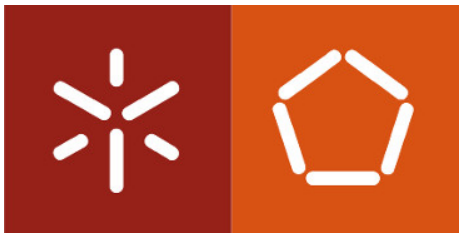
Processamento de Linguagens

Trabalho Prático 2

28 de Junho de 2020

Grupo nr.	36
a83899	André Morais
a85954	Luís Ribeiro
a84783	Pedro Rodrigues



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1 Introdução

1.1 Contexto

Este relatório foi produzido em conformidade com a UC de **Processamento de Linguagens**, correspondente ao segundo semestre do terceiro ano do Mestrado Integrado em Engenharia Informática da Universidade do Minho.

1.2 Problema

O problema consiste em pegar num ficheiro de texto, resultado da conversão automática PDF-TXT de um dicionário EN-PT, e aplicar *reverse engineering* de modo a tirar o máximo partido da informação. O seguinte extrato do dicionário de negócios e finanças EN-PT:

```
dispute:
labour -                conflito (m) trabalhista
dissolution             dissolução (f)
distribution:
- costs                 custos de distribuição
channels of -           canais (mpl) de distribuição
physical - management   controle (m) da distribuição física
distributor             distribuidor (m), atacadista (m)
diversification:        diversificação (f)
- strategy              estratégia (f) de diversificação
product -               diversificação (f) de produtos
```

Deve gerar o seguinte output:

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

EN dissolution
PT dissolução (f)

EN distribution costs
+base distribution:
PT custos de distribuição

EN channels of distribution
+base distribution:
PT canais (mpl) de distribuição

EN physical distribution management
+base distribution:
PT controle (m) da distribuição física

EN distributor
PT distribuidor (m)
PT atacadista (m)

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
```

1.3 Objetivos

Este trabalho pratico tem como principais objectivos:

- aumentar a experiência de uso do ambiente Linux, da linguagem imperativa C (para codificação das estruturas de dados e respectivos algoritmos de manipulação), e de algumas ferramentas de apoio à programação;
- rever e aumentar a capacidade de escrever gramáticas independentes de contexto (GIC), que satisfaçam a condição LR(), para criar Linguagens de Domínio Específico (DSL);
- desenvolver processadores de linguagens segundo o método da tradução dirigida pela sintaxe, suportado numa gramática tradutora (GT);
- utilizar geradores de compiladores como o par flex/yacc.

1.4 Utilização

Na pasta do programa existe uma pasta de input e outra de output, na pasta input são guardados todos os ficheiros resultantes do pré-processamento com o mesmo nome do ficheiro input acrescentando `_corrigido` à frente. Para utilizar o programa basta executar o `program.sh`, existem duas hipóteses:

- `./program.sh input.txt` → não realiza pré-processamento, cria output na pasta output com o mesmo nome do ficheiro de input;
- `./program.sh -p input.txt` → realiza pré-processamento, cria resultado do pré-processamento na pasta input com o mesmo nome do ficheiro input, acrescentando `_corrigido`, depois é executado o programa principal gerando o ficheiro output na pasta output com o nome do ficheiro resultante após pré-processamento.

2 Análise e Especificação

Após uma análise ao problema, consegue-se identificar uma série de requisitos necessários para a resolução do problema

2.1 Requisitos:

- Análise do ficheiro de entrada;
- Definir regras gramaticais necessárias para conversão do ficheiro;
- Reconhecer símbolos terminais presentes no ficheiro;
- Corrigir certas entradas do ficheiro, possíveis de corrigir, de modo a respeitar as regras gramaticais definidas acima.

3 Concepção e Codificação da Resolução

3.1 Gramática

Observando o problema e os requisitos, verificamos que inicalmente, era necessário estudar e analisar o ficheiro de entrada. A partir desta análise foram identificados alguns padrões:

- O ficheiro começa sempre com um texto de introdução que deve ser ignorado pois não contém qualquer termo pretendido para tradução. Posto isto, o nosso problema apenas começa quando encontramos a sequência (__**BEGIN**__);
- É sempre apresentada a letra do abecedário, identificando um grupo de palavras inglesas começadas por essa mesma letra;

- A tradução é composta por uma expressão em inglês, seguida de vários espaços e uma expressão em português, seguida por um \n;
- O número de espaços entre a expressão inglesa e a expressão portuguesa deve ser, no mínimo, 3;
- A componente inglesa é composta por três situações diferentes:
  - **Atribuição da base** (palavra-chave para possivelmente substituir nas seguintes).
    Exemplo:

    ```
    abandonment:
    ```

  - **Expressões com substituição** (expressão contém um -, significando que este - deve ser substituido pela base).
    Exemplo:

    ```
    product -
    ```

  - **Expressão inglesa normal**, que terá uma tradução direta.
    Exemplo:

    ```
    acceleration clause
    ```

- A componente portuguesa pode ser composta por:
  - **Uma expressão portuguesa normal**
    Exemplo:

    ```
    absenteísmo (m)
    ```

  - **Um conjunto de expressões sinónimas separadas por , ou ;**
    Exemplo:

    ```
    com ágio; acimda da paridade
    ```

- As expressões portuguesas podem conter uma indicação do género de cada palavra, identificada entre parêntesis.

  Exemplo:

  ```
  custeio (f) de absorção
  ```

- As atribuições de base podem ou não conter tradução

Após uma análise mais cuidadosa ao ficheiro de entrada e uma aplicação de um simples protótipo da gramática, notou-se que algumas componentes inglesas e portuguesas continuavam na linha seguinte. Então, foi adicionada a hipótese de continuar as expressões nas linhas seguintes, respeitando algumas regras.

Com esta análise, foi possível construir a gramática pretendida para a resolução do problema.

Foi definido o axioma **dic**, que identifica o dicionário EN-PT. Este axioma identifica, entrada a entrada, de forma recursiva, as diferentes hipóteses definidas acima. Primeiramente, é necessário identificar o fim de cada entrada, visto que há a possibilidade de continuar as expressões na linha seguinte. Para isto foi criado o token **NEWLINE**, que identifica o fim de uma entrada e o início da seguinte.

Então, este axioma é definido da seguinte forma:

```
dic: dic linha NEWLINE    {printf("%s",$2);useBase=0;}
   | error NEWLINE        {yyerrok;}
   |
   ;
```

Para que fosse possível identificar erros sintáticos e manter a execução do programa, foi adicionada a segunda derivação do axioma `error NEWLINE`.

Como vimos em cima, uma entrada é identificada pelo símbolo não terminal **linha** e o token `NEWLINE`. Para isto, é necessário criar o axioma *linha*, que identifica todas as possíveis situações de entrada, respeitando as regras definidas acima.

```
1 linha: LETRA                    {$$=strdup("");}
2      | en ENSEP listpt          {if (useBase)
           asprintf(&$$,"EN %s\n+base %s\n%s\n",$1,base,$3);
           else asprintf(&$$,"EN %s\n%s\n",$1,$3);}
3      | en ':'                   {base = strdup($1);
           $$ = strdup("");}
4      | en ':' ENSEP listpt      {base=strdup($1);
           asprintf(&$$,"EN %s\n%s\n",$1,$4);}
     ;
```

1. Corresponde ao caso em que é encontrada a letra, neste caso não tem de fazer rigorosamente nada;

2. Caso em que obtemos uma tradução normal e direta, é composta pela expressão inglesa, a separação entre a componente inglesa e portuguesa e a lista de traduções em português, podendo ser uma ou mais expressões

3. Identifica uma base, sem qualquer tradução.

4. Identifica uma base e a sua tradução à frente, separadas por um conjunto de espaços, identificados pelo token **ENSEP**

Quando existe uma identificação de base, é guardada na variável `base` a palavra correspondente à base, para que depois seja possível substituir. Nos restantos casos de tradução, é definido o valor do axioma como a mensagem de output pretendida.

O axioma acima recorre a dois axiomas diferentes **en** e **listpt**, estes correspondem a ter expressões inglesas e portuguesas,respetivamente.

```
listpt: pt                      {asprintf(&$$,"PT %s\n",$1);}
      | listpt ',' pt           {char* aux;
                asprintf(&aux,"PT %s\n",$3); strcat($$,aux);}
      | listpt ';' pt           {char* aux;
                asprintf(&aux,"PT %s\n",$3); strcat($$,aux);}

en: en palavraEN                {asprintf(&$$,"%s %s",$1,$2);}
  | palavraEN                   {$$ = strdup($1);}
  ;

pt: pt palavraPT                {asprintf(&$$,"%s %s",$1,$2);}
  | palavraPT                   {$$ = strdup($1);}
  ;
```

O axioma `listpt` representa as diferentes expressões sinónimas possíveis, pode conter apenas uma expressão (`pt`) ou uma lista de expressões separada por vírgula ou ponto e vírgula.

O axioma `pt` e `en` representam uma expressão em português e inglês, respetivamente. Estas expressões são compostas por uma palavra apenas, ou por uma lista de palavras separadas por espaços.

Estes axiomas recorrem aos dois últimos axiomas da gramática que indentificam uma palavra inglesa e uma palavra portuguesa.

