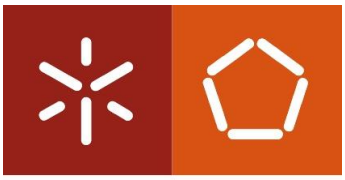

Universidade do Minho

Departamento de Informática

DSL para Cadernos de Anotações em HD

TP2 - Exercício 2 - Grupo 7

João Teixeira (A85504)
José Filipe Ferreira (A83683)
Miguel Solino (A86435)

7 de julho de 2020

Resumo

O objetivo deste projeto é construir um processador para cadernos de anotações, sendo capaz de converter um conjunto de cadernos num site *HTML* bem estruturado.

Conteúdo

Capítulo 1

Introdução

Com este projeto pretende-se processar um conjunto de cadernos de anotações, com o formato descrito no enunciado fornecido.

Para a construção de um processador capaz disto, primeiro foi escrita uma gramática capaz de definir a DSL de *caderno de anotação*. Em seguida foi escrito um filtro de texto com recurso a *FLex* de forma a ser capaz de fazer corresponder o texto de entrada à gramática anteriormente descrita. Para finalizar, com recurso a *yacc*, foram definidas as ações que correspondem a cada parte da gramática para efetuar o *parsing* e posterior criação do site *HTML*.

Ao longo deste relatório iremos expor o problema e explicar a solução encontrada, começando por falar da arquitetura e estruturas de dados, explicando a gramática definida e entrando em maior detalhe sobre como é efetuado o processamento de cada caderno. Por fim, iremos apresentar também um pequeno manual de utilização do programa criado.

Capítulo 2

Problema

O programa tem que cumprir os seguintes requisitos:

- Reconhecer e validar vários cadernos de anotações;
- Criar uma página *HTML* por cada tópico, sujeito ou objeto presente no caderno agrupando toda a informação respetiva ao mesmo;
- Incluir na página de um sujeito uma imagem sempre que apareça no caderno um par do tipo *:x :img "img.jpg"*;
- Ter a possibilidade de referir triplos inversos por forma a melhorar a organização da informação no site criado;

Capítulo 3

Solução

3.1 Arquitetura do Projeto

A nossa solução divide-se em dois módulos: um correspondente ao filtro de texto e um correspondente à parte da gramática, armazenamento e processamento do ficheiro de entrada e criação do site *HTML*.

Quanto a estruturas de dados, numa primeira iteração do desenvolvimento apenas existia uma lista contendo informação referente aos triplos inversos, sendo todo o resto escrito diretamente na página correspondente ao sujeito em processamento.

Mas por forma a obter um aspeto mais consistente em todas as páginas construídas, são agora armazenadas todas as informações dos triplos. Para conseguirmos isto, criamos estruturas de dados semelhantes ao formato dos triplos. Sendo este um *HashMap* indexado por sujeito e fazendo corresponder a cada índice um segundo *HashMap*, sendo este indexado pela relação e contendo uma lista de objetos.

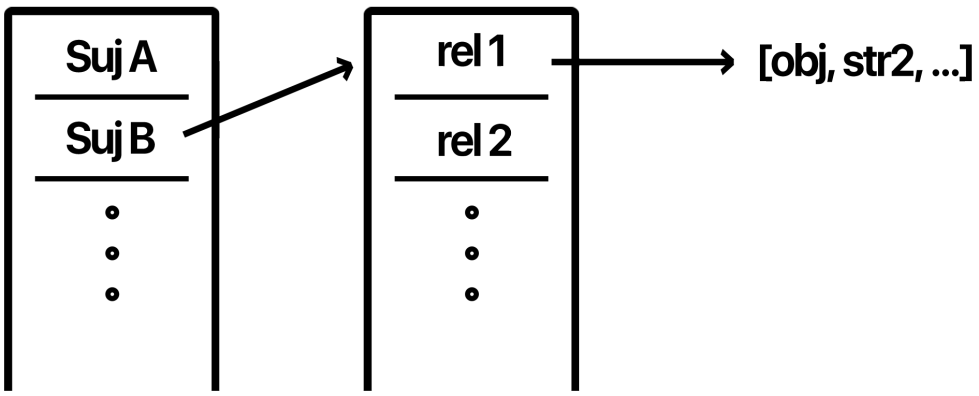

Figura 3.1: Esquema da estrutura de dados do programa

3.2 Especificação da gramática

3.2.1 Símbolos terminais

Para a especificação desta gramática, foram definidos nove símbolos terminais, sendo a maioria marcadores ou separadores de estados (INIT, TR, IMG, MT, INV). Os restantes correspondem a dados vindos do ficheiro de entrada, sendo estes, o STR, que corresponde a quando um objeto de um triplo é uma string, o CONTEUDO, correspondendo a um parágrafo do texto de um tópico, o TIT que corresponde ao título do texto de um tópico e por fim os URI, que correspondem aos tópicos, sujeitos ou objetos (normalmente começados por *:*).

3.2.2 Meta

Esta secção corresponde ao reconhecimento do ficheiro que contém os triplos inversos, reconhecidos com a ajuda do símbolo terminal INV, retornado pelo flex quando é encontrada a relação *:inverseOf*.

3.2.3 Conceito

Um caderno é um conjunto de pares (Documento, Triplos), sendo a representação de cada um deste pares um conceito. É aqui que é utilizado o símbolo terminal TR, que marca a separação entre o Documento e os Triplos.

3.2.4 Documento

Um Documento é onde é reconhecido o texto de um conceito, e é separado em quatro partes, um símbolo terminal INIT, que marca o inicio de um documento, o Sujeito, dizendo o objeto ao qual o documento se refere, um título, com o título do texto, e o conteúdo, sendo este um conjunto de parágrafos, que formam o texto em si.

3.2.5 Triplos

Os Triplos correspondem ao reconhecimento dos triplos em notação turtle, sendo estes com a forma (sujeito, relação, objeto), suportando a possibilidade de por sujeitos iguais em evidência. Na nossa gramática cada triplo, ou triplos com o mesmo sujeito, é representado começando com um Sujeito do triplo e a uma lista de pares explicados abaixo.

3.2.6 Par

Um Par nesta gramática corresponde a uma Relação e a uma lista de objetos e strings, denominada de Comps.

3.2.7 Rel

Rel corresponde às relações entre os sujeitos e objetos, ao segundo elemento dos triplos da notação turtle. Existem vários tipos de relações, existindo duas que possuem um comportamento diferente das outras, a relação 'a' e a relação ':img'. Numa relação do tipo 'a' os objetos correspondem ao tipo do sujeito do triplo. Numa relação do tipo ':img' é especificada uma imagem para ser mostrada no site *HTML*, na pagina do sujeito do triplo.

3.3 Processamento de um Caderno

Para processar um caderno, constituído por vários pares (documento, triplos), processam-se cada um dos pares.

Para processar um destes pares, começa-se por validar e tratar a parte do documento, verificando se começa pelo identificador de início de documento, valida-se o sujeito, e cria-se a sua respetiva página *HTML* caso ainda não exista, e adiciona-se informação sobre este na pagina do índice. Depois é processado o título do documento, escrevendo-o com a formatação respetiva para títulos *HTML* no ficheiro do sujeito. Em seguida é processado o texto em si, formatando-o e escrevendo-o diretamente na página do sujeito em questão.

Depois do processamento da parte inicial do par, passa-se à parte dos triplos. O processamento dos triplos corresponde processar recursivamente pares do tipo (Sujeito, Pares). O processamento do sujeito decorre da forma anteriormente explicada. Para processar os Pares, que é uma lista de pares (Rel, Comps), percorre-se esta lista recursivamente, processando cada um dos pares individualmente.

Para processar um par do tipo (Rel, Comps), começamos por processar o primeiro elemento, Rel, que corresponde ao segundo elemento dos tripos em notação turtle. Depois de validada pelo flex, é procurada a relação inversa e guardada, para quando forem processados os objetos presentes no terceiro e ultimo elemento dos triplos da notação turtle, ser possível guardar de imediato nas nossas estruturas de dados a informação sobre não só a relação a ser processada, mas também o seu inverso.

Por último, processamos o tipo Comps, que corresponde a uma lista de strings ou objetos (URI). Aqui é criada uma lista guardada em memória, com todos os elementos da relação, e guardados na estrutura de dados criada para o efeito, já tendo em conta as relações inversas.

Processados todos os cadernos passados ao programa, é percorrida a nossa estrutura de dados, por forma a preencher as páginas *HTML* com a respetiva informação.

Capítulo 4

Manual de Utilização

Este programa é de utilização simples, recebendo como argumento os cadernos a ser processado, podendo ser chamado desta forma: `./turtle path_to_input_file1 path_to_input_file2`.

Para explorar o site resultante do processamento dos ficheiros de entrada, pode abrir o ficheiro *index.html* com um qualquer browser.

Capítulo 5

Conclusão

Para concluir, fazemos um balanço bastante positivo do projeto, conseguindo atingir todos os requisitos definidos, conseguindo com sucesso aplicar os conhecimentos adquiridos ao longo desta Unidade Curricular num contexto mais prático.

Como trabalho futuro, gostariamos de melhorar o aspeto do site produzido, introduzindo formatação para o texto de cada conceito, e organizando a informação apresentada de melhor forma.

Apêndice A

FLex

```
%x TT CT
%%
===                         { return INIT; }
:reverse_of                 { return INV; }
:[^\ ,;.\n]+                { yylval.str = strdup(yytext + 1); return URI; }

@tit:[\ \n]                 { BEGIN TT; }
<TT>[^\n]+                  { BEGIN CT;
                              yylval.str = strdup(yytext);
                              return TIT; }

@triplos:[\ \n]             { return TR; }

<CT>@meta:[\ \n]            { BEGIN INITIAL; return MT; }

<CT>([^@\n]+\n)+/[\n@]      { yylval.str = strdup(yytext); return CONTEUDO; }
<CT>@triplos:[\ \n]         { BEGIN INITIAL; return TR; }
<CT>\n                      { }

[a,;.]                      { return yytext[0]; }
\".+\"                      { yylval.str = strdup(yytext); return STR; }
.|\n                        {  }
%%
```

Apêndice B

Yacc

```
%union {
    char* str;
    struct cmp* comp;
    GArray* comps;
}

%token INIT TR MT INV
%token<str> STR CONTEUDO TIT URI
%type<str> Sujeito Par Pares Conteudo Rel
%type<comp> Comp
%type<comps> Comps

%%

Caderno : Conceitos
        | Meta
        ;

Conceitos   : Conceito
            | Conceitos Conceito

Conceito    : Documento TR Triplos
            ;

Documento   : INIT Sujeito Titulo Conteudo          { fprintf(file, "%s", $4); fclose(file);
                                                      free($4); free($2); curr_rev = NULL; }
            ;

Meta : Meta URI INV URI '.'             { reverse[revptr++] = strdup($2); reverse[revptr++] = strdup($4); }
     | URI INV URI '.'                  { reverse[revptr++] = strdup($1); reverse[revptr++] = strdup($3); }
     ;

Triplos : Triplos Sujeito Pares '.'     { fclose(file); }
        | Sujeito Pares '.'             { fclose(file); }
        ;

Pares   : Par                           { }
        | Pares ';' Par                 { }
        ;

Par     : Rel Comps                    { GArray* list = get_or_insert_list($1, suj);
                                         g_array_append_vals(list, $2->data, $2->len);
                                         curr_rev = NULL; }
        ;

Comps   : Comp                          { $$ = g_array_new(FALSE, FALSE, sizeof(struct cmp*));
                                          g_array_append_val($$, $1); }
        | Comps ',' Comp                { $$ = $1; g_array_append_val($$, $3); }
        ;

Comp    : URI                           { mkindex($1);
                                          mkrev($1);
                                          $$ = malloc(sizeof(struct cmp));
                                          $$->str = strdup($1);
                                          $$->type = 1;
                                          free($1);
```

```
                                        }
        | STR                           { $$ = malloc(sizeof(struct cmp));
                                               $$->str = strdup($1);
                                               $$->type = 0;
                                               free($1); }
        ;

Sujeito : URI                           { suj = mkindex($1);
                                          $$ = strdup($1);
                                          sujeito = $$;
                                          file = fopen($1, "a");
                                          free($1); }
        ;

Titulo  : TIT                           { fprintf(file, "<h1>%s</h1>\n", $1); free($1); }
        ;

Rel     : 'a'                           { $$ = strdup("a"); }
        | URI                           { curr_rev = findrev($1); $$ = strdup($1); }
        ;

Conteudo : CONTEUDO                     { asprintf(&$$, "<p>%s</p>\n", $1); free($1); }
         | Conteudo CONTEUDO            { asprintf(&$$, "%s<p>%s</p>\n", $1, $2); free($1); free($2); }
         ;
%%
```