COMPILERS LAB - ASSIGNMENT 2
PART 2

GROUP 24
K.SOWMYA - 130101028
PREETHAM KAMIDI - 130101031
BHAVANA LAKHINENA - 130101043

CODE FOR TOKENIZATION:

```
/* Lexical analyser for a sample c like language. */
%option noyywrap

%Start    BL_CMNT

DIGIT    [0-9]
LETTER    [a-zA-Z]
LETTER_    [a-zA-Z_]
ID    {LETTER_}({LETTER_}|{DIGIT})*
MATH_EXP    "+"|"*"|"-"|"/"
UNARY    "++"|"--"|"!"
RELATIONAL    "=="|"!="|"<"|">"|"<="|">="
ASSIGN    "="
DT    "int"|"bool"
CONDITIONAL    "if"|"else"|"else if"|"switch"|"case"|"default"
LOOP    "do"|"while"|"for"
BRACKET    "{"|"}"|"("|")"|"["|"]"
IO    "scan"|"print"
DELIMITER ";"
BOOL "TRUE"|"FALSE"
NUMBER {DIGIT}+
KEYWORD "GLOBAL"|"void"|"RETURN"|"BREAK"|"CONTINUE"
COMMA    ","

%%
<INITIAL>"/*"    {BEGIN BL_CMNT;}
<BL_CMNT>"*/"    {BEGIN 0;}
<BL_CMNT>.    /* eat up the block comment characters */
<BL_CMNT>\n    /* eat up lines in block comments */
<INITIAL>"*/"    printf("Unmatched end of comment\n");
<INITIAL>{MATH_EXP}    {printf("<MATH_OPERATOR,'%s'>\n",yytext);}
<INITIAL>{UNARY}    {printf("<UNARY_OPERATOR,'%s'>\n",yytext);}
<INITIAL>{RELATIONAL}    {printf("<RELATIONAL_OPERATOR,'%s'>\n",yytext);}
<INITIAL>{ASSIGN}    {printf("<ASSIGN_OPERATOR,'%s'>\n",yytext);}
<INITIAL>{DT}    {printf("<DATATYPE,'%s'>\n",yytext);}
```

```
<INITIAL>{CONDITIONAL}    {printf("<CONDITIONAL,'%s'>\n",yytext);}
<INITIAL>{LOOP}    {printf("<LOOP,'%s'>\n",yytext);}
<INITIAL>{BRACKET}    {printf("<BRACKET,'%s'>\n",yytext);}
<INITIAL>{IO}    {printf("<IO,'%s'>\n",yytext);}
<INITIAL>{DELIMITER}    {printf("<DELIMITER,'%s'>\n",yytext);}
<INITIAL>{BOOL}    {printf("<BOOL,'%s'>\n",yytext);}
<INITIAL>{KEYWORD}    {printf("<KEYWORD,'%s'>\n",yytext);}
<INITIAL>{ID}    {printf("<IDENTIFER,'%s'>\n",yytext);}
<INITIAL>{NUMBER}    {printf("<CONSTANT,'%s'>\n",yytext);}
<INITIAL>{COMMA}    {printf("<COMMA,'%s'>\n",yytext);}
<INITIAL>"//"[^\n]*    /* eat up one line comments    */
<INITIAL>[ \t\n]+    /* eat up white spaces */
<INITIAL>.    printf("Invalid characters: %s\n",yytext);    /* All other erroneous characters */
%%
```

Language Used: Flex
To run:

```
        flex input_code.flex
        gcc lex.yy.c -lfl
        ./a.out input_file
```

DEFINED GRAMMAR:

```
definition_list -> definition_list definition | ϵ
definition -> variable_definition | function_definition
variable_definition -> datatype identifier_list;
function_defintiion -> function_datatype id(parameter_list) opt_stmts
identifier_list -> identifier_list,var | var
var -> assign_stmt | id
parameter_list -> parameter list,datatype id| datatype id
datatype -> int|bool
function_datetype -> void|int|bool
opt_stms -> {stmt_list}|stmt_list|ϵ
stmt_list -> stmt_list; stmt; | stmt;
stmt -> conditional_stmt|loop_stmt|break_stmt|continue_stmt|f_call_stmt|return_stmt|assign_stmt|input_stmt|output_stmt
conditional_stmt->if_stmt|switch_stmt
if_stmt->mif|uif
mif->if (expr) mif else mif | opt_stms
uif->if (expr) mif|if (expr) mif else uif
switch_stmt-> switch(expr){switch_case}
switch_case-> case number: opt_stmts switch_case | case number: opt_stmts | default: opt_stmts
loop_stmt -> do_stmt | while_stmt | for_stmt
for_stmt -> for(init;condition;update)opt_stmt
```

```
init ->  id assign_op expr init_a | ε
init_a -> ,id assign_op expr init_a | ε
condition -> expr condition_a | ε
condition_a ->, expr condition_a | ε
update -> id assign_op expr init_a | ε
while_stmt -> while (expr) opt_stmt
do_stmt -> do opt_stmt while (expr)
break_stmt -> break
continue_stmt -> continue
f_call_stmt -> id(arg_list)|id assign_op id(arg_list)
arg_list -> arg , arg_list| arg
arg -> id|num
input_stmt -> scan(id)
output_stmt -> print(arg_list_p)
expr ->  exp expr_p | !expr
expr_p ->  rel_op exp expr_p |  ε
exp ->  term exp_p
exp_p ->  +term exp_p |term exp_p | ε
term ->  factor term_p
term_p ->  * factor term_p | / factor term_p | ε
factor ->  (exp) | id | number
assign_op -> "="
rel_op -> "=="|"!="|">"|"<"|">="|"<="|
digit -> [0-9]
letter_ -> [a-zA-Z_]
number -> {digit}+
id -> letter_(letter_ | digit)*
```