Link to github: https://github.com/octaviaah/flcd

lang.y

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

#define YYDEBUG 1
%}

%token INTEGER
%token WORD
%token CHARACTER
%token IF
%token ELSE
%token FOR
%token WHILE
%token RETURN
%token CIN
%token COUT
%token OR
%token AND

%token IDENTIFIER
%token CONSTANT
```

```
%token ATRIB
%token EQ
%token NE
%token LT
%token LTE
%token GT
%token GTE
%token NOT

%left '+' '-' '/' '*' '%'

%token ADD
%token SUB
%token DIV
%token MUL
%token MOD
%token ADDEQ
%token SUBEQ
%token DIVEQ
%token MULEQ

%token OPEN_CURLY_BRACKET
%token CLOSED_CURLY_BRACKET
```

```
%token OPEN_ROUND_BRACKET
%token CLOSED_ROUND_BRACKET
%token OPEN_RIGHT_BRACKET
%token CLOSED_RIGHT_BRACKET


%token COMMA
%token DOT
%token SEMI_COLON
%token COLON
%token SPACE

%token READ_OP
%token WRITE_OP

%start program

%%
program : stmtlist
        ;
declaration :  type IDENTIFIER
            ;
type :  INTEGER | WORD | typeTemp
            ;
```

typeTemp : /*Empty*/ | OPEN_RIGHT_BRACKET CONSTANT CLOSED_RIGHT_BRACKET

;

cmpdstmt : OPEN_CURLY_BRACKET stmtlist CLOSED_CURLY_BRACKET

;

stmtlist : stmt stmtTemp

;

stmtTemp : /*Empty*/ | stmtlist

;

stmt : simplstmt SEMI_COLON | structstmt

;

simplstmt : assignstmt | iostmt | declaration

;

structstmt : cmpdstmt | ifstmt | whilestmt | forstmt

;

ifstmt : IF OPEN_ROUND_BRACKET boolean_condition CLOSED_ROUND_BRACKET stmtlist tempIf

;

tempIf : /*Empty*/ | ELSE stmtlist

;

forstmt : FOR forheader cmpdstmt

;

forheader : OPEN_ROUND_BRACKET INTEGER assignstmt SEMI_COLON OPEN_ROUND_BRACKET boolean_condition CLOSED_ROUND_BRACKET SEMI_COLON assignstmt CLOSED_ROUND_BRACKET

;

whilestmt : WHILE OPEN_ROUND_BRACKET boolean_condition CLOSED_ROUND_BRACKET cmpdstmt

;

assignstmt : IDENTIFIER ATRIB expression

;

expression : arithmetic2 arithmetic1

;

arithmetic1 : ADD arithmetic2 arithmetic1 | SUB arithmetic2 arithmetic1 | /*Empty*/

;

arithmetic2 : multiply2 multiply1

;

multiply1 : MUL multiply2 multiply1 | DIV multiply2 multiply1 | /*Empty*/

;

multiply2 : OPEN_ROUND_BRACKET expression CLOSED_ROUND_BRACKET | CONSTANT | IDENTIFIER | IndexedIdentifier

;

IndexedIdentifier : IDENTIFIER OPEN_RIGHT_BRACKET CONSTANT CLOSED_RIGHT_BRACKET

;

iostmt : CIN READ_OP IDENTIFIER | COUT WRITE_OP IDENTIFIER | COUT WRITE_OP CONSTANT

;

condition : expression GT expression |

expression GTE expression |

expression LT expression |

```
        expression LTE expression |
        expression EQ expression |
        expression NE expression
        ;
boolean_condition : condition boolean_cond_temp
                ;
boolean_cond_temp : /*Empty*/ | AND boolean_condition | OR
boolean_condition
        ;
%%
yyerror(char *s)
{
        printf("%s\n", s);
}

extern FILE *yyin;

main(int argc, char **argv)
{
        if(argc>1) yyin :  fopen(argv[1], "r");
        if(argc>2 && !strcmp(argv[2], "-d")) yydebug: 1;
        if(!yyparse()) fprintf(stderr, "\tO.K. \n");
}
```

specif.lxi

```
%{
#include <stdio.h>
#include <string.h>
#include "lang.tab.h"
int lines = 0;
%}