```
palavraEN: '-'                      {$$ = strdup(base);useBase=1;}
    | PAL                           {$$ = strdup($1);}
    | '(' en ')'                    {asprintf(&$$,"(%s)",$2);}
    | ','                           {$$ = strdup(",");}
    | palavraEN NEWLINE palavraEN   {asprintf(&$$,"%s %s",$1,$3);}
    ;
palavraPT: PAL              {$$ = strdup($1);}
    | '(' pt ')'            {asprintf(&$$,"(%s)",$2);}
    | '+'                   {$$ = strdup("+");}
    | ENSEP                 {$$ = strdup("");}
    | PAL TRACO palavraPT   {asprintf(&$$,"%s%s",$1,$3);}
    ;
```

Como apresentado na gramática acima, uma palavra em inglês pode ser representada por:

- **traço de substituição**, significa que esta palavra corresponde à base;
- **token** `PAL`, corresponde a uma palavra;
- **expressão entre parêntesis**, se abrir parêntesis, deve fechar também;
- **vírgula**, para permitir a inclusão de vírgulas nas expressões inglesas;
- **permissão de um** `NEWLINE` **entre duas palavras inglesas**, permitindo, assim, que se possa continuar a expressão na linha seguinte.

Uma palavra em português pode ser representada por:

- **token** `PAL`, corresponde a uma palavra;
- **expressão entre parêntesis**, se abrir parêntesis, deve fechar também;
- **caracter** +, para permitir a inclusão deste caracter nas expressões portuguesas;
- **permissão de um** `ENSEP`, significa que permite mais do que 3 espaços para separar as palavras portugesas;
- **permissão de um** `TRACO` **entre um** `PAL` **e uma palavra portuguesa**, para tratar os casos em que continuamos a expressão portugesa na próxima linha e esta divisão ocorra no meio de uma palavra.

3.2 Analisador Léxico

Com a gramática concluída, é necessário implementar um analisador léxico, de forma a retornar os elementos necessários que definem a gramática.

Foi necessário criar uma start condition `CORPO` para distinguir texto introdutório do conteúdo do dicionário.

```
__BEGIN__                {BEGIN CORPO; fst=1;}
```

O conteúdo do dicionário começa com a palavra-chave (__BEGIN__), por isso, quando encontrar essa palavra, entro na start condition `CORPO`.

O próximo passo é identificar as letras e retornar o token `LETRA`:

```
^{letra}/\n              {fst=0; return LETRA;}
```

No entanto, quando encontar uma letra e esta letra for a primeira, não é necessário retornar o token `NEWLINE`, mas, caso esta letra não seja a primeira, significa que a linha anterior terminou, então é necessário enviar esse mesmo token. Para isso é utilizada a variável `fst` e a seguinte ER:

```
\n/{letra}\n              {if (fst); else return NEWLINE;}
```

Agora, sempre que encontrar uma palavra, deve retornar o valor da mesma e o token `PAL`:

```
{palavra}|\/    {yylval.s = strdup(yytext); return PAL;}
```

Sempre que encontrar um \n seguido de 10 ou mais espaços e com uma palavra à frente, ignoramos. Isto permite-nos continuar as expressões portuguesas na linha seguinte.

No caso de encontrar a sequência de caracteres anterior, mas com um - no ínicio, significa que a expressão em português foi divida no meio de uma palavra e deve retornar o token `TRACO`

```
\n[ ]{10,}/{palavra}          {;}
-\n[ ]{10,}/{palavra}         {return TRACO;}
```

O token `ENSEP`, caracteriza a separação entre expressão portuguesa e expressão inglesa. Tal como mostrado acima, esta separação é composta por 3 ou mais espaços e pode ser representada na seguinte ER:

```
[ ]{3,}/({palavra}|\(|\")      {return ENSEP;}
```

Neste momento, apenas nos falta retornar o token `NEWLINE` no fim de cada entrada. Isto pode ser definido como um \n seguido de 0 até 10 espaços, em que há frente, encontre uma palavra ou outros caracteres permitidos:

