Computer Science and Engineering

Compiler Design Laboratory

Course No : CSE-3212

Project : FootballScript, A Sports-Themed Programming Language Compiler

Name : Anirban Ghosh Argha

Roll : 2007094

Section : B

Date : 14/01/2025

Introduction:

This project implements a football-themed programming language compiler using Flex and Bison. The language features include:

- Football-themed keywords (e.g., 'striker' for int, 'keeper' for float, 'goal' for print)
- Basic programming constructs: variable declarations, arithmetic operations, if-else conditions, loops (for/while), functions, and error handling
- Symbol table management for variable tracking and type checking
- Input/output operations with proper error detection for type mismatches and undefined variables

The compiler reads an input file containing the source code and produces appropriate output including variable declarations, computations, and error messages, while maintaining a symbol table for tracking variables and their types.

Flex Code:

```
%{
#include <stdio.h>
#include <string.h>
#include "2007094.tab.h"
int line_number = 1;
```

```
%}

IDENTIFIER    [a-zA-Z_][a-zA-Z0-9_]*
STRING        \"[^\"]*\"
INTEGER       [0-9]+
FLOAT         [0-9]+\.[0-9]+
OPERATOR      [+\-*/=<>!&|]
DELIMITER     [,;\(\)\[\]\{\}:]
WHITESPACE    [ \t]+
NEWLINE       \n
SLC           "//".*
MLC           "/\*".*?"\*/"

%%

"goal"            { return PRINT; }
"shot"            { return SCANF; }
"kickoff"         { return MAIN; }
"dribble"         { return FOR; }
"run"             { return WHILE; }
"var"             { return IF; }
```

```
"penalty"          { return ELIF; }
"no_penalty"       { return ELSE; }
"red_card"         { return BREAK; }
"yellow_card"      { return CONTINUE; }
"final_whistle"    { return RETURN; }
"team"             { return CLASS; }
"stadium"          { return INCLUDE; }
"coach"            { return FUNCTION; }
"injury_time"      { return TRY; }
"replay"           { return CATCH; }
"trophy"           { return SUCCESS; }
"scoreboard"       { return ARRAY; }
"striker"          { return INT; }
"keeper"           { return FLOAT; }
"defender"         { return CHAR; }
"midfielder"       { return STRING; }
"squad"            { return STRUCT; }
"league"           { return ENUM; }
"pass"             { return ASSIGN; }
"tackle"           { return AND; }
"assist"           { return OR; }
```

```
"offside"          { return NOT; }
"formation"         { return VOID; }
"substitute"       { return SWITCH; }
"bench"            { return CASE; }
"training"         { return DO; }

"=="               { return EQ; }
"!="               { return NE; }
"<="               { return LE; }
">="               { return GE; }
"&&"               { return AND; }
"||"               { return OR; }
"!"                { return NOT; }

{IDENTIFIER}       { yylval.string = strdup(yytext); return IDENTIFIER; }
{INTEGER}          { yylval.number = atoi(yytext); return INTEGER; }
{FLOAT}            { yylval.floating = atof(yytext); return FLOAT_VAL; }
{STRING}           { yylval.string = strdup(yytext); return
STRING_LITERAL; }
{OPERATOR}         { return yytext[0]; }
{DELIMITER}        { return yytext[0]; }
{WHITESPACE}       { /* ignore whitespace */ }
```

```
{NEWLINE}          { line_number++; }

{SLC}              { /* Ignore single line comments */ }

{MLC}              { /* Ignore multi line comments */ }

.                  { printf("Error: Unknown token '%s' at line %d\n", yytext,
line_number); }


%%


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

Grammar rules:

```
program
    : global_declarations
    | program global_declarations
    ;


global_declarations
    : function_declaration
    | struct_declaration
```

```
    | MAIN '{' statement_list '}'
    ;

function_declaration
    : FUNCTION type IDENTIFIER '(' parameter_list ')' '{' statement_list
'}'
    | FUNCTION type IDENTIFIER '(' ')' '{' statement_list '}'
    ;

parameter_list
    : parameter
    | parameter_list ',' parameter
    ;

parameter
    : type IDENTIFIER
    ;

struct_declaration
    : STRUCT IDENTIFIER '{' member_list '}'
    ;
```

```
member_list
    : declaration ';'
    | member_list declaration ';'
    ;

statement_list
    : statement
    | statement_list statement
    ;

statement
    : declaration ';'
    | assignment ';'
    | print_statement ';'
    | scanf_statement ';'
    | if_statement
    | while_statement
    | for_statement
    | return_statement ';'
    | BREAK ';'
    | CONTINUE ';'
```

```
    | function_call ';'
    | struct_access ';'
    ;

declaration
    : type IDENTIFIER
    | STRUCT IDENTIFIER IDENTIFIER
    ;

type
    : INT
    | FLOAT
    | CHAR
    | STRING
    | VOID
    | ARRAY type '[' INTEGER ']'
    ;

assignment
    : IDENTIFIER '=' expression
    | struct_access '=' expression
```

```
    | array_access '=' expression
    ;

expression
    : INTEGER
    | FLOAT_VAL
    | IDENTIFIER
    | STRING_LITERAL
    | function_call
    | struct_access
    | array_access
    | expression '+' expression
    | expression '-' expression
    | expression '*' expression
    | expression '/' expression
    | expression AND expression
    | expression OR expression
    | NOT expression
    | '(' expression ')'
    ;
```

```
function_call
    : IDENTIFIER '(' argument_list ')'
    | IDENTIFIER '(' ')'
    ;

argument_list
    : expression
    | argument_list ',' expression
    ;

struct_access
    : IDENTIFIER '.' IDENTIFIER
    ;

array_access
    : IDENTIFIER '[' expression ']'
    ;

print_statement
    : PRINT '(' expression ')'
    ;
```

```
scanf_statement
    : SCANF '(' IDENTIFIER ')'
    ;

if_statement
    : IF '(' condition ')' '{' statement_list '}' %prec LOWER_THAN_ELSE
    | IF '(' condition ')' '{' statement_list '}' ELSE '{' statement_list '}'
    | IF '(' condition ')' '{' statement_list '}' ELIF '(' condition ')' '{' statement_list '}'
    | IF '(' condition ')' '{' statement_list '}' ELIF '(' condition ')' '{' statement_list '}' ELSE '{' statement_list '}'
    ;

while_statement
    : WHILE '(' condition ')' '{' statement_list '}'
    ;

for_statement
    : FOR '(' for_init ';' condition ';' assignment ')' '{' statement_list '}'
    ;
```

```
for_init
    : declaration '=' expression
    | assignment
    ;

condition
    : expression comparison_op expression
    | expression
    | condition AND condition
    | condition OR condition
    | NOT condition
    | '(' condition ')'
    ;

comparison_op
    : '<'
    | '>'
    | EQ
    | NE
    | LE
    | GE
```

```
    ;

return_statement
    : RETURN expression
    ;

try_catch_statement
    : TRY '{' statement_list '}' CATCH '{' statement_list '}'
    ;

switch_statement
    : SWITCH '(' expression ')' '{' case_list '}'
    ;

case_list
    : CASE expression ':' statement_list
    | case_list CASE expression ':' statement_list
    ;

enum_declaration
    : ENUM IDENTIFIER '{' enum_list '}'
```

```
    ;


enum_list
    : IDENTIFIER
    | enum_list ',' IDENTIFIER
    ;
```