Trabalho Prático 3 de Processamento de Linguagens

André Vieira (A78322) Eduardo Rocha (A77048)
Ricardo Neves (A78764)

2 de Junho de 2018

Resumo

Este documento apresenta o relatório do terceiro projeto de Processamento de Linguagens, de Mestrado Integrado em Engenharia Informática da Universidade do Minho. Aqui, será realizada a descrição de todo o trabalho realizado pelo grupo, que consistiu em escolher uma gramatica e desenvolver um reconhecador lexico.

Conteúdo

1 Introdução

Este trabalho prático foi realizado no ambito da Unidade Curricular de Processamento de Linguagens. Aqui, iremos dicutir toda a estratégia e linha de pensamento que o grupo tomou, de modo a cumprir os requisitos pedidos.

Uma vez que o menor número mecanográfico é igual a 77048, e ao dividir este mesmo número por 8, constatamos que o resto desta operação é igual a 0. Assim, 0+1=1, o que corresponde ao número do exercicio a realizar.

Deste modo, o grupo constatou que o enunciado atribuido foi o 3.1 - Rede Semântica do Museu da Emigração. Este trabalho engloba, em geral, 2 exercícios distintos, que iremos mencionar mais à frente, em pormenor. O objetivo máximo do grupo em relação a este trabalho foi o de realizar o exercício com a maior eficiência, ganhando, assim, experiência e conhecimento em relação ao FLEX e a reconhecedores léxicos, que irá ser importante no seguimento desta Unidade Curricular. Com o fecho da introdução deste relatório, é adequado mencionar a estrutura do mesmo. Este relatório contém o resumo do trabalho que foi realizado durante o período dado para tal, a descrição e a implementação do exercício que do trabalho prático (incluíndo a linha de pensamento seguida pelo grupo, bem como imagens que ajudam à interpretação), e no final, uma pequena conclusão que reflete o trabalho realizado.

2 Resumo

Neste terceiro exercício, pretende-se descrever parte da rede semantica que suporta o Museu da Emigração, das Comunidades e da Luso-Descendencia. Deste modo, gera-se um grafo (usando GraphVIZ ou DOT) que permita uma navegação sobre esta base de dados. Esta rede conta com 3 nodos, ou vértices: o emigrante (caracterizado pelo nome, idade, concelho, destino e código), a obra (podendo se tratar de um palácio, fábrica, hospital, etc.) ou um evento (por exemplo, baile, sarau, etc.). A isto, alia-se os nodos entre os vértices: fez (que liga o emigrante a uma obra), e participou (que liga um emigrante a um evento).

3 Rede

De modo a ir testando a implementação do grupo, criamos uma pequena rede com exemplos retirados do site indicado pelos docentes. Assim, esta é a rede semantica utilizada, e que será descrita na forma de um grafo:

```
 9  emigrante:
10  JoseCoutinho,
11  14,
12  Estoraos,
13  Brasil,
14  8A1834,
15  ;
16
17  emigrante:
18  FranciscoMeneses,
19  17,
20  Quinchaes,
21  Brasil,
22  9A1834,
23  ;
24
25  obra:
26  Palacio,
27  ;
28
29  obra:
30  Fabrica,
31  ;
```

Figura 1: Rede semântica de teste

Em cima, podemos verificar que o tipo do nodo é seguido por dois pontos ':' e, só depois, todos os dados relativos a esse emigrante, por exemplo. No fim de um dado, deve-se acabar com uma vírgula. Para fechar este nodo, utiliza-se o ponto e virgula. Aqui, o programa irá saber que este nodo terminou, e irá procurar por mais. Esta foi a gramática escolhida consensualmente pelo grupo, depois de vários testes a outras gramáticas.

4 YACC

Uma vez concebida a gramática do projeto, e todos os dados inseridos na mesma, é altura de ler esse ficheiro e armazenar em estruturas de dados, essas mesmas informações. O grupo decidiu dividir todas as informações nos seus Arrays de Strings apropriados. Por exemplo, todos os nomes de todos os emigrantes estão guardados no Array Nome. Com isto, é muito mais fácil, posteriormente, construir o grafo. Para isto, foi implementado um ficheiro YACC, onde dividimos a grmática em vários blocos. Cada bloco criado é responsável pela leitura e junção de todas as informações que são recebidas pelo ficheiro onde se encontram todos os dados dos emigrantes, obras e eventos.

```
{ $$ = "\n"; }
{
    if(i == 4){
        strcpy(nome[contNome++], $1);
        i = 0;
    }
    else {
        if(i == 1){
            strcpy(destino[contDestino++], $1);
            i++;
        }
        else{
            if(i == 0){
                strcpy(codigo[contCodigo++], $1);
                i++;
            }
            else{
                if(i == 2){
                    strcpy(concelho[contConcelho++], $1);
                    i++;
                }
                else{
                    if(i == 3){
                        strcpy(idade[contIdade++], $1);
                        i++;
                    }
                }
            }
        }
    }
}
```

