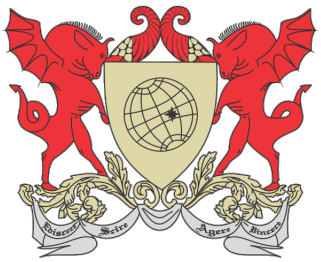


UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS FLORESTAL

Trabalho 0
Compiladores

Luciano Belo de Alcântara Júnior - 3897

Trabalho 0 apresentado à disciplina de Compiladores do curso de Ciência da Computação da Universidade Federal de Viçosa.

Florestal
14 de Maio de 2023

CCF 441 - Compiladores

Trabalho 0

Luciano Belo de Alcântara Júnior - 3897

14 de Maio de 2023

Contents

1 Introdução

O presente trabalho tem como objetivo o entendimento e aplicação de conteúdos vistos na disciplina de CCF 480 em especial sobre analisadores léxicos.

Um analisador léxico é uma das etapas iniciais de um compilador. Ele é responsável por ler o código fonte do programa e dividir as suas sequências de caracteres em tokens, que são unidades léxicas com signigficado semântico. Os tokens gerados como saída do analisador léxico são passados para o analisador léxico, que utiliza esses tokens para construir uma árvore sintática que representa a estrutura do programa.

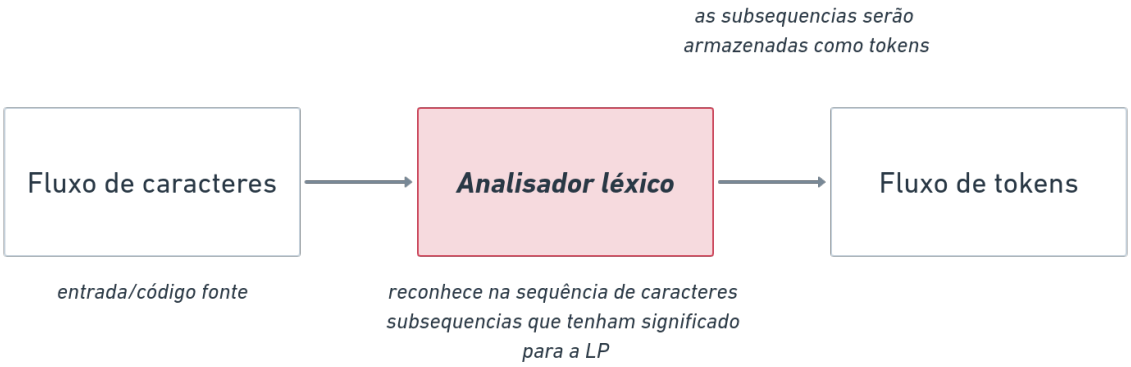
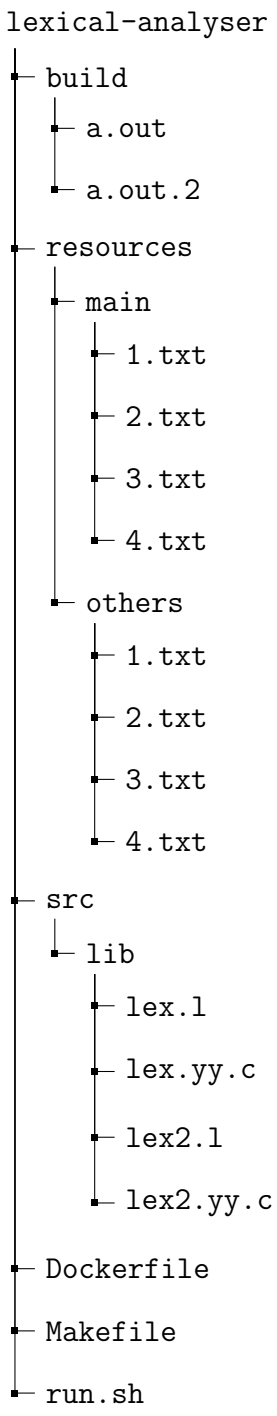


Figura 1: Analisador léxico

2 Implementação

Para implementar os analisadores léxicos foi seguido como base teórico o material disponibilizado: *A Guide to Lex Yacc* [1].

O trabalho segue a seguinte estrutura de pastas:



Sendo que na pasta *build* temos os arquivos executáveis, na pasta *resources* temos outras duas pastas, a pasta *main* que contém os arquivos de entrada do arquivo **lex.l** e a pasta *others* contém os arquivos de entrada do arquivo **lex2.l**, temos a pasta *src/lib* que contem os arquivos *.l* e *.c*. Além disso, temos o arquivo *Dockerfile* sendo que podemos executar o trabalho criar uma imagem docker executando os comandos a seguir:

```
docker build -t lexical-analyzer .
docker run lexical-analyzer
```

Temos o arquivo *Makefile* para compilar e executar os testes e por fim arquivo o *run.sh*

que executa os quatro casos de teste para cada analisador léxico criado.

Para compilar e arquivo principal (lex.l) e executar os quatro arquivos de testes execute:

```
make
make main-first-test
make main-second-test
make main-third-test
make main-fourth-test
```

Para compilar e arquivo secundário (lex2.l) no contexto de um banco e executar os quatro arquivos de testes execute:

```
make bank
make bank-first-test
make bank-second-test
make bank-third-test
make bank-fourth-test
```

2.1 Arquivo Principal - lex.l

2.1.1 Padrões

Nesse arquivo principal os padrões a serem reconhecidos pelo analisador léxico devem ser:

1. Espaços em branco, tabulação e quebra de linha devem ser ignorados.
2. Número inteiro positivo: qualquer sequencia de um ou mais dígitos, precedidos ou não do símbolo +.
3. Número inteiro negativo: qualquer sequencia de um ou mais dígitos, precedidos do símbolo -.
4. Número decimal: qualquer sequencia de um ou mais dígitos seguida de um ponto (.) e de outra sequencia de um ou mais dígitos. Obs.: o número decimal também pode ser positivo ou negativo.
5. Placa: três letras maiúsculas seguidas de um hífen (mesmo caractere dos números negativos) e de 4 dígitos.
6. Palavra: qualquer sequencia de uma ou mais letras maiúsculas ou minúsculas (sem caractere especial ou letras acentuadas).

7. Telefone: 4 dígitos seguidos de um hífen (mesmo caractere dos números negativos) e de mais 4 dígitos.

8. Nome próprio: três ou quatro palavras, tendo necessariamente e exatamente um espaço em branco entre cada par dessas palavras. Um espaço ao final não é necessário.

2.1.2 Código

Para implementar os padrões descritos anteriormente, foi utilizado o código a seguir:

```
/* \t => tabulação; \n => quebra de linha; */
ws [ \t\n]

digit [0-9]
positive [+]?{digit}+
negative [-]{digit}+
decimal [+]?[-]?{digit}+[.]{digit}+
/* or decimal ({positive}|{negative})+[.]{digit}+ */

low [a-z]
upper [A-Z]
word ({low}|{upper})+

license-plate {upper}{3}[-]{digit}{4}
phone {digit}{4}[-]{digit}{4}
name {word}[ ]{word}[ ]{word}([ ]{word})?

%%
{ws}+  {/* 1 */}
{positive} {printf(...)} {/* 2 */}
{negative} {printf(...)} {/* 3 */}
{decimal} {printf(...)}  {/* 4 */}
{license-plate} {printf(...)} {/* 5 */}
{word} {printf(...)} {/* 6 */}
{phone} {printf(...)} {/* 7 */}
{name} {printf(...)} {/* 8 */}
. { printf("Token não reconhecido. LEXEMA: %s\n", yytext); }
%%
```

2.1.3 Arquivos de entrada

Além do arquivo principal de entrada já especificado foram criados outros três arquivos .txt para fins de explicação utilizaremos o arquivos *4.txt*.

