UNIVERSIDADE FEDERAL DE LAVRAS

RELATÓRIO DE COMPILADORES

Professor: Ricardo Terra Nunes Bueno Villela

Estevão Augusto da Fonseca Santos

Felipe Crisóstomo Silva Oliveira

Bernardo Coelho Pavani Marinho

1. INTRODUÇÃO

O processo de análise léxica é uma das etapas fundamentais na construção de compiladores e interpretadores. Seu principal objetivo é identificar e classificar os símbolos válidos em uma linguagem de programação, transformando uma sequência de caracteres em uma sequência de tokens significativos. Este trabalho apresenta a implementação de um analisador léxico utilizando a ferramenta *Flex* (Fast Lexical Analyzer Generator), demonstrando suas funcionalidades, estratégias de implementação, testes realizados e os resultados obtidos.

2. REFERENCIAL TEÓRICO

A análise léxica é a primeira fase da compilação e tem como função principal escanear o código fonte e dividi-lo em unidades léxicas (tokens), que são passadas à etapa de análise sintática. Um token pode representar palavras-chave, identificadores, operadores, literais, entre outros.

O *Flex* é uma ferramenta para geração de analisadores léxicos baseada em expressões regulares. Ele gera código em C para reconhecer padrões definidos pelo programador. Os padrões são especificados em uma linguagem declarativa dividida em três seções: definição, regras e código do usuário.

```
Unset
%% // separa as seções
```

```
<expressão regular>  <ação>

%%

int main() {
	yylex();
	return 0;
}
```

3. DESCRIÇÃO DO TRABALHO E ESTRATÉGIAS DE SOLUÇÃO

Neste projeto, desenvolvemos um analisador léxico para a linguagem de programação C- (subconjunto de C), contendo os seguintes elementos léxicos:

- Tipos de dados: inteiro, real, caractere, arranjo e registro.
- Funções: recursão, parâmetros passados por valor.
- Comandos: Atribuição, if/else, while, E/S simples (tratados como funções).
- Comentários: texto entre /* e */ (sem comentários aninhados).
- Palavras reservadas: int, oat, struct, if, else, while, void, return (caixa baixa)
- Símbolo Inicial: <programa>

3.1 EXPRESSÕES REGULARES UTILIZADAS

```
Unset
programa                          {declaracao_lista}
declaracao_lista                  {declaracao}+
declaracao                        {var_declaracao}|{func_declaracao}
var_declaracao
{tipo_especificador}{ident};|{tipo_especificador}{ident}({abre_colchete}{num
_int}{fecha_colchete})+
```

```
tipo_especificador
(int|float|char|void|struct{ident}{abre_chave}{atributos_declaracao}{fecha_c
have})
atributos_declaracao             {var_declaracao}+
func_declaracao
{tipo_especificador}{ident}\({params}\){composto_decl}
params                           {params_lista}|void
params_lista                     {param}(,{param})*
param
({tipo_especificador}{ident}|{tipo_especificador}{ident}{abre_colchete}{fech
a_colchete})
composto_decl
{abre_chave}{local_declaracoes}{comando_lista}{fecha_chave}
local_declaracoes                {var_declaracao}*
comando_lista                    {comando}*
comando
{expressao_decl}|{composto_decl}|{selecao_decl}|{iteracao_decl}|{retorno_dec
l}
expressao_decl                   {expressao};|;
selecao_decl
(if\({expressao}\){comando}|if\({expressao}\){comando}else{comando})
iteracao_decl                    while\({expressao}\){comando}
retorno_decl                     return;|return{expressao};
expressao                        ({var}={expressao}|{expressao_simples})
var
{ident}|{ident}({abre_colchete}{expressao}{fecha_colchete})+
expressao_simples
{expressao_soma}{relacional}{expressao_soma}|{expressao_soma}
relacional                       <=|<|>=|>|==|!=
expressao_soma                   {termo}({soma}{termo})*
soma                             \+|\-
termo                            {fator}({multi}{fator})*
multi                            \/|\*
fator
\({expressao}\)|{var}|{ativacao}|{num}|{num_int}
ativacao                         {ident}\({args}\)
args                             {arg_lista}?
arg_lista                        {expressao}(,{expressao})*
num                              [+-]?[0-9]+(\.[0-9]+)?([E][+-]?[0-9]+)?
num_int                          {digito}+
digito                           [0-9]
ident                            {letra}({letra}|{digito})*
letra                            [a-z]
abre_chave                       \{
fecha_chave                      \}
abre_colchete                    \[
fecha_colchete                   \]
other                            .
```

```
delim                           [ \t\n]
ws                              {delim}+
comment                         "/*"([^*]|\*+[^/])*\*+"/"
```

4. TESTES EXECUTADOS E RESULTADOS OBTIDOS

4.1 Entrada: int main() { return 0; }

Resultado obtido:

1(1): int (KEYWORD)

1(4): main (IDENTIFIER)

1(8): ((DELIMITER)

1(9):) (DELIMITER)

1(10): { (DELIMITER)

1(11): return (KEYWORD)

1(17): 0 (CONSTINT)

1(18): ; (DELIMITER)

1(19): } (DELIMITER)

4.2 Entrada: char []txt = “texto”;

Resultado obtido:

1(1): char (KEYWORD)

1(5): [(DELIMITER)

1(6):] (DELIMITER)

1(7): txt (IDENTIFIER)

1(10): = (RELOP)

1(11): “texto” (CONSTSTRING)

1(18): ; (DELIMITER)

4.3 Entrada: float quadrado(float x){ return x*x; }

Resultado obtido:

1(1): float (KEYWORD)

1(6): quadrado (IDENTIFIER)

1(14): ((DELIMITER)

1(15): float (KEYWORD)

1(20): x (IDENTIFIER)

1(21):) (DELIMITER)

1(22): { (DELIMITER)

1(23): return (KEYWORD)

1(29): x (IDENTIFIER)

1(30): * (ARITHOP)

1(31): x (IDENTIFIER)

1(32): ; (DELIMITER)

1(33): } (DELIMITER)

4.4 Entrada: int res = 2*a - 9;

Resultado obtido:

1(1): int (KEYWORD)

1(4): res (IDENTIFIER)

1(7): = (RELOP)

1(8): 2 (CONSTINT)

1(9): * (ARITHOP)

1(10): a (IDENTIFIER)

1(11): - (ARITHOP)

1(12): 9 (CONSTINT)

1(13): ; (DELIMITER)

5. CONCLUSÃO

A implementação de um analisador léxico com *Flex* demonstrou ser eficiente e prática para identificar estruturas léxicas em uma linguagem de programação simples. A clareza das expressões regulares e a flexibilidade do Flex permitem adaptações rápidas para novas linguagens. A ferramenta provou ser útil tanto em contextos acadêmicos quanto como base para compiladores reais. Futuramente,

este analisador pode ser integrado com ferramentas de análise sintática, como o *Bison*, para formar um compilador completo.

6. REFERÊNCIAS BIBLIOGRÁFICAS.

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compiladores: Princípios, Técnicas e Ferramentas* (2ª ed.). Pearson.

Levine, J. R., Mason, T., & Brown, D. (1992). *Lex & Yacc*. O'Reilly Media.

Projeto GNU. *Flex - The Fast Lexical Analyzer*. Disponível em: https://www.gnu.org/software/flex/