Date : 05/03/2022

CSPC62 : COMPILER DESIGN
LAB-4

Roll no. : **106119100**

Name : **Rajneesh Pandey**

Section : **CSE-B**

1. Write a code for syntax analysis of if, if-else and nested if conditional constructs in C.

Code:

lexer.l

```
%option yylineno
%option noyywrap
%{
#include "parser.tab.h"
extern int yylval;
%}
NUMBER ([0-9]+(".")?([0-9])*)
IDENTIFIER ([a-zA-z_][a-zA-z_0-9]*)
%%
[\t ] /* ignore whitespaces */ ;
if {return IF;}
else {return ELSE;}
"&&" {return AND;}
"||" {return OR;}
"<=" {return LE;}
">=" {return GE;}
">" {return GT;}
"<" {return LT;}
"!=" {return NE;}
"++" {return INC;}
"--" {return DEC;}
"==" {return EQ;}
{NUMBER} {return NUM;}
{IDENTIFIER} {return ID;}
. {return yytext[0];}
\n {yylval = yylineno;}
\n\n {return 0;}
%%
```

parser.y

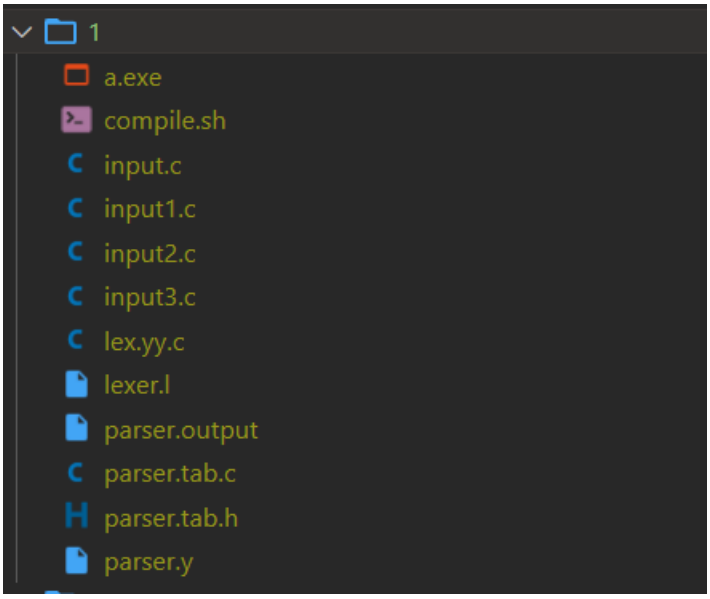
```
%{
#include<stdio.h>
#include<stdlib.h>
int yylex(void);
int yyerror(const char *s);
int success = 1;
%}
%token NUM ID LT GT EQ LE GE NE AND OR INC DEC END
%left '+' '-'
%left '*''/'
%right '^'
%right '='
%nonassoc UMINUS
%nonassoc IF
%nonassoc ELSE
%left GE NE LT GT LE EQ
%left AND OR

%%
S : IF '(' F ')' '{' S '}' %prec IF
 | IF '(' F ')' '{' S '}' ELSE '{' S '}'
 | E ';'
 | E ';' S
 ;
F : C LO C
 | C
 ;
LO : AND
 | OR
 ;
C : E RELOP E
 | E
```

```
 ;
E : ID '=' E
 | E '+' E
 | E '-' E
 | E '*' E
 | E '/' E
 | E '^' E
 | '(' E ')'
 | '-' E %prec UMINUS
 | ID
 | NUM
 | ID INC
 | ID DEC
 ;
RELOP :LT
 | GT
 | EQ
 | LE
 | GE
 | NE
 ;
%%


int main (void)
{
    yyparse();
    if(success)
    printf("Result of input............. \n");

    printf("Parsing Successful....coditions detected!\n");
    return 0;
}
```

```
int yyerror(const char *msg)
{
    printf("Parsing Failed.\n");
    success = 0;
    return 0;
}
```

Compile.sh

```
flex lexer.l
bison -vd parser.y
gcc lex.yy.c parser.tab.c -lm
./a.exe<input.c
./a.exe<input1.c
./a.exe<input2.c
./a.exe<input3.c
```



