedef struct expression_node
{
    struct expression_node* left;       //pointer to the left child
    char* val;                          //value of the node
    struct expression_node* right;      // pointer to the right child
}expression_node;

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

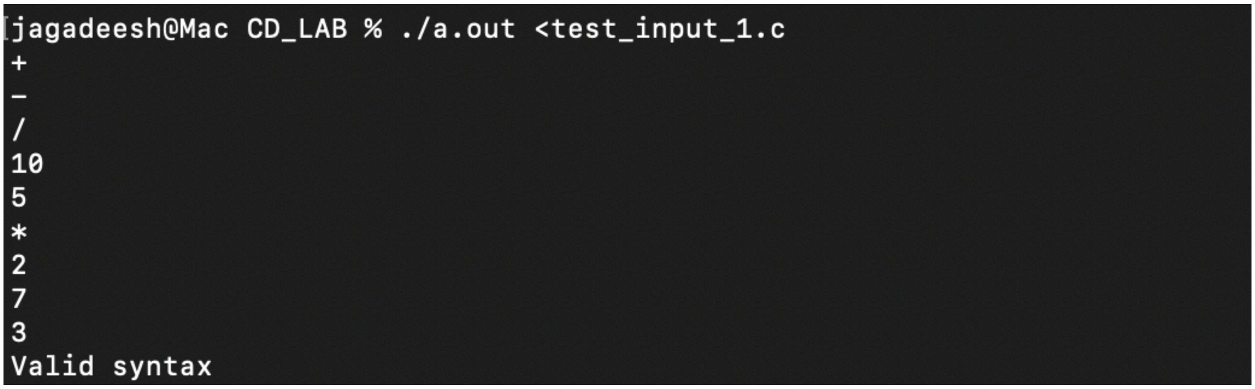
TEST_INPUT_1.C

Code:



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

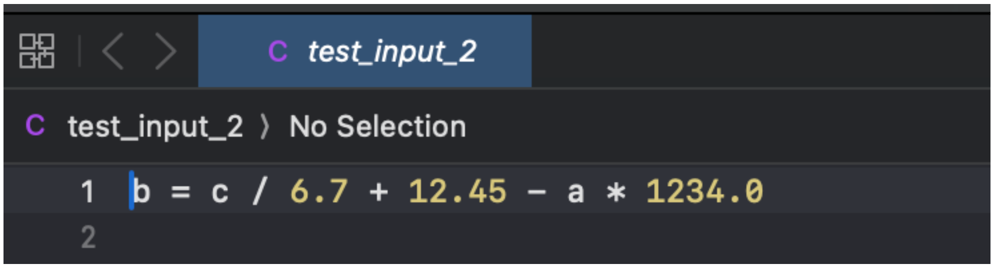
Output:



```
[jagadeesh@Mac CD_LAB % ./a.out <test_input_1.c
+
-
/
10
5
*
2
7
3
Valid syntax
```

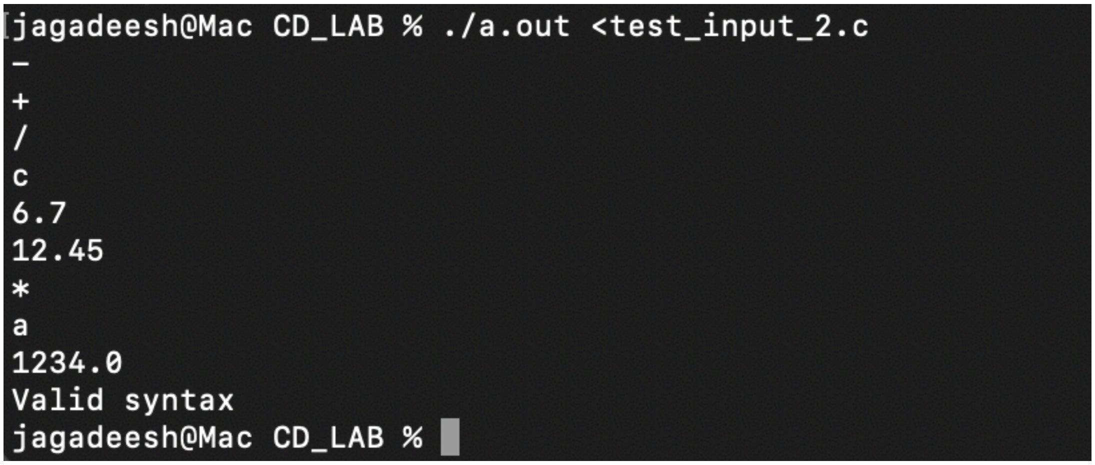
TEST_INPUT_2.C

Code:



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

```

Output:



```
[jagadeesh@Mac CD_LAB % ./a.out <test_input_2.c
-
+
/
c
6.7
12.45
*
a
1234.0
Valid syntax
jagadeesh@Mac CD_LAB %
```