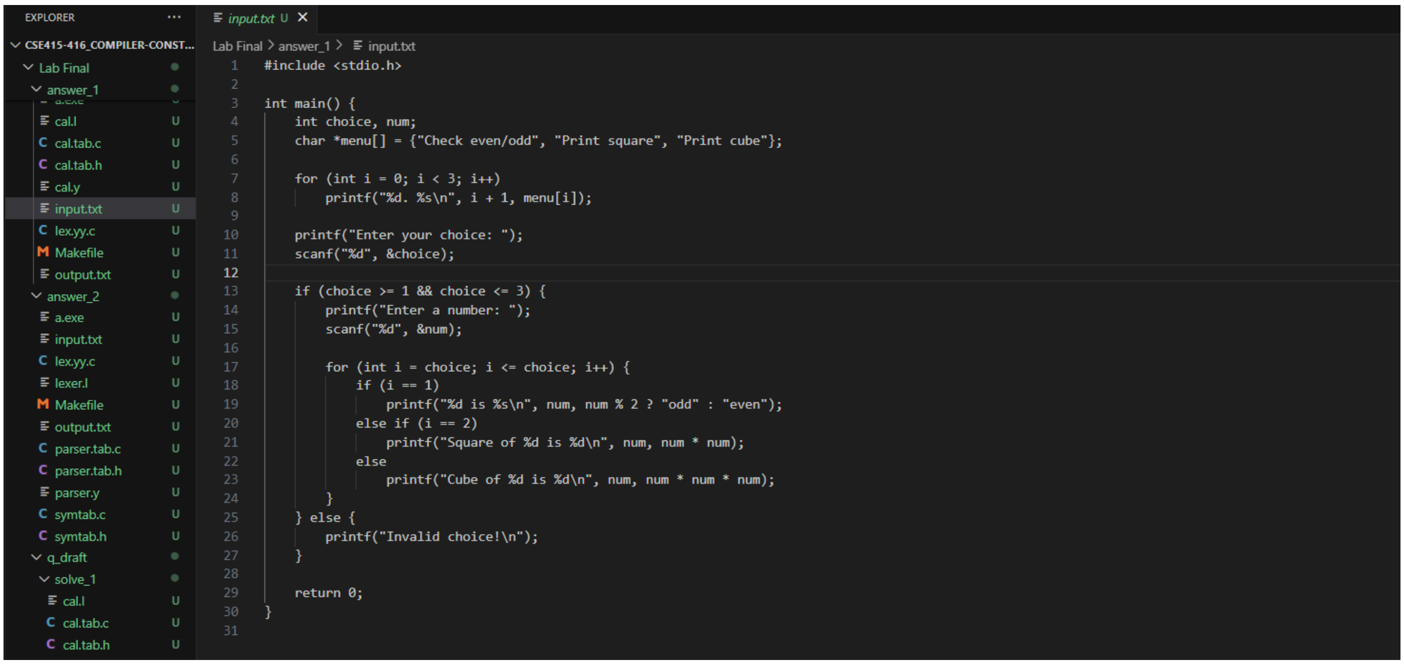
<u>**Set-A**</u>

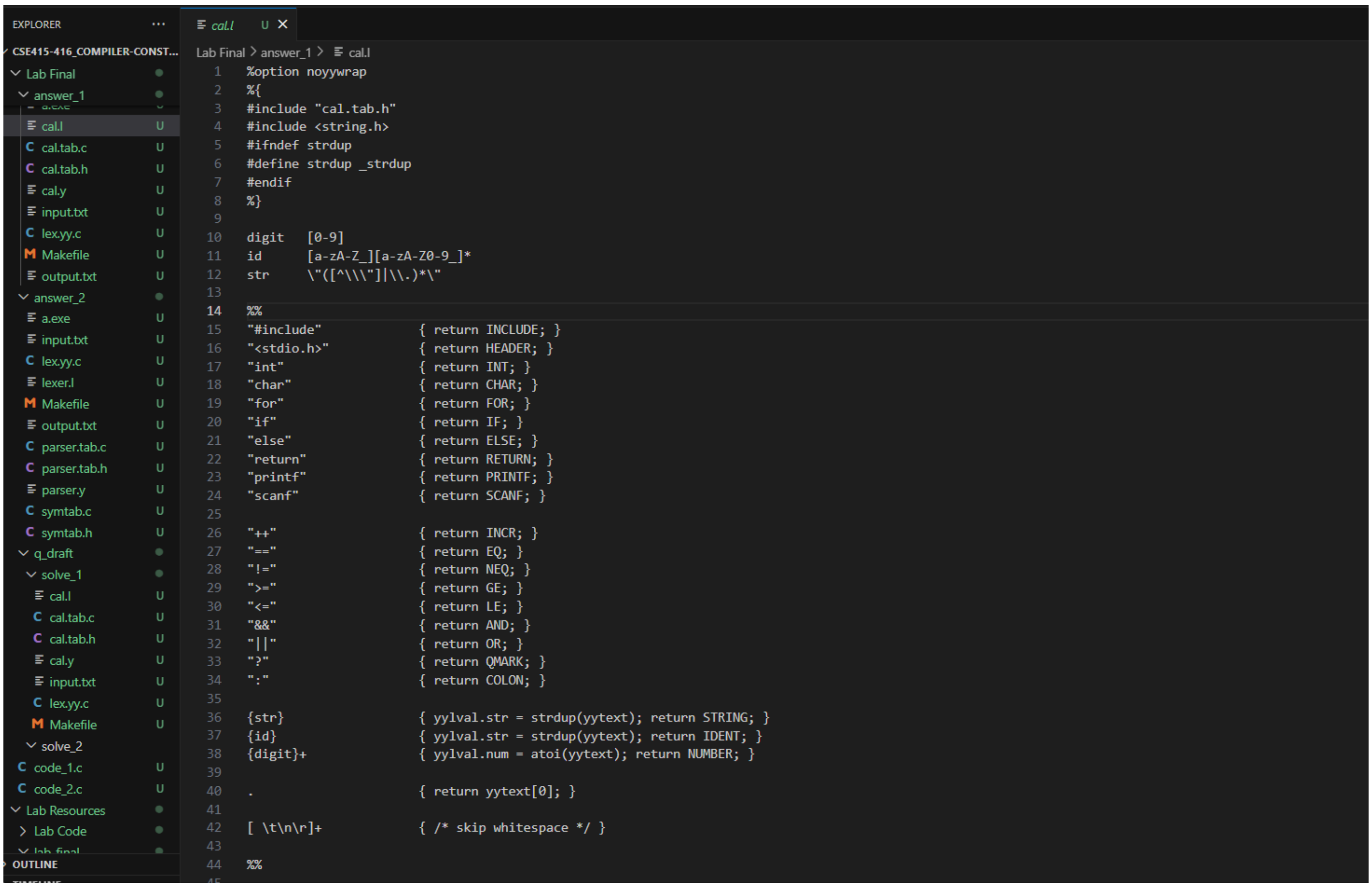
<u>**Ans. to the Q. No. (1)**</u>

**input.txt**



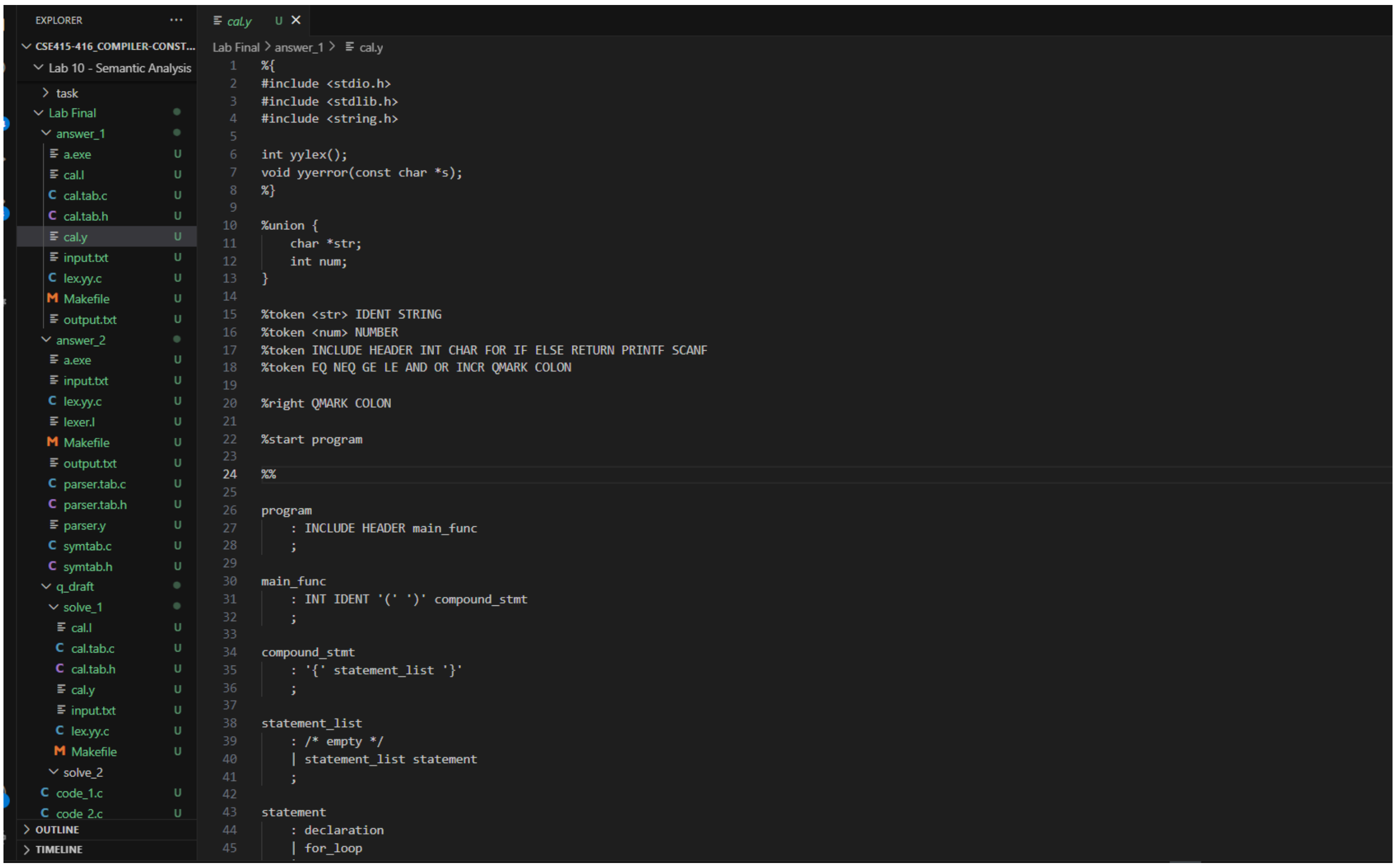
```c
#include <stdio.h>

int main() {
    int choice, num;
    char *menu[] = {"Check even/odd", "Print square", "Print cube"};

    for (int i = 0; i < 3; i++)
        printf("%d. %s\n", i + 1, menu[i]);

    printf("Enter your choice: ");
    scanf("%d", &choice);

    if (choice >= 1 && choice <= 3) {
        printf("Enter a number: ");
        scanf("%d", &num);

        for (int i = choice; i <= choice; i++) {
            if (i == 1)
                printf("%d is %s\n", num, num % 2 ? "odd" : "even");
            else if (i == 2)
                printf("Square of %d is %d\n", num, num * num);
            else
                printf("Cube of %d is %d\n", num, num * num * num);
        }
    } else {
        printf("Invalid choice!\n");
    }

    return 0;
}
```

**cal.l**

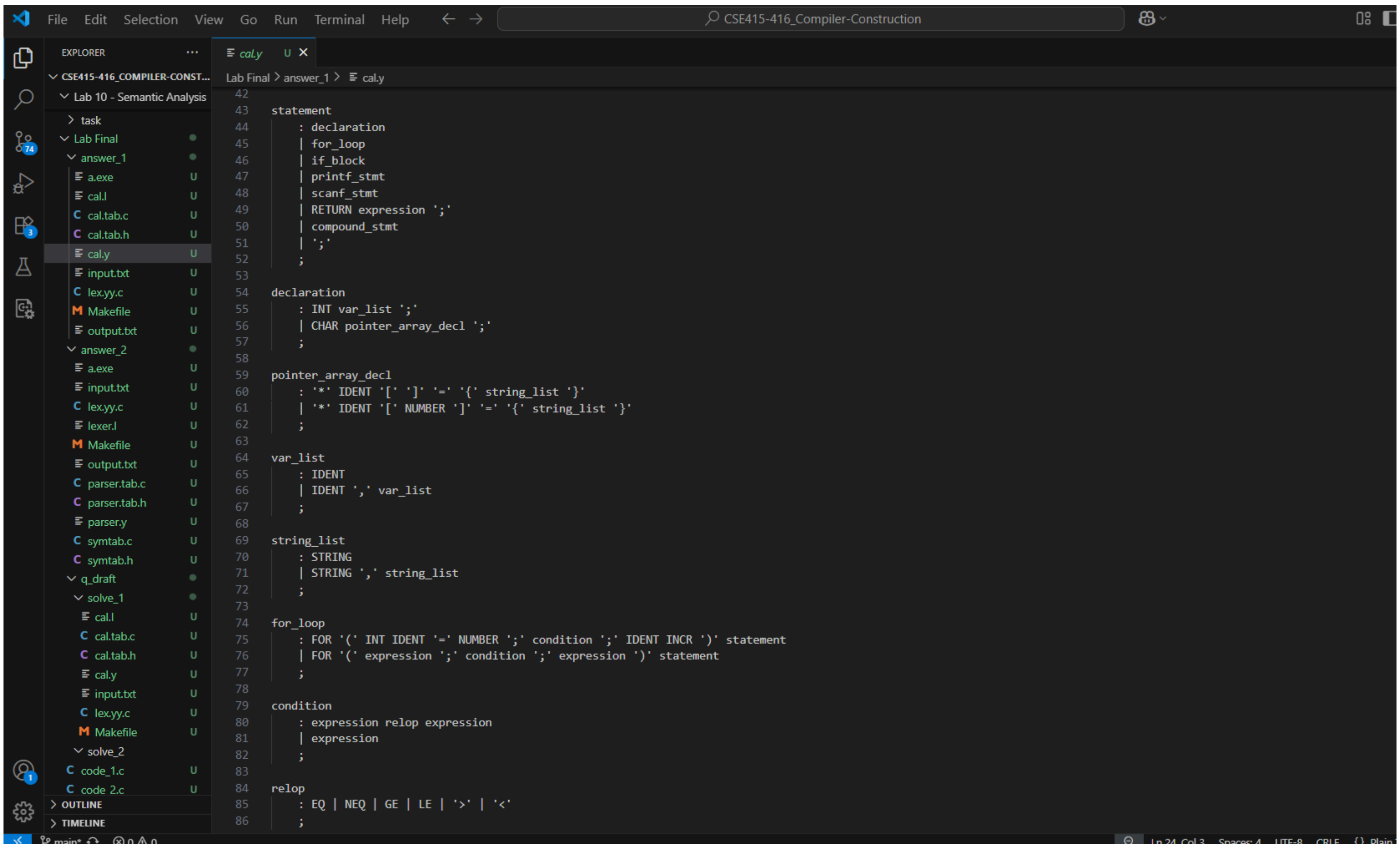


```
%option noyywrap
%{
#include "cal.tab.h"
#include <string.h>
#ifndef strdup
#define strdup _strdup
#endif
%}

digit   [0-9]
id      [a-zA-Z_][a-zA-Z0-9_]*
str     \"([^\\\"]|\\.)*\"

%%
"#include"              { return INCLUDE; }
"<stdio.h>"             { return HEADER; }
"int"                   { return INT; }
"char"                  { return CHAR; }
"for"                   { return FOR; }
"if"                    { return IF; }
"else"                  { return ELSE; }
"return"                { return RETURN; }
"printf"                { return PRINTF; }
"scanf"                 { return SCANF; }

"++"                    { return INCR; }
"=="                    { return EQ; }
"!="                    { return NEQ; }
">="                    { return GE; }
"<="                    { return LE; }
"&&"                    { return AND; }
"||"                    { return OR; }
"?"                     { return QMARK; }
":"                     { return COLON; }

{str}                   { yylval.str = strdup(yytext); return STRING; }
{id}                    { yylval.str = strdup(yytext); return IDENT; }
{digit}+                { yylval.num = atoi(yytext); return NUMBER; }

.                       { return yytext[0]; }

[ \t\n\r]+              { /* skip whitespace */ }

%%
```