```
\n[ ]{0,10}/({palavra}|-|\()  {return NEWLINE;}
```

Quando apanharmos caracteres permitidos e identificados como símbolos terminais na gramática, devemos retorná-los:

```
[\(\),;\:\-\+]                {return yytext[0];}
```

Visto que toda a gramática está definida pelos tokens já se encontram definidos, quando apanharmos um \n ou um espaço, devemos ignorá-lo:

```
[ ]                           {;}
\n                            {;}
```

Todos os restantes caracteres serão considerados caracteres inválidos:

```
.                           {yyerror("Caracter Inválido");}
```

Como no final do ficheiro é necessário retornar um `NEWLINE` para que a última entrada termine, então foi alterado o funcionamento do analisador léxico ao encontar `EOF`:

```
<<EOF>>                       {BEGIN 0; return NEWLINE;}
```

Isto garante que no final do ficheiro é retornado o token `NEWLINE` e, ao realizar `BEGIN 0`, garanto que o analisador léxico acabe e não fique em *loop* infinito.

Na start condition `INITIAL`, apenas ignoramos todos os caracteres e \n que recebemos, de forma a ignorar o conteúdo introdutório do ficheiro de entrada.

Neste analisador léxico escrito em `Flex`, foram definidas ER gerais:

```
acentos    \xc3[\x80-\xbf]
letra      [a-zA-Z]|{acentos}
palavra    {letra}+(-{letra}+|'{letra}+|\+{letra}+)*
```

3.3 Problemas e Erros

Após a execução do programa, foram verificados vários erros e incoerências presentes no ficheiro de entrada. Para minimizar o número de erros, foi criado um filtro em `Flex`, de forma a pré-processar os dados.

Neste filtro corrigimos algumas situações:

- **Três ou mais espaços entre o traço de substituição e a restante expressão inglesa**, isto fornecia um output não desejado porque considerava esses espaços como uma separação entre inglês e português.
- **Informações sobre seções do PDF no meio do conteúdo útil**, acabava por dar erro sintático porque continha expressões resultantes da conversão PDF-TXT. Facilmente foi identificado que estas expressões ocorriam entre _ e foram facilmente retiradas.
- **Espaçamento nas palavras de substituição incorreto**, dificultava a identificação destas expressões relativamente à continuação das expressões nas linhas seguintes, assim devem começar apenas só com um espaço.

Nas outras diferentes situações de erros em que não foram corrigidas com o pré-processamento, este mesmo erro é reportado num ficheiro designado por `y.log`, contendo a linha e o caracter que causou esse erro.

4 Testes e Resultados

4.1 Teste 1

Input

```
__BEGIN__


A
ADP (automatic data processing)   processamento (m) automático de
                                   dados
above par                         com ágio; acima da paridade
acceptance:                       aceitação (f)
 brand -                          aceitação (f) de uma marca
advertising:

    - agent             agente (m) de publicidade
B
budget:                           orçamento (m)
 - appropriation                  verba ff) orçamentária
budgeting:                        contabilidade (f) orçamentária
 performance -                    controle (m) orçamentário de rendimen-
                                  to


c

COBOL (comon business
 oriented language)                 Cobol, linguagem (f) comum para
                                     negócios
s
sales:
 -   expansion effort   esforço (m) para aumentar vendas
```

4.1.1 Sem pré-processamento

Output

```
EN ADP (automatic data processing)
PT processamento (m) automático de dados

EN above par
PT com ágio
PT acima da paridade

EN acceptance
PT aceitação (f)

EN brand acceptance
+base acceptance
PT aceitação (f) de uma marca

EN advertising agent
+base advertising
PT agente (m) de publicidade

EN budget
PT orçamento (m)

EN budgeting
PT contabilidade (f) orçamentária

EN performance budgeting
+base budgeting
PT controle (m) orçamentário de rendimento

EN COBOL (comon business oriented language)
PT Cobol
PT linguagem (f) comum para negócios

EN sales
+base sales
PT expansion effort  esforço (m) para aumentar vendas
```

Ficheiro log (erros)

```
Linha 15: syntax error )
```

Este erro ocorre porque na linha 15 contém a expressão `verba ff) orçamentária`, ou seja, há um fecho de parêntesis mas não uma abertura, resultando num erro sintático.

Apesar de apenas ter sido apresentado um erro, há incoerências no resultado do programa. Na última entrada, como há 3 espaços entre o - e a expressão `expansion effort`, ele considera esses espaços como a separação entre inglês e português, resultando numa mistura de palavras inglesas e portuguesas no output.

4.1.2 Com pré-processamento

Input após pré-processamento