Input:

Terminal Help — input3.c - Lab4 - Visual Studio Code

input.c 1, U

```c
if(i+j){

}
else{

}
```

input1.c 1, U

```c
if(y > a)
{
    a = a + 1;
}
```

input2.c 1, U

```c
if (b < 5)
{
    if (b > 5)
    {
        c= c + 1;
    }
    else
    {
        b = b - 10;
    }
}
```

input3.c 1, U

```c
if (c > 3)
{
    a = 2 + 3;
}
else
{
    a = 9 * 5 + 4;
}
```

Output:

Terminal Help — Lab4 - Visual Studio Code

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE GITLENS

```
rajne (main *) Lab4
$ cd 1
rajne (main *) 1
$ sh compile.sh
Parsing Failed.
Parsing Successful....coditions detected!
Result of input..............
Parsing Successful....coditions detected!
Result of input..............
Parsing Successful....coditions detected!
Result of input..............
Parsing Successful....coditions detected!
rajne (main *) 1
$
```

2. Write a code for syntax analysis of while loop constructs in C.

Code:

lexer.l

```
%option yylineno
%{
#include "parser.tab.h"
extern int yylval;
%}
NUMBER ([0-9]+(".")?([0-9])*)
IDENTIFIER ([a-zA-z_][a-zA-z_0-9]*)
%%
[\t ] /* ignore whitespaces */ ;
while {return WHILE;}
{NUMBER} {return NUM;}
{IDENTIFIER} {return ID;}
"<=" {return LE;}
">=" {return GE;}
"==" {return EQ;}
"!=" {return NE;}
"||" {return OR;}
"&&" {return AND;}
. {return yytext[0];}
\n {yylval = yylineno;}
\n\n {return 0;}
%%
int yywrap() {
return 1;
}
```

parser.y

```
%{
#include<stdio.h>
#include<stdlib.h>
int yylex(void);
int yyerror(const char *s);
int success = 1;
%}
%token ID NUM WHILE LE GE EQ NE OR AND
%right '='
%left OR AND
%left '>' '<' LE GE EQ NE
%left '+' '-'
%left '*' '/'
%right UMINUS
%left '!'
%%
S : WHILE '(' E2 ')' DEF
 ;
DEF : '{' BODY '}'
 | E';'
 | S
 ;
BODY : BODY BODY
 | E ';'
 | S
 |
 ;
E : ID '=' E
 | E '+' E
 | E '-' E
 | E '*' E
```

```
 | E '/' E
 | E '<' E
 | E '>' E
 | E LE E
 | E GE E
 | E EQ E
 | E NE E
 | E OR E
 | E AND E
 | E '+' '+'
 | E '-' '-'
 | ID
 | NUM
 ;
E2 : E'<'E
 | E'>'E
 | E LE E
 | E GE E
 | E EQ E
 | E NE E
 | E OR E
 | E AND E
 ;
%%
int main (void)
{
yyparse();
if(success)
 printf("Result of input............... \n");

printf("Parsing Successful....WHILE loop!\n");
return 0;
}
```

```
int yyerror(const char *msg)
{
printf("Parsing Failed\n");
 success = 0;
return 0;
}
```

Compile.sh

```
flex lexer.l
bison -vd parser.y
gcc lex.yy.c parser.tab.c -lm
./a.exe<input.c
./a.exe<input1.c
./a.exe<input2.c
./a.exe<input3.c
```

Input:

Terminal Help input.c - Lab4 - Visual Studio Code

input.c 1, U

2 > input.c

```c
while(i<j){
    y = z;
}
```

input1.c 1, U

2 > input1.c

```c
while(i+j){
    i++;
    j--;
}
```

Output:

Terminal Help Lab4 - Visual Studio Code

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE GITLENS

```
rajne (main *) 2
$ cd ..
rajne (main *) Lab4
$ cd 2
rajne (main *) 2
$ sh compile.sh
parser.y: conflicts: 10 shift/reduce, 4 reduce/reduce
Result of input..............
Parsing Successful....WHILE loop!
Parsing Failed
Parsing Successful....WHILE loop!
rajne (main *) 2
$
```

