Universidade do Minho

Mestrado Integrado em Engenharia Informática

Processamento de Linguagens

Processador de Inglês Corrente

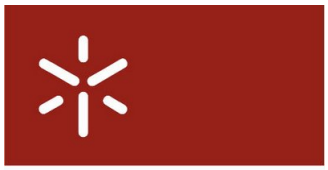
Autores:

A75625 André Almeida Gonçalves

A74576 José António Dantas Silva

A75315 Ricardo Jorge Barroso Certo

16 de Junho de 2017



Universidade do Minho

Conteúdo

1 Introdução

Na realização deste segundo trabalho prático foi desenvolvido um filtro de texto em Flex no âmbito da unidade curricular de Processamento de Linguagens. Tal exercício prático, tem um reforço no incentivo à utilização de expressões regulares e filtros Flex, sendo aplicados todos os conceitos e conteúdos lecionados nas aulas práticas sobre esta matéria.

No desenvolvimento deste relatório serão apresentadas todas as etapas e decisões tomadas ao longo da elaboração da solução para o problema apresentado, desde a descrição do problema até à apresentação do código fonte final em anexo com a devida explicação das opções tomadas para cada problema encontrado.

2 Análise e Especificação

2.1 Descrição do Problema

Após a leitura do enunciado a escolha do grupo recaiu sobre a elaboração do processador de inglês corrente pois trata uma linguagem natural falada através de um algoritmo, sendo sempre um desafio devido à diversidade que estas apresentam. Neste enunciado eram requeridas duas funcionalidades, a expansão das contrações encontradas num dado texto e a apresentação das formas verbais não flexionadas ordenadamente. A apresentação dos dados será efetuada em HTML para uma melhor apresentação, sendo visualizado numa opção o texto com a expansão das contrações efetuadas tendo na segunda opção disponível a visualização de uma tabela com as formas verbais não flexionadas.

2.2 Enunciado

Neste projeto pretende-se que leiam um texto corrente em inglês que faça uso das habituais contrações do tipo I'm, I'll, W're, can't, it's e:

- **a)** reproduza o texto de entrada na saída expandindo todas as contrações que encontrar.
- **b)** gere em HTML uma lista ordenada de todos os verbos (no infinitivo não flexionado) que encontrar no texto, recordando que essa forma verbal em inglês é precedida pela palavra to, sendo a mesma palavra retirada se o dito verbo for antecedido por can, could, shall, should, will, would, may, might. Considere também a forma interrogativa em que o verbo no infinitivo é precedido por do/does ou did.

2.3 Contrações

Sendo uma linguagem natural falada o principal meio de comunicação usado pelos seres humanos, existe uma necessidade de tornar fluente o seu uso, neste caso de estudo concreto na escrita. São exemplo dessa necessidade de fluidez na escrita da linguagem inglesa as contrações. Após uma análise da língua inglesa foram encontradas as seguintes contrações:

Original	***Contração***
am	'm
have	've
is	's
are	're
will	'll
cannot	can't
had	'd
not	n't
have not	haven't
will not	won't

Tabela 1: Tabela com as contrações encontradas

Existe assim a necessidade de identificar as contrações referidas na tabela no input recebido para retornar no output a respetiva expansão.

2.4 Formas Verbais

A língua inglesa como todas seguem um conjunto de regras semânticas que regem a mesma. As formas verbais no infinitivo não flexionado por exemplo são a base de um verbo e, apesar de existirem exceções usualmente estas formas verbais são encontradas das seguintes formas:

- Seguidas pela palavra **to** como são exemplo:
  - *He is going to **live** in the same house as Sarah.*
- Seguidas pela palavra **can, could, shall, should, will, would, may, might** como são exemplos:
  - *You can **make** great things in life.*
  - *One day you will **be** able to **buy** a house.*

- Seguida pela palavra **did, do/does** e um pronome como são exemplo:
  - *Do you **live** in the same house as Sarah?*

