GITHUB: https://github.com/RazvanAndreiLazar/FLCD - l8

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

int lines = 0;
char **stringST;
int sizeStringST = 10;
int lenStringST = 0;
char **identifierST;
int sizeIdentST = 10;
int lenIdentST = 0;
int *intST;
int sizeIntST = 10;
int lenIntST = 0;
bool *boolST;
int sizeBoolST = 10;
int lenBoolST = 0;
char **tokens;
int sizeTokens = 10;
int lenTokens = 0;

struct Entry{
   int code; //0 - token, 1 - identifier, 2 - int const, 3 - string const, 4 - bool const
   char *token;
   int position;
};

struct Entry* pif;
int sizePIF = 10;
int lenPIF = 0;

void AddToken (char *token)
{
   if (sizeTokens == lenTokens) {
      char **newST = malloc(sizeTokens * 2 * sizeof (char*));
      for (int i = 0; i < sizeTokens; ++i)
         newST[i] = tokens[i];
      free(tokens);
      sizeTokens *= 2;
      tokens = newST;
   }
   tokens[lenStringST++] = token;
}

void InitTokens()
```

```
{
    tokens = malloc(sizeTokens * sizeof(char*));
    AddToken("const");
    AddToken("ident");
    char buff[256];
    FILE *f = fopen("tokens.in", "r");
    while(fgets(buff, 255, f) != 0){
        AddToken(buff);
    }
}

int GetTokenPosition(char* token){
    for (int i = 0; i < lenTokens; i++){
        if (strcmp(token, tokens[i]) == 0)
            return i;
    }
    return -1;
}

void InitSymbolTables()
{
    stringST = malloc(sizeStringST * sizeof(char*));
    identifierST = malloc(sizeIdentST * sizeof(char*));
    intST = malloc(sizeIntST * sizeof(int));
    boolST = malloc(sizeBoolST * sizeof(bool));
    pif = malloc(sizePIF * sizeof(struct Entry));
}

void AddStringConst (char *string)
{
    if (sizeStringST == lenStringST) {
        char **newST = malloc(sizeStringST * 2 * sizeof (char*));
        for (int i = 0; i < sizeStringST; ++i)
            newST[i] = stringST[i];
        free(stringST);
        sizeStringST *= 2;
        stringST = newST;
    }
    stringST[lenStringST++] = string;
}

void AddIdentifier (char *identifier)
{
    if (sizeIdentST == lenIdentST) {
        char **newST = malloc(sizeIdentST * 2 * sizeof (char*));
        for (int i = 0; i < sizeIdentST; ++i)
            newST[i] = identifierST[i];
        free(identifierST);
        sizeIdentST *= 2;
```

```
        identifierST = newST;
    }
    identifierST[lenIdentST++] = identifier;
}

void AddIntConst (int number)
{
    if (sizeIntST == lenIntST) {
        int *newST = malloc(sizeIntST * 2 * sizeof (int));
        for (int i = 0; i < sizeIntST; ++i)
            newST[i] = intST[i];
        free(intST);
        sizeIntST *= 2;
        intST = newST;
    }
    intST[lenIntST++] = number;
}

void AddBoolConst (bool boolean)
{
    if (sizeBoolST == lenBoolST) {
        bool *newST = malloc(sizeBoolST * 2 * sizeof (bool));
        for (int i = 0; i < sizeBoolST; ++i)
            newST[i] = boolST[i];
        free(boolST);
        sizeBoolST *= 2;
        boolST = newST;
    }
    boolST[lenBoolST++] = boolean;
}

int GetStringIndex(char *string)
{
    for (int i = 0; i < lenStringST; ++i)
        if (strcmp(string, stringST[i]) == 0)
            return i;
    AddStringConst(string);
    return lenStringST-1;
}

int GetIdentifierIndex(char *identifier)
{
    for (int i = 0; i < lenIdentST; ++i)
        if (strcmp(identifier, identifierST[i]) == 0)
            return i;
    AddIdentifier(identifier);
    return lenIdentST-1;
}
```