%option noyywrap
%option caseless

DIGIT           [0-9]
WORD            \"[a-zA-Z0-9 ]*\"
INTEGER         0|[+-]?[1-9][0-9]*
CHARACTER \'[a-zA-Z0-9]\'
constant        {WORD}|{INTEGER}|{CHARACTER}
identifier      [a-zA-Z][a-zA-Z0-9]*

%%

corner          {printf( "Reserved word: %s\n", yytext); return ELSE;}
cross       {printf( "Reserved word: %s\n", yytext); return IF;}
dribble         {printf( "Reserved word: %s\n", yytext); return FOR;}
freekick        {printf( "Reserved word: %s\n", yytext); return WHILE;}
```

```
fulltime            {printf( "Reserved word: %s\n", yytext); return RETURN;}
lineup      {printf( "Reserved word: %s\n", yytext); return WORD;}
score       {printf( "Reserved word: %s\n", yytext); return COUT;}
pass        {printf( "Reserved word: %s\n", yytext); return CIN;}
player      {printf( "Reserved word: %s\n", yytext); return INTEGER;}
referee             {printf( "Reserved word: %s\n", yytext); return CHARACTER;}

{identifier}  {printf( "Identifier: %s\n", yytext); return IDENTIFIER;}
{constant}    {printf( "Constant: %s\n", yytext); return CONSTANT;}

"{"         {printf( "Separator: %s\n", yytext); return OPEN_CURLY_BRACKET;}
"}"         {printf( "Separator: %s\n", yytext); return CLOSED_CURLY_BRACKET;}
"("         {printf( "Separator: %s\n", yytext); return OPEN_ROUND_BRACKET;}
")"         {printf( "Separator: %s\n", yytext); return CLOSED_ROUND_BRACKET;}
"."         {printf( "Separator: %s\n", yytext); return DOT;}
","         {printf( "Separator: %s\n", yytext); return COMMA;}
":"         {printf( "Separator: %s\n", yytext); return COLON;}
";"         {printf( "Separator: %s\n", yytext); return SEMI_COLON;}

"+"         {printf( "Operator: %s\n", yytext); return ADD;}
"-"         {printf( "Operator: %s\n", yytext); return SUB;}
"*"         {printf( "Operator: %s\n", yytext); return MUL;}
"/"         {printf( "Operator: %s\n", yytext); return DIV;}
"%"         {printf( "Operator: %s\n", yytext); return MOD;}
```

```
"="             {printf( "Operator: %s\n", yytext); return ATRIB;}
"<"             {printf( "Operator: %s\n", yytext); return LT;}
"<<"            {printf( "Operator: %s\n", yytext); return WRITE_OP;}
"<="            {printf( "Operator: %s\n", yytext); return LTE;}
"=="            {printf( "Operator: %s\n", yytext); return EQ;}
">="            {printf( "Operator: %s\n", yytext); return GTE;}
">>"            {printf( "Operator: %s\n", yytext); return READ_OP;}
">"             {printf( "Operator: %s\n", yytext); return GT;}
"!"             {printf( "Operator: %s\n", yytext); return NOT;}
"!="            {printf( "Operator: %s\n", yytext); return NE;}
"&&"            {printf( "Operator: %s\n", yytext); return AND;}
"||"            {printf( "Operator: %s\n", yytext); return OR;}
"+="            {printf( "Operator: %s\n", yytext); return ADDEQ;}
"-="            {printf( "Operator: %s\n", yytext); return SUBEQ;}
"*="            {printf( "Operator: %s\n", yytext); return MULEQ;}
"/="            {printf( "Operator: %s\n", yytext); return DIVEQ;}


[ \t]+
[\n]+


[+-]?0[0-9]*                printf("Illegal integer at line \n");
[0-9]+[a-zA-Z_]+[a-zA-Z0-9_]*   printf("Illegal identifier\n");
\'[a-zA-Z0-9]{2,}\'          printf("Character of length >=2 at line \n");
```

```
.                       printf("Lexical error\n");
%%
```

p1.txt

```
player max;
player n;
player sum;

max = 0;
sum = 0;

pass >> n;

freekick (n > 0) {
        player x;
        pass >> x;
        sum = sum + x;
        n = n - 1;
}

score << sum;
```

p2.txt

```
player a;
player b;
player c;
player max;

max = 0;

pass >> a;
pass >> b;
pass >> c;

cross (a >= b && a >= c) max = a;
corner max = b;

score << max;
```