Processamento de Linguagens
MiEI (3ano)

Trabalho Prático nº 2 (YACC)

Ano lectivo 19/20

1 Objectivos e Organização

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

Para o efeito, esta folha contém 6 enunciados, dos quais deverá resolver um escolhido em função do número do grupo (NGr), usando a fórmula $exe = (NGr \ \% \ 6) \ + \ 1$.
Notem que podem escolher outro enunciado por troca com outro grupo que esteja de acordo.
Neste TP, que se pretende que seja resolvido com agilidade, os resultados pedidos são simples e curtos. Aprecia-se a imaginação/criatividade dos grupos ao incluir outros processamentos!

Deve entregar a sua solução **até Domingo dia 7 de Junho**. O ficheiro zipado a entregar (com relatório, ficheiros Lex, Yacc e C) deve ter o nome `'pl19TP2NAl1-NAl2-NAl3'` em que `NALx` é o numero de cada aluno do grupo — *perto da data de entrega serão dadas indicações precisas sobre a forma de submissão e calendário de avaliações a decorrer ao longo de Junho.*
O **relatório** a elaborar, deve ser claro e, além do respectivo enunciado, da descrição do problema, das decisões que lideraram o desenho da solução e sua implementação (incluir a especificação FLex e Yacc em Apêndice), deverá conter exemplos de utilização (textos fontes diversos e respectivo resultado produzido). Como é de tradição, o relatório será escrito em LaTeX (ver template no Bb).

2 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

2.1 Reverse Engineering dum Dicionário financeiro

Em `natura.di.uminho.pt/~jj/pl-20/TP2` está o texto de um dicionário EN-PT. Considere o seguinte extrato de um dicionário de negócio 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
```

Este ficheiro de partida foi resultado da conversão automática PDF-TXT- Pretendemos agora fazer *reverse engineering* de modo a tirar o máximo partido da informação.

No contexto deste TP pede-se que:

- Escreva uma gramática que cubra tanto quanto possível as entradas do dicionário.
- Construa um processador (flex, yacc) que converta o ficheiro num formato mais tratável—ver abaixo um exemplo de saída pretendida.
- Detete e trate os casos de erro sintático, nas situações em que o formato da entrada seja incoerente com a gramática criada acima; nessas circunstâncias, avise a linha onde ocorreu o erro, e continue o processamento.

```
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
```

2.2 DSL para Cadernos de Anotações em HD

Para guardar informação no domínio das *Humanidade Digitais*, pretendemos criar um formato e um processador de *cadernos de anotações*.

Cada *caderno* é composto por diversos pares `(doc,triplos)`. Cada documento inclui um tópico principal (conceito), um título e um texto. Os triplos seguem a notação turtle (versão simplificada), o que corresponde a `(sujeito, relação, objecto)` em que `sujeito` e `predicado` são de novo conceitos.

O texto a seguir é um fragmento de um *caderno de anotações*, conforme pretendido.

```
=== :OnorioL
@tit: Onório de Lima

# Infância no Brasil

Onório de Lima viveu ...

# Percurso cultural no Porto

Entre 1873 e 1930 Onório de Lima viveu no Porto, Cedofeita e foi...

...parágrafos separados por linhas em branco

@triplos:
:OnórioL
   a     :Person  ,
         :Mecenas_cultural  ;
   :Name  "Onório de Lima" ;
   :birth_place :Brasil ;
   :viveu_em :Porto,
             :Gerês ;
   :img "OnórioDL1.jpg" .

:OnórioL
   :pai_de :OndinaL .

:CarolinaDA
   :filho_de :OntinaL;
```

```
   :name "Carolina Dias Antunes".
   :casou_com :JorgeDomingos

=== :JorgeDomingos
@tit: António Jorge Domingos
...
@triplos:
...
```

No contexto deste TP pede-se que:

- Escreva uma gramática para definir a DSL de *caderno de anotação*, acima descrita.
- Construa um processador (flex, yacc) que reconheça e valide cadernos de anotação, escritos na DSL definida acima, gerando um site HTML de acordo com as seguintes regras:
  - crie uma página por cada elemento (conceito) – tópico, sujeito, ou objeto (:OnórioL, :Person, :Mecenas_cultural, :Brasil, etc) que agrupe a informação recolhida, e que hiperlink com tudo o que for possível.
  - incluia na página HTML de um sujeito `:x` uma imagem `"la.jpg"` sempre que encontrar no *caderno* um triplo da forma `(:x :img "la.jpg")`.

Extra: permita a existência de uma secção *meta* onde se indiquem triplos que definam relações inversas (Ex `(:filho_de :inverseOf :pai_de)`) de modo a ser possível a inferência de triplos inversos.

2.3 Conversor de Pug^{--} para HTML

Pug (inicialmente designado por Jade) é um *template engine* que oferece uma notação simples para escrever código HTML reutilizável (permite ter partes do documento que serão preenchidas com valores extraídos de bases de dados ou outras fontes) e que gera HTML puro para mostrar essa informação em páginas Web.
Poe exemplo, o documento a seguir, escrito em notação Pug

```
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) bar(1 + 5)
  body
    h1 Pug - node template engine
    #container.col
      if youAreUsingPug
        p You are amazing
      else
        p Get on it!
      p.
        Pug is a terse and simple templating language with a
        strong focus on performance and powerful features
```

será transformado no documento HTML abaixo:

```
<html lang="en">
  <head><title></title>
      <script type="text/javascript">
        if (foo) bar(1 + 5)</script>
  </head>
  <body>
     <h1>Pug - node template engine</h1>
     <div class="col" id="container">
```

```
        <p>Get on it!</p>
        <p>Pug is a terse and simple templating language with a
strong focus on performance and powerful features</p>
     </div>
  </body>
</html>
```

Neste trabalho pretende-se que:

- Escreva uma gramática que cubra um subconjunto da linguagem Pug à sua escolha (e claramente identificado no relatório de desenvolvimento do TP). Para fazer a escolha veja a documentação em:
  `https://pugjs.org/language/tags.html`
  `https://pugjs.org/language/plain-text.html`
  `https://pugjs.org/language/attributes.html`

- Construa um processador (flex, yacc) que reconheça e valide os documentos Pug^{--}, escritos na sintaxe definida acima, gerando o HTML correspondente.

2.4 Conversor toml2json

TOML[1] é uma linguagem simples, fácil de ler e escrever, para descrever estruturas complexas (usada frequentemente em ficheiros de configuração), desenhada para ser mapeada diretamente e sem ambiguidades para dicionários generalizados. TOML, cujo nome significa *Tom's Obvious, Minimal Language* devido ao seu criador *Tom Preston-Werner*, é usada em muitos projetos de software sendo equiparável a JSON ou YAML.

Pode ver mais detalhes em: `https://github.com/toml-lang/toml`

Para ilustrar esta proposta de trabalho, considere o seguinte exemplo:

```
title = "TOML Example"

[owner]
name = "Tom Preston-Werner"
date = 2010-04-23
time = 21:30:00

[database]
server = "192.168.1.1"
ports = [ 8001, 8001, 8002 ]
connection_max = 5000
enabled = true

[servers]
[servers.alpha]
  ip = "10.0.0.1"
  dc = "eqdc10"

[servers.beta]
  ip = "10.0.0.2"
  dc = "eqdc10"

# Line breaks are OK when inside arrays
hosts = [
  "alpha",
  "omega"
]
```

[1] Consulte `https://en.wikipedia.org/wiki/TOML`

No contexto deste TP pede-se que:

- Escreva uma gramática que cubra um subconjunto da linguagem TOML à sua escolha (e claramente identificado no relatório de desenvolvimento do TP).
- Construa um processador (flex, yacc) que reconheça e valide estruturas/dicionários escritos na DSL TOML^{--} definida acima, gerando o JSON correspondente.

2.5 Conversor de GEDCOM

A Genealogia tem desde sempre atraído muitas pessoas. Investigadores, historiadores, membros da nobreza, pessoas comuns curiosas ou que, por um motivo ou outro, precisam de construir a sua árvore genealógica.

Ao longo dos últimos anos, o formato GEDCOM ganhou popularidade e uma alargada comunidade de utilizadores, o que faz com que seja o formato de eleição para a transmissão e intercâmbio de informação nesta área.

Na internet é possível encontrar muitos recursos e informação sobre este formato.

Neste projeto, pretende-se que o grupo de trabalho desenvolva um conversor de GEDCOM 5.5 para XML/HTML de modo a poder mostrar de imediato numa página Web a árvore genealógica especificada na notação em causa.

Sugerem-se as seguintes etapas para o desenvolvimento deste projeto:

1. Estudar o formato, sua estrutura e alguns casos de estudo (prepare os seus próprios exemplos). Para isso pode consultar os seguintes links:

   `http://homepages.rootsweb.com/~pmcbride/gedcom/55gcch1.htm` – Contem uma descrição do formato juntamente com um rascunho da gramática;

   `http://www4.di.uminho.pt/~jcr/AULAS/didac/ontologias/BibleFamilyTree.ged.txt` – Dataset com as famílias da Bíblia;

   `http://www4.di.uminho.pt/~jcr/AULAS/didac/ontologias/GreekGods.ged.txt` – Dataset com as famílias da mitologia grega;

   `http://www4.di.uminho.pt/~jcr/AULAS/didac/ontologias/RomanGods.ged.txt` – Dataset com as famílias da mitologia romana;

   `http://www4.di.uminho.pt/~jcr/AULAS/didac/ontologias/Royal92.ged.txt` – Dataset com as famílias da realeza europeia.

2. Escrever uma gramática para o GEDCOM 5.5
3. Definir o formato de saída em XML/HTML[2]. Como fonte de inspiração, apresenta-se a seguir um possível formato de saída[3]:

```
<genoa>
  <pessoa>
    <id>I1</id>
    <nome>Victoria Hanover</nome>
    <título>Queen of England</título>
    <sexo>F</sexo>
    <dataNasc>1819-05-24</dataNasc>
    <localNasc>Kensington,Palace,London,England</localNasc>
    <dataÓbito>1901-01-22</dataÓbito>
    <pai>I133</pai>
    <mae>I138</mae>
  </pessoa>
  ...
</genoa>
```

4. Construir um processador (flex, yacc) que reconheça e valide árvores genealógicas escritas na DSL GEDCOM definida acima, gerando o XML/HTML pretendido de acordo com o passo anterior.

[2] Peça orientação ao professor JCRamalho.

[3] Veja pormenores em `http://www4.di.uminho.pt/~jcr/AULAS/didac/ontologias/royalFamilies.xml`

2.6 Visualizador de árvores genealógicas

A Genealogia tem desde sempre atraído muitas pessoas. Investigadores, historiadores, membros da nobreza, pessoas comuns curiosas ou que, por um motivo ou outro, precisam de construir a sua árvore genealógica.

Nas aulas de Processamento e Representação de Conhecimento definiu-se uma linguagem para descrever ontologias (notação ilustrada abaixo) e desenhou-se uma ontologia muito simples para a representação de famílias. Essa ontologia é formada pelos *conceitos* `Pessoa` e `Sexo`, e pelas *relações* `temPai` e `temMãe`.

Neste projeto, pretende-se que o grupo de trabalho desenvolva um conversor de ontologias escritas nessa linguagem para grafos escritos em DOT, que depois podem ser visualizados em GraphViz ou convertidos em SVG para serem usados em páginas Web.

A seguir apresenta-se um breve excerto de uma das ontologias que serão usadas como material de teste e análise:

```
    ###################################################################
    #    Individuals
    ###################################################################

    ###  http://www.di.uminho.pt/prc/ontologies/2020/prc-genoa#Abilio_Silva_Ramalho
    :Abilio_Silva_Ramalho rdf:type owl:NamedIndividual ,
                                  :Pessoa ;
                          :temMae :Custodia_Azevedo_1867 ;
                          :temPai :Manuel_Silva_Ramalho_1866 .


    ###  http://www.di.uminho.pt/prc/ontologies/2020/prc-genoa#Albina_Esteves_Araujo_1910
    :Albina_Esteves_Araujo_1910 rdf:type owl:NamedIndividual ,
                                         :Pessoa ;
                                :temMae :Maria_Araujo_1884 ;
                                :temPai :Henrique_Luiz_Araujo_1867 .
...
```

Seguem-se algumas explicações sobre a notação usada acima:

- Linhas iniciadas por "#"são comentários e devem ser ignoradas;
- Uma ontologia nesta forma é uma lista de triplos: (sujeito, predicado, objeto);
- O sujeito é sempre um identificador (URI) dum indivíduo;
- O predicado é sempre um identificador (URI) duma relação ou propriedade;
- O objeto pode ser um identificador (URI) dum indivíduo ou conceito, ou ainda um valor escalar: string, número, ...
- Um triplo normal termina por ".";
- Um triplo terminado por ","significa que o próximo triplo está abreviado e vai partilhar o mesmo sujeito e predicado;
- Um triplo terminado por ";"significa que o próximo triplo está abreviado e vai partilhar o mesmo sujeito.

Sugerem-se as seguintes etapas para o desenvolvimento deste projeto:

1. Estudar o formato, sua estrutura e alguns casos de estudo (prepare os seus próprios exemplos). Para isso pode consultar o seguinte link:

   `http://www4.di.uminho.pt/~jcr/AULAS/didac/ontologias/mini-familia.ttl` – Contem o exemplo completo do excerto acima;

   Outros – O docente irá fornecer mais exemplos.

2. Escrever uma gramática para a notação ilustrada para este tipo de ontologias.
3. Estudar e definir o formato de saída: `https://www.graphviz.org/` ou `https://graphviz.gitlab.io/_pages/pdf/dotguide.pdf`
4. Construir um processador (flex, yacc) que reconheça e valide ontologias genealógicas escritas na DSL definida acima, gerando o DOT pretendido de acordo com o passo anterior.
5. Seria interessante incluir relações de parentesco inferidas a partir da informação dada: `temAvô`, `temIrmão`, `temTio`...