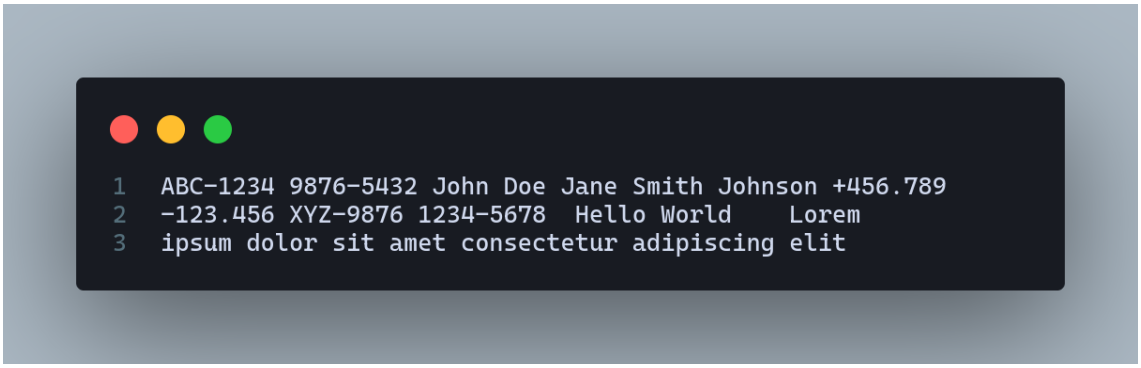


Figura 2: Entrada 4.txt

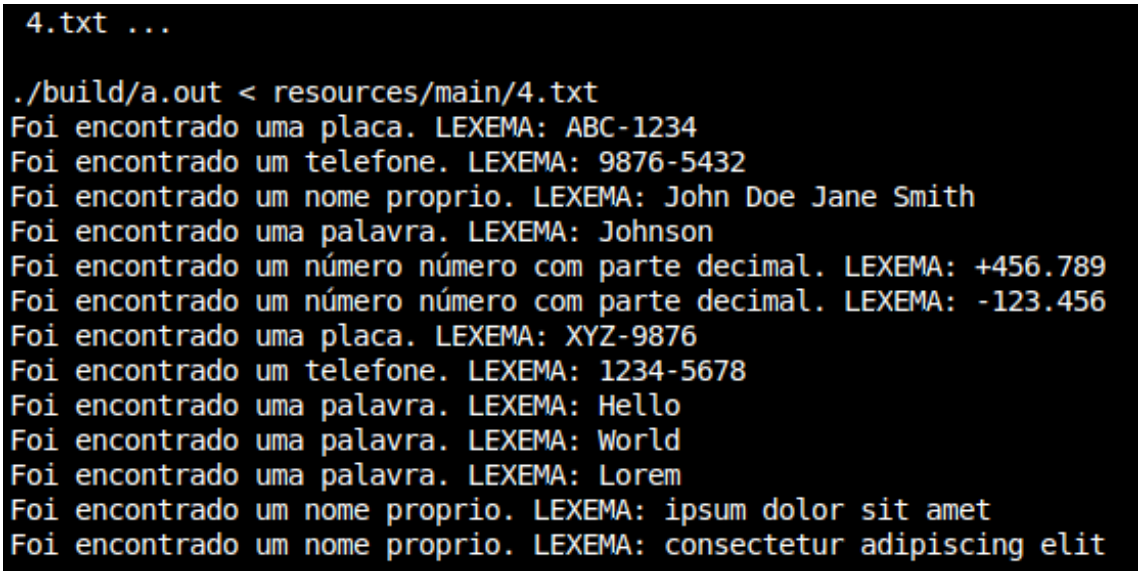


Figura 3: Sáida da entrada 4.txt

Para cada um dos resultados encontrados, iremos fazer uma enumeração com a finalizade de elucidar melhor o resultado atingido:

1. placa (upper3[-]digit4): sendo assim, ABC-1234 atende
2. telefone: temos 4 dígitos seguidos de "-" e por fim 4 dígitos
3. nome própria: seguida de letras (uma palavra) separadas por espaço sendo que um numero próprio deve ter 3 ou 4 palavras
4. palavra(Johnson): sequência de letras que não se encaixa como nome próprio pela ausencia de outras 2 ou 3 palavras em sequência

5. número decimal: sequencia de dígitos seguido do símbolo "." e posteriormente seguido por uma sequência de dígitos

6. número decimal: sequencia de dígitos seguido do símbolo "." e posteriormente seguido por uma sequência de dígitos

7. placa (upper3[-]digit4): sendo assim, XYZ-9876 atende

8. telefone: temos 4 dígitos seguidos de "-" e por fim 4 dígitos

9. palavra (Hello): sequência de letras que não se encaixa como nome próprio pela ausencia de outras 2 ou 3 palavras em sequência

10. palavra (World): sequência de letras que não se encaixa como nome próprio pela ausencia de outras 2 ou 3 palavras em sequência

11. palavra (Lorem): sequência de letras que não se encaixa como nome próprio pela ausencia de outras 2 ou 3 palavras em sequência, vale observar que a sequência Hello World Lorem não se encaixou como nome própria pela presença de uma tabulação após a palavra *World*

12. nome própria: seguida de letras (uma palavra) separadas por espaço sendo que um numero próprio deve ter 3 ou 4 palavras

13. nome própria: seguida de letras (uma palavra) separadas por espaço sendo que um numero próprio deve ter 3 ou 4 palavras

2.2 Arquivo Secundário - lex2.l

2.2.1 Padrões

Para o segundo analisador léxico (*lex2.l*) o contexto escolhido foi de um banco digital, sendo assim os padrões a serem reconhecidos pelo analisador léxico devem ser:

1. **Número de conta bancária**: Uma sequência de dígitos com tamanho fixo 10.

2. **CPF (Cadastro de Pessoa Física)**: Uma sequência de 3 digitos seguido do símbolo "\", depois 3 dígitos seguido do símbolo "\", depois 3 dígitos seguido do simbolo "-" e por fim 2 dígitos. Formato = xxx.xxx.xxx-xx

3. **CNPJ (Cadastro Nacional de Pessoa Jurídica)**: Uma sequência de 14 dígitos. Formato = xx.xxx.xxx/xxxx-xx

4. **Data de nascimento**: Uma sequência no formato dd/mm/aaaa.

5. **Código de segurança (CVV)**: Uma sequência de três dígitos.

6. **Senha**: Uma sequência de caracteres alfanuméricos e especiais com tamanho mínimo 8 e máximo 15.
7. **E-mail**: Uma sequência de caracteres alfanuméricos, pontos, underscores, hífens e arrobas.
8. **Agência bancária**: Uma sequência de dígitos com tamanho fixo 4.
9. **Número de cartão de crédito**: Uma sequência de 16 dígitos.
10. **Tipo de conta bancária**: Uma sequência de caracteres letras em caixa alta.
11. **Código de barras**: Uma sequência numérica com tamanho fixo 44.
12. **Número do cheque**: Uma sequência numérica com tamanho fixo 6.
13. **CEP**: Uma sequência de 5 dígitos seguidos de um símbolo "-" e posteriormente seguido de 3 dígitos.
14. **Token de segurança avançado**: Uma sequência alfanumérica com tamanho variável (entre 8 e 16) e caracteres especiais.
15. **Número de lote de processamento**: Uma sequência alfanumérica com tamanho fixo 6.
16. **Hora**: Uma sequência no formato hh:mm:ss.
17. **Código de verificação de imagem (CAPTCHA)**: Uma sequência alfanumérica com tamanho fixo 5.

2.2.2 Código

Para implementar os padrões descritos anteriormente, foi utilizado o código a seguir:

```
/* definicoes regulares */

/* \t => tabulação; \n => quebra de linha; */
ws [ \t\n]

digit [0-9]

low [a-z]
upper [A-Z]

bank-account {digit}{10}
```

```
/* cpf =>  000\. 000\. 000-00*/
cpf {digit}{3}[.]{digit}{3}[.]{digit}{3}[-]{digit}{2}

/* CNPJ =>  XX. XXX. XXX/0001-XX */
cnpj {digit}{2}[.]{digit}{3}[.]{digit}{3}[/]{digit}{4}[-]{digit}{2}

/* dd/mm/aaaa */
birth {digit}{2}[/]{digit}{2}[/]{digit}{4}
/* xxx */
CVV {digit}{3}

password [a-zA-Z0-9@#$%^&!+=]{8,15}
email [A-Za-z0-9_.-]+[@][A-Za-z0-9_.-]+[.][A-Za-z]{2,3}
bank-branch {digit}{4}
credit-card {digit}{16}
bank-account-type {upper}+
bar-code {digit}{44}
check-number {digit}{6}
CEP {digit}{5}[-]{digit}{3}
token [A-Za-z0-9!@$#%^&*()_+=-]{8,16}
processing-batch [A-Z0-9]{6}
hour {digit}{2}[:]{digit}{2}[:]{digit}{2}
captcha [A-Za-z0-9]{5}

%%
{ws}+ { /*Espaços em branco, tabulação e quebra de linha devem ser ignorados.*/ }
{bank-account} {{ printf(...); }} {/* 1 */}
{cpf} {{ printf(...); }} {/* 2 */}
{cnpj} {{ printf(...); }} {/* 3 */}
{birth} {{ printf(...); }} {/* 4 */}
{CVV} {{ printf(...); }} {/* 5 */}
{password} {{ printf(...); }} {/* 6 */}
{email} {{ printf(...); }} {/* 7 */}
{bank-branch} {{ printf(...); }} {/* 8 */}
{credit-card} {{ printf(...); }} {/* 9 */}
{bank-account-type} {{ printf(...); }} {/* 10 */}
{bar-code} {{ printf(...); }} {/* 11 */}
{check-number} {{ printf(...); }} {/* 12 */}
```

```
{CEP} {{ printf(...); }} {/* 13 */}
{token} {{ printf(...); }} {/* 14 */}
{processing-batch} {{ printf(...); }} {/* 15 */}
{hour} {{ printf(...); }} {/* 16 */}
{captcha} {{ printf(...); }} {/* 17 */}
. { printf("Token não reconhecido. LEXEMA: %s\n", yytext); }
%%
```

2.2.3 Arquivos de entrada

Para este contexto foram desenvolivos quatro arquivos de entrada, porém para fins de explicação utilizaremos o arquivos *4.txt*.

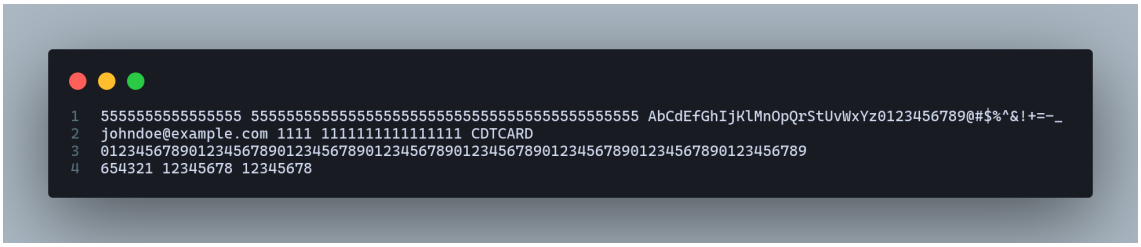


Figura 4: Entrada 4.txt

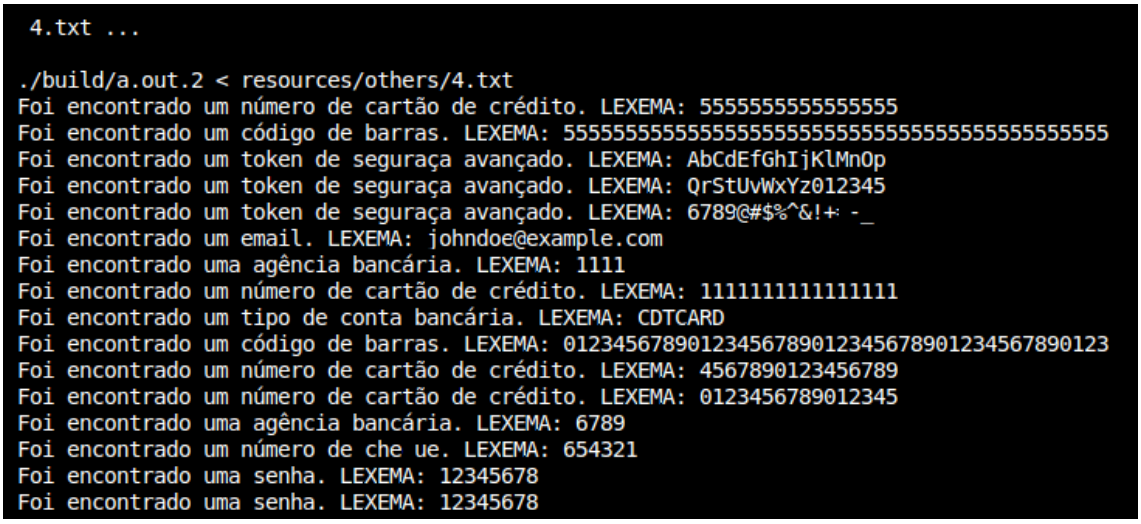


Figura 5: Resultado da entrada 4.txt

Para cada um dos resultados encontrados, iremos fazer uma enumeração com a finalizade de elucidar melhor o resultado atingido:

1. cartão de crédito: sequência de 16 dígitos
2. código de barras: sequência de 44 dígitos
3. token de segurança: sequência alfanumérica com tamanho 16

4. token de segurança: sequência alfanumérica com tamanho 16
5. token de segurança: sequência alfanumérica com tamanho 15
6. e-mail: sequência de símbolos ou letras ou dígitos sequido do símbolo "@", seguido de uma sequência de símbolos ou letras ou dígitos, seguido do símbolo ".", seguido de 2 ou 3 letras maiúsculas ou minúsculas
7. agência bancária: sequência de 4 dígitos
8. cartão crédito: sequência de 16 dígitos
9. tipo de conta bancária: sequencia de letras em maiúsculas
10. código de barras: sequência de 44 dígitos
11. cartão de crédito: sequência de 16 dígitos
12. cartão de crédito: sequência de 16 dígitos
13. agência bancária: sequência de 4 dígitos
14. número de cheque: sequência de 6 dígitos
15. senha: sequência de letras, dígitos ou simbolos com tamanho 8
16. senha: sequência de letras, dígitos ou simbolos com tamanho 8

References

[1] A guide to lex yacc. https://redirect.cs.umbc.edu/courses/331/papers/compactGuideLexYacc.pdf. (Accessed on 14/05/2023).