3. Write a code for syntax analysis of for loop constructs in C

Code:

lexer.l

```
%{
    #include "parser.tab.h"
%}

num     [0-9]+
id      [a-zA-Z]+
binary  =|<|>|!=|<=|>=|==|&&|"||"|[+-/*]
unary   "++"|"--"

%%

"for"       { return FOR; }
{binary}    { return BINARY; }
{unary}     { return UNARY; }
{num}       { return NUMBER; }
{id}        { return ID; }
[ \n\t]     { ; }
.           {return *yytext; }

%%

int yywrap() {
    return 1;
}
```

parser.y

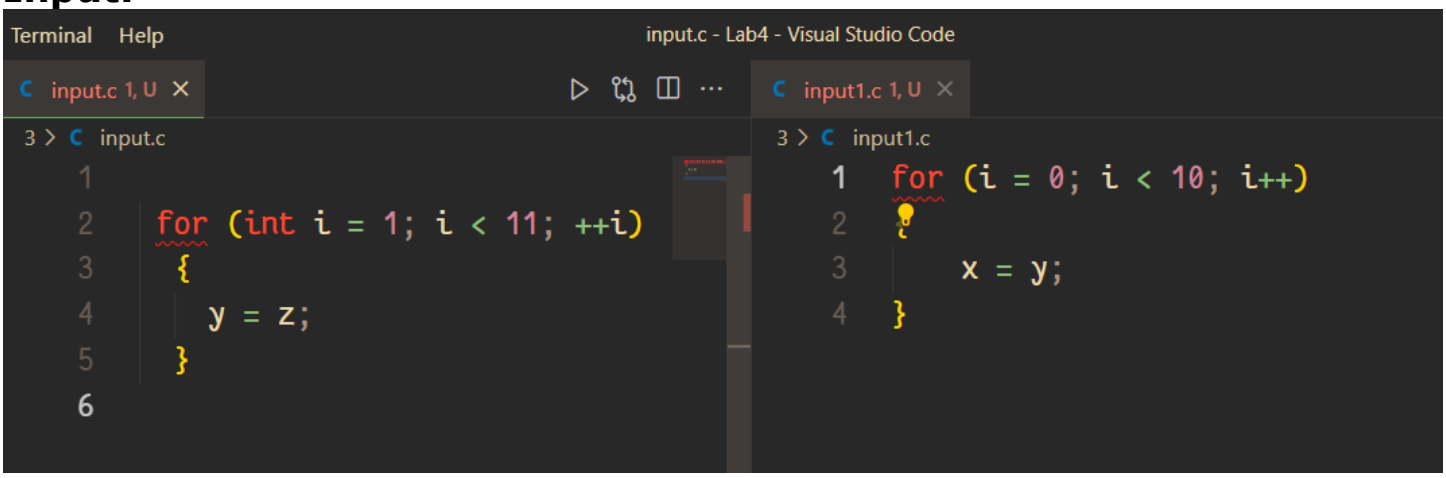
```
%{
#include<stdio.h>
void yyerror(const char *s);
int yylex();
%}

%token FOR ID NUMBER UNARY BINARY

%%

program: program loop body                    { printf("Loops and
more? \n"); }
| loop body                                   { printf("Loops! \n");
}
;

loop: FOR '(' for_statements ')'              { printf("For loop!
\n"); }
;

body: statement                               { printf("Nothin'
much... \n"); }
| '{' statements '}'                          { printf("Code Block!
\n"); }
| '{' loop '}' body                           { printf("Nested For?
\n"); }
;

for_statements: statement ';' statement ';' statement
;
```

```
statements: statements statement ';'          { printf("Lecture
begins! \n"); }
| statement ';'                               { printf("One liner!
\n"); }
;

statement: ID BINARY statement
| ID UNARY
| UNARY ID
| ID
| NUMBER
;

%%

int main() {
    printf("Result of input............. \n");
    yyparse();
}

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

Compile.sh

```
flex lexer.l
bison -vd parser.y
gcc lex.yy.c parser.tab.c -lm
./a.exe<input.c
./a.exe<input1.c
./a.exe<input2.c
./a.exe<input3.c
```

Input:



Output: