Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 3 - SATI

Relatório de Desenvolvimento

Rui Azevedo	Joana Cruz	Maurício Salgado
(80789)	(76270)	(71407)

10 de Junho de 2019

Resumo

Neste relatório são apresentadas as várias decisões no desenvolvimento de um parser Flex/Yacc que processe textos tendo em conta um dicionário que respeite a gramática construída. Este processamento de textos deve identificar palavras do texto que estão presentes no dicionário e sublinhá-las, adicionar um footnote e, no fim do texto, criar um apêndice com as palavras encontradas e o seu significado, convertendo o ficheiro inicial(do tipo `.txt` por exemplo) num ficheiro LaTeX(`.tex`).

Conteúdo

Capítulo 1

Introdução

O objetivo deste projeto consiste no desenvolvimento que faz parse a um dicionário de palavras relacionadas com informática, com base numa gramática que definimos. Posteriormente, usamos o resultado do dicionário processado para processar ficheiros de texto e gerar pdf com footnotes e um apéndice que resulta da associação entre as palavras do texto e o nosso dicionário. Inicialmente será exposto, de uma forma aprofundada, o problema a resolver sendo, de seguida, apresentada uma solução para o mesmo. São depois ilustrados vários exemplos de teste, finalizando com uma conclusão relativamente aos resultados do trabalho.

Estrutura do Relatório

Em relação à estrutura do relatório, inicialmente será exposto, de uma forma aprofundada, o problema a resolver sendo, de seguida, apresentada uma solução para o mesmo. São depois ilustrados vários exemplos de teste, finalizando com uma conclusão relativamente aos resultados do trabalho.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Pretende-se criar um Dicionário de palavras utilizadas na área de Informática. Associado a cada palavra pretende-se ter: o significado, a tradução comum em inglês e uma lista de sinónimos. O objetivo do projeto consiste em gerar ficheiros LaTeX com as palavras que têm correspondência com o nosso dicionário sublinhadas, com footnotes para cada match, e um apéndice onde aparece o significado das palavras encontradas. Para o efeito desenvolveu-se uma gramática para processar um dicionário (com uma estrutura por nós definida) e carregar a informação útil para as devidas estruturas de dados.

2.2 Especificação dos Requisitos

O sistema desenvolvido deve permitir que sejam usados dicionários de diferentes áreas do saber. Isto deve ser alcançado de maneira eficiente para garantir a possibilidade de processar grandes quantidades de texto. Deve, também, ser capaz de processar diversos ficheiros de texto, e, para cada um deles, gerar um ficheiro LaTeX para cada um.

Capítulo 3

Concepção/desenho da Resolução

3.1 Gramática do Dicionário

Com o intuito de criar um formato para os dicionários que devem ser utilizados para consulta foi criada uma gramática para os mesmos, gramática esta que permite associar cada palavra ao seu significado, o termo em inglês e os seus sinónimos. O formarto do Dicionário é o seguinte:

```
palavra{
    def: "definição da palavra"
    trd: "tradução da palavra"
    sin: "primeiro sinónimo"
    sin: "segundo sinónimo"
}
```

A estrutura da nossa gramática é a seguinte:

```
DICIONARIO : PALAVRAS
           ;
PALAVRAS   : PALAVRAS PALAVRA            {dic = insert_word(dic,pal);}
           |
           ;
PALAVRA    : NOME ARGUMENTOS
           ;
NOME       : WORD                        {pal = create_word() ; add_name(pal,$1);}
           ;
ARGUMENTOS : ARGUMENTOS ARGUMENTO
           |
           ;
ARGUMENTO  : DEF                          {add_def(pal,$1);}
           | TRD                          {add_trd(pal,$1);}
           | SIN                          {add_sin(pal,$1);}
           ;
```

3.2 Estruturas de Dados

Por forma a permitir o processamento de vários textos com um dicionário é necessário guardar o mesmo em memória. Para tal foram criadas as seguintes estruturas:

```
typedef struct sinonimos{
    char *nome;
    struct sinonimos *next;
}*Sinonimos;

typedef struct palavra{
    char *nome;
    char *definicao;
    char *traducao;
    Sinonimos sinonimo;
    int referencia;
}*Palavra;

typedef struct dicionario{
    Palavra pal;
    struct dicionario *next;
}*Dicionario;
```

Figura 3.1: Concepção da Estrutura de Dados - Dicionário

A inserção da variável `referencia` na estrutura permite saber, findo o processamento de cada texto, que palavras foram encontradas no texto, sendo que, após criado o apêndice, esse valor volta a ser colocado a 0.

3.3 Parse dos ficheiros

O nosso parsing é feito em duas fases:

- Parse do ficheiro com o dicionário, processo durante o qual são identificadas as diferentes partes da gramática, retornando uma stream de tokens para o Yacc. No Yacc, consoante os tokens identificados, executa a ação a eles associados. No nosso caso vai-se identificar a que estrutura de dados adicionar a informação a ser processada.
- Parse dos ficheiros com os textos, sublinhando e criando notas de rodapé para as palavras que sejam comuns ao dicionário e ao texto, criando ainda um apêndice com as palavras encontradas e respetivos significados.

Isto é alcançado através de dois estados, DIC e FILES. Antes de cada iniciação do Flex é chamado uma das funções parseDic e, posteriormente, parseFiles para que o Flex aplique apenas as regras associadas ao contexto certo. Sendo assim, dentro de cada condição é colocado as regras especificas para cada tipo de ficheiro, permitindo assim realizar o parse de vários ficheiros com estruturas diferentes com o mesmo ficheiro Flex.

3.4 Underline e footnote

Para cada palavra presente nos ficheiros processados, a ferramenta realiza uma consulta no dicionário e, caso seja encontrado o termo em causa, a palavra é sublinhada no ficheiro de output, sendo ainda criada uma nota de rodapé com a sua tradução:

```
traducao = get_trd(yytext, &flag);
    if(traducao){
        if( !flag )
            underlineRef(yytext,traducao);
        else
            underline(yytext);

    }
    else{
        fprintf(yyout,"%s ",yytext);
    }

...

void underlineRef(char *pt, char *en){
    fprintf(yyout,"\\underline{%s}\\footnote{%s} ",pt,en);
}

void underline(char *pt){
    fprintf(yyout,"\\underline{%s} ",pt);
}
```

3.5 Apêndice

A partir da variável referencia da estrutura palavra é possível saber se a palavra foi ou não encontrada no texto. Sendo assim basta verificar esse valor e caso seja igual a 1 imprime-se um item de uma lista com o nome do termo e a definição:

```
while(tmp){
    if(tmp->pal->referencia == 1){
        fprintf(f,"\\item %s $\\to$ Def: %s\n",tmp->pal->nome,tmp->pal->definicao);
        tmp->pal->referencia = 0;
    }
    tmp = tmp->next;
}
```

Contudo, inicialmente é criada uma secção Apendice e a abertura da lista:

```
fprintf(f,"\n\\appendix\n");
fprintf(f,"\\section{Apendice}\n");
fprintf(f,"\\begin{itemize}\n");
```

E por fim:

```
fprintf(f,"\\end{itemize}\n");
fprintf(f,"\n\\end{document}\n");
```

Capítulo 4

Codificação e Testes

4.1 Testes realizados e Resultados

Foi criada uma Makefile de modo a compilar o código sendo que a ideia base é:

- Gerar `.c` do flex: `flex sati.l`
- Gerar `.c` do yacc: `yacc -d -v sati.y`
- Gerar o executável: `gcc -o sati y.tab.c`

O executável pode ser usado da seguinte maneira:

```
./sati <FicheiroDicionario> <FicheiroARealizarParse1> <FicheiroArealizarParse2> ...
```

4.1.1 Exemplo de teste

Comando executado: `./sati dic.txt text.txt`

4.1.2 Input - dic.txt

```
computador {
    def: "aparelho eletrónico que é capaz de receber, armazenar e processar grande quantidade
de informação em função de um conjunto de instruções com que é programado"
    trd: "computer"
    sin: "PC"
}

Internet{
    def: "conjunto de redes de computadores que, espalhados por todas as regiões do planeta,
    conseguem trocar dados e mensagens utilizando um protocolo comum"
    trd: "Internet"
    sin: "net"
    sin: "rede"
}
```

```
algoritmo{
    def: "conjunto de operações, sequenciais, lógicas e não ambíguas, que, aplicadas a
    um conjunto de dados, permitem encontrar a solução para um problema num número finito de
passos"
    trd: "algorithm"
    sin: "operação"
    sin: "cálculo"
}
```

Input - text.txt

```
Informática é um termo usado para descrever o conjunto das ciências relacionadas à coleta,
armazenamento, transmissão e processamento de informações em meios digitais, estando incluídas
neste grupo: a ciência da computação, os sistemas de informação, a teoria da informação, o
processo de cálculo, a análise numérica e os métodos teóricos da representação dos conhecimentos
e da modelagem dos problemas. Mas também a informática pode ser entendida como ciência que
estuda o conjunto de informações e conhecimentos por meios digitais. O algoritmo foi desenhado
para que um dia uma máquina pudesse tratar informações. O uso da Internet também é um caso
importante. De nada adianta pedir para um aluno fazer uma pesquisa na Internet sem as devidas
orientações. Cabe ao professor instruir os alunos para que estes não façam simples cópias de
textos encontrados em sites. Apenas copiando, os alunos não vão aprender. Computador é uma
máquina capaz de variados tipos de tratamento automático de informações ou processamento de
dados. Um computador pode possuir inúmeros atributos, dentre eles armazenamento de dados, process
de dados, cálculo em grande escala, desenho industrial, tratamento de imagens gráficas, realidade
virtual, entretenimento e cultura.
```

Output

```
\documentclass{article}
\usepackage[bottom]{footmisc}

\begin{document}
Informática é um termo usado para descrever o conjunto das ciências relacionadas à coleta,
armazenamento, transmissão e processamento de informações em meios digitais, estando incluídas
neste grupo a ciência da computação, os sistemas de informação, a teoria da informação, o proces
de cálculo, a análise numérica e os métodos teóricos da representação dos conhecimentos e da
modelagem dos problemas. Mas também a informática pode ser entendida como ciência que estuda
o conjunto de informações e conhecimentos por meios digitais. O \underline{algoritmo}\footnote{a
foi desenhado para que um dia uma máquina pudesse tratar informações. O uso da \underline{Interne
também é um caso importante. De nada adianta pedir para um aluno fazer uma pesquisa na \underlin
sem as devidas orientações. Cabe ao professor instruir os alunos para que estes não façam simples
cópias de textos encontrados em sites. Apenas copiando, os alunos não vão aprender. Computador
é uma máquina capaz de variados tipos de tratamento automático de informações ou processamento
de dados. Um \underline{computador}\footnote{computer} pode possuir inúmeros atributos, dentre
eles armazenamento de dados, processamento de dados, cálculo em grande escala, desenho industria
tratamento de imagens gráficas, realidade virtual, entretenimento e cultura.
\appendix
\section{Apendice}
```

```
\begin{itemize}
\item algoritmo $\to$ Def: conjunto de operações, sequenciais, lógicas e não ambíguas, que,
aplicadas a um conjunto de dados, permitem encontrar a solução para um problema num número
finito de passos
\item Internet $\to$ Def: conjunto de redes de computadores que, espalhados por todas as regiões
do planeta, conseguem trocar dados e mensagens utilizando um protocolo comum
\item computador $\to$ Def: aparelho eletrónico que é capaz de receber, armazenar e processar
grande quantidade de informação em função de um conjunto de instruções com que é programado

\end{itemize}
\end{document}
```

Capítulo 5

Conclusão

Neste trabalho foi possível perceber os diferentes passos de desenvolvimento de uma gramática. O caso de estudo foi desenvolver um dicionário que continha palavras relacionadas com a área da informática mas, no entanto, a implementação proposta pode ser estendida a qualquer tipo de área do conhecimento. Tal é possível pois a implementação do dicionário é genérica, qualquer que seja o dicionário e qualquer que sejam os ficheiros a processar, a peça de software desenvolvida vai aplicar a mesma lógica a esses ficheiros segundo as regras gramaticais definidas pelo Yacc.

O Yacc providencia uma ferramenta genérica para a descrição do *input* de um programa. Desta modo, e junto com o Flex, a criação de uma linguagem que representa um dicionário tornou-se muito mais fácil de desenvolver.

Apêndice A

Código do Programa

A.1 sati.l

```
%option noyywrap
%option yylineno

%{

#include "y.tab.h"

int flag;
char *traducao;
void underlineRef(char *, char *);
char *get_trd(char *, int *);
void underline(char *pt);
%}

STRING ([a-zA-Z0-9]|[áàâãéèêíóôõúçñÁÀÂÃÉÈÍÓÔÕÚÇ,-.])

%x DIC FILES def sin trd word

%%
<DIC>{
({STRING}|\ )+\{
                                    {
                                        yytext[yyleng-1]='\0';
                                        yylval.texto = strdup(yytext);
                                        BEGIN (word); return WORD;
                                    }

<word>\}                                {BEGIN DIC;}

<word>\n                                {;}
```

```
<word>[\ \t]*def:[\ \t]*\"              {BEGIN (def);}

<def>\"                                 {BEGIN (word);}

<def>({STRING}|\ )+                     {
                                            yylval.texto = strdup(yytext);
                                            return DEF;
                                        }

<word>[\ \t]*sin:[\ \t]*\"              {BEGIN (sin);}

<sin>\"                                 {BEGIN (word);}

<sin>\n                                 {;}

<sin>({STRING}|\ )+                     {
                                            yylval.texto = strdup(yytext);
                                            return SIN;
                                        }

<word>[\ \t]*trd:[\ \t]*\"              {BEGIN (trd);}

<trd>\"                                 {BEGIN (word);}

<trd>\n                                 {;}

<trd>({STRING}|\ )+                     {
                                            yylval.texto = strdup(yytext);
                                            return TRD;
                                        }


}
<FILES>{
{STRING}+                               {
                                            traducao = get_trd(yytext, &flag);
                                            if(traducao){
                                                if( !flag )
                                                    underlineRef(yytext,traducao);
                                                else
                                                    underline(yytext);
                                             }
                                            else{
                                                fprintf(yyout,"%s ",yytext);
                                            }
                                        }
<*>.|\n                                 {;}
}
```

```
%%


void parseDic(){
    BEGIN DIC;
}

void parseFiles(){
    BEGIN FILES;
}

void underlineRef(char *pt, char *en){
      fprintf(yyout,"\\underline{%s}\\footnote{%s} ",pt,en);
}

void underline(char *pt){
    fprintf(yyout,"\\underline{%s} ",pt);
}
```

A.2 sati.y

```
    %{
     extern int yylex();
     extern int yylineno;
     extern char *yytext;
     void yyerror(char*);
     #include <stdlib.h>
     #include <stdio.h>
     #include <string.h>
     #include <ctype.h>

/* Estruturas de dados */
     typedef struct sinonimos{
          char *nome;
          struct sinonimos *next;
     }*Sinonimos;
     typedef struct palavra{
          char *nome;
          char *definicao;
          char *traducao;
          Sinonimos sinonimo;
          int referencia;
     }*Palavra;
     typedef struct dicionario{
          Palavra pal;
          struct dicionario *next;
     }*Dicionario;
```

```
/* Prototipos */
     Palavra create_word();
     void add_name(Palavra , char *);
     void add_def(Palavra , char *);
     void add_trd(Palavra , char *);
     void add_sin(Palavra , char *);
     void set_ref(Palavra);
     char *get_trd(char *, int *);
     Dicionario insert_word(Dicionario , Palavra );
     Palavra lookup(char *);
     Dicionario dic = NULL;
     Palavra pal = NULL;
%}

%union{
  char* texto;
}

%token DEF SIN TRD WORD
%type<texto> DEF SIN TRD WORD NOME ARGUMENTO
%%

/* ------------------------------ Gramatica ---------------------------------- */
DICIONARIO : PALAVRAS
           ;
PALAVRAS   : PALAVRAS PALAVRA               {dic = insert_word(dic,pal);}
           |
           ;
PALAVRA    : NOME ARGUMENTOS
           ;
NOME       : WORD                           {pal = create_word() ; add_name(pal,$1);}
           ;
ARGUMENTOS : ARGUMENTOS ARGUMENTO
           |
           ;
ARGUMENTO  : DEF                            {add_def(pal,$1);}
           | TRD                            {add_trd(pal,$1);}
           | SIN                            {add_sin(pal,$1);}
           ;
%%
#include "lex.yy.c"

/* ------------------------------ Codigo ------------------------------------- */

Palavra create_word(){
     Palavra p = malloc(sizeof(struct palavra));
     p->nome = NULL;
```

```
     p->definicao  = NULL;
     p->traducao  = NULL;
     p->sinonimo  = NULL;
     p->referencia  = 0;
     return p;
}
/* Adiciona o nome da palavra */
void add_name(Palavra p, char *name){
     p->nome = strdup(name);
}
/* Adiciona a definicacao de uma palavra */
void add_def(Palavra p, char *definicao){
     p->definicao = strdup(definicao);
}
/* Adiciona a traducao em ingles de uma palavra */
void add_trd(Palavra p, char *traducao){
     p->traducao = strdup(traducao);
}
/* Adiciona um sinonimo de um palavra */
void add_sin(Palavra p, char *sinonimo){
     Sinonimos s = malloc(sizeof(struct sinonimos));
     s->nome = strdup(sinonimo);
     s->next = p->sinonimo;
     p->sinonimo  = s;
}
/* Palavra mencionada */
void set_ref(Palavra p){
     p->referencia = 1;
}
/* Adiciona uma palavra ao dicionario */
Dicionario insert_word(Dicionario dic, Palavra p){
     Dicionario new = malloc(sizeof(struct dicionario));
     new->pal = p;
     new->next = dic;
     dic = new;
     return new;
}
char* get_trd(char *p, int *val){
    Dicionario tmp = dic;
    while(tmp){
      if(!strcmp(tmp->pal->nome, p)) {
          if(tmp->pal->referencia)
            *val = 1;
          else{
            *val = 0;
            tmp->pal->referencia = 1;
          }
          return tmp->pal->traducao;
```

```
      }
      tmp = tmp->next;
    }
    return NULL;
}

/* --------------------------------- Latex ------------------------------------*/
void initLatex(FILE *f){
     fprintf(f,"\\documentclass{article}\n");
     fprintf(f,"\\usepackage[bottom]{footmisc}\n\n");
     fprintf(f,"\\begin{document}\n");
}
void makeAppendix(FILE *f){
    Dicionario tmp = dic;
    fprintf(f,"\n\\appendix\n");
    fprintf(f,"\\section{Apendice}\n");
    fprintf(f,"\\begin{itemize}\n");
    while(tmp){
        if(tmp->pal->referencia == 1){
            fprintf(f,"\\item %s $\\to$ Def: %s\n",tmp->pal->nome,tmp->pal->definicao);
            tmp->pal->referencia = 0;
        }
        tmp = tmp->next;
    }
    fprintf(f,"\\end{itemize}\n");
    fprintf(f,"\n\\end{document}\n");
}
/* --------------------------------- Main ------------------------------------*/
void yyerror(char *error){
  fprintf(stderr, "ERROR : %s \n", error);
}
int main(int argc, char **argv){
     char outfile[200];
     parseDic();
     yyin = fopen(argv[1], "r");
     yyparse();
     fclose(yyin);
     parseFiles();
     for(int i = 2; i < argc ; i++){
       yyin = fopen(argv[i], "r");
       if(yyin){
         strcat(outfile,argv[i]);
         strcat(outfile,".tex");
         yyout = fopen(outfile,"w");
         initLatex(yyout);
         yylex();
         makeAppendix(yyout);
         fclose(yyin);
```

```
          fclose(yyout);
        }
        memset(outfile,0,200);
      }
      return 0;
}
```