TAREFA EXTRACLASSE – GERADOR AUTOMÁTICO FLEX

Com base no exemplo do analisador para expressões aritméticas, apresentado na vídeo-aula 9 disponível no Moodle da disciplina, formado pelos três arquivos (exp.h, exp.c e exp.lex) a seguir:

/* ===== Arquivo exp.h ===== */

```
// constantes booleanas
#define TRUE 1
#define FALSE 0

// constantes para nome de token
#define TOK_NUM 0
#define TOK_OP 1
#define TOK_PONT 2
#define TOK_ERRO 3

// constantes para operadores
#define SOMA 0
#define SUB 1
#define MULT 2
#define DIV 3

// constantes para parenteses
#define PARESQ 0
#define PARDIR 1

// estrutura de um token
typedef struct {
    int tipo;
    int atributo;
} Token;

// funcao para criar um token
extern Token *token();

// funcao do analisador lexico
extern Token *yylex();
```

/* ===== Arquivo exp.lex ===== */

```
%option noyywrap
%option nodefault
%option outfile="lexer.c" header-file="lexer.h"

%{ #include "exp.h“ %}
```

```
NUM [0-9]+

%%

[[:space:]] { } /* ignora espaços */
{NUM} { return token(TOK_NUM, atoi(yytext)); }
\+ { return token(TOK_OP, SOMA); }
- { return token(TOK_OP, SUB); }
\* { return token(TOK_OP, MULT); }
\/ { return token(TOK_OP, DIV); }
\( { return token(TOK_PONT, PARESQ); }
\) { return token(TOK_PONT, PARDIR); }
/* Tratamento para token desconhecido */
. { return token(TOK_ERRO, 0); }

%%

// variavel global para um token
Token tok;
Token * token (int tipo, int valor) {
    tok.tipo = tipo;
    tok.atributo = valor;
    return &tok;
}


/* ===== Arquivo exp.c ===== */

#include "lexer.h"
#include “exp.h“

/* Carrega uma string como entrada */
YY_BUFFER_STATE buffer;

void inicializa(char *str) {
    buffer = yy_scan_string(str);
}

Token *proximo_token() {
    return yylex();
}

void imprime_token( Token *tok) {
    /* Aqui deve implementar a lógica para mostrar na tela o token retornado pelo analisador léxico <token, atributo> de acordo com o tipo do token (obs: lembre-se que alguns tokens não possuem atributo)*/
}
```

```
int main(int argc, char **argv) {
    Token *tok;

// Definicao da entrada
    char entrada[200];
    printf("\nAnalise Lexica da expressao: ");
    fgets(entrada, 200, stdin);
    inicializa(entrada);

    tok = proximo_token();
    while (tok != NULL) {
        imprime_token(tok);
        tok = proximo_token();
    }

    return 0;
}
```

Implemente um analisador léxico (lexer) no FLEX para reconhecer os tokens: id (identificadores), relop (operadores relacionais), separadores (espaço, \t e \n), num_int (constante numérica inteira), num_float (constante numérica float: com ponto decimal e/ou notação científica), comentários (apenas os comentários de bloc: "/* comentário */"), EOF (final de arquivo), atribuição (":=") e as palavras reservadas "**if**", "**then**", "**else**", "**while**", "**repeat**" e "**until**". Destaca-se que as expressões regulares e os diagramas de transição de alguns desses tokens também estão disponibilizados nos slides do material sobre análise léxica. Para os tokens id, relop, num_int e num_float, além do tipo do token, o analisador deve retornar o lexema encontrado como atributo. Os demais tokens não possuem atributos e devem-se retornar "none" no campo correspondente. Destaca-se que o tipo do atributo deve ser alterado para atender as especificações da questão.

OBS: caso utilize alguma instrução do flex ou alguma notação estendida de especificação das ERs que não foram vistas nos slides das aulas, inclua a referência na frente do comando, na forma de comentário.

Também escreva um programa principal para testar o analisador criado. Esse programa deve solicitar ao usuário o nome do arquivo com o código fonte a ser analisado e passá-lo para o lexer, conforme apresentado na vídeo-aula. Em seguida, inicia-se o processo de reconhecimento dos tokens, com a solicitação de um novo *token* ao analisador léxico. O token retornado deve ser mostrado na tela, no seguinte formato: <*token, atributo*>, sendo que *token* refere-se ao nome do token e não ao seu código. Esse processo se repetirá até que não haja mais *tokens* no arquivo, ou seja, o analisador léxico retorne o token <EOF, none >, o qual também deve ser exibido.

Compile os arquivos lex e c, a fim de gerar a versão executável do analisador, conforme indicado na vídeo-aula. Compacte todos os arquivos (lex, c e executável) em um único **arquivo zip** com o nome **TarefaFlex_NomeAluno.zip** e envio-o pela tarefa no *Teams* dentro do prazo especificado.

OBS: O FLEX é a versão GNU do LEX e pode ser instalado em qualquer ambiente linux