Figura 2: Excerto do ficheiro YACC

Em cima, apresentamos um excerto do YACC criado para a leitura da gramática desenvolvida. Colocar aqui todo o ficheiro YACC não é o mais aconselhado, devido á dimensão do mesmo. No entanto, neste excerto de código, podemos ver a parte responsável por ler e armazenar os dados dos vários emigrantes. Assim, usamos várias variáveis auxiliares e, dependendo da ordem em que os dados apareciam, eram guardados no Array destinado.

5 FLEX

O código YACC acima referido tem de ser auxiliado por uma implementação FLEX que ajuda no parse dos dados.

```
1  %option noyywrap yylineno
2
3  %%
4
5  emigrante       { return EMI; }
6  obra            { return OBR; }
7  evento          { return EVE; }
8  fez             { return FEZ; }
9  participou      { return PAR; }
10 [a-zA-Z]*       { yylval.s = strdup(yytext); return STR; }
11 [0-9]*          { yylval.s = strdup(yytext); return STR; }
12 \_              { yylval.s = strdup(yytext); return STR; }
13 ,|;|:           { return yytext[0]; }
14 [ \n\t. ]       { }
15
16
17
18 %%
```

Figura 3: Ficheiro FLEX

Como podemos ver, utilizamos uma série de expressões regulares para dividir os dados. Por exemplo, sempre que é encontrada a palavra emigrante, o FLEX dá a conhecer ao YACC que irá ler, de seguida, os dados de um emigrante. Isto é igual para o resto das entidades. Foram também criadas expressões regulares para a leitura de palavras, números e underscores. Tudo o resto será ignorado pelo programa.

6 Resultado Final

Para um teste mais rápido deste exercício, o grupo criou a seguinte Makefile:

```
1  tp3: Emigrante.y Emigrante.fl
2       flex Emigrante.fl
3       yacc Emigrante.y
4       cc -o tp3 y.tab.c
5       ./tp3 < Emigrante.txt > Graph.dot
6       dot -Tpng Graph.dot > Graph.png
7       display Graph.png
8
```

Figura 4: Makefile do exercício

Ao correr esta Makefile, será aberta uma imagem do grafo concebido ao longo do exercício.

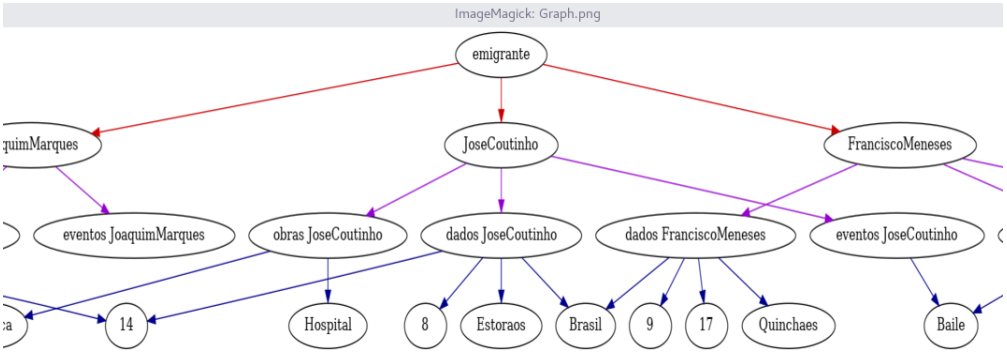

Figura 5: Grafo resultante

Como se pode observar, no topo, estão presentes todos os emigrantes da base de dados. Cada emigrante desenvolve-se em 3 grupos: dados, obras e eventos a que o emigrante foi. Em baixo de cada nodo, é apresentada a informação relativa.

Depois de algumas pesquisas, o grupo não conseguiu implementar um grafo interativo onde, clicando em cada nodo, iria ser possível observar toda a informação contida no mesmo. Temos a noção que é um ponto onde o grupo falhou e que, com mais tempo e uma pesquisa mais aprofundada, este requisito seria completamente coberto.

7 Conclusões

Com este trabalho prático, adquirimos e, maioritariamente, aprofundamos os nossos conhecimentos acerca do gerador FLEX e do YACC. Com isto, constatamos toda a utilidade que estas ferramentas, juntas, nos podem oferecer, sendo que agora, com estes exercícios, conseguimos dominar melhor as mesmas. Em suma, estamos satisfeitos com o trabalho realizado até então, sendo que o grupo se sente preparado para, após este desenvolvimento dos conhecimentos em FLEX e YACC que irão ser importantes no futuro.