```
__BEGIN__


A
ADP (automatic data processing)   processamento (m) automático de
                                   dados
above par                         com ágio; acima da paridade
acceptance:                       aceitação (f)
 brand -                          aceitação (f) de uma marca
advertising:

 - agent             agente (m) de publicidade
B
budget:                           orçamento (m)
 - appropriation                  verba ff) orçamentária
budgeting:                        contabilidade (f) orçamentária
 performance -                    controle (m) orçamentário de rendimen-
                                  to


c

COBOL (comon business
 oriented language)                 Cobol, linguagem (f) comum para
                                     negócios
s
sales:
 - expansion effort   esforço (m) para aumentar vendas
```

Output

```
EN ADP (automatic data processing)
PT processamento (m) automático de dados

EN above par
PT com ágio
PT acima da paridade

EN acceptance
PT aceitação (f)

EN brand acceptance
+base acceptance
PT aceitação (f) de uma marca

EN advertising agent
+base advertising
PT agente (m) de publicidade

EN budget
PT orçamento (m)

EN budgeting
PT contabilidade (f) orçamentária

EN performance budgeting
+base budgeting
PT controle (m) orçamentário de rendimento

EN COBOL (comon business oriented language)
PT Cobol
PT linguagem (f) comum para negócios

EN sales expansion effort
+base sales
PT esforço (m) para aumentar vendas
```

4.1.3 Reflexão

Apesar do ficheiro log ter mantido o mesmo erro, visto que o pré-processamento não está implementado para corrigir situações que causaram esse mesmo erro, o conteúdo do ficheiro output foi diferente. Como podemos reparar, a última entrada está agora como pretendida, isto porque com o pré-processamento, o número de espaços entre o - e a expressão em inglês passou de 3 para 1.

4.2 Teste 2

Como último teste, foi verificado as linhas dos ficheiros gerados após execução do programa ao ficheiro fornecido pelos docentes `dic-finance-en.pt.txt`.

```
$ wc -l input/dic-finance-en.pt.txt output/output.txt
  >> 3779 input/dic-finance-en.pt.txt
  >> 10387 output/output.txt

$ wc -l input/corrigido.txt output/outputCorrigido.txt
  >> 3779 input/corrigido.txt
  >> 10393 output/outputCorrigido.txt

$ diff -y output.txt outputCorrigido.txt | grep '|' | wc -l
  >> 200
```

4.2.1 Reflexão

Observando os comandos apresentados acima e os seus respetivos outputs, reparamos que após o pré-processamento, o ficheiro inicial manteve o mesmo número de linhas, no entanto, o output gerado obteve mais linhas do que sem pré-processamento.

Isto acontece porque foram corrigidos alguns erros, mas no entanto, se compararmos as linhas diferentes entre os ficheiros, reparamos que temos várias linhas/traduções diferentes entre o ficheiro output sem pré-processamento e com pré-processamento.

Uma vez mais, provando que sem o pré-processamento obtemos algumas incoerências no resultado.

5 Conclusão

Depois de terminado o segundo e último trabalho da Unidade Curricular de Processamento de Linguagens, concluímos que a realização deste permitiu-nos consolidar os conhecimentos estudados nas aulas da disciplina.

Houve assim, uma aprendizagem notória relativamente ao analisador sintático (Yacc) e às suas funcionalidades, como a declaração de Tokens, declaração dos tipos para os Tokens e Símbolos Não Terminais (caso se justificasse o uso do valor semântico desse não terminal) através do `%union`, e a própria recursividade que o Yacc oferece, permitindo que seja identificado e analisado todo o ficheiro, e por sua vez se corresponde ou não à gramática desenvolvida.

Para além disso, o uso do Flex permitiu-nos que fosse possível a análise léxica também, possibilitando uma "comunicação" Flex+Yacc (Análise Léxica e Sintática). Assim, há uma relação entre os dados de entrada com os Tokens do Yacc.

Para o analisador sintático usar os valores das expressões identificados pelo analisador léxico, para além de receber o token retornado pelo Flex, o valor da expressão é retornado através da variável global previamente combinada de nome yylval.

O uso da linguagem C permitiu que houvesse uma ação relacionada com cada derivação identificada na gramática.

Em jeito de conclusão, consideramos o nosso desempenho bastante satisfatório, pois foi possível desenvolver e resolver todos os objetivos pretendidos no enunciado deste trabalho.