Felipe Ferreira Carvalho Silva
Lorena Kérollen Botelho Tavares
Rodrigo Pinto Herculano
William Davi Coelho

RELATÓRIO DA PRIMEIRA ETAPA DO TRABALHO DE COMPILADORES
ANALISADOR LÉXICO

LAVRAS - MG
Abril de 2019

INTRODUÇÃO

Na primeira etapa do trabalho prático, implementamos o analisador léxico para a linguagem C--. O projeto foi feito na linguagem Python e tem como entrada um arquivo texto contendo um exemplo de código na linguagem C--, e como saída temos os possíveis erros léxicos (caso existam), a lista de lexemas, os tokens gerados e a tabela de símbolos. O programa também trata erros léxicos e ignora/remove comentários e espaços em branco.

REFERENCIAL TEÓRICO

O material de análise léxica do livro "Compiladores: princípios, técnicas e ferramentas" - Aho, Lam, Sethi, Ullman (2ª edição) em junção com o material teórico explicado em sala de aula contribuíram significativamente para a conclusão da implementação do analisador léxico para a linguagem C-- neste trabalho.

DESENVOLVIMENTO DO PROJETO

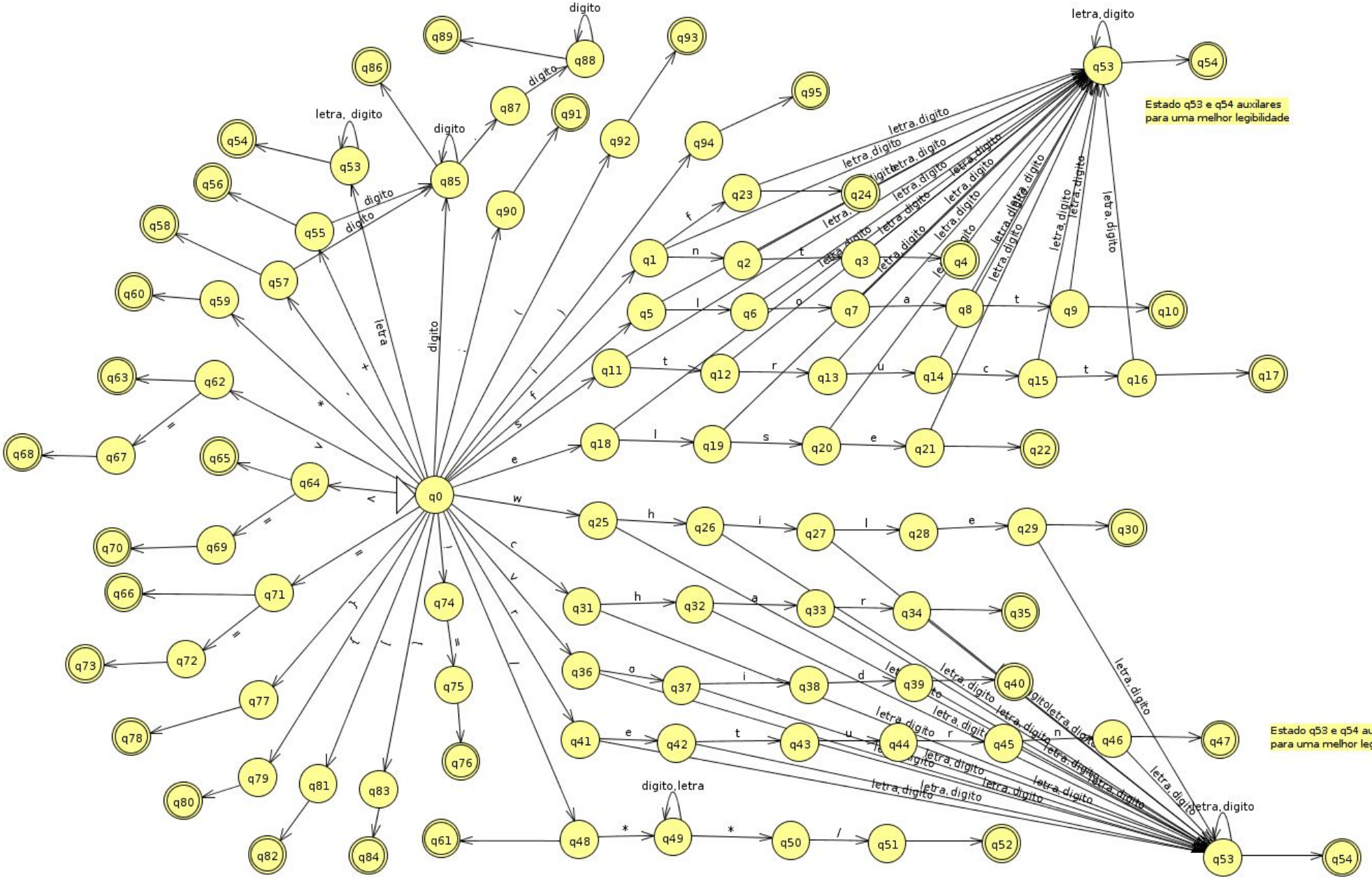
Na primeira parte foi feita a leitura do arquivo de entrada (caractere por caractere), a separação dos lexemas encontrados e a retirada dos espaços em branco e comentários.

Os erros léxicos foram tratados usando o panic mode. A abordagem utilizada foi a de deletar a palavra (inteira) que está fora do padrão léxico da linguagem e continuar a leitura do próximo lexema.

Para identificar o tipo léxico da palavra de entrada e gerar os tokens, utilizamos a tabela dirigida à autômato. Com ela, computamos cada transição do autômato reconhecedor da linguagem C--, e ao finalizar essa computação, são gerados os tokens dos lexemas de entrada.

Após isso, todos os lexemas que são identificadores ou constantes são inseridos na tabela de símbolos. Na inserção, é verificado se o lexema já está presente na tabela de símbolos, a posição do mesmo é retornada, caso contrário, ele é inserido no final da tabela.

Na página seguinte está a representação do autômato reconhecedor da linguagem C--:


Estado q53 e q54 auxilares para uma melhor legibilidade
Estado q53 e q54 auxilares para uma melhor legibilidade
letra, digito
digito
digito, letra
letra
digito

TESTES E RESULTADOS

Arquivo sem erros léxicos

```
----- Lexemas -----

['struct', 'aluno', '{', 'float', 'nota', ';', 'char', 'sexo', ';', 'int', 'periodo', ';', '}', 'int', 'nota', '(', 'int', 'periodo', ')', '{', 'int', 'nota', '=', '0',
';', 'if', '(', 'periodo', '==', '3', ')', '{', 'nota', '=', '+10', ';', '}', 'return', 'nota', ';', '}', 'void', 'imprimealunos', '(', ')', '{', '}', 'int', 'main', '(
', ')', '{', 'int', 'x', '=', '10', ';', 'int', 'i', '=', '0', ';', 'while', '(', 'i', '<', '10', ')', '{', 'x', '=', 'x', '+10', ';', 'i', '=', 'i', '+', '1', ';', '}',
'if', '(', 'x', '>=', '100', ')', '{', 'x', '=', '-10', ';', '}', 'else', '{', 'x', '=', '15', ';', '}', 'return', '0', ';', '}']
```

```c
struct aluno {
     float nota;
     char sexo;
     int periodo;
}
int nota (int periodo){
     int nota = 0;
     if (periodo == 3){
        nota = +10;
     }
     return nota;
}

void imprimealunos() {
     /* imprime todos os alunos */
}

int main() {

     int x = 10;
     int i = 0;

     /* laço para somar x = x + 10, 10 vezes */
     while (i < 10)  {
          x = x +10;
          i = i+ 1;
     }

     if(x >= 100){
          x = -10;
     } else {
               x = 15;
     }

     return 0;
}
```

```
     ----- Tabela de Símbolos -----

+---------+---------------+---------------+
| Entrada |     Lexema    |      Tipo     |
+---------+---------------+---------------+
|    1    |     aluno     | identificador |
|    2    |      nota     | identificador |
|    3    |      sexo     | identificador |
|    4    |    periodo    | identificador |
|    5    |       0       | constNumerica |
|    6    |       3       | constNumerica |
|    7    |      +10      | constNumerica |
|    8    | imprimealunos | identificador |
|    9    |      main     | identificador |
|    10   |       x       | identificador |
|    11   |       10      | constNumerica |
|    12   |       i       | identificador |
|    13   |       1       | constNumerica |
|    14   |      100      | constNumerica |
|    15   |      -10      | constNumerica |
|    16   |       15      | constNumerica |
+---------+---------------+---------------+
```

```
  ----- Tokens -----

< struct , struct >
< identificador , 1 >
< abreChave , { >
< float , float >
< identificador , 2 >
< pontoVirgula , ; >
< char , char >
< identificador , 3 >
< pontoVirgula , ; >
< int , int >
< identificador , 4 >
< pontoVirgula , ; >
< fechaChave , } >
< int , int >
< identificador , 2 >
< abreParenteses , ( >
< int , int >
< identificador , 4 >
< fechaParenteses , ) >
< abreChave , { >
< int , int >
< identificador , 2 >
< = , = >
< constNumerica , 5 >
< pontoVirgula , ; >
< if , if >
< abreParenteses , ( >
< identificador , 4 >
< == , == >
< constNumerica , 6 >
< fechaParenteses , ) >
< abreChave , { >
< identificador , 2 >
< = , = >
< constNumerica , 7 >
< pontoVirgula , ; >
< fechaChave , } >
< return , return >
< identificador , 2 >
< pontoVirgula , ; >
< fechaChave , } >
< void , void >
< identificador , 8 >
< abreParenteses , ( >
< fechaParenteses , ) >
< abreChave , { >
< fechaChave , } >
< int , int >
< identificador , 9 >
< abreParenteses , ( >
< fechaParenteses , ) >
< abreChave , { >
< int , int >
< identificador , 10 >
< = , = >
< constNumerica , 11 >
< pontoVirgula , ; >
< int , int >
< identificador , 12 >
< = , = >
< constNumerica , 5 >
< pontoVirgula , ; >
< while , while >
< abreParenteses , ( >
< identificador , 12 >
< < , < >
< constNumerica , 11 >
< fechaParenteses , ) >
< abreChave , { >
< identificador , 10 >
< = , = >
< identificador , 10 >
< constNumerica , 7 >
< pontoVirgula , ; >
< identificador , 12 >
< = , = >
< identificador , 12 >
< + , + >
< constNumerica , 13 >
< pontoVirgula , ; >
< fechaChave , } >
< if , if >
< abreParenteses , ( >
< identificador , 10 >
< >= , >= >
< constNumerica , 14 >
< fechaParenteses , ) >
< abreChave , { >
< identificador , 10 >
< = , = >
< constNumerica , 15 >
< pontoVirgula , ; >
< fechaChave , } >
< else , else >
< abreChave , { >
< identificador , 10 >
< = , = >
< constNumerica , 16 >
< pontoVirgula , ; >
< fechaChave , } >
< return , return >
< constNumerica , 5 >
< pontoVirgula , ; >
< fechaChave , } >
```

Arquivo com erros léxicos

```
----- Lexemas -----

['struct', 'aluno', '{', 'float', 'char', 'sexo', ';', 'int', 'periodo', ';', '}', 'int', 'nota', '(', 'int', 'periodo', ')', '{', 'int', 'nota', '=', '0', ';', 'if', '
=', '3', ')', '{', 'nota', '=', '+10', ';', '}', 'return', '}', 'void', 'imprimealunos', '(', ')', '{', '}', 'int', 'main', '(', ')', '{', 'int', 'x', '=', 'int', 'i',
=', '0', ';', 'while', '(', '<', '10', ')', '{', 'x', '=', 'x', '+10', ';', 'i', '=', 'i', '+', '1', ';', '}', 'if', '(', 'x', '>=', '100', ')', '{', 'x', '=', '-10', '
', '}', 'else', '{', 'x', '=', '}', 'return', '0', ';', '}']
```

```
struct aluno {
    float 4no34ta;
    char sexo;
    int periodo;
}
int nota (int periodo){
    int nota = 0;
    if 34a(periodo == 3){
        nota = +10;
    }
    return 24no42ta39;
}

void imprimealunos() {
    /* imprime todos os alunos */
}

int main() {

    int x = 1x0;
    int i = 0;

    /* laço para somar x = x + 10, 10 vezes */
    while (10i < 10)  {
        x = x +10;
        i = i+ 1;
    }

    if(x >= 100){
        x = -10;
    } else {
        x = 1abc5;
    }

    return 0;
}
```

----- Tabela de Símbolos -----

Entrada	Lexema	Tipo
1	aluno	identificador
2	sexo	identificador
3	periodo	identificador
4	nota	identificador
5	0	constNumerica
6	3	constNumerica
7	+10	constNumerica
8	imprimealunos	identificador
9	main	identificador
10	x	identificador
11	i	identificador
12	10	constNumerica
13	1	constNumerica
14	100	constNumerica
15	-10	constNumerica

```
***** Erro lexico na linha  2  coluna  11  *****

***** Erro lexico na linha  8  coluna  8  *****

***** Erro lexico na linha  11  coluna  12  *****

***** Erro lexico na linha  20  coluna  13  *****

***** Erro lexico na linha  24  coluna  12  *****

***** Erro lexico na linha  32  coluna  13  *****
```

```
  ----- Tokens -----

< struct , struct >
< identificador , 1 >
< abreChave , { >
< float , float >
< char , char >
< identificador , 2 >
< pontoVirgula , ; >
< int , int >
< identificador , 3 >
< pontoVirgula , ; >
< fechaChave , } >
< int , int >
< identificador , 4 >
< abreParenteses , ( >
< int , int >
< identificador , 3 >
< fechaParenteses , ) >
< abreChave , { >
< int , int >
< identificador , 4 >
< = , = >
< constNumerica , 5 >
< pontoVirgula , ; >
< if , if >
< == , == >
< constNumerica , 6 >
< fechaParenteses , ) >
< abreChave , { >
< identificador , 4 >
< = , = >
< constNumerica , 7 >
< pontoVirgula , ; >
< fechaChave , } >
< return , return >
< fechaChave , } >
< void , void >
< identificador , 8 >
< abreParenteses , ( >
< fechaParenteses , ) >
< abreChave , { >
< fechaChave , } >
< int , int >
< identificador , 9 >
< abreParenteses , ( >
< fechaParenteses , ) >
< abreChave , { >
< int , int >
< identificador , 10 >
```

```
< = , = >
< int , int >
< identificador , 11 >
< = , = >
< constNumerica , 5 >
< pontoVirgula , ; >
< while , while >
< abreParenteses , ( >
< < , < >
< constNumerica , 12 >
< fechaParenteses , ) >
< abreChave , { >
< identificador , 10 >
< = , = >
< identificador , 10 >
< constNumerica , 7 >
< pontoVirgula , ; >
< identificador , 11 >
< = , = >
< identificador , 11 >
< + , + >
< constNumerica , 13 >
< pontoVirgula , ; >
< fechaChave , } >
< if , if >
< abreParenteses , ( >
< identificador , 10 >
< >= , >= >
< constNumerica , 14 >
< fechaParenteses , ) >
< abreChave , { >
< identificador , 10 >
< = , = >
< constNumerica , 15 >
< pontoVirgula , ; >
< fechaChave , } >
< else , else >
< abreChave , { >
< identificador , 10 >
< = , = >
< fechaChave , } >
< return , return >
< constNumerica , 5 >
< pontoVirgula , ; >
< fechaChave , } >
```

CONCLUSÃO

O desenvolvimento deste trabalho, desde o planejamento teórico até a implementação do trabalho em si, foram de suma importância para o aprendizado dessa etapa de um compilador. O entendimento do conteúdo teórico da análise léxica agregado com o desenvolvimento e implementação do trabalho, proporcionou uma boa aquisição de conhecimento para os membros do grupo para a realização e conclusão do mesmo.