UNIVERSIDADE FEDERAL DE VIÇOSA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RONIEL NUNES BARBOSA - 3464

Trabalho Prático - CCF 441 - Compiladores

Primeiro trabalho prático da disciplina CCF 441 - Compiladores, do curso de Ciência da Computação da Universidade Federal de Viçosa - Campus Florestal/MG.

Professor(a): Daniel Mendes Barbosa

Florestal/MG
2022

Introdução

O trabalho prático teve como objetivo colocar em prática os conhecimentos adquiridos em ambiente de sala de aula. No trabalho foi realizada a criação de dois arquivos, sendo eles: lex.l e lex2.l.

O primeiro arquivo foi criado para reconhecimento de tokens conforme a definição presente na especificação do trabalho prático. Já o segundo arquivo se baseou em um analisador léxico que reconhece um padrão de lexema estabelecido pelo aluno. Dessa maneira, os lexemas identificados fazem referência a uma carteira de identificação contendo informações como: rg, data, nome, cpf, naturalidade, doc origem e via.

Execução

Para realizar a compilação e execução desse trabalho prático, é necessário ter como sistema operacional uma distribuição Linux ou em caso da utilização do sistema Windows, o Windows Subsystem for Linux(WSL) instalado em sua máquina.

Para executar basta seguir os procedimentos listados abaixo:

- Compilação:
    1. flex lex.l
    2. gcc lex.yy.c
- Execução
    3. ./a.out ou ./a.out < entrada.txt

Ferramentas e tecnologias utilizadas

- Visual Studio Code
- Flex

Desenvolvimento

Neste tópico abordaremos os códigos presente nos arquivos lex.l e lex2.l e será discutido as decisões tomadas para realização do trabalho.

Arquivo Lex.l

Abaixo temos o código fonte do analisador lex.l, contendo as definições regulares desenvolvidas.

```
%{
    /*código colocado aqui aparece no arquivo gerado pelo flex*/
#include <stdio.h>
%}

/*isso diz ao flex para ler apenas um arquivo de entrada*/
%option noyywrap

/* definições regulares */

delim                   [ \t\n]
ws                      {delim}+

inteiro_positivo         (\+?)[0-9]+
inteiro_negativo         (\-?)[0-9]+
numero_decimal           ((\-|\+)?)[0-9]+(\.)[0-9]+
placa                    [A-Z][A-Z][A-Z](\-)[0-9][0-9][0-9][0-9]
palavra                  [A-Za-z]+
telefone                 [0-9][0-9][0-9][0-9](\-)[0-9][0-9][0-9][0-9]
nome_proprio             {palavra}[" "]{palavra}[" "]{palavra}[" "]{palavra}*

%%

{ws}                            {/*nenhuma acao e nenhum retorno*/}
{inteiro_positivo}              printf("Foi encontrado um numero inteiro positivo. LEXEMA: %s\n", yytext);
{inteiro_negativo}              printf("Foi encontrado um numero negativo negativo. LEXEMA: %s\n", yytext);
{numero_decimal}                printf("Foi encontrado um numero com parte decinal. LEXEMA: %s\n", yytext);
{placa}                         printf("Foi encontrado uma placa. LEXEMA: %s\n", yytext);
{palavra}                       printf("Foi encontrado uma palavra. LEXEMA: %s\n", yytext);
{telefone}                      printf("Foi encontrado um telefone. LEXEMA: %s\n", yytext);
{nome_proprio}                  printf("Foi encontrado um nome proprio. LEXEMA: %s\n", yytext);

%%

/*codigo em C. Foi criado o main, mas podem ser criadas outras funcoes aqui.*/

int main(void)
{
    /* Call the lexer, then quit. */
    yylex();
    return 0;
}
```

Fonte: Visual Studio Code

Esse primeiro código, contém os tokens descritos na especificação. Temos a criação das expressões regulares para chegar em uma resposta satisfatória ao que foi pedido. Foram criadas expressões para reconhecer lexemas contendo números inteiros positivos, negativos, números decimais, placas, palavras, telefone e nomes próprios.

Em reflexão à essa primeira parte, pode-se dizer que a dificuldade inicial foi aprender a sintaxe correta para a criação de definições regulares que satisfizesse o que era pedido. Contudo, com pesquisas realizadas foi possível desenvolver sem grandes problemas.

Arquivo Lex2.l

Abaixo temos o código fonte do analisador lex2.l, contendo as definições regulares desenvolvidas.

```
%{
    /*Definições de constantes em manifesto.*/
#include <stdio.h>
%}

/*isso diz ao flex para ler apenas um arquivo de entrada*/
%option noyywrap

/* definicoes regulares */

delim                   [ \t\n]
ws                      {delim}+

palavra                 [A-Za-z]+

rg                      [A-Z]{2}[-][0-9]{2}[.][0-9]{3}[.][0-9]{3}
data                    [0-9]{2}[/][0-9]{2}[/][0-9]{4}
nome                    {palavra}[" "]{palavra}[" "]{palavra}[" "]{palavra}*
cpf                     [0-9]{3}[.][0-9]{3}[.][0-9]{3}[-][0-9]{2}
naturalidade            [A-Z]+[\-][A-Z]{2}
doc_origem              [A-Z]{4}[.][" "][A-Z]{2}[-][0-9]{3}[" "][A-Z]{2}[-][0-9]{2}
via                     [1-9]+[.][V][I][A]

%%

{ws}                            {/*nenhuma acao e nenhum retorno*/}
{rg}                            printf("Foi encontrado um registo geral. LEXEMA: %s\n", yytext);
{data}                          printf("Foi encontrado uma data. LEXEMA: %s\n", yytext);
{nome}                          printf("Foi encontrado um nome proprio. LEXEMA: %s\n", yytext);
{cpf}                           printf("Foi encontrado um CPF. LEXEMA: %s\n", yytext);
{naturalidade}                  printf("Foi encontrado uma naturalidade. LEXEMA: %s\n", yytext);
{doc_origem}                    printf("Foi encontrado o doc origem. LEXEMA: %s\n", yytext);
{via}                           printf("Foi encontrado a via. LEXEMA %s\n", yytext);

%%

/*codigo em C. Foi criado o main, mas podem ser criadas outras funcoes aqui.*/

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

Fonte: Visual Studio Code

Este código fonte segue a especificação criada pelo aluno, onde temos o tema de lexema fazendo referência a informações contidas em um documento já citado anteriormente.

Os padrões que foram seguidos para reconhecimento pelo analisador léxico são:

- Espaços em branco, tabulações e quebras de linha são ignorados

- Registro Geral: seguindo o padrão sigla do estado, hífen, 2 dígitos, ponto, 3 dígitos, ponto, 3 dígitos.
- Data: seguindo o padrão dia / mês / ano. Exemplo: 15/02/1946.
- Nome: três ou mais palavras separadas por espaços.
- CPF: seguindo o padrão 3 dígitos, ponto, 3 dígitos, ponto, 3 dígitos, hífen, 2 dígitos.
- Naturalidade: cidade natal em maiúsculo, concatenado com um hífen, seguido pela sigla do estado.
- Doc origem: segue o padrão de caracteres 4 caracteres em maiúsculo, ponto, espaço, 2 caracteres, hífen, 3 dígitos, espaço, 2 caracteres, hífen, 2 caracteres.
- Via: seguindo o padrão, dígito, ponto, palavra “VIA”.

Esses foram os padrões seguidos para realizar a criação do arquivo 2.

Resultados das execuções

Nesta parte do trabalho será mostrado os arquivos de entrada e saída. Temos abaixo o arquivo de entrada para o lex.l e sua respectiva saída, sendo o mesmo disponibilizado pelo professor.

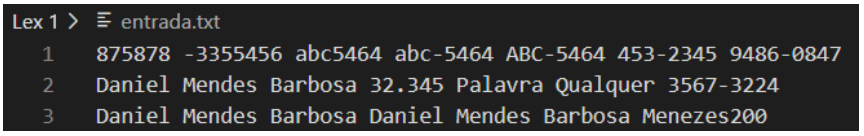

Figura 01: arquivo de entrada.txt

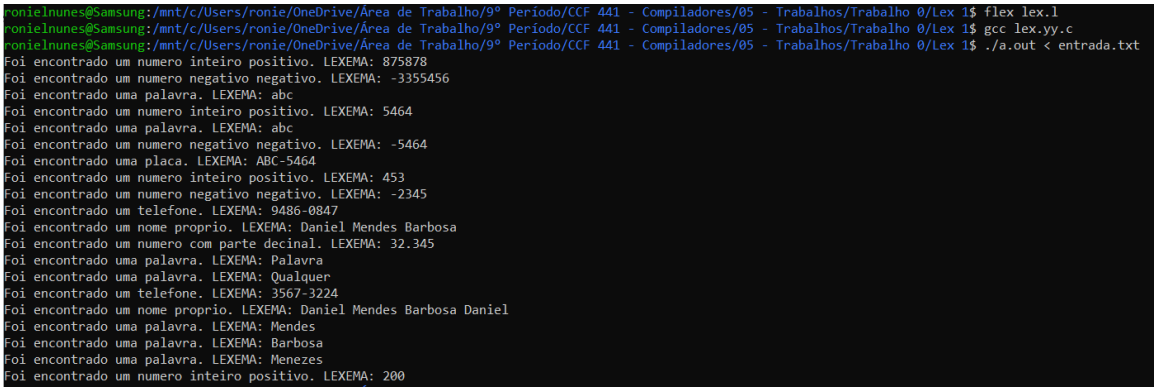

Figura 02: resultado do arquivo de entrada.txt para o lex.l

Podemos ver que os resultados foram os mesmos especificados na documentação. Assim o arquivo lex.l satisfaz os tópicos pedidos na atividade prática.

Agora discutiremos o segundo código fonte produzido. Contudo, como já foi supracitado, o lex2.l faz referência a informações contidas em uma carteira de identidade. Ou seja, seguem padrões de informações pessoais, assim o arquivo de teste produzido não contém informações reais, mas sim sua forma de escrita correspondente.

Figura 03: arquivo de entrada para o lex2.l

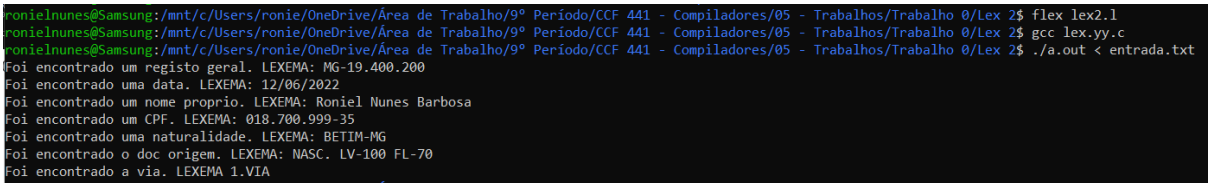

Figura 04: resultado do arquivo de entrada.txt para o lex2.l

Como podemos ver os resultados satisfizeram os padrões estabelecidos e também o arquivo de entrada. Dessa maneira, podemos concluir que as especificações foram concluídas de forma correta.

Conclusão

O desenvolvimento desse trabalho prático baseou-se na criação de analisadores léxicos com o auxílio da ferramenta Flex para gerar o analisador a partir de definições regulares. Um desafio presente neste trabalho foi entender inicialmente a sintaxe de forma correta, sendo resolvida com a pesquisa de materiais didáticos que visam documentar exemplos e apresentar a sintaxe.

Contudo, foi uma experiência bastante produtiva e positiva para o desenvolvimento na disciplina, uma vez que mostra por meio da prática uma ferramenta que facilita na criação de uma das etapas de um compilador.

Referências

GESSER, Carlos. FURTADO, Olinto. GALS GERADOR DE ANALISADORES LÉXICOS E SINTÁTICOS. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/183898/Gals.pdf?sequence=-. Acesso em 12 de junho de 2022.

DEI/ISEP. Introdução ao FLEX e expressões regulares. Disponível em: https://www.dei.isep.ipp.pt/~goreti/ficha1.pdf. Acesso em 12 de junho de 2022.

UNICAMP .Especificação das sentenças regulares. Disponível em: https://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node52.html.Acesso em 12 de junho de 2022.