Processamento de Linguagens I
LESI + LMCC (3º ano)

Trabalho Prático nº 1
(Lex e Yacc)

Ano lectivo 05/06

1 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

Para o efeito, esta folha contém 8 enunciados, dos quais deverá resolver pelo menos um[1].
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 até ao dia **26 de Abril (4ªfeira)**, através do sistema electrónico para recepção de TPs cujo endereço é disponibilizado na página WWW da disciplina (os alunos devem fazer a inscrição do grupo no dito sistema o mais cedo possível).

[1] Se resolver mais, cada um será avaliado separadamente e a nota final será a média das notas individuais obtidas.

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.

2.1 SOS Home

Considere que existe em Portugal um *Serviço de SOS*, SOS-Home, criado especificamente para problemas domésticos. Este serviço funciona com base em mensagens SMS que indicam o tipo de problema, a descriçao sucinta do problema (opcional), o carácter de urgência (URG ou nada), a morada e nome de quem quer ser socorrido. Os vários items são separados por um sinal de pontuação (por exemplo ","").

```
SOS tipo_problema, descrição, n_urgencia, morada, nome.
```

No tipo de problema devem ser usadas as seguintes palavras-chave: falha de electricidade (fee); avaria de electrodoméstico (ava); problema na canalização (can); incêndio (inc); inundação (inu); chave por dentro da porta (cha); assalto (ass); acidente pessoal (aci); etc.

A central que recebe estas mensagens precisa de uma aplicação que interprete estas mensagens e crie um sistema de informação, baseado numa interface Web (acessível em qualquer parte através de um *browser WWW*), que permita à equipa, especializada em resolver cada tipo de problema, saber o trabalho que tem a fazer. A interface do sistema tem uma página central, com informações diversas (não relevantes para este contexto), e um link para a página auxiliar relativa a cada equipa.

A empresa SOS-Home está organizada nas seguintes equipas: electricistas e técnicos de electrodomésticos (elect); canalizadores (canal); bombeiros (bom); para-médicos (para); e polícia (pol).

A página central, com links para páginas de pedidos vazias, é criada quando o sistema central da SOS-Home recebe uma mensagem SMS de inicialização que além da data actual contém a descrição nominal dos membros de cada equipa, do tipo

```
INIT 2006.03.25,
     elect(a,b,c) canal(x,y) bom(m,n,o,p) para(d,f) pol(i,j).
```

Cada vez que uma mensagem chega, é analisada pelo sistema, é-lhe atribuido um número de código em série, único (número do pedido) e o pedido é acrescentado no fim da lista de pedidos da página WWW correspondente à equipa em causa.

Quando uma equipa atende uma chamada, envia outro SMS à central, indicando a sigla da equipa (ver lista acima) e o número do pedido servido, para que seja retirado da lista.

```
SOSrecep equipa, num_pedido.
```

Caso tenha socorrido o pedido, mas não tenha podido resolver por não ser da sua especialidade, a mensagem respectiva deve ainda indicar a que equipa se destina, de modo a que o pedido (mantendo o número) seja transferido automaticamente de listas.

```
SOStransf equipa nova_equipa num_pedido.
```

Construa então um interpretador de mensagens que permita realizar o sistema de informação pretendido para gestão da SOS-Home.
É desejável que a *lista de equipas* e a *lista dos tipos de problema* possam ser facilmente alteradas (em cada novo sistema) sem ter de se alterar a gramática.

2.2 Correcção automática de documentos

As tarefas associadas à correcção de documentos baseiam-se na inserção, remoção e substituição de palavras ou frases. Essas acções são, por vezes, muito enfadonhas porque se repetem ao longo do documento. Considere a criação de uma mini-linguagem composta por um conjunto de comandos tais como:

```
inserir texto linha coluna
```

Este comando indica que deve ser inserida uma palavra ou frase, genericamente um *texto*, na linha (a primeira terá o número 1) e coluna (a primeira será a 1) indicadas.
Os comandos de substituição serão:

```
substituir N texto1 texto2
substitodas texto1 texto2
```

Estes comandos permitem substituir a N-ésima ocorrência de texto1 pelo texto2 (caso N seja omisso, será a 1ª), ou todas as ocorrências.
Os comandos de remoção serão:

```
apagar N texto
apagartodas texto
apagarlinha L
```

Estes comandos permitem apagar um texto na sua N-ésima ocorrência, ou todas as suas ocorrências, ou então apagar a linha L.
É necessário então criar um sistema que recebe um texto (o nome do ficheiro será passado como parâmetro do programa na linha de comandos) e um comando e gere automaticamente um ficheiro Lex que efectue a alteração pedida no documento original.
Será interessante que o seu editor (interpretador de cada comando de edição) gere uma makefile para activar o Flex de modo a gerar o filtro pretendido e aplique esse filtro ao texto de entrada; no fim o ficheiro produzido pelo filtro deve ser renomeado, ficando com o nome do texto-fonte. O editor deve aceitar mais comandos que serão aplicados sucessivamente ao texto editado.

2.3 Criação automática de visualizadores de redes de transportes

Considere que se pretende gerar automaticamente uma representação visual duma rede de transportes descrita através de uma especificação textual. Esse texto poderá ser organizado de duas formas: em função dos diversos meios de transporte (considere-se "avião", "comboio"e "camionete"), indicando o horário das viagens possíveis entre cada dois pontos (localidades) da rede; ou em função da ligação entre dois locais, indicando as alternativas possíveis de meios de transporte. Claro está que no mesmo texto os dois tipos de especificação podem ser misturados.

No primeiro tipo de especificação e para cada meio de transporte deve ser indicado, para cada duas localidades, a distância, o tempo previsto, o custo e uma lista de horários possíveis. Exemplo:

```
avião
porto frankfurt 2500k 3h30 400e 9h30 16h20
porto madrid 500k 1h00 250e 6h30 11h45 14h20 18h00
comboio
porto lisboa 300k 3h30 50e 8h45 14h15
porto madrid 600k 5h15 105e 5h30 13h30
```

No segundo tipo de especificação e para cada par de localidades, deverá ser indicado o meio de transporte, a distância, o tempo previsto, o custo e uma lista de horários possíveis. Exemplo:

```
porto madrid
avião 500k 1h00 250e 6h30 11h45 14h20 18h00
comboio 600k 5h15 105e 5h30 13h30
```

Desenvolva um processador que analise o programa fonte e gere instruções de desenho em GraphViz. Essas instruções de desenho serão posteriormente usadas pelo interpretador de GraphViz para visualização de um grafo que represente as localidades e as ligações entre estas, usando cores diferentes para cada tipo transporte e indicando sobre cada ramo o custo e o horário. O sistema deve detectar e rejeitar definições de viagens repetidas (entre as mesmas localidades e no mesmo meio), rejeitando-as.

A informação sobre GraphViz encontra-se na página
`http://www.ipb.pt/~mjoao/ProcLing.html`.

2.4 Skelcal

A linguagem Skelcal – *skeleton calculator* – é uma linguagem para descrição e manipulação de *esqueletos ("skeletons") de texto* (*templates*) quaisquer (programas, documentos, etc.). As instruções dessa linguagem terão de permitir:

- construir templates,
- instanciar um template,
- escrever ou ler templates

- executar comandos

Considere que a linguagem Skelcal é definida pela seguinte gramática (acrescente outro açucar sintáctico se achar necessário):

```
prog : eesk *

eesk : id = eesk                      // associa um skel a um ID
     | id                             // refere-se ao skel associado ao ID
     | ≪ (TXT | param)* ≫              // skel constante
     | id < file                      // associa ao ID um skel lido de um ficheiro
     | eesk( (idp = eesk)* )          // instancia alguns param de um skel
     | eesk + eesk                    // concatena dois skel
     | eesk > file                    // grava skel num ficheiro
     | ‘comando linux‘                // skel obtido pela stdout da execuÃ§Ã£o
     | id:param ’*’ eesk-list         // CONCAT  <id(param=li) | li in list>

eesk-list :  ≪$ lista $≫
eesk-list :  ≪$ $≫

lista : eesk
      | lista $ eesk

param : ≪id≫
```

Para clarificar a ideia pretendida com a linguagem Skelcal e o processador pretendido, segue-se um exemplo (tente adivinhar o que ficava em output.tex...):

```
relat = ≪\documentclass[portuges,a4paper]{article}
\usepackage{babel}
\usepackage[latin1]{inputenc}
\usepackage{t1enc}
\usepackage{aeguill}
\begin{document}
\title{≪tit≫}
\author{≪aut≫}
\date{\today} \maketitle
\begin{abstract}
  ≪abstract≫
\end{abstract}
  ≪corpo≫
\end{document}
≫

ver=≪\begin{vverbatim}
≪x≫
\end{vverbatim}
≫

manual= relat(tit=≪Relatório ≪tool≫≫
              aut=≪JJoao≫
```

```
            corpo=verb(x=≪c1≫))

f < "skelcal.c"

manual(tool=≪skelcal≫ c1=f) > "output.tex"
```

Desenvolva então um processador que interprete (reconheça e execute) os comandos da linguagem Skelcal.

2.5 Concurso de Programação

No âmbito da realização de um concurso de programação pretende-se criar e manter de forma expedita um sítio WWW com todas as informações necessárias para os interessados. A estrutura base desse sítio seguirá um formato standard que deixamos ao seu cuidado. Contudo terá de conter, no mínimo, 2 páginas: uma, a **página de entrada**, com o nome do concurso e toda a informação elementar; outra, com as equipes inscritas e sua pontuação, chamemos-lhe a **tabela classificativa**.

A novidade desta proposta de trabalho é que se quer permitir que o Organizador do Concurso, os Concorrentes e os membros do Júri interajam com o sistema via mensagens curtas enviadas por SMS. Assim o sítio irá sendo construído e alterado à medida que são recebidas mensagens que podem ser de vários tipos: *inicialização*, *inscrição*, *classificação* e *ordenação*.

Existe um conjunto de informação que tem que ser transmitida ao sistema no início da organização do evento: local, datas, nº de questões da prova, etc. Um exemplo de mensagem de inicialização seria:

```
INIT Braganca, IPB. 14/05/2005. 8?
```

Esta mensagem significa que o concurso decorrerá no IPB, em Bragança, no dia 14 de Maio de 2005 e a prova é composta por 8 problemas. Com esta informação é construído a página de entrada (base) do sítio, a qual irá ser guardado num ficheiro HTML, que é criado nesse momento.

Depois o sítio deve ser automaticamente actualizado à medida que são recebidas mensagens SMS com inscrição de equipas. Cada inscrição contém o nome da equipa, instituição, elementos da equipa (nome1, idade1, nome2, idade2) e nome do tutor. Uma mensagem de inscrição poderá ser:

```
INS cascoes UM (Pedro,20,Luis,19) Costa
```

Esta mensagem significa que a equipa formada pelo Pedro e pelo Luís chama-se "cascoes"e vem da Universidade do Minho, tendo como tutor o Prof. Costa. Ao receber esta mensagem o programa de gestão terá de acrescentar mais uma linha à tabela de equipas inscritas que constitui a 2ª página do sítio. Caso ainda não exista, o respectivo ficheiro HTML terá de ser criado.

No dia do concurso os professores corrigem as soluções apresentadas pelos alunos e actualizam a tabela de classificações usando também mensagens SMS. Logo que um problema é corrigido e classificado é enviada uma mensagem do tipo:

```
CLASS cascoes 3 5
```

Esta mensagem indica que a equipa "cascoes"teve cotação 5 no problema 3. Ao recebê-la, o sistema de gestão terá de actualizar a tabela classificativa na linha e coluna correspondentes à equipa e à questão indicadas.

No final é feito o cálculo dos totais (somatório da pontuação em cada questão, considerando-se 0 como valor por omissão), e a ordenação da tabela classificativa. Para realizar tal ordenação e actualização do ficheiro HTML correspondente, será enviada uma mensagem do tipo que se segue e onde se indica a hora de conclusão da avaliação do concurso:

```
FECHO 18h30
```

Construa então um interpretador de mensagens que permita recolher toda a informação sobre o concurso, sobre as equipas e sobre as classificações e actualize permanentemente o sítio.

2.6 Sistema SMS do Teatro de Bragança

Pretende-se implementar um sistema de reservas via SMS para os espectáculos do Teatro de Bragança. O auditório do teatro tem um conjunto de lugares disponíveis, cada um identificado por um nº de fila e um nº de lugar.

O sistema deve interpretar as mensagens recebidas, consultar a base de dados do teatro (estruturas que indicam os espectáculos em cena, os lugares disponíveis, etc.) e escrever uma mensagem de retorno com a informação pretendida ou a confirmação da reserva.

Através de mensagem SMS deverá ser possível consultar os espectáculos, escrevendo:

```
CONSULTA 23/5
```

Esta mensagem deverá obter como resposta por parte do sistema, todos os espectáculos do dia 23 de Maio e as respectivas horas.

Se se pretender informação sobre um espectáculo em particular poderá ser feita uma consulta do tipo:

```
CONSULTA 23/5 21:30
```

Neste caso, é dado o nome do espectáculo e o numero de lugares vagos.

Para efectuar a reserva deve ser escrito um outro tipo de mensagem, por exemplo:

```
RES nºBI 2 + 25/5 21:30
```

Este comando tenta reservar dois lugares consecutivos mais perto do palco para o espectáculo das 21:30 do dia 25 de Maio. Esta mensagem pode ter dois tipos de resposta: a reserva está feita (fica associada ao nºBI apresentado na mensagem) e é indicado o número dos lugares e o número da fila; ou não é possível fazer a reserva de dois lugares consecutivos.

Deve ser possível também cancelar reservas, com uma mensagem do tipo:

```
CANC nºBI 34 G 25/5 21:30
```

Trata-se de cancelar a reserva efectuada pelo nº de BI especificado, para o lugar 34 da fila G, do espectáculo das 21:30 do dia 25 de Maio. As reservas têm de ser canceladas uma a uma.

Nota: A informação dos lugares do auditório e respectivas reservas deve estar em memória e será carregada inicialmente a partir de uma mensagem de inicialização que seguirá o formatado que achar conveniente.

Construa então um interpretador de mensagens que permita inicializar a estrutura de dados básica e responder às questões ou reservas colocadas pelos utilizadores.

2.7 Composição Automática de Testes

Pretende-se desenvolver um sistema que permita a criação automática de testes com base num conjunto de questões previamente definidas. Cada questão tem uma disciplina, um texto, um tema e um grau de dificuldade associado. Quando é requerido um teste de determinada disciplina é indicado o número de questões pretendido e o grau de dificuldade que as questões devem ter. Há portanto duas formas de interagir com o sistema: inserção de questões e composição de novos testes. Estas operações serão realizadas com base no conjunto de comandos que se explica a seguir.

O comando:

```
INSERT <processamento de linguagens> <compiladores>
       "Quais as fases de compilação?" (B)
```

insere uma questão sobre *compiladores* da disciplina de *processamento de linguagens* com um *grau de dificuldade baixo* (B-baixo, M-médio, A-alto).

Após a inserção de um conjunto de questões razoavelmente grande é possível executar um comando do tipo:

```
COMPOSE 20 M <processamento de linguagens>
        [ANO: 3; CURSO: LESI/LMCC; CHM: 1]
```

Este comando requer a composição de um teste para a disciplina de *processamento de linguagens*, com 20 questões de grau de dificuldade Médio (os temas, referentes à disciplina, serão quaisquer). O teste será destinado ao 3ª ano do curso de `LESI/LMCC` e corresponde à 1ª chamada. O resultado final deste comando será a criação de um ficheiro "`.tex`" com o teste requerido.

Para produzir o teste em LaTeX use o *template* disponível no sítio W3 da disciplina (`www.di.uminho.pt/~prh/curpl105.html`), na rubrica "`Trