

Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 2

Relatório de Desenvolvimento

Grupo 24

Adriana Meireles
(a82582)

Mariana Pereira
(a81146)

28 de Junho de 2020

Resumo

Este relatório tem como objetivo documentar o segundo trabalho prático desenvolvido na unidade curricular de Processamento de Linguagens, utilizando a ferramenta Flex e Yacc como auxiliares para gerar um reconhecedor léxico e sintático.

Deste modo, é apresentada uma introdução ao projeto bem como a descrição do enunciado. De seguida é explicada de que forma é implementada a solução e as decisões que foram tomadas durante a sua realização. No final é feita uma apreciação crítica ao trabalho desenvolvido.

Conteúdo

1	Introdução	2
2	Análise e Especificação	3
2.1	Descrição do problema	3
2.2	Especificação dos Requisitos	3
3	Ficheiro - Reverse Engineering dum Dicionário financeiro	4
4	Concepção/desenho da Resolução	5
4.1	Especificação da GIC	5
4.2	Analisador Léxico	6
4.3	Analisador Sintático	7
5	Codificação e Testes	10
5.1	Alternativas, Decisões e Problemas de Implementação	10
5.2	Modo de Utilização	10
5.2.1	Compilação	10
5.2.2	Execução	11
5.3	Testes realizados e Resultados	11
6	Conclusão	13

Capítulo 1

Introdução

No segundo trabalho prático da Unidade Curricular de Processamento de Linguagens, o exercício que nos foi atribuído foi o 1, que consiste em fazer *Reverse Engineering* de um dicionário de negócios e finanças EN-PT.

Deste modo, pretende-se que seja elaborada uma gramática independente de contexto, bem como o respetivo analisador léxico, utilizando a ferramenta Flex que permita reconhecer um determinado Dicionário Financeiro.

É ainda necessário desenvolver um analisador sintático recorrendo à ferramenta Yacc.

Assim, no presente relatório, iremos apresentar as estratégias utilizadas, bem como a linha de pensamento seguida para resolver todos os exercícios propostos.

Capítulo 2

Análise e Especificação

2.1 Descrição do problema

Como referido anteriormente, coube-nos o enunciado número 1, referente a um dicionário financeiro. Assim, tendo em conta o ficheiro disponibilizado contendo um extrato de um dicionário de negócios e finanças EN-PT pretende-se:

- Escrever uma gramática que cubra tanto quanto possível as entradas do dicionário.
- Construir um processador (flex, yacc) que converta o ficheiro num formato mais tratável.
- Detetar e tratar os casos de erro sintático, nas situações em que o formato da entrada seja incoerente com a gramática criada acima; nessas circunstâncias, avisar a linha onde ocorreu o erro, e continuar o processamento.

2.2 Especificação dos Requisitos

O enunciado que nos foi atribuído, para além das indicações específicas do próprio tema deve também seguir algumas indicações gerais:

- Especificar a gramática concreta da linguagem de entrada.
- Desenvolver um reconhecedor léxico e sintático para essa linguagem com recurso ao par de ferramentas geradoras Flex/Yacc.
- Construir o gerador de código que produza a resposta solicitada. Esse gerador de código é construído associando acções semânticas de tradução às produções da gramática, recorrendo uma vez mais ao gerador Yacc.

Capítulo 3

Ficheiro - Reverse Engineering dum Dicionário financeiro

Depois de analisar o ficheiro que nos foi fornecido reparamos que existiam várias formas de representar as palavras como podemos ver na tabela a baixo.

Vimos que o caracter ':' representa que o texto vai ter uma base e significa que nas linhas seguintes começam todos com um ou mais espaços antes de aparecer a palavra. Também terá que ter um caracter '-' que irá indicar a posição da base.

Caso a linha não contenha caracter ':' vai ser uma tradução simples. Em primeiro lugar vem a palavra em inglês e só depois vem a palavra em português. Neste último caso, é possível ter o caracter ',' e ';' que permitem ter mais do que uma tradução em português.

Dicionario Financeiro		Reverse Engineering
dissolution	dissolução (f)	EN dissolution PT dissolução
distributor	distribuidor (m), atacadista (m)	EN distributor PT distribuidor (m) PT atacadista (m)
dispute: labour -	conflito (m) trabalhista	EN labour dispute +base dispute PT conflito (m) trabalhista
diversification: - strategy	diversificação (f) estratégia (f) de diversificação	EN diversification PT diversificação (f) EN diversification strategy +base diversification PT estratégia (f) de diversificação

Capítulo 4

Concepção/desenho da Resolução

4.1 Especificação da GIC

Nesta secção é apresentada a gramática independente de contexto desenvolvida para o formato que especifica o *Reverse Engineering* do nosso dicionário que é a seguinte:

$T = \{\text{START, PALPORT, BASEAUX, PALING, DEFING, BASE}\}$

$NT = \{\text{ListaPalavras, Palavras, Definicao, ListaBase, Ingles, InglesAux, Portugues, PortuguesAux}\}$

$S = \{\text{Dicionario}\}$

Dicionario -> START ListaPalavras

ListaPalavras : ListaPalavras Palavras
 | Palavras

Palavras : DEFING Portugues
 | error
 | BASE Definicao ListaBase

Definicao: PALPORT
 |

ListaBase : ListaBase Ingles Portugues
 | Ingles Portugues

Ingles : PALING BASEAUX InglesAux
 | BASEAUX PALING

InglesAux :
 | PALING

Portugues : PALPORT PortuguesAux

```
PortuguesAux :
    | ' , ' PALPORT
    | ' ; ' PALPORT
```

4.2 Analisador Léxico

Para procedermos à análise léxica do documento num formato mais tratável como pedido no enunciado utilizamos a ferramenta Flex. Este deverá transformar o texto original de modo a ser interpretado pela GIC anteriormente criada. Assim, os '- ', bases(palavras em inglês seguidas de :), palavras em inglês fora das bases, palavras em inglês dentro das bases e palavras em português devem ser transformadas em tokens BASEAUX, BASE, DEFINING, PALING e PALPORT, respetivamente.

Começamos por definir algumas expressões regulares assim como uma start condition que nos indicará quando estivermos na parte que nos interessa do dicionário:

```
%x dicionario

acentos \xc3[\x80-\xbf]
palavraPort ([a-z\/\-+()]{acentos})+([ ]([a-z\/\-+()]{acentos})+)*

palavraIng [A-Z]+([ \-][A-Z()]+)*
```

Estas foram utilizadas para apanhar as palavras em Português e Inglês(já com acentos), respetivamente. Ignora-se tudo o que estiver antes do __BEGIN__. Os símbolos ', ' e '; ' já estão presentes no texto original.

```
[^\n]+ ;

<dicionario>[\-]      {      if(aux!=NULL){
                        yylval.spal = strdup(aux);
                        return BASEAUX;
                      }
                      }

<dicionario>[,;]      {      return yytext[0];}

<dicionario>~{palavraIng}: {  yytext[yyleng-1] = '\0';
                              lower_string(yytext);
                              yylval.spal = strdup(yytext);
                              aux = strdup(yytext);
                              return BASE;
                            }

<dicionario>~{palavraIng} {  lower_string(yytext);
                              yylval.spal = strdup(yytext) ;
                              return DEFINING;
                            }

<dicionario>{palavraIng} {  lower_string(yytext);
```



```

        yylval.spal = strdup(yytext) ;
        return PALING;
    }

<dicionario>{palavraPort}  {   yylval.spal = strdup(yytext) ;
                                return PALPORT;
                                }

```

4.3 Analisador Sintático

Tendo a GIC já sido escrita e o analisador léxico já preparado para a conversão de texto, é possível guardar nos tokens os valores relevantes, utilizando o Yacc como ferramenta de análise sintática. Após a análise léxica do ficheiro de texto, é necessário garantir que se encontra de acordo com as regras da gramática previamente definida e ainda processar e tratar os dados fornecidos de modo a dar resposta ao pretendido, sendo estas as principais funcionalidades do nosso analisador sintático.

Para começar a desenhar a gramática, em primeiro lugar, constatámos que a parte do documento que nos interessa começa por "__BEGIN__"e, de seguida, surge um conjunto de pares EN(inglês)/PT(português). Este conjunto podia constituir apenas um par de definições EN/PT, ou várias, tal como podemos ver na segunda regra da gramática.

```

Dicionario : START  ListaPalavras
            ;

```

```

ListaPalavras : ListaPalavras Palavras {printf("%s", $2);}
              | Palavras {printf("%s", $1);}
              ;

```

Por sua vez, uma definição EN/PT pode ser apresentada de várias formas. Podemos ter:

- **DEFING Portugues** - situação em que se encontra fora de uma base e, portanto, a definição em inglês encontra-se à esquerda e a respetiva definição em português encontra-se à direita. Essa definição em inglês é identificada pelo token **DEFING**.
- **error** - caso detete um erro sintático, permite prosseguir o processamento.
- **BASE Definicao ListaBase** - situação em que se encontra dentro de uma base. A palavra da base é identificada pelo token **BASE**. O que se recebe do símbolo não terminal *ListaBase*, vai ser separado em tokens segundo o delimitador '#'. No values[0] vão estar as palavras em inglês e no values[1] vão estar as palavras em português para depois poderem ser ambas apresentadas juntamente com a base segundo o output esperado.

```

Palavras : DEFING Portugues {asprintf(&$$, "EN %s\n%s\n\n", $1, $2);}
| error {asprintf(&$$, "");}
| BASE Definicao ListaBase { char* tokens = strtok ($3,"#");
                           char* values[2];
                           char* aux[i];
                           int j=0,size=0;

                           while( tokens != NULL && j<i) {
                               values[0] = strdup(tokens);
                               tokens = strtok(NULL, "#");
                               values[1] = strdup(tokens);
                               asprintf(&aux[j], "EN %s\n+base %s\n
                               %s\n\n",values[0], $1,values[1]);
                               size += strlen(aux[j]);
                               j++;
                               tokens = strtok(NULL, "#");
                           }

                           if($2!=NULL){
                               tokens = NULL;
                               asprintf(&tokens, "EN %s\nPT %s\n\n",$1,$2);
                               $$ = malloc(sizeof(char)*(size + strlen(tokens)));
                               strcat($$,tokens);

                           }
                           else $$ = malloc(sizeof(char)*size);

                           for(j=0; j < i; j++){
                               strcat($$,aux[j]);
                           }
                           i=0;
                           }

;

```

A base pode ou não ter definição em português à sua direita, sendo que essa definição é identificada pelo token **PALPORT**.

```

Definicao: PALPORT {$$ = strdup($1);}
| {$$=NULL;}
;

```

Para o caso em que dentro de uma base tem vários pares de definições EN/PT , sendo, EN/PT uma lista. Esta define-se da forma que podemos ver a seguir. A lista pode ter apenas uma definição EN/PT ou pode ser uma sequência de definições EN/PT. Aqui, as palavras em inglês e português, são separadas pelo caracter '#' para serem processadas no símbolo não terminal *Palavras*.

```

ListaBase : ListaBase Ingles Portugues {i++;asprintf(&$$, "%s%s#%s#", $1, $2,$3);}

```

```

| Ingles Portugues          {i++;asprintf(&$$, "%s#%s#", $1, $2);}
;

```

No que toca ao valor do tipo Inglês, há que ter em atenção a posição onde se situa, dentro da base, o caracter '-' que é identificado pelo token **BASEAUX**. Pode estar antes de uma palavra em inglês, depois de uma palavra em inglês ou entre duas palavras em inglês. Essa palavra é identificada pelo token **PALING**.

```

Ingles : PALING BASEAUX InglesAux{
                                if($3!=NULL)
                                    asprintf(&$$, "%s %s %s", $1,$2,$3);
                                else
                                    asprintf(&$$, "%s %s", $1,$2);
                                }
| BASEAUX PALING              { asprintf(&$$, "%s %s", $1, $2);}
;

InglesAux : {$$=NULL;}
| PALING {$$ = strdup($1);}
;

```

No que toca ao valor do tipo Português, pode ter apenas uma palavra em português, pode ter duas palavras em português separadas por ',' ou ';'. Essa palavra é identificada pelo token **PALPORT**.

```

Portugues : PALPORT PortuguesAux{
                                if($2!=NULL)
                                    asprintf(&$$, "PT %s\n%s", $1,$2);
                                else
                                    asprintf(&$$, "PT %s", $1);
                                }
;

PortuguesAux : {$$=NULL;}
| ', ' PALPORT {asprintf(&$$, "PT %s", $2);}
| '; ' PALPORT {asprintf(&$$, "PT %s", $2);}
;

```

Capítulo 5

Codificação e Testes

5.1 Alternativas, Decisões e Problemas de Implementação

O principal problema incidiu em definir uma regra para separar as palavras em inglês do português, pois o separador que nos é dado, espaço, dificulta bastante este processo. Deste modo tentamos criar uma regra em que o espaço era o separador, mas tanto a palavra em inglês como em português poderiam ter mais que um espaço antes de começar, tornando esta solução menos viável. De seguida tentamos usar o tab como separador entre as palavras de línguas diferentes mas não conseguimos obter o resultado esperado. Por fim, depois de falarmos com o professor José João, decidimos mudar o texto que nos foi fornecido, pondo as palavras em inglês em maiúsculo e as palavras em português em minúsculo.

Com a alteração referida em cima ao texto de input, para obtermos um output igual ao que nos é dado no enunciado criamos uma função, *lower_string(char* s)*, que é responsável por passar as palavras escritas em maiúsculas para minúsculas.

5.2 Modo de Utilização

5.2.1 Compilação

Apresentamos de seguida a makefile que permite a compilação do projeto, correndo o comando **make** que cria o executável *ReverseEngineering*. Esta makefile permite também a limpeza dos ficheiros criados na compilação bem como os que resultam da execução do programa. Executa-se o comando **make clean**, para esta funcionalidade.

```
ReverseEngineering : y.tab.o lex.yy.o
gcc -o ReverseEngineering y.tab.o lex.yy.o
```

```
y.tab.o : y.tab.c
gcc -c y.tab.c
```

```
lex.yy.o : lex.yy.c
gcc -c lex.yy.c
```

```
y.tab.c y.tab.h : ReverseEngineering.y
yacc -d ReverseEngineering.y
```

```
lex.yy.c : ReverseEngineering.l y.tab.h
flex ReverseEngineering.l
```

clean:

```
rm -f ReverseEngineering lex.yy.c lex.yy.o y.tab.h y.tab.o y.tab.c
```

5.2.2 Execução

Depois de compilar os programas, a sua execução é feita do seguinte modo:

- Programa ReverseEngineering :
cat dic-finance-en.pt.txt | ./ReverseEngineering

Sendo o dic-finance-en.pt.txt um exemplo de um input de dicionário.

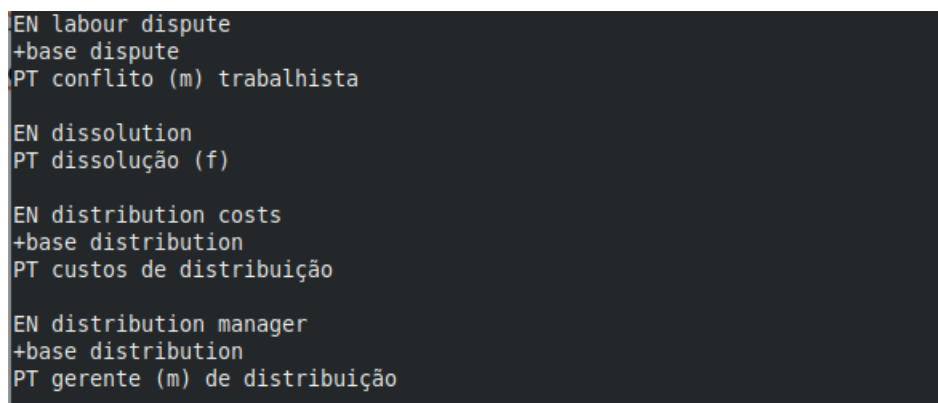
5.3 Testes realizados e Resultados

De modo a demonstrar tudo o que foi referido ao longo do relatório apresentamos de seguida alguns textos de input e os respectivos resultados obtidos.

Dado este texto de input:

```
__BEGIN__
DISPUTE:
  LABOUR -          conflito (m) trabalhista
DISSOLUTION        dissolução (f)
DISTRIBUTION:
  - COSTS           custos de distribuição
  - MANAGER         gerente (m) de distribuição
```

O resultado obtido:



```
EN labour dispute
+base dispute
PT conflito (m) trabalhista

EN dissolution
PT dissolução (f)

EN distribution costs
+base distribution
PT custos de distribuição

EN distribution manager
+base distribution
PT gerente (m) de distribuição
```

Figura 5.1: Resultado da aplicação do programa

Dado este texto de input:

```

__BEGIN__
DISPUTE:
  LABOUR -          conflito (m) trabalhista
distributor        distribuidor (m),atacadista (m)
DIVERSIFICATION:   diversificação (f)
  - STRATEGY        estratégia (f) de diversificação
PRODUCT -          diversificação (f) de produtos

```

O resultado obtido:

```

EN labour dispute
+base dispute
PT conflito (m) trabalhista

Linha 5: syntax error (distributor)

EN diversification
PT diversificação (f)

EN diversification strategy
+base diversification
PT estratégia (f) de diversificação

EN product diversification
+base diversification
PT diversificação (f) de produtos

```

Figura 5.2: Resultado da aplicação do programa

Capítulo 6

Conclusão

Este segundo trabalho prático desta unidade curricular foi mais exigente em relação ao primeiro no que toca ao tempo dispensado para a sua execução. Foi necessário um conhecimento mais profundo das ferramentas de *Yacc* e *Flex*.

Com a sua realização conseguimos aprofundar e consolidar todos os conhecimentos obtidos com o primeiro trabalho prático e com as aulas práticas.

De uma forma geral, terminado este trabalho, acreditamos que os objetivos foram concluídos com sucesso apesar dos obstáculos que enfrentámos.