```
int GetIntIndex(char *number)
{
   int x = atoi(number);
   for (int i = 0; i < lenIntST; ++i)
      if (x == intST[i])
         return i;
   AddIntConst(x);
   return lenIntST-1;
}

int GetBoolIndex(char *boolean)
{
   bool x = strcmp(boolean, "true") == 0 ? true : false;
   for (int i = 0; i < lenBoolST; ++i)
      if (x == boolST[i])
         return i;
   AddBoolConst(x);
   return lenBoolST-1;
}

void AddToPIF(struct Entry entry)
{
   if (sizePIF == lenPIF) {
      struct Entry *new_pif = malloc(sizePIF * 2 * sizeof (struct Entry));
      for (int i = 0; i < sizePIF; ++i)
         new_pif[i] = pif[i];
      free(pif);
      sizePIF *= 2;
      pif = new_pif;
   }
   pif[lenPIF++] = entry;
}

struct Entry GetEntry(int opcode, char *token, int pos) {
   struct Entry entry;
   entry.code = opcode;
   entry.position = pos;
   entry.token = token;
   return entry;
};

char* StringCopy(char *string) {
   int size = (int)strlen(string);
   char* new_string = malloc((size+1) * sizeof (char));
   for (int i = 0; i <= size; ++i)
      new_string[i] = string[i];
   return new_string;
}
```

```
%}

%option noyywrap
%option caseless

DIGIT [0-9]
NON_ZERO_DIGIT [1-9]
INT_CONSTANT [+-]?{NON_ZERO_DIGIT}{DIGIT}*|0
LETTER [a-zA-Z_]
STRING_CONSTANT \".*\\"
BOOL_CONSTANT true|false
IDENTIFIER {LETTER}({LETTER}|{DIGIT})*

%%

"int"|"string"|"bool"|"array"|"if"|"else"|"while"|"READ"|"PRINT"|"BEGIN"|"END"|"STOP" { char *token = StringCopy(yytext); AddToPIF(GetEntry(0, token, -1)); printf("%s - reserved word\n", yytext);}

{INT_CONSTANT} {char *intConst = StringCopy(yytext); AddToPIF(GetEntry(2, intConst, GetIntIndex(intConst))); printf("%s - int constant\n", yytext);}

{STRING_CONSTANT} {char *strConst = StringCopy(yytext); AddToPIF(GetEntry(3, s, GetStringIndex(strConst))); printf("%s - str constant\n", yytext);}

{BOOL_CONSTANT} {char *boolConst = StringCopy(yytext); AddToPIF(GetEntry(4, boolConst, GetBoolIndex(boolConst))); printf("%s - bool constant\n", yytext);}

{IDENTIFIER} {char *id = StringCopy(yytext); AddToPIF(GetEntry(1, id, GetIdentifierIndex(id))); printf("%s - identifier\n", yytext);}

"+"|"-"|"*"|"/"|"%"|"<"|">"|"="|"<="|">="|"=="|"!="|"&&"|"||" {char *token = StringCopy(yytext); AddToPIF(GetEntry(0, token, -1)); printf("%s - operator\n", yytext);}

"("|")"|"{"|"}"|","|"["|"]"|"\"" {char *token = StringCopy(yytext); AddToPIF(GetEntry(0, token, -1)); printf("%s - separator\n", yytext);}

[ \t]+ {}

[\n]+ {++lines; AddToPIF(GetEntry(0, "endline", -1));}

. {printf("Error at token %s at line %d\n", yytext, lines); exit(1);}

%%

int main(int argc, char **argv )
{
   if ( argc > 1 )
                yyin = fopen(argv[1], "r");
```

```
    else
            yyin = stdin;
            InitSymbolTables();
    yylex();
    printf("INT SYMBOL TABLE\n\n");
    for (int i = 0; i < lenIntST; ++i)
            printf("%d\n", intST[i]);
    printf("\n");
    printf("STRING SYMBOL TABLE\n\n");
    for (int i = 0; i < lenStringST; ++i)
            printf("%s\n", stringST[i]);
    printf("\n");
    printf("BOOL SYMBOL TABLE\n\n");
    for (int i = 0; i < lenBoolST; ++i)
            printf("%d\n", boolST[i]);
    printf("\n");
    printf("IDENTIFIER SYMBOL TABLE\n\n");
    for (int i = 0; i < lenIdentST; ++i)
            printf("%s\n", identifierST[i]);
    printf("\n");
    printf("PIF\n\n");
    for (int i = 0; i < lenPIF; ++i)
            printf("%s: (%d, %d)\n", pif[i].token, pif[i].code, pif[i].position);
}
```