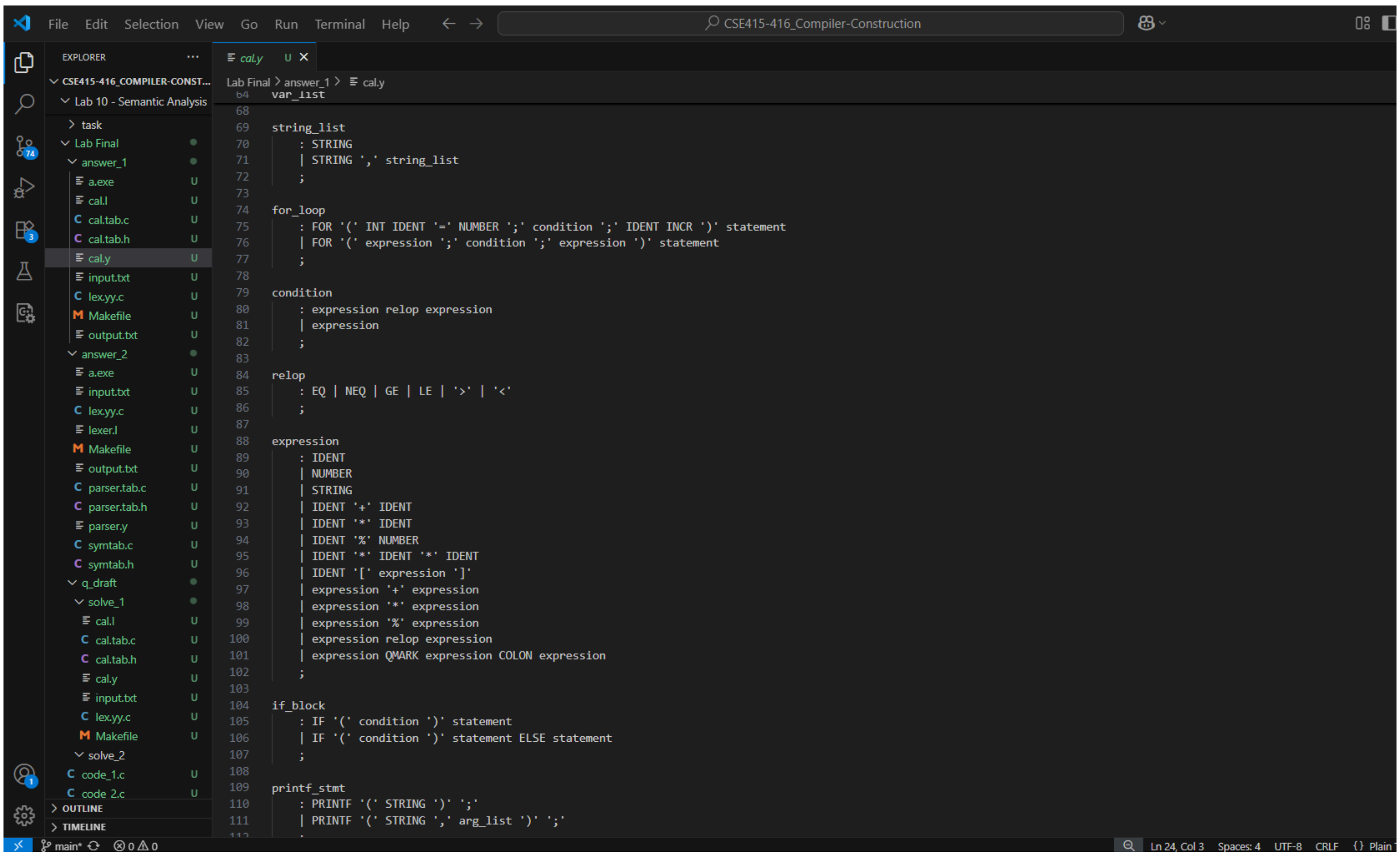
## cal.y



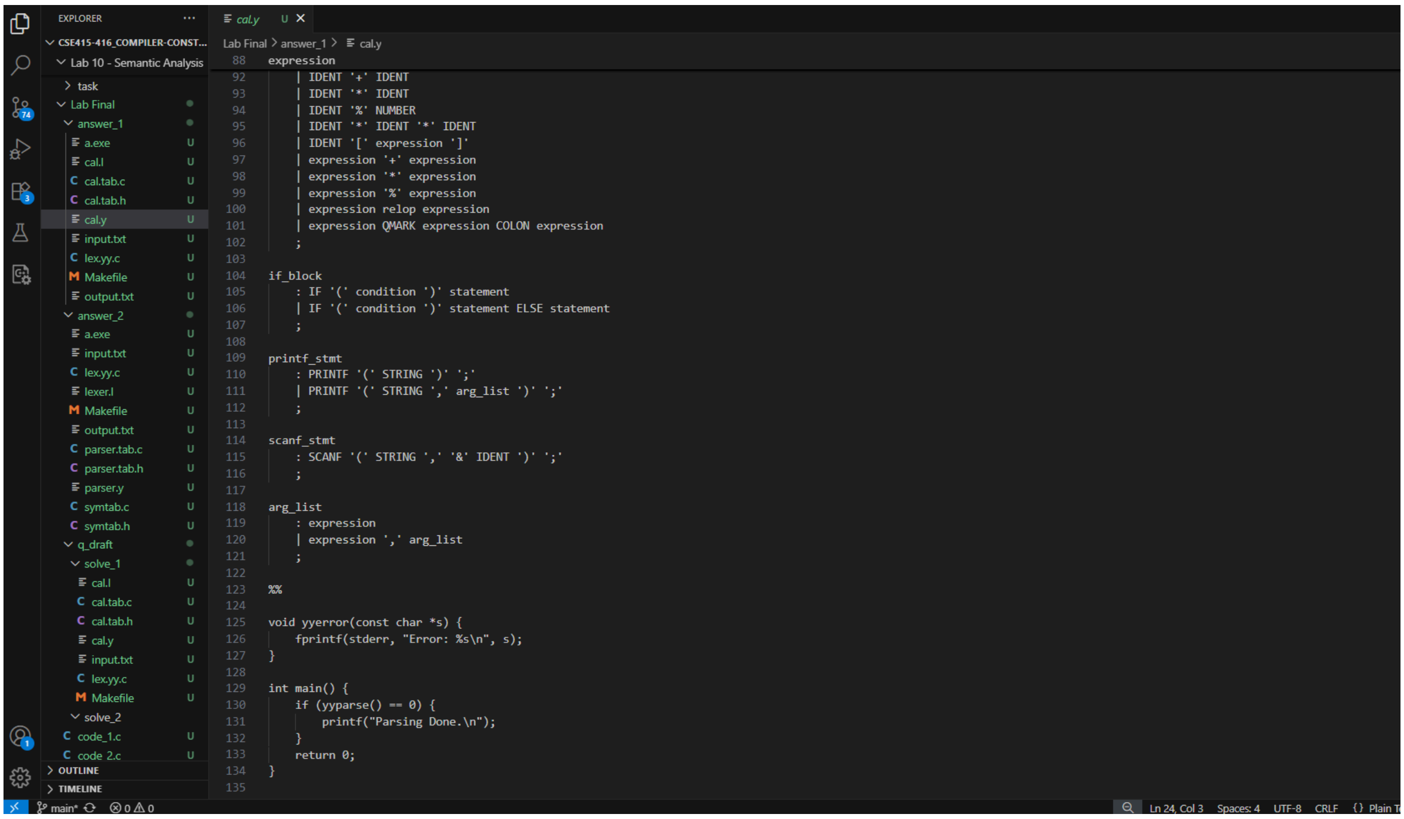
```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int yylex();
void yyerror(const char *s);
%}

%union {
    char *str;
    int num;
}

%token <str> IDENT STRING
%token <num> NUMBER
%token INCLUDE HEADER INT CHAR FOR IF ELSE RETURN PRINTF SCANF
%token EQ NEQ GE LE AND OR INCR QMARK COLON

%right QMARK COLON

%start program

%%

program
    : INCLUDE HEADER main_func
    ;

main_func
    : INT IDENT '(' ')' compound_stmt
    ;

compound_stmt
    : '{' statement_list '}'
    ;

statement_list
    : /* empty */
    | statement_list statement
    ;

statement
    : declaration
    | for_loop
```



```
statement
    : declaration
    | for_loop
    | if_block
    | printf_stmt
    | scanf_stmt
    | RETURN expression ';'
    | compound_stmt
    | ';'
    ;

declaration
    : INT var_list ';'
    | CHAR pointer_array_decl ';'
    ;

pointer_array_decl
    : '*' IDENT '[' ']' '=' '{' string_list '}'
    | '*' IDENT '[' NUMBER ']' '=' '{' string_list '}'
    ;

var_list
    : IDENT
    | IDENT ',' var_list
    ;

string_list
    : STRING
    | STRING ',' string_list
    ;

for_loop
    : FOR '(' INT IDENT '=' NUMBER ';' condition ';' IDENT INCR ')' statement
    | FOR '(' expression ';' condition ';' expression ')' statement
    ;

condition
    : expression relop expression
    | expression
    ;

relop
    : EQ | NEQ | GE | LE | '>' | '<'
    ;
```

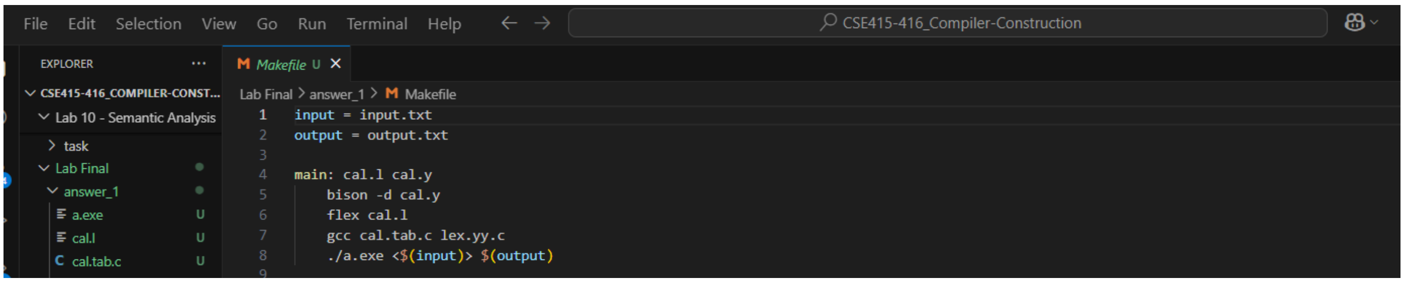

CSE415-416_Compiler-Construction
cal.y
Lab Final > answer_1 > cal.y

```
var_list

string_list
    : STRING
    | STRING ',' string_list
    ;

for_loop
    : FOR '(' INT IDENT '=' NUMBER ';' condition ';' IDENT INCR ')' statement
    | FOR '(' expression ';' condition ';' expression ')' statement
    ;

condition
    : expression relop expression
    | expression
    ;

relop
    : EQ | NEQ | GE | LE | '>' | '<'
    ;

expression
    : IDENT
    | NUMBER
    | STRING
    | IDENT '+' IDENT
    | IDENT '*' IDENT
    | IDENT '%' NUMBER
    | IDENT '*' IDENT '*' IDENT
    | IDENT '[' expression ']'
    | expression '+' expression
    | expression '*' expression
    | expression '%' expression
    | expression relop expression
    | expression QMARK expression COLON expression
    ;

if_block
    : IF '(' condition ')' statement
    | IF '(' condition ')' statement ELSE statement
    ;

printf_stmt
    : PRINTF '(' STRING ')' ';'
    | PRINTF '(' STRING ',' arg_list ')' ';'
```



```
expression
    | IDENT '+' IDENT
    | IDENT '*' IDENT
    | IDENT '%' NUMBER
    | IDENT '*' IDENT '*' IDENT
    | IDENT '[' expression ']'
    | expression '+' expression
    | expression '*' expression
    | expression '%' expression
    | expression relop expression
    | expression QMARK expression COLON expression
    ;

if_block
    : IF '(' condition ')' statement
    | IF '(' condition ')' statement ELSE statement
    ;

printf_stmt
    : PRINTF '(' STRING ')' ';'
    | PRINTF '(' STRING ',' arg_list ')' ';'
    ;

scanf_stmt
    : SCANF '(' STRING ',' '&' IDENT ')' ';'
    ;

arg_list
    : expression
    | expression ',' arg_list
    ;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main() {
    if (yyparse() == 0) {
        printf("Parsing Done.\n");
    }
    return 0;
}
```

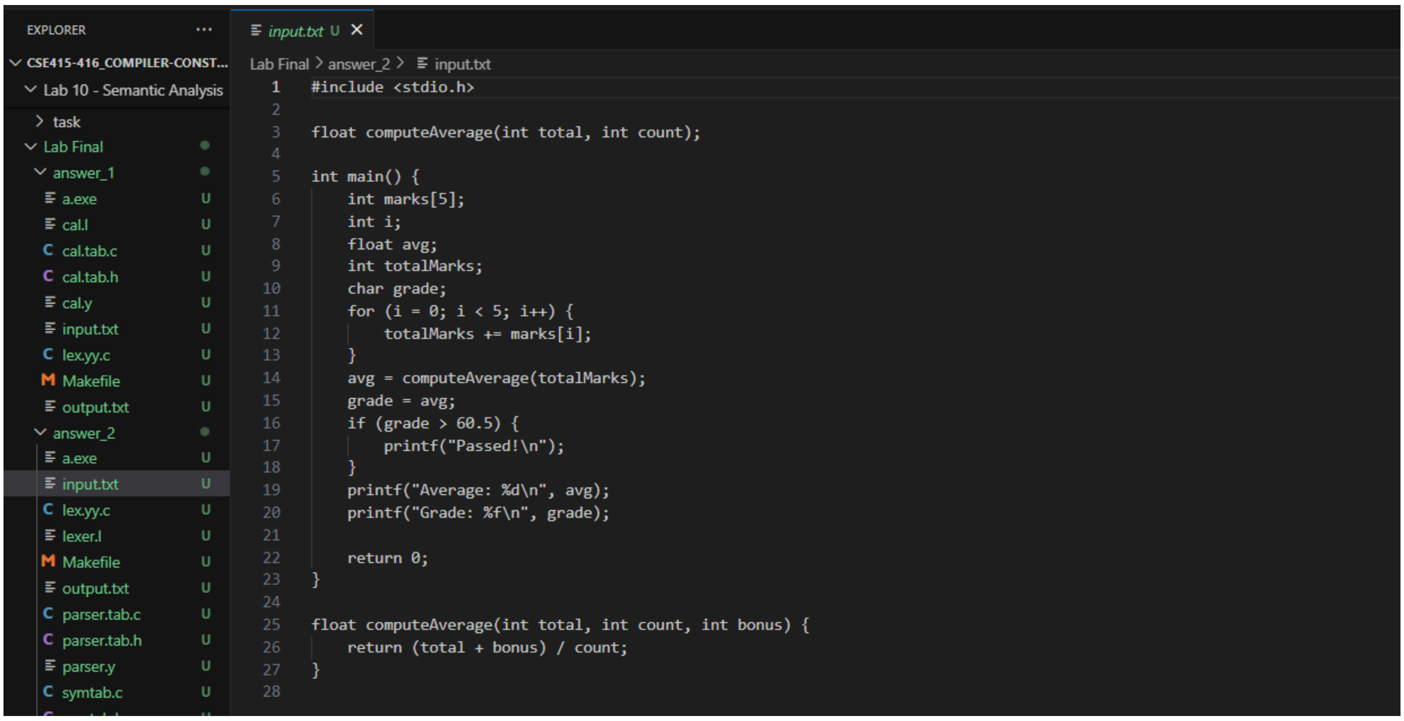
## Makefile


CSE415-416_Compiler-Construction
Makefile
Lab Final > answer_1 > Makefile

```
input = input.txt
output = output.txt

main: cal.l cal.y
	bison -d cal.y
	flex cal.l
	gcc cal.tab.c lex.yy.c
	./a.exe <$(input)> $(output)
```

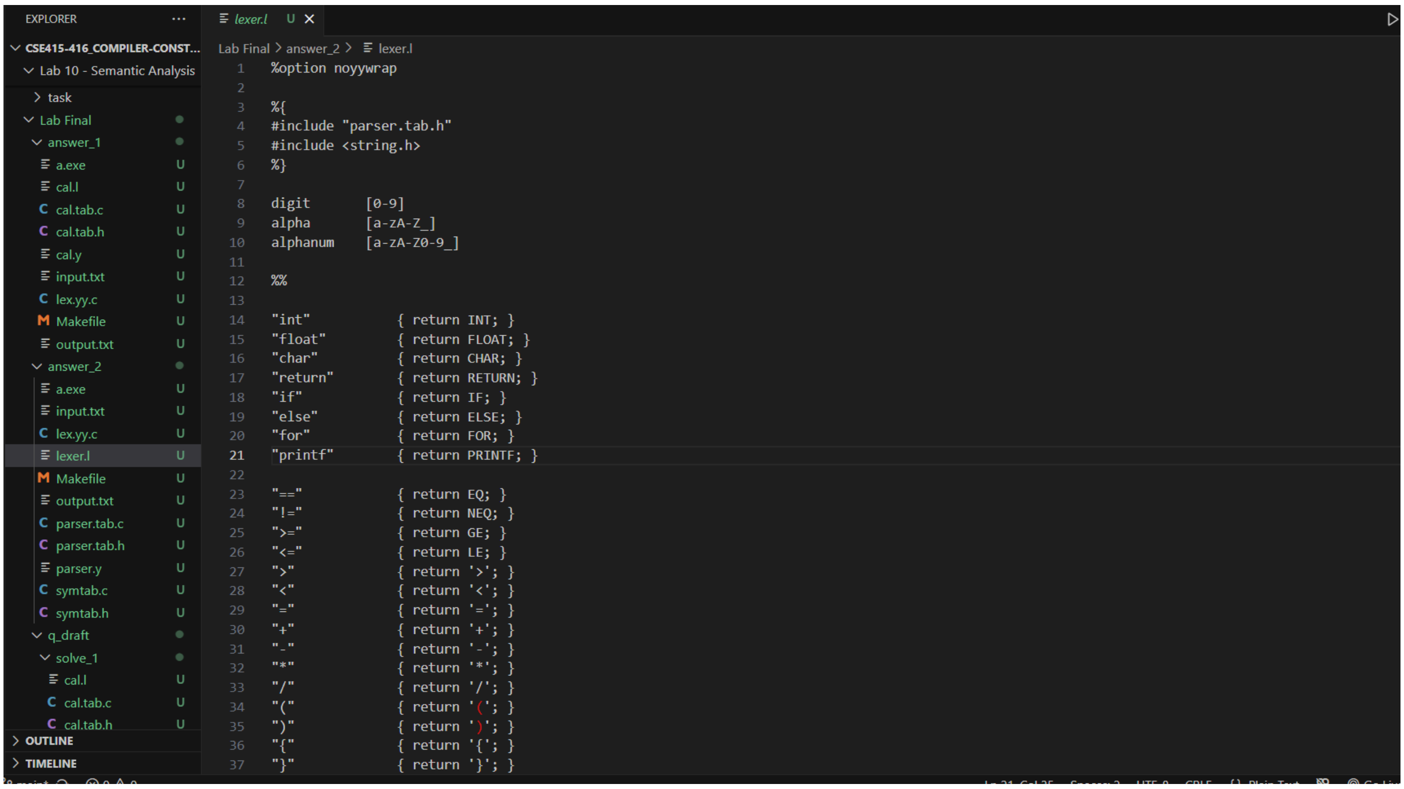
## Ans. to the Q. No. (2)

### input.txt

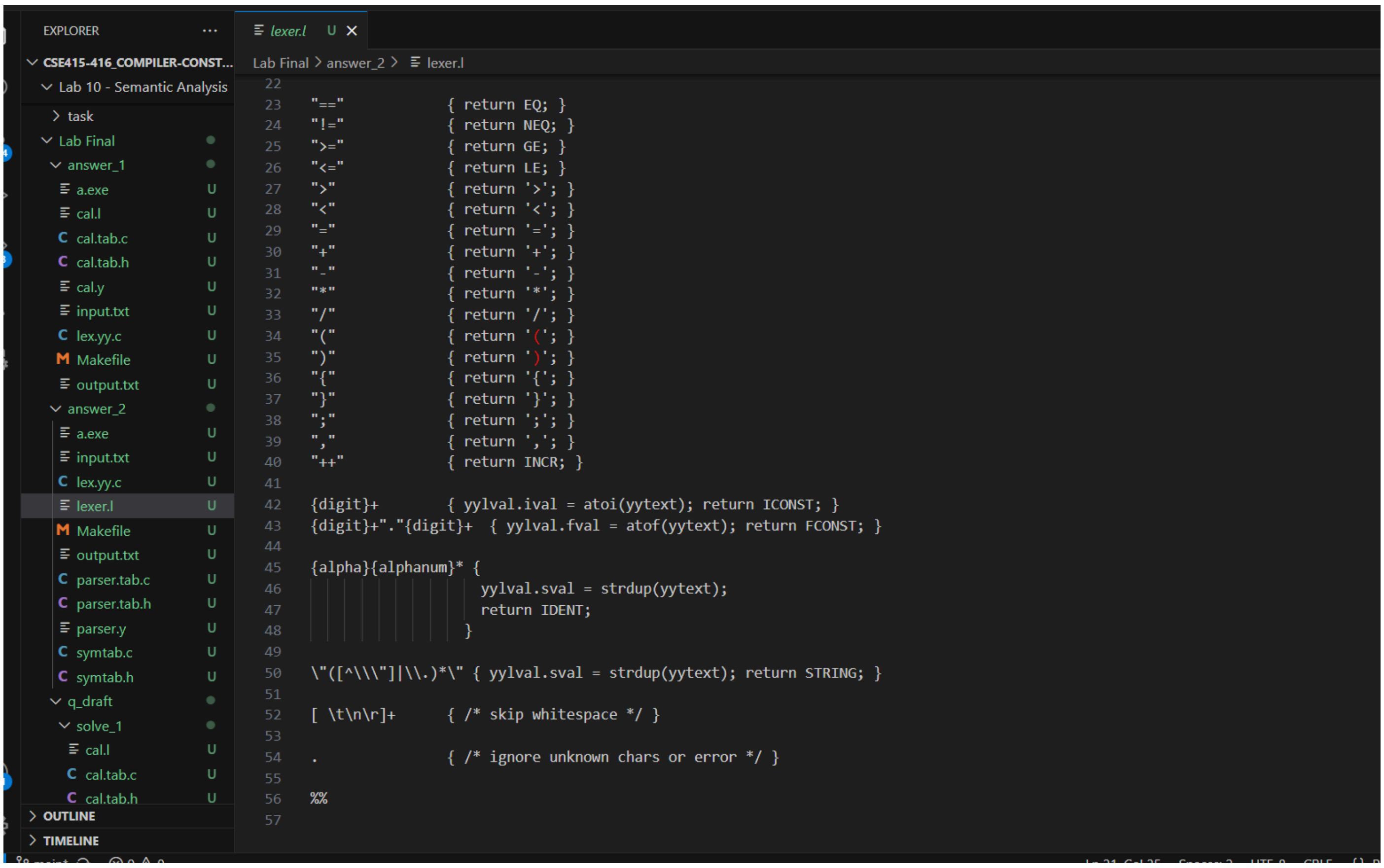


```c
#include <stdio.h>

float computeAverage(int total, int count);

int main() {
    int marks[5];
    int i;
    float avg;
    int totalMarks;
    char grade;
    for (i = 0; i < 5; i++) {
        totalMarks += marks[i];
    }
    avg = computeAverage(totalMarks);
    grade = avg;
    if (grade > 60.5) {
        printf("Passed!\n");
    }
    printf("Average: %d\n", avg);
    printf("Grade: %f\n", grade);

    return 0;
}

float computeAverage(int total, int count, int bonus) {
    return (total + bonus) / count;
}
```

### lexer.l

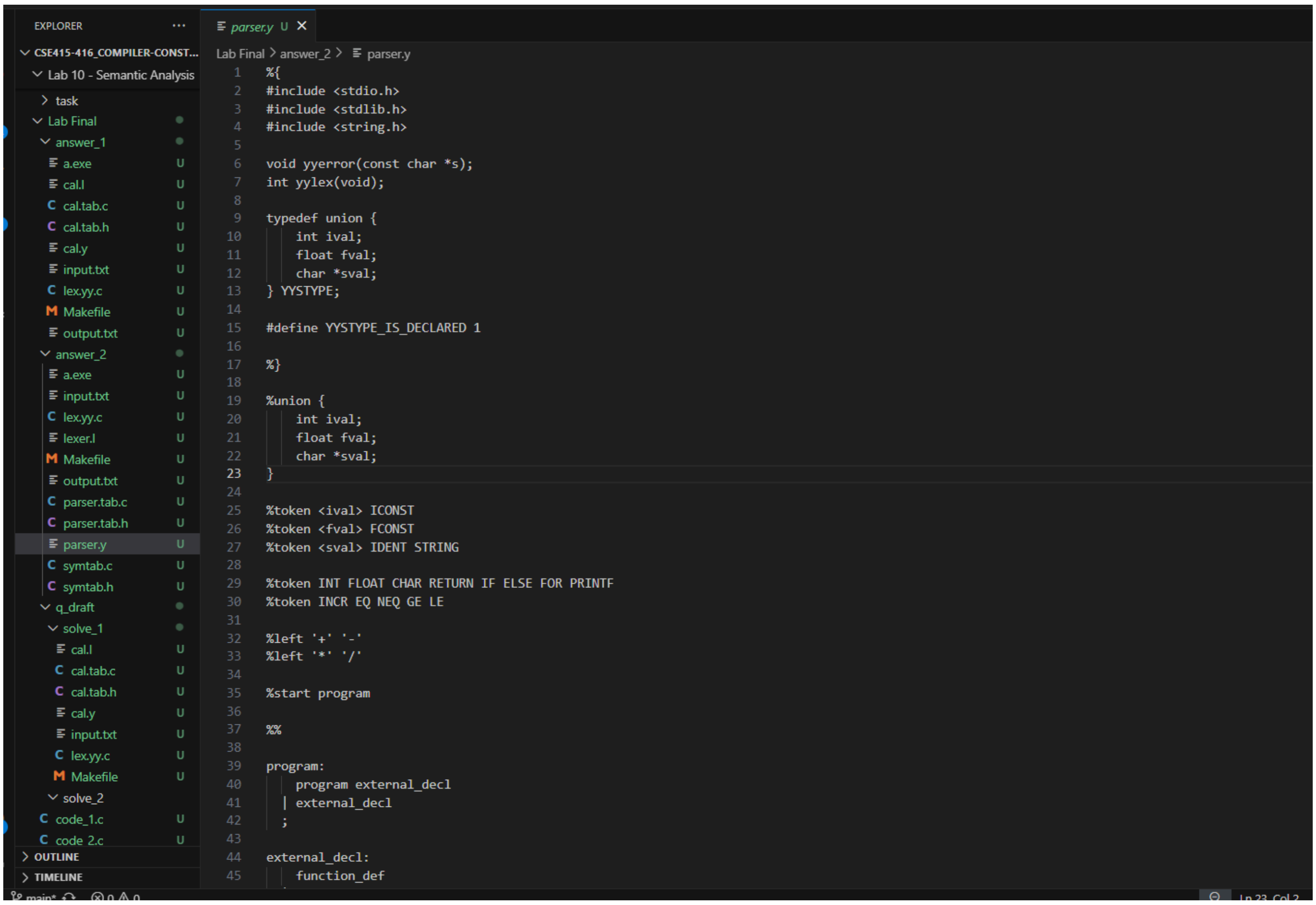


```
%option noyywrap

%{
#include "parser.tab.h"
#include <string.h>
%}

digit       [0-9]
alpha       [a-zA-Z_]
alphanum    [a-zA-Z0-9_]

%%

"int"           { return INT; }
"float"         { return FLOAT; }
"char"          { return CHAR; }
"return"        { return RETURN; }
"if"            { return IF; }
"else"          { return ELSE; }
"for"           { return FOR; }
"printf"        { return PRINTF; }

"=="            { return EQ; }
"!="            { return NEQ; }
">="            { return GE; }
"<="            { return LE; }
">"             { return '>'; }
"<"             { return '<'; }
"="             { return '='; }
"+"             { return '+'; }
"-"             { return '-'; }
"*"             { return '*'; }
"/"             { return '/'; }
"("             { return '('; }
")"             { return ')'; }
"{"             { return '{'; }
"}"             { return '}'; }
```

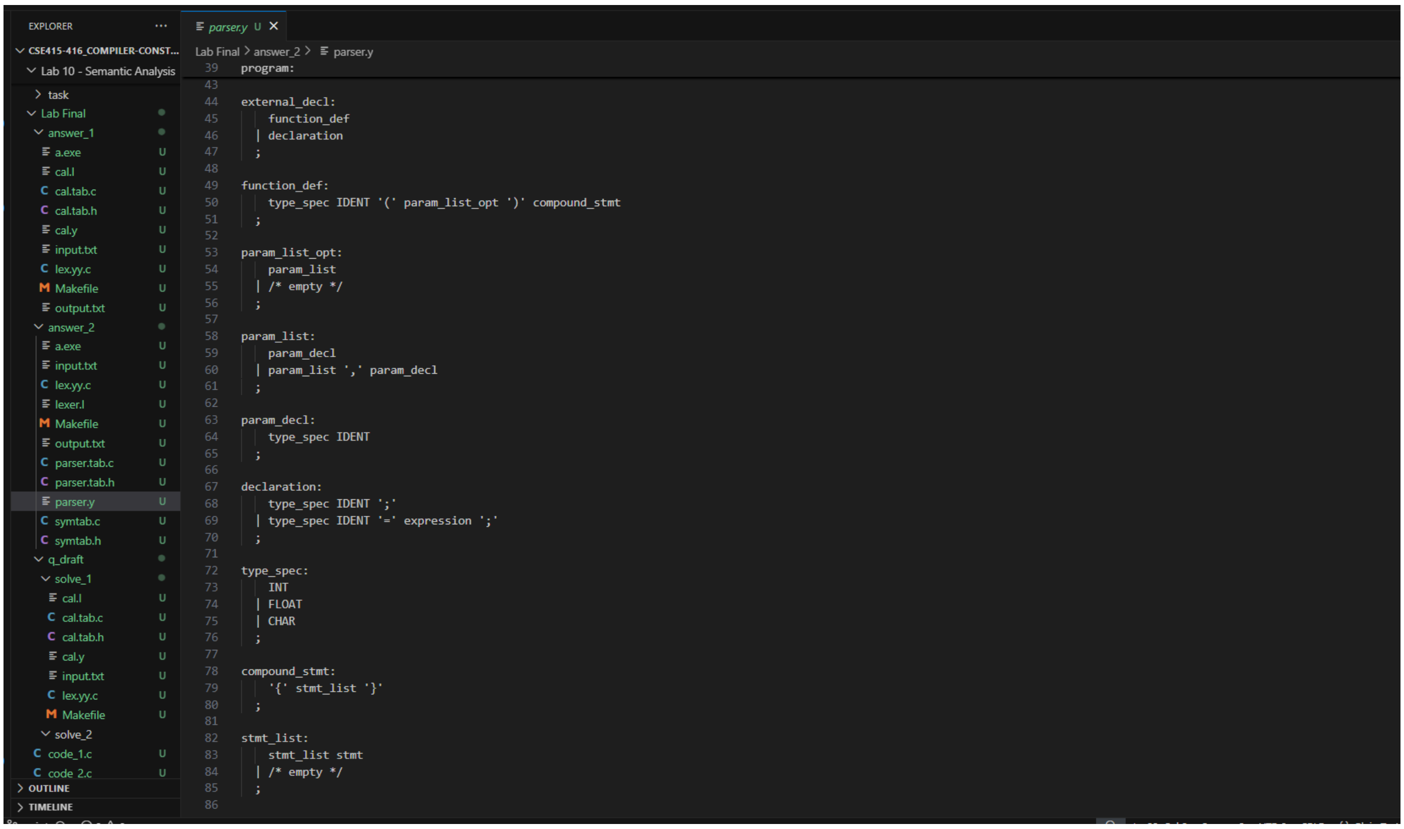


```
"=="            { return EQ; }
"!="            { return NEQ; }
">="            { return GE; }
"<="            { return LE; }
">"             { return '>'; }
"<"             { return '<'; }
"="             { return '='; }
"+"             { return '+'; }
"-"             { return '-'; }
"*"             { return '*'; }
"/"             { return '/'; }
"("             { return '('; }
")"             { return ')'; }
"{"             { return '{'; }
"}"             { return '}'; }
";"             { return ';'; }
","             { return ','; }
"++"            { return INCR; }

{digit}+        { yylval.ival = atoi(yytext); return ICONST; }
{digit}+"."{digit}+  { yylval.fval = atof(yytext); return FCONST; }

{alpha}{alphanum}* {
                    yylval.sval = strdup(yytext);
                    return IDENT;
                   }

\"([^\\\"]|\\.)*\" { yylval.sval = strdup(yytext); return STRING; }

[ \t\n\r]+      { /* skip whitespace */ }

.               { /* ignore unknown chars or error */ }

%%
```

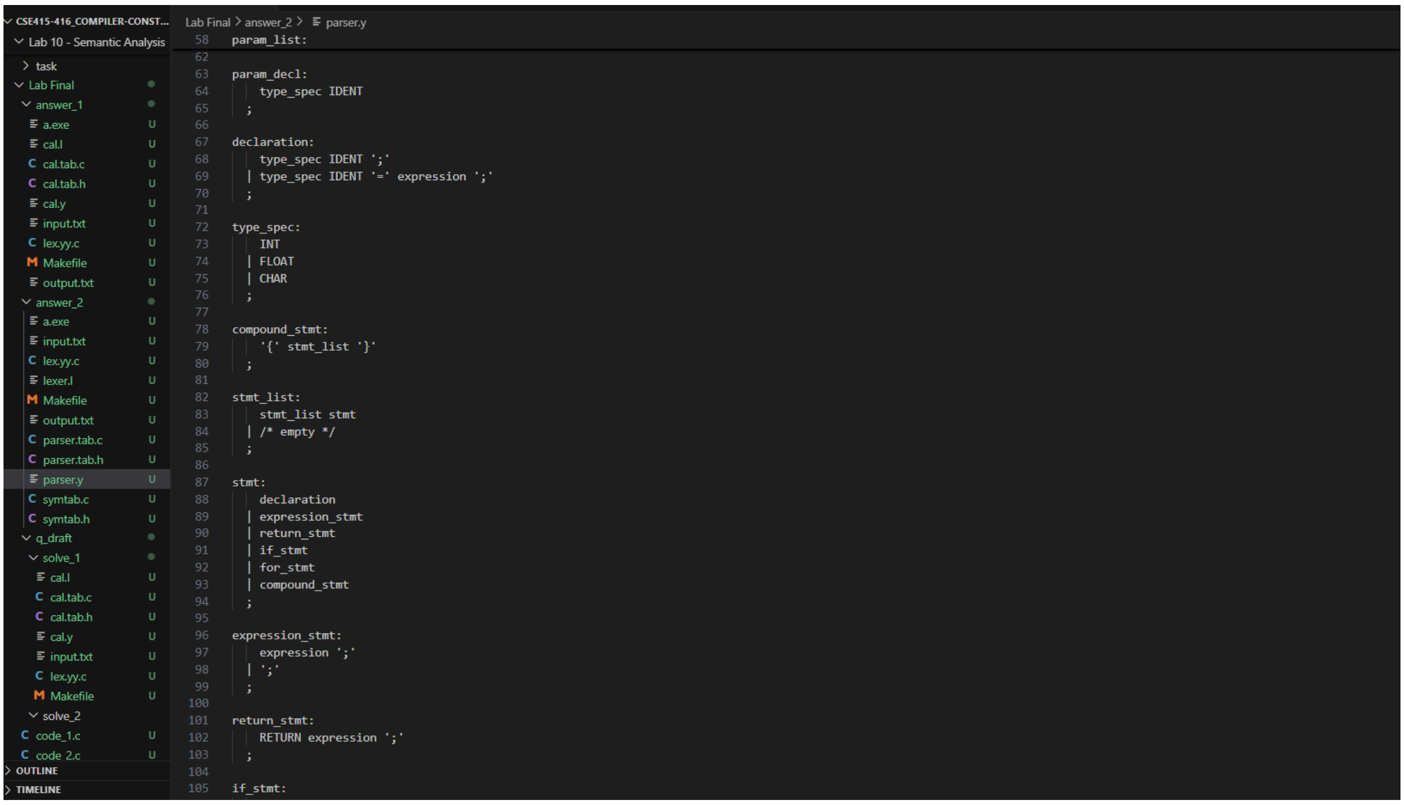
## parser.y



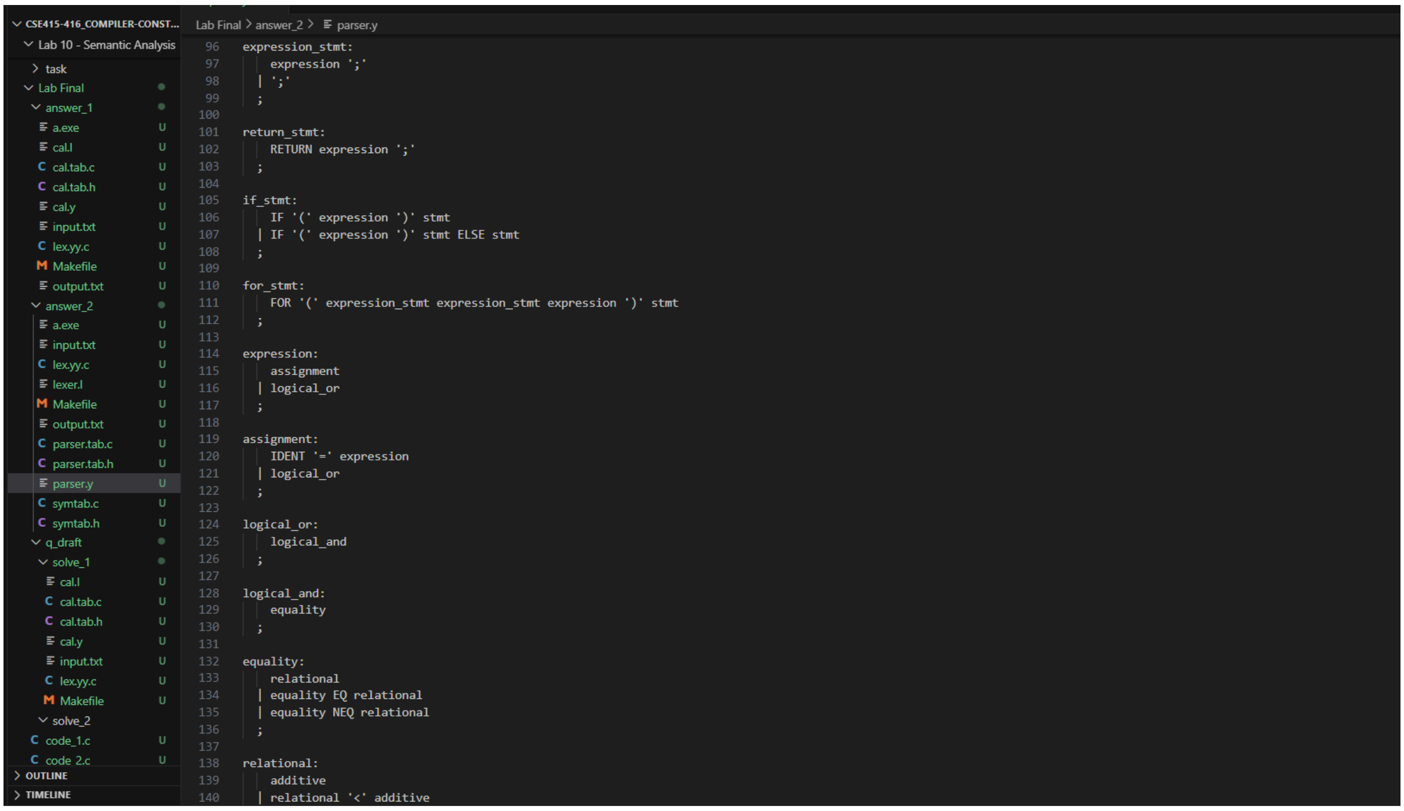
```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void yyerror(const char *s);
int yylex(void);

typedef union {
    int ival;
    float fval;
    char *sval;
} YYSTYPE;

#define YYSTYPE_IS_DECLARED 1

%}

%union {
    int ival;
    float fval;
    char *sval;
}

%token <ival> ICONST
%token <fval> FCONST
%token <sval> IDENT STRING

%token INT FLOAT CHAR RETURN IF ELSE FOR PRINTF
%token INCR EQ NEQ GE LE

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

%start program

%%

program:
    program external_decl
  | external_decl
  ;

external_decl:
    function_def
```



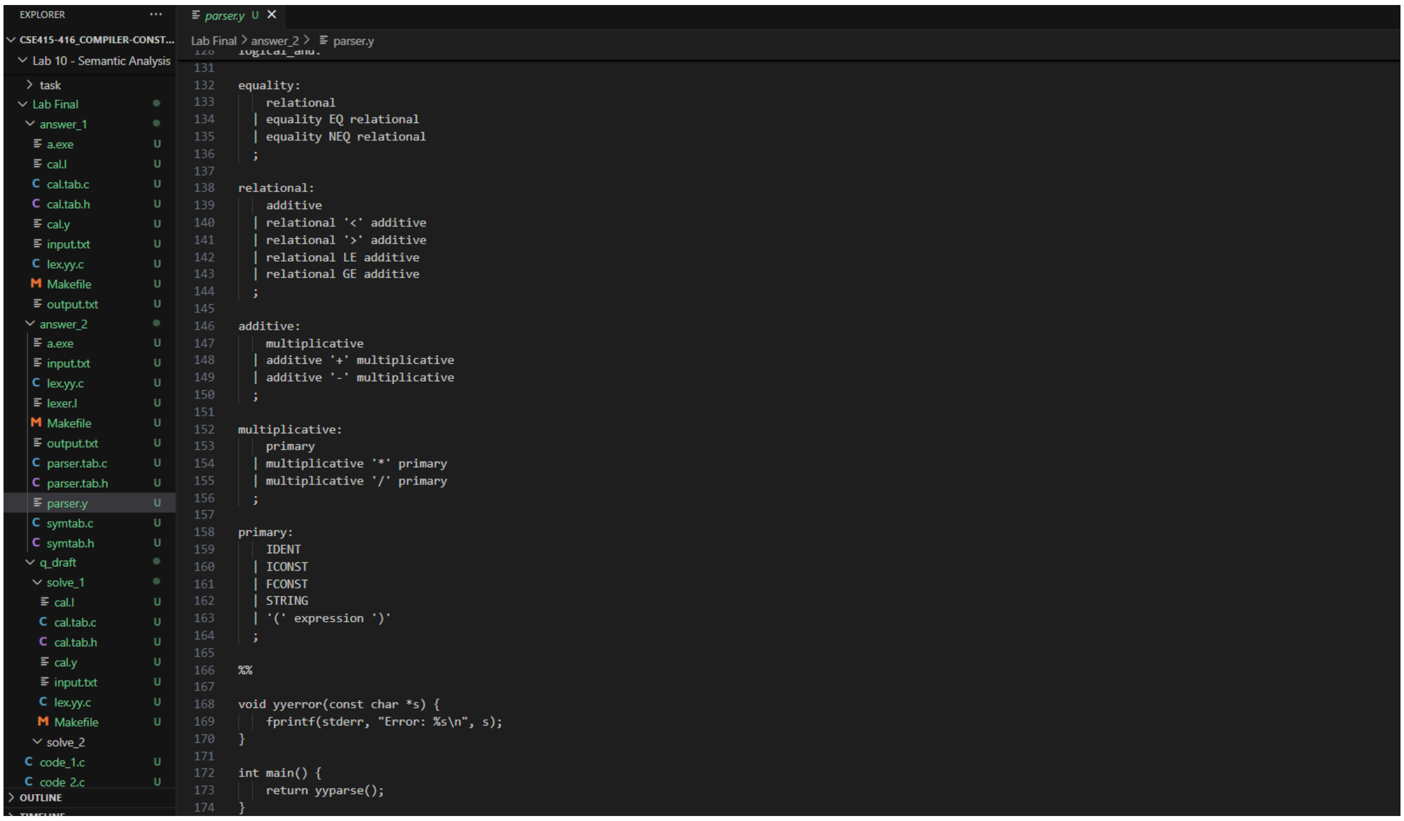
```
program:

external_decl:
    function_def
    | declaration
    ;

function_def:
    type_spec IDENT '(' param_list_opt ')' compound_stmt
    ;

param_list_opt:
    param_list
    | /* empty */
    ;

param_list:
    param_decl
    | param_list ',' param_decl
    ;

param_decl:
    type_spec IDENT
    ;

declaration:
    type_spec IDENT ';'
    | type_spec IDENT '=' expression ';'
    ;

type_spec:
    INT
    | FLOAT
    | CHAR
    ;

compound_stmt:
    '{' stmt_list '}'
    ;

stmt_list:
    stmt_list stmt
    | /* empty */
    ;

```



```
param_list:

param_decl:
    type_spec IDENT
    ;

declaration:
    type_spec IDENT ';'
    | type_spec IDENT '=' expression ';'
    ;

type_spec:
    INT
    | FLOAT
    | CHAR
    ;

compound_stmt:
    '{' stmt_list '}'
    ;

stmt_list:
    stmt_list stmt
    | /* empty */
    ;

stmt:
    declaration
    | expression_stmt
    | return_stmt
    | if_stmt
    | for_stmt
    | compound_stmt
    ;

expression_stmt:
    expression ';'
    | ';'
    ;

return_stmt:
    RETURN expression ';'
    ;

if_stmt:
```

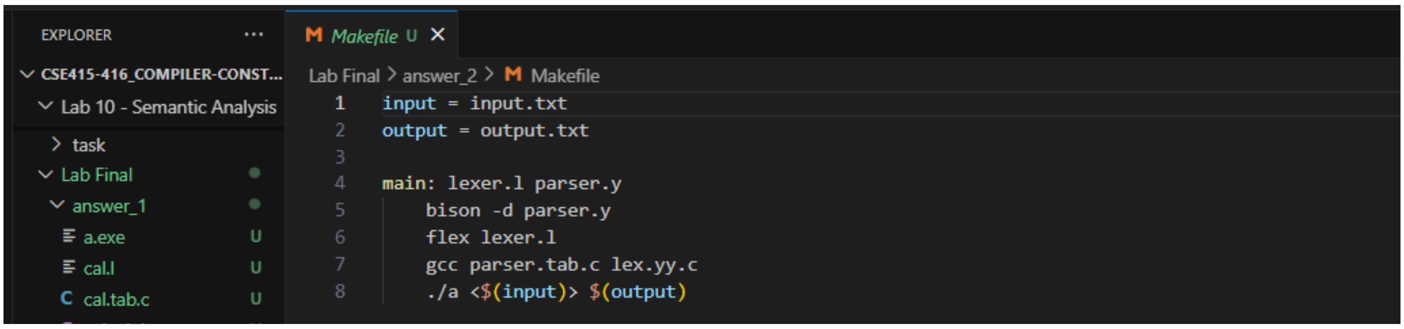


```
expression_stmt:
    expression ';'
  | ';'
  ;

return_stmt:
    RETURN expression ';'
  ;

if_stmt:
    IF '(' expression ')' stmt
  | IF '(' expression ')' stmt ELSE stmt
  ;

for_stmt:
    FOR '(' expression_stmt expression_stmt expression ')' stmt
  ;

expression:
    assignment
  | logical_or
  ;

assignment:
    IDENT '=' expression
  | logical_or
  ;

logical_or:
    logical_and
  ;

logical_and:
    equality
  ;

equality:
    relational
  | equality EQ relational
  | equality NEQ relational
  ;

relational:
    additive
  | relational '<' additive
```



```

equality:
    relational
  | equality EQ relational
  | equality NEQ relational
  ;

relational:
    additive
  | relational '<' additive
  | relational '>' additive
  | relational LE additive
  | relational GE additive
  ;

additive:
    multiplicative
  | additive '+' multiplicative
  | additive '-' multiplicative
  ;

multiplicative:
    primary
  | multiplicative '*' primary
  | multiplicative '/' primary
  ;

primary:
    IDENT
  | ICONST
  | FCONST
  | STRING
  | '(' expression ')'
  ;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main() {
    return yyparse();
}
```

## Makefile


EXPLORER
Makefile U
CSE415-416_COMPILER-CONST...
Lab 10 - Semantic Analysis
task
Lab Final
answer_1
a.exe
cal.l
cal.tab.c
Lab Final > answer_2 > Makefile

```
input = input.txt
output = output.txt

main: lexer.l parser.y
	bison -d parser.y
	flex lexer.l
	gcc parser.tab.c lex.yy.c
	./a <$(input)> $(output)
```