Compiler Design Hands On- 1

Name: Suchitra Shankar
Class: 6J
SRN: PES2UG23CS608

This assignment required the implementation of a lexical analyzer using Flex and a syntax analyzer using Bison to validate a simplified subset of the C language.

The lexer scans the input program and converts it into tokens such as keywords (`int`, `if`, `while`, etc.), identifiers, numbers, operators, and punctuation symbols. Regular expressions were used to define patterns for identifiers and numbers. Comments and whitespace are ignored. The lexer returns tokens to the parser and keeps track of line numbers for error reporting.

The parser checks whether the sequence of tokens follows the defined grammar. The grammar supports:

- Multiple variable declarations (e.g., `int a, b;`)
- Assignments
- `if` and `if-else` statements
- `do-while` loops
- Nested blocks using `{ }`
- Arithmetic, relational, and logical expressions

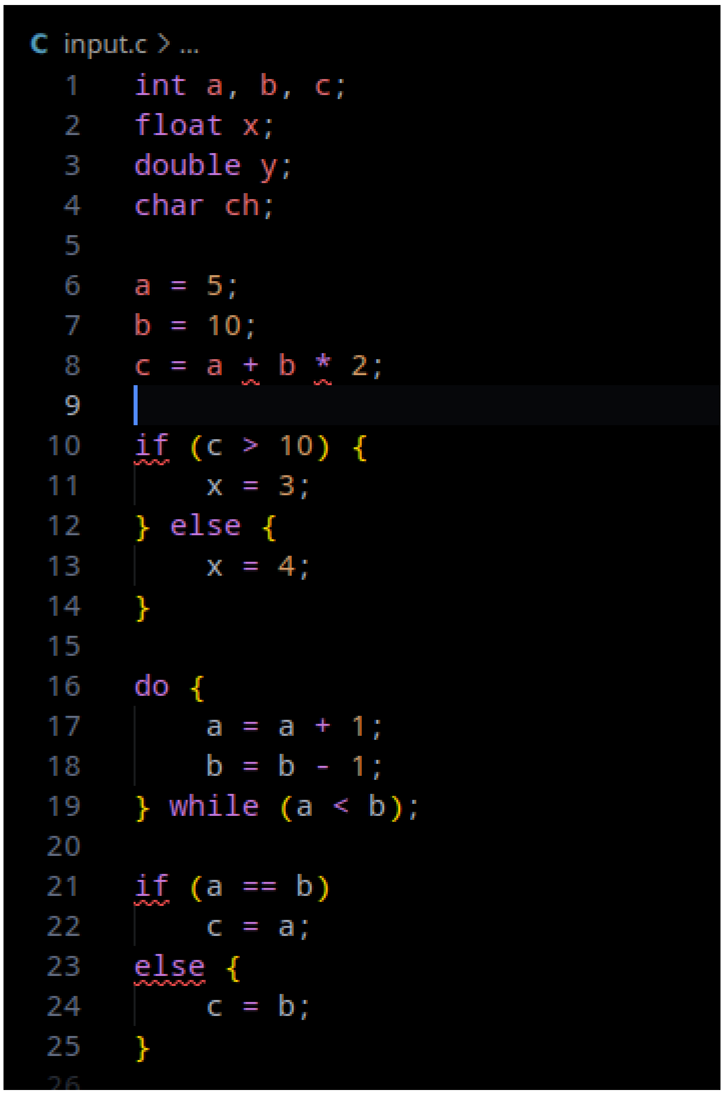
Operator precedence and associativity were defined in Bison to correctly parse expressions. The dangling-else problem was handled using precedence rules.

If the program follows the grammar, the output is:

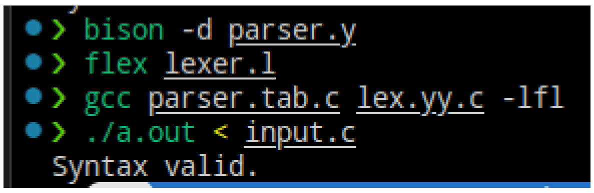
Syntax valid.

If there is an error, the parser reports the line number and the token where the syntax error occurred.

A Makefile was used to automate compilation using Flex and Bison.

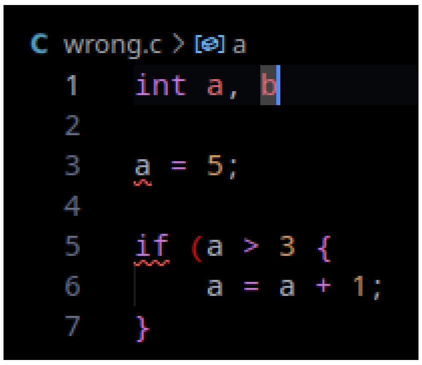


```
input.c > ...
int a, b, c;
float x;
double y;
char ch;

a = 5;
b = 10;
c = a + b * 2;

if (c > 10) {
    x = 3;
} else {
    x = 4;
}

do {
    a = a + 1;
    b = b - 1;
} while (a < b);

if (a == b)
    c = a;
else {
    c = b;
}
```



