Processamento de Linguagens
LEI (3ºano)

Trabalho Prático nº 3
(Yacc)

Ano lectivo 17/18

Objectivos e Organização

Este trabalho prático 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* que satisfaçam a condição LR();
- desenvolver processadores de linguagens segundo o método da *tradução dirigida pela sintaxe*, suportado numa *gramática tradutora*;
- utilizar *geradores de compiladores* como o par lex/yacc

O programa desenvolvido será apresentado a um dos membros da equipa docente, totalmente pronto e a funcionar (acompanhado do respectivo relatório de desenvolvimento) e será defendido por todos os elementos do grupo (3 alunos), em data a marcar.
O **relatório** a elaborar, deve ser claro e, além do respectivo enunciado, da descrição do problema, das decisões que lideraram o desenho da linguagem e a concepção da gramática, do esquema de tradução e respectivas acções semânticas (incluir as especificações `lex` e `yacc`), deverá conter exemplos de utilização (textos fontes diversos e respectivo resultado produzido). Como é de tradição, o relatório será escrito em LaTeX.
O pacote de software desenvolvido (um ficheiro compactado, "`.tgz`", contendo os ficheiros "`.l`", "`.y`", algum "`.c`" ou "`.h`" que precise, os ficheiros de teste "`.txt`", o relatório "`.tex`" e a respectiva "`makefile`") deve ser entregue através do sistema de submissão de TPs, até ao dia **2 de Junho**.

Enunciados

Para sistematizar o trabalho que se lhe pede em cada uma das propostas seguintes, considere que deve, em qualquer um dos casos, realizar a seguinte lista de tarefas:

1. Especificar a gramática concreta da linguagem de entrada.
2. Desenvolver um reconhecedor léxico e sintáctico para essa linguagem com recurso ao par de ferramentas geradoras flex/yacc.

3. 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.

Conteúdo

1 Rede Semântica do Museu da Emigração

Neste projecto pretende-se descrever parte da *rede semântica* que suporta o Museu da Emigração, das Comunidades e da Luso-descendência (www.museu-emigrantes.org) e gerar um grafo (usando GraphViz / WebDot) que permita uma navegação conceptual sobre esse repositório de conhecimento; além de poder percorrer os caminhos do grafo, pode seleccionar um nodo e saltar para uma página com info sobre esse elemento.
Para o efeito, deve definir e processar uma linguagem que lhe permita descrever três tipos de nodos, ou vértices, da rede—emigrante (com dados pessoais e dados relativos ao processo de emigração), obra (palácio, fábrica, hospital, escola, capela, etc.), evento (baile, sarau literário, sarau musical, etc.)—e os respectivos arcos—fez (liga emigrante a obra), participou (liga emigrante a evento).

2 Dicionário Cinematográfico

Considere que, no âmbito da realização de filmes, se pretende criar um **Dicionário** de palavras utilizadas nesta área. Associado a cada palavra pretende-se ter: o significado; a designação comum em inglês; e uma lista de sinónimos.

O seu sistema de tratamento de textos cinematográficos, STTC, deve aceitar uma linguagem que especifique o ficheiro onde está o dicionário (escrito na linguagem que terá de definir) e uma lista de ficheiros onde se encontram textos a anotar.

O objectivo do STTC é ler o dicionário e carregá-lo para um estrutura interna de tal modo que depois possa processar os ditos textos. Para isso, deve procurar em cada um as palavras que estejam presentes no dicionário assinalando-as (sublinhado) e associando-lhes uma `footnote` com o termo em inglês. No fim do texto será gerado um apêndice com a lista de palavras encontradas e respectivos significados.

3 Informação Geográfica

O objectivo deste projecto é construir um *site* para as localidades de Portugal face à sua hierarquia geográfica definida à custa da linguagem OrgGeo abaixo descrita.

OrgGeo é uma Linguagem de Domínio Específico (DSL) que se destina a descrever a organização geo-política de um país, ou sua região. Para o efeito, pretende-se identificar as Localidades de cada Concelho de cada Distrito da região em causa.
A gramática independente de contexto G, abaixo apresentada, define o tipo de linguagem pretendida—cada grupo pode criar a sua variante.
O Símbolo Inicial é `OrgGeo`, os Símbolos Terminais são escritos em minúsculas (pseudo-terminais), ou em maiúscula (palavras-reservadas), ou entre apostrofes (sinais de pontuação) e a string nula é denotada por ε; os restantes serão os Símbolos Não-Terminais.

```
OrgGeo    ⟶  Distritos
Distritos ⟶  IdD  DescD  '.'
           |  Distritos  IdD  DescD  '.'
DescD     ⟶  Concelhos
Concelhos ⟶  Concelho
           |  Concelhos  ';'  Concelho
Concelho  ⟶  Locais  ':'  IdC
Locais    ⟶  Local  LstIds
LstIds    ⟶  ε
           |  ','  Locais
IdD       ⟶  id
IdC       ⟶  id
Local     ⟶  IdL .....
IdL       ⟶  id
```

A informação a incluir em cada `Local` fica ao cuidado do grupo, sendo que se deve contemplar a possibilidade de fornecer um URL para um site do local; caso tal suceda, no site gerado existirá um link para esse site do local.

Uma possível forma de desenvolver o projecto é definida a seguir: Transforme G numa **gramática tradutora**, GT, reconhecível pelo yacc, para gerar um sítio HTML com uma página por cada Distrito, contendo uma lista de items com os Concelhos e para cada um uma outra lista com suas localidades. Complete a GT anterior de modo a gerar também uma página inicial (a *homepage* do seu sítio) com um título (fixo) do projecto e links para as páginas dos Distritos.
Adicionalmente, estenda o seu processador para incluir a construção duma **Tabela de Localidades** que associe a cada identificador de Localidade os respectivos Concelho e Distrito. Note que podem existir Localidades com nomes repetidos, desde que tal não aconteça no mesmo Concelho. Nomes de Concelhos e de Distritos é que não podem ser duplicados.

4 Formato Biblio::Thesaurus

O formato thesaurus ISO 2788 (T2788) é usado para representar ontologias / thesaurus. Dum modo simplificado contem metadados (indicação das linguas, relações externas, inversas das relações, titulo, etc) e um conjunto de conceitos.

Cada conceito inclui:

- um representante na linguagem de base.
- traduções noutras línguas.

- relações conceptuais com outros conceitos.

Pretende-se:

- Escrever uma gramática de T2788.
- Escrever uma gramática e analisador léxico capaz de reconhecer um documento em formato Thesaurus ISO 2788.
- Armazená-lo num estrutura de dados em memória (Hash table glib ?).
- Fazer uma travessia da estrutura de dados e gerar uma página HTML para cada conceito, sendo construído hiperligações de acordo com as relações conceptuais.

Exemplo:

```
                          # directivas / metadados
%language PT EN           # línguas: PT EN
%baselang EN              # língua de base: EN
%inv NT BT                # NT, BT são relações inversas  a NT B => b BT a

                          # conceitos:
animal                    # termo na baselang
PT animal                 # LINGUA termo
NT cat, dog, cow,         # NT = narrow term = termo específico
  fish, ant
NT camel
BT Life being             # BT = broader term = termo genérico
                          # linha em branco : separador de conceitos
cat
PT gato
BT animal
SN animal que tem sete    # scope note = nota explicativa
  vidas e m(e)ia

#comentário               # desde o símbolo '#' até ao fim da linha
```

5 Linguagem para definição de dados genealógicos

Ao falar de histórias de vida, histórias de família, etc é muitas vezes necessário incluir dados referentes às relações de parentesco e outros dados de cariz genealógico.

Pretende-se definir uma notação compacta e formal para definir este tipo de dados.

Esta notação genealógica (ngen) deve cobrir elementos como:

- Nomes. Exemplo:
  - `João Manuel Rodrigues da Silva`
  - `João Manuel/Rodrigues da Silva` (separação nome aplelido)
  - `João Manuel Rodrigues da Silva%2` (distingue entre 2 elementos com o mesmo nome)
- Eventos e Datas. Exemplo:
  - `*1996` (evento=Nasceu ; data 1996)

  - `+1996` (evento=Faleceu; data 1996)
  - `+c1996` (evento=Faleceu; data cerca de 1996)
  - `cc(1996) P` (evento=casamento; data 1996; com a pessoa P)
  - `ev(1996:x)` (evento=x; data 1996)
- Parentescos. Exemplo dada uma pessoa P1:
  - `P P2` (relação P1 tem como pai P2)
  - `PM P2` (relação P1 tem como Avó paterna P2)
  - `-P P2` (relação P2 tem como pai P1)
  - `-PP P2` (relação P2 tem como Avô paterno P1)
  - `F P2` (relação de P2 com casamento atrás descrito)
- Estas relações têm inverso
  - Ficheiros / documentos auxiliares. Exemplo dada uma pessoa P1:
  - `FOTO file.jpg`
  - `HIST file.tex` (em file.tex há uma história em que P1 participa)

```
Manuel da Silva *1977 +2011 [3]  // nome, nascimento, morte, II=3
M  Maria da Silva +2009          // mãe nome, morte da mãe
P  Joaquim Oliveira da Silva     // Pai
MM Joaquina *1930                // mãe da mãe
MP [45]                          // pai da mãe é o #I45, descrito anteriormente
FOTO f.jpg
HIST h1.tex
CC 2000 [2]                      // #I3 casou-se em 2000 com #I8, IF=2
Maria Felisbina *1980 [8]        // Conjugue, nome, nascimento, II=8
F  Serafim da Silva *2004        // Filho (ref. ao CC anterior [3][8])
F  Ana da Silva *2006 [7]{       //
     FOTO f1.jpg                 // dados extra referentes à Ana #I7
     HIST h1.tex
   }
```

Pretende-se portanto:

- definir a linguagem
- escrever uma gramática tradutora
- gerar factos elementares (ou o que acharem interessante gerar)

Exemplo de alguns factos elementares (pode alterar os detalhes)

```
#I3 nome Manuel da Silva
#I3 data-nacimento  1977
#I3 data-falecimento 2011
#I3 tem-como-mae #aut1
#aut1 nome Maria da Silva
#aut1 data-nascimento 2009
#I3 tem-como-mae #aut2
#aut2 nome Joaquim Oliveira da Silva
```

```
#I3 temo-como-MM #aut3
#aut3 nome Joaquina
#aut3 data-nascimento 1930
#I3 temo-como-MP #I45
#I3 FOTO f.jpg
#I3 HIST h1.tex
#F2 = #I3 #I8
#F2 data-casamento 2000
#I8 nome Maria Felisbina
#I8 data-nascimento 1980
#aut4 nome Serafim da Silva
#aut4 data-nascimento 2004
#F2 tem-como-filho #aut4
#I7 nome Ana da Silva
#I7 data-nascimento 2006
#F2 tem-como-filho #I7
#I7 FOTO f1.jpg
#I7 HIST h1.tex
```

6 Escolha espiritual de música

O objectivo deste projecto é construir uma musiteca espiritual, a MEspiritual – a escolha de músicas consoante o estado de espírito.

Para o efeito será necessário definir uma linguagem para descrever cada música disponível na musiteca (nome, autor(es), ano, editor, letra da música, estilo (rock, pop, clássico, ...), tipo, ...). Quanto ao tipo, podem associar-se um ou mais descritores (tais como: calmo, agitada, relaxante, alegre, triste, espevitante, dançante, ...) afectados por uma percentagem.

A partir da descrição desse repositório musical, pretende-se gerar um site que aceite um pedido (escrito numa linguagem específica) baseado no estado de espírito do visitante (enérgico, calmo, ...) e mostre um grafo com todas as músicas disponíveis que se adeqúem melhor ou pior a esse pedido espiritual.
A noção de adequar é deixada ao critério do grupo (quer o conceito, quer a métrica).

7 Tradutor YAML – JSON

Construa um tradutor YAML to JSON.

Sugestão veja a página https://www.json2yaml.com/convert-yaml-to-json.

Se necessario, simplifique alguns detalhes que lhe pareçam demasiado complexos.

Yaml:

```
---
# <- yaml supports comments, json does not
# did you know you can embed json in yaml?
# try uncommenting the next line
# { foo: 'bar' }

json:
  - rigid
  - better for data interchange
yaml:
  - slim and flexible
```

```
  - better for configuration
object:
        key: value
  array:
    - null_value:
    - boolean: true
    - integer: 1
paragraph: >
   Blank lines denote

   paragraph breaks
content: |-
   Or we
   can auto
   convert line breaks
   to save space
```

Json:

```
{
  "json": [
    "rigid",
    "better for data interchange"
  ],
  "yaml": [
    "slim and flexible",
    "better for configuration"
  ],
  "object": {
    "key": "value",
    "array": [
      {
        "null_value": null
      },
      {
        "boolean": true
      },
      {
        "integer": 1
      }
    ]
  },
  "paragraph": "Blank lines denote\nparagraph breaks\n",
  "content": "Or we\ncan auto\nconvert line breaks\nto save space"
}
```

8 Construtor de diaporama

Para apresentar as novidade do DI, pretende-se construir um tradutor que aceite uma descrição de um *diaporama* e gere uma sequências de páginas HTML temporizadas (que ao fim de k segundos, carreguem a pagina seguinte)

O diaporama é uma sequência de elementos. Cada elemento pode ser/conter:

- Uma página inicial de abertura e créditos

- imagens
- páginas simples (exemplo: uma lista de items)
- video[1]
- optionalmente título
- optionalmente audio

As páginas têm associado um tempo de permanência. Ao carregar num browser a primeira página o diaporama deverá decorrer ciclicamente.

1. Defina uma linguagem para descrever o diaporama;
2. Construa um reconhecedor que gere uma sequência de páginas HTML temporizadas;
3. Construa um exemplo demonstrador.

Exemplo de uma página HTML temporizada 'p1.html':

```
<html>
<head>
  <meta charset="UTF-8"/>
  <meta http-equiv="REFRESH" content="34 ;URL=p2.html">
</head>
<body>
 <h1>Pagina p1</h1> ...ao fim de 34s carrega p2.html
 <hr/>
 <img width="100%" src="img1.jpg"/>
</body>
</html>
```

[1]Ver os elementos HTML5 para audio e video