

**Set-A**

**Ans. to the Q. No. (1)**

**input.txt**

EXPLORER

...  
CSE415-416\_COMPILER-CONST...  
Lab Final  
answer\_1  
a.cac  
cal.l  
cal.tab.c  
cal.tab.h  
cal.y  
input.txt  
lex.yy.c  
Makefile  
output.txt  
answer\_2  
a.exe  
input.txt  
lex.yy.c  
lexer.l  
Makefile  
output.txt  
parser.tab.c  
parser.tab.h  
parsery  
syntab.c  
syntab.h  
q\_draft  
solve\_1  
cal.l  
cal.tab.c  
cal.tab.h

input.txt U X  
Lab Final > answer\_1 >   
1 #include <stdio.h>  
2  
3 int main() {  
4 int choice, num;  
5 char \*menu[] = {"Check even/odd", "Print square", "Print cube"};  
6  
7 for (int i = 0; i < 3; i++)  
8 printf("%d. %s\n", i + 1, menu[i]);  
9  
10 printf("Enter your choice: ");  
11 scanf("%d", &choice);  
12  
13 if (choice >= 1 && choice <= 3) {  
14 printf("Enter a number: ");  
15 scanf("%d", &num);  
16  
17 for (int i = choice; i <= choice; i++) {  
18 if (i == 1)  
19 printf("%d is %s\n", num, num % 2 ? "odd" : "even");  
20 else if (i == 2)  
21 printf("Square of %d is %d\n", num, num \* num);  
22 else  
23 printf("Cube of %d is %d\n", num, num \* num \* num);  
24 }  
25 } else {  
26 printf("Invalid choice!\n");  
27 }  
28  
29 return 0;  
30 }  
31

**cal.l**

```
EXPLORER  ...  ≡ cal.l  U X
CSE415-416 COMPILER-CONST...  Lab Final > answer_1 > ≡ cal.l
└─ Lab Final  ●
└─ answer_1  ●
└─ cal.l  ●
  ≡ cal.l  U
  C cal.tab.c  U
  C cal.tab.h  U
  ≡ cal.y  U
  ≡ input.txt  U
  C lex.yy.c  U
  M Makefile  U
  ≡ output.txt  U
└─ answer_2  ●
  ≡ a.exe  U
  ≡ input.txt  U
  C lex.yy.c  U
  ≡ lexer.l  U
  M Makefile  U
  ≡ output.txt  U
  C parser.tab.c  U
  C parser.tab.h  U
  ≡ parser.y  U
  C symtab.c  U
  C symtab.h  U
└─ q_draft  ●
└─ solve_1  ●
  ≡ cal.l  U
  C cal.tab.c  U
  C cal.tab.h  U
  ≡ cal.y  U
  ≡ input.txt  U
  C lex.yy.c  U
  M Makefile  U
└─ solve_2  ●
  C code_1.c  U
  C code_2.c  U
└─ Lab Resources  ●
└─ Lab Code  ●
└─ Lab Final  ●
OUTLINE  44  %%
```

```
1  %option noyywrap
2  %{
3  #include "cal.tab.h"
4  #include <string.h>
5  #ifdef strdup
6  #define strdup _strdup
7  #endif
8  %}
9
10 digit  [0-9]
11 id      [a-zA-Z_][a-zA-Z0-9_]*
12 str     \"([^\"]|\\\"\\.)*\"
13
14 %%%
15 "#include"      { return INCLUDE; }
16 "<stdio.h>\""    { return HEADER; }
17 "int"           { return INT; }
18 "char"          { return CHAR; }
19 "for"           { return FOR; }
20 "if"            { return IF; }
21 "else"          { return ELSE; }
22 "return"        { return RETURN; }
23 "printf"        { return PRINTF; }
24 "scanf"         { return SCANF; }
25
26 "++"            { return INCR; }
27 "=="           { return EQ; }
28 "!="           { return NEQ; }
29 ">="           { return GE; }
30 "<="           { return LE; }
31 "&&"          { return AND; }
32 "||"           { return OR; }
33 "?"            { return QMARK; }
34 ":"            { return COLON; }
35
36 {str}           { yyval.str = strdup(yytext); return STRING; }
37 {id}            { yyval.str = strdup(yytext); return IDENT; }
38 {digit}+        { yyval.num = atoi(yytext); return NUMBER; }
39
40 .               { return yytext[0]; }
41
42 [ \t\n\r]+      { /* skip whitespace */ }
43
44 %%%
```

```

EXPLORER
CSE415-416_COMPILER-CONST...
  Lab 10 - Semantic Analysis
    task
    Lab Final
      answer_1
        a.exe
        cal
        cal.tab.c
        cal.tab.h
        cal.y
        input.txt
        lex.yy.c
        Makefile
        output.txt
      answer_2
        a.exe
        input.txt
        lex.yy.c
        lexer.l
        Makefile
        output.txt
        parser.tab.c
        parser.tab.h
        parser.y
        symtab.c
        symtab.h
      q_draft
      solve_1
        cal
        cal.tab.c
        cal.tab.h
        cal.y
        input.txt
        lex.yy.c
        Makefile
      solve_2
        code_1.c
        code_2.c
    OUTLINE
    TIMELINE

Lab Final > answer_1 > cal.y
1  %{
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  int yylex();
7  void yyerror(const char *s);
8  %}
9
10 %union {
11     char *str;
12     int num;
13 }
14
15 %token <str> IDENT STRING
16 %token <num> NUMBER
17 %token INCLUDE HEADER INT CHAR FOR IF ELSE RETURN PRINTF SCANF
18 %token EQ NEQ GE LE AND OR INCR QMARK COLON
19
20 %right QMARK COLON
21
22 %start program
23
24 %%
25
26 program
27     : INCLUDE HEADER main_func
28     ;
29
30 main_func
31     : INT IDENT '(' ')' compound_stmt
32     ;
33
34 compound_stmt
35     : '{' statement_list '}'
36     ;
37
38 statement_list
39     : /* empty */
40     | statement_list statement
41     ;
42
43 statement
44     : declaration
45     | for_loop