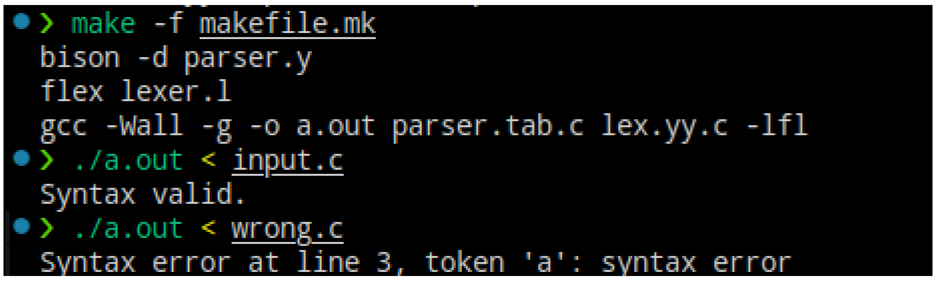
```
> bison -d parser.y
> flex lexer.l
> gcc parser.tab.c lex.yy.c -lfl
> ./a.out < input.c
Syntax valid.
```

Added a makefile:
Note the line number being tracked, for when errors arise
The line number cited is the first place where the compiler panics



```
C wrong.c > [⊘] a
1   int a, b
2
3   a = 5;
4
5   if (a > 3 {
6       a = a + 1;
7   }
```

the lack of semicolon on line 1, makes compiler detect it right after that, which is why the line 3 is cited for error instead of line 1



```
● ❯ make -f makefile.mk
  bison -d parser.y
  flex lexer.l
  gcc -Wall -g -o a.out parser.tab.c lex.yy.c -lfl
● ❯ ./a.out < input.c
  Syntax valid.
● ❯ ./a.out < wrong.c
  Syntax error at line 3, token 'a': syntax error
```

Input.c:

```c
int a, b, c;
float x;
double y;
char ch;

a = 5;
b = 10;
c = a + b * 2;

if (c > 10) {
   x = 3;
} else {
   x = 4;
}

do {
   a = a + 1;
   b = b - 1;
} while (a < b);

if (a == b)
   c = a;
else {
   c = b;
}
```

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

extern int yylex();
extern int yylineno;
extern char *yytext;

void yyerror(const char *s);
%}

/* ----------- TOKENS (must match lexer) ----------- */

%token INT FLOAT CHAR DOUBLE
%token IF ELSE DO WHILE
%token ID NUM
%token FOR MAIN INCLUDE INC DEC STRLITERAL HEADER
%token EQCOMP NOTEQ GREATEREQ LESSEREQ
%token ANDAND OROR

/* Precedence */
%left OROR
%left ANDAND
%left EQCOMP NOTEQ
%left '<' '>' GREATEREQ LESSEREQ
%left '+' '-'
%left '*' '/'
%right '='
%nonassoc IFX
%nonassoc ELSE

%start program

%%

program
        : stmt_list
        ;

stmt_list
        : stmt_list statement
        | /* empty */
        ;

statement
        : declaration ';'
        | assignment ';'
        | if_stmt
        | do_while_stmt
        | block
        ;

block
        : '{' stmt_list '}'
        ;

declaration
        : type declarator_list
        ;

type
        : INT
        | FLOAT
        | CHAR
        | DOUBLE
        ;

declarator_list
        : declarator_list ',' ID
        | ID
        ;

assignment
        : ID '=' expression
        ;

if_stmt
        : IF '(' expression ')' statement %prec IFX
        | IF '(' expression ')' statement ELSE statement
        ;

do_while_stmt
        : DO statement WHILE '(' expression ')' ';'
        ;

expression
        : expression '+' expression
        | expression '-' expression
        | expression '*' expression
        | expression '/' expression
        | expression '<' expression
        | expression '>' expression
        | expression GREATEREQ expression
        | expression LESSEREQ expression
        | expression EQCOMP expres%{
#include <stdio.h>
#include "parser.tab.h"
void yyerror(const char *s);
%}
%option noinput nounput

digit      [0-9]
letter     [a-zA-Z]
id         {letter}({letter}|{digit})*
number     {digit}+
header     \<{id}\.h\>
string     \"([^\"\n])*\"

%%


"int"        { return INT; }
"char"       { return CHAR; }
"float"      { return FLOAT; }
"double"     { return DOUBLE; }
"while"      { return WHILE; }
"for"        { return FOR; }
"do"         { return DO; }
"if"         { return IF; }
"else"       { return ELSE; }
"main"       { return MAIN; }
"#include"   { return INCLUDE; }

"//".*       ;


"/*"([^*]|\*+[^*/])*\*+"/" ;

"=="         { return EQCOMP; }
">="         { return GREATEREQ; }
"<="         { return LESSEREQ; }
"!="         { return NOTEQ; }
"&&"         { return ANDAND; }
"||"         { return OROR; }
"++"         { return INC; }
"--"         { return DEC; }
```

```
">"        { return *yytext; }
"<"        { return *yytext; }
"="        { return *yytext; }
"!"        { return *yytext; }
"+"        { return *yytext; }
"-"        { return *yytext; }
"*"        { return *yytext; }
"/"        { return *yytext; }


"("        { return *yytext; }
")"        { return *yytext; }
"["        { return *yytext; }
"]"        { return *yytext; }
"{"        { return *yytext; }
"}"        { return *yytext; }
";"        { return *yytext; }
","        { return *yytext; }

{string}     { return STRLITERAL; }
{number}     { return NUM; }


{header}     { return HEADER; }
{id}         { return ID; }


"\n"        { yylineno++; }
[ \t\r]+     ;


.           { return *yytext; }

%%


int yywrap() {
   return 1;
}

sion
       | expression NOTEQ expression
       | expression ANDAND expression
       | expression OROR expression
       | '(' expression ')'
       | ID
       | NUM
       ;

%%
```

```
void yyerror(const char *s)
{
   fprintf(stderr,
           "Syntax error at line %d, token '%s': %s\n",
           yylineno,
           yytext,
           s);
}

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