3 Desenho e Implementação da Solução

3.1 Expansão das Contrações

Após a análise das contrações a serem detetadas, era necessário encontrar a sua ocorrência nos ficheiros de input. Para tal foi feito recurso de expressões regulares que capturam as contrações descritas no capitulo anterior e efetuando a ação de escrita num ficheiro que recolherá o texto de input com a expansão das contrações encontradas.

```
\'m     { fprintf(texto,"%s"," am"); }

\'s     { fprintf(texto,"%s"," is"); }

\'re    { fprintf(texto,"%s"," are"); }

\'ll    { fprintf(texto,"%s"," will"); }

\'ve    { fprintf(texto,"%s"," have"); }

can\'t  { fprintf(texto,"%s","nnot"); }

n\'t    { fprintf(texto,"%s"," not"); }

\'d     { fprintf(texto,"%s"," had"); }

haven\'t { fprintf(texto,"%s","have not"); }

won\'t { fprintf(texto,"%s","will not"); }
```

Para que todo o conteúdo do texto seja escrito no ficheiro de output **result.txt** na função **main** antes de ser chamada a função **yylex()** foi efetuada a redireção do output para o ficheiro.

```
texto = fopen("result.txt","w");
    yyout = texto;
```

3.2 Listagem dos Verbos

Como descrito no capitulo anterior é necessária a identificação das palavras que têm de seguida ou com intervalo de um pronome um verbo no infinitivo não flexionado. Há então a necessidade de separação destes dois casos, sendo para tal definida uma variável que descreve a distância do verbo

para a palavra que indica a sua presença. Assim sendo foram efetuadas as seguintes expressões regulares para recolha do texto necessário para a captura dos verbos.

```
(to|can|could|shall|should|will|would|may|might)\ [a-z]* { ECHO;
    adicionaVerbo(yytext,1); }

(do|does|did)\ [a-z]*\ [a-z]*  { ECHO; adicionaVerbo(yytext,2); }
```

As expressões regulares anteriores encontram uma das palavras necessárias descritas, seguidas de um espaço uma ou duas novas palavras através da expressão **[a-z]***. A ação para tal expressão é a de **ECHO;** para que o texto não seja removido no ficheiro de output, bem como **adicionaVerbo(yytext,1);** que terá a função de adicionar o verbo à estrutura utilizada para armazenamento dos verbos.

```
void adicionaVerbo(char* texto, int posicao) {

    char* verbo = (char*) malloc(100 * sizeof(char));
    gpointer existe = NULL;

    char* palavra = strtok(texto," ");
    while(palavra != NULL && posicao) {
        palavra = strtok(NULL," ");
        posicao--;
    }

    if(palavra && (strlen(palavra) > 1))
        verbo = strdup(palavra);

    existe = g_tree_lookup(verbos,verbo);

    if(!existe)
        g_tree_insert(verbos,verbo," ");

}
```

Tal função efetua a separação do texto recebido por espaços, ou seja, por palavras, tendo no argumento posição a indicação de que tenha sido encontrada a palavra pretendida como verbo. Após a obtenção dos dados é necessário verificar que a mesma não é nula e caso se confirme é efetuada a cópia para uma variável **verbo** para procura na estrutura de armazenamento dos verbos existentes, sendo adicionada a palavra encontrada caso a mesma ainda não exista na estrutura de armazenamento.

3.2.1 Estruturas de dados

Por forma a tornar a organização dos dados na listagem dos verbos correta e feita de forma prática, foi usada a biblioteca do Glib, mais concretamente a implementação de árvores balanceadas da mesma. Esta permite armazenar os dados ordenadamente através do uso de

```
    verbos = g_tree_new((GCompareFunc)g_ascii_strcasecmp);
```

efetuando a comparação alfabética na inserção das keys, que neste caso serão os verbos encontrados. O uso da função existente **foreach** permite percorrer toda a árvore pela sua ordem alfabética e aplicar a cada elemento a seguinte função:

```
void imprime_dicionario() {
    g_tree_foreach(verbos, imprime_verbo, NULL);
}

gboolean imprime_verbo(gpointer k, gpointer v, gpointer d) {
    fprintf(tabela,"<tr> <td> %s </td> </tr>",k);
    return FALSE;
}
```

Assim conseguimos obter a construção dos elementos da tabela em HTML através de uma simples função já pré-definida pelo Glib.

4 Exemplos de Utilização

Figura 1: Página HTML inicial.

```
PyChecker is a static analysis tool that finds bugs in Python source code and
warns about code complexity and style.  You can get PyChecker from
http://pychecker.sourceforge.net/.

`Pylint <https://www.pylint.org/>`_ is another tool that checks
if a module satisfies a coding standard, and also makes it possible to write
plug-ins to add a custom feature.  In addition to the bug checking that
PyChecker performs, Pylint offers some additional features such as checking line
length, whether variable names are well-formed according to your coding
standard, whether declared interfaces are fully implemented, and more.
https://docs.pylint.org/ provides a full list of Pylint is features.


How can I create a stand-alone binary from a Python script?
----------------------------------------------------------

You do not need the ability to compile Python to C code if all you want is a
stand-alone program that users can download and run without having to install
the Python distribution first.  There are a number of tools that determine the
set of modules required by a program and bind these modules together with a
Python binary to produce a single executable.

One is to use the freeze tool, which is included in the Python source tree as
``Tools/freeze``. It converts Python byte code to C arrays; a C compiler you can
embed all your modules into a new program, which is then linked with the
standard Python modules.

It works by scanning your source recursively for import statements (in both
forms) and looking for the modules in the standard Python path as well as in the
source directory (for built-in modules).  It then turns the bytecode for modules
written in Python into C code (array initializers that can be turned into code
objects using the marshal module) and creates a custom-made config file that
only contains those built-in modules which are actually used in the program.  It
then compiles the generated C code and links it with the rest of the Python
interpreter to form a self-contained binary which acts exactly like your script.

Obviously, freeze requires a C compiler.  There are several other utilities
which do not. One is Thomas Heller is py2exe (Windows only) at

    http://www.py2exe.org/

Another tool is Anthony Tuininga is `cx_Freeze <http://cx-freeze.sourceforge.net/>`_.


Are there coding standards or a style guide for Python programs?
---------------------------------------------------------------
```

Figura 2: Apresentação do texto expandido.

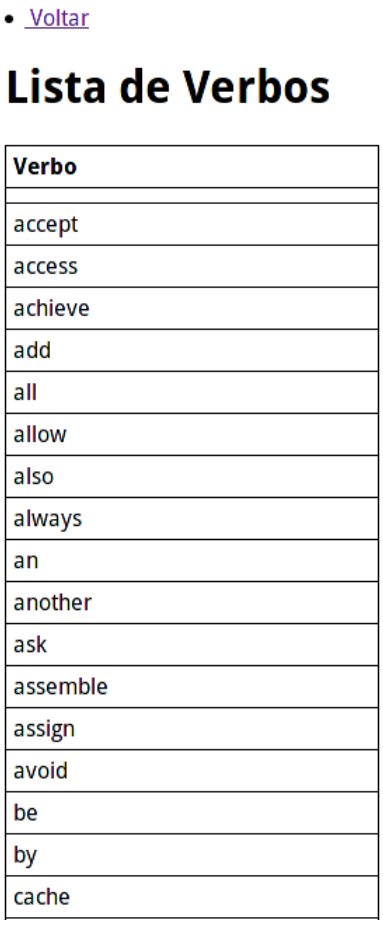


Figura 3: Listagem dos Verbos encontrados.

5 Conclusão

Após ter sido descrito todo o processo de desenvolvimento deste trabalho prático, desde a descrição do problema em causa até á implementação da solução passando pela análise de dados resta agora apresentar uma breve conclusão sobre todo o processo.

A realização deste projeto permitiu-nos desenvolver e consolidarmos a matéria lecionada até ao momento quer nas aulas teóricas, quer nas aulas práticas. As técnicas de utilização de expressões regulares foram de fácil utilização, uma vez que já tinha sido consolidadas no anterior projeto. Estas técnicas estão inseridas no uso do **FLEX** (fast lexical analyzer generator), que funciona como um filtro de texto. O uso do **FLEX** serviu para adquirimos outra sensibilidade relativamente ao uso dos filtros de texto e com isso verificarmos que se trata de uma forma mais simples de desenvolver estes filtros.