```

```

File Edit Selection View Go Run Terminal Help
CSE415-416_Compiler-Construction

EXPLORER
CSE415-416_COMPILER-CONST...
  Lab 10 - Semantic Analysis
    task
    Lab Final
      answer_1
        a.exe
        cal
        cal.tab.c
        cal.tab.h
        cal.y
        input.txt
        lex.yy.c
        Makefile
        output.txt
      answer_2
        a.exe
        input.txt
        lex.yy.c
        lexer.l
        Makefile
        output.txt
        parser.tab.c
        parser.tab.h
        parser.y
        symtab.c
        symtab.h
      q_draft
      solve_1
        cal
        cal.tab.c
        cal.tab.h
        cal.y
        input.txt
        lex.yy.c
        Makefile
      solve_2
        code_1.c
        code_2.c
    OUTLINE
    TIMELINE

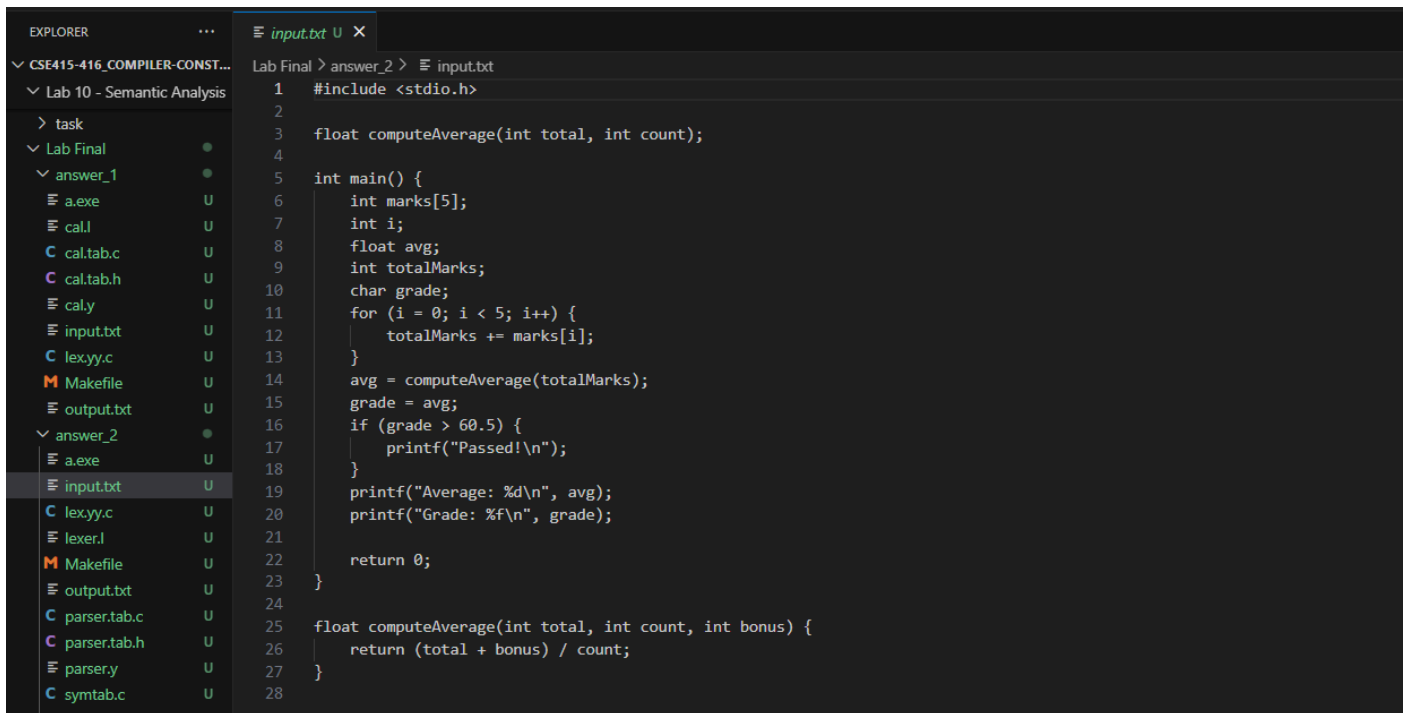
Lab Final > answer_1 > cal.y
42
43 statement
44     : declaration
45     | for_loop
46     | if_block
47     | printf_stmt
48     | scanf_stmt
49     | RETURN expression ';'
50     | compound_stmt
51     | ';'
52     ;
53
54 declaration
55     : INT var_list ';'
56     | CHAR pointer_array_decl ';'
57     ;
58
59 pointer_array_decl
60     : '*' IDENT '[' ']' '=' '{' string_list '}'
61     | '*' IDENT '[' NUMBER ']' '=' '{' string_list '}'
62     ;
63
64 var_list
65     : IDENT
66     | IDENT ',' var_list
67     ;
68
69 string_list
70     : STRING
71     | STRING ',' string_list
72     ;
73
74 for_loop
75     : FOR '(' INT IDENT '=' NUMBER ';' condition ';' IDENT INCR ')' statement
76     | FOR '(' expression ';' condition ';' expression ')' statement
77     ;
78
79 condition
80     : expression relop expression
81     | expression
82     ;
83
84 relop
85     : EQ | NEQ | GE | LE | '>' | '<'
86     ;

```



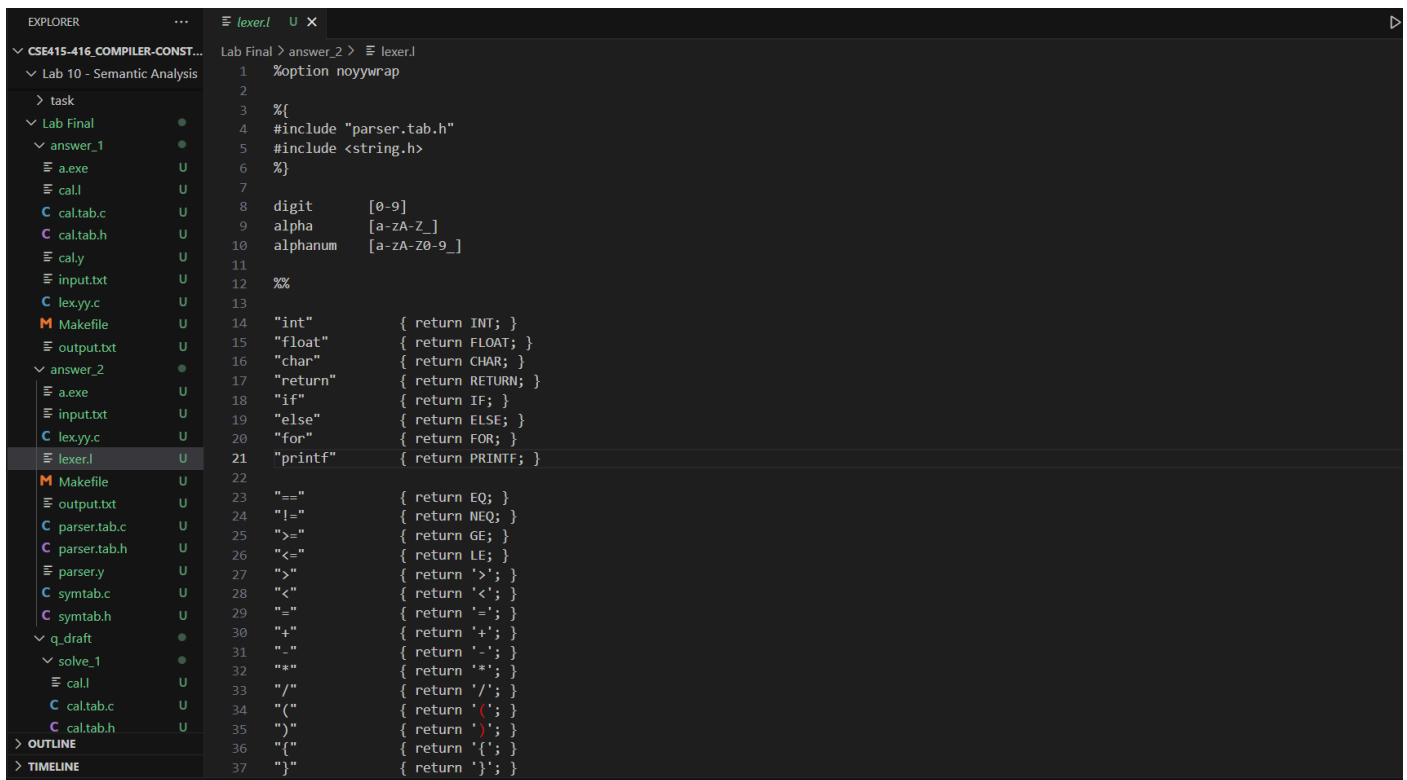
## Ans. to the Q. No. (2)

input.txt



```
1 #include <stdio.h>
2
3 float computeAverage(int total, int count);
4
5 int main() {
6     int marks[5];
7     int i;
8     float avg;
9     int totalMarks;
10    char grade;
11    for (i = 0; i < 5; i++) {
12        totalMarks += marks[i];
13    }
14    avg = computeAverage(totalMarks);
15    grade = avg;
16    if (grade > 60.5) {
17        printf("Passed!\n");
18    }
19    printf("Average: %d\n", avg);
20    printf("Grade: %f\n", grade);
21
22    return 0;
23 }
24
25 float computeAverage(int total, int count, int bonus) {
26     return (total + bonus) / count;
27 }
28
```

lexer.l



```
1 %option noyywrap
2
3 %{
4     #include "parser.tab.h"
5     #include <string.h>
6 %}
7
8 digit      [0-9]
9 alpha      [a-zA-Z_]
10 alphanum   [a-zA-Z0-9_]
11
12 %%
13
14 "int"      { return INT; }
15 "float"    { return FLOAT; }
16 "char"     { return CHAR; }
17 "return"   { return RETURN; }
18 "if"       { return IF; }
19 "else"     { return ELSE; }
20 "for"      { return FOR; }
21 "printf"   { return PRINTF; }
22
23 "=="       { return EQ; }
24 "!="       { return NEQ; }
25 ">="        { return GE; }
26 "<="        { return LE; }
27 ">"         { return '>'; }
28 "<"         { return '<'; }
29 "="        { return '='; }
30 "+"        { return '+'; }
31 "-"        { return '-'; }
32 "*"        { return '*'; }
33 "/"        { return '/'; }
34 "("        { return '('; }
35 ")"        { return ')'; }
36 "{"        { return '{'; }
37 "}"        { return '}'; }
```

```
22
23 "=="      { return EQ; }
24 "!="      { return NEQ; }
25 ">="      { return GE; }
26 "<="      { return LE; }
27 ">"      { return '>'; }
28 "<"      { return '<'; }
29 "="      { return '='; }
30 "+"      { return '+'; }
31 "-"      { return '-'; }
32 "*"      { return '*'; }
33 "/"      { return '/'; }
34 "("      { return '('; }
35 ")"      { return ')'; }
36 "{"      { return '{'; }
37 "}"      { return '}'; }
38 ";"      { return ';'; }
39 ","      { return ','; }
40 "++"      { return INCR; }
41
42 {digit}+   { yylval.ival = atoi(yytext); return ICONST; }
43 {digit}+"."{digit}+ { yylval.fval = atof(yytext); return FCONST; }
44
45 {alpha}{alphanum}* {
46     yylval.sval = strdup(yytext);
47     return IDENT;
48 }
49
50 \"([^\\""]|\\.)*\\" { yylval.sval = strdup(yytext); return STRING; }
51
52 [ \t\n\r]+      { /* skip whitespace */ }
53
54 .               { /* ignore unknown chars or error */ }
55
56 %%
57
```

## parser.y

```
1 %{
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 void yyerror(const char *s);
7 int yylex(void);
8
9 typedef union {
10     int ival;
11     float fval;
12     char *sval;
13 } YYSTYPE;
14
15 #define YYSTYPE_IS_DECLARED 1
16
17 %}
18
19 %union {
20     int ival;
21     float fval;
22     char *sval;
23 }
24
25 %token <ival> ICONST
26 %token <fval> FCONST
27 %token <sval> IDENT STRING
28
29 %token INT FLOAT CHAR RETURN IF ELSE FOR PRINTF
30 %token INCR EQ NEQ GE LE
31
32 %left '+' '-'
33 %left '*' '/'
34
35 %start program
36
37 %%
38
39 program:
40     program external_decl
41     | external_decl
42     ;
43
44 external_decl:
45     function_def
46
```

EXPLORER

...

parser.y

Lab Final > answer\_2 > parser.y

Lab 10 - Semantic Analysis

> task

Lab Final

answer\_1

a.exe

call

caltab.c

caltab.h

caly

input.txt

lex.yy.c

Makefile

output.txt

answer\_2

a.exe

input.txt

lex.yy.c

lexer.l

Makefile

output.txt

parser.tab.c

parser.tab.h

parser.y

syntab.c

syntab.h

q\_draft

solve\_1

call

caltab.c

caltab.h

caly

input.txt

lex.yy.c

Makefile

solve\_2

code\_1.c

code\_2.c

OUTLINE

TIMELINE

39  
program:  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
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 \*/  
    ;  
86

CSE415-416\_COMPILER\_CONST...

Lab Final > answer\_2 > parser.y

Lab 10 - Semantic Analysis

> task

Lab Final

answer\_1

a.exe

call

caltab.c

caltab.h

caly

input.txt

lex.yy.c

Makefile

output.txt

answer\_2

a.exe

input.txt

lex.yy.c

lexer.l

Makefile

output.txt

parser.tab.c

parser.tab.h

parser.y

syntab.c

syntab.h

q\_draft

solve\_1

call

caltab.c

caltab.h

caly

input.txt

lex.yy.c

Makefile

solve\_2

code\_1.c

code\_2.c

OUTLINE

TIMELINE

58  
param\_list:  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
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:  
105

```
Lab Final > answer_2 > parser.y
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
```

```
Lab Final > answer_2 > parser.y
equality:
| relational
| equality EQ relational
| equality NEQ relational
;

relational:
| additive
| relational '<' additive
| relational '>' additive
| relational '!' 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

```
Lab Final > answer_2 > Makefile
1 input = input.txt
2 output = output.txt
3
4 main: lexer.l parser.y
5     bison -d parser.y
6     flex lexer.l
7     gcc parser.tab.c lex.yy.c
8     ./a <$(input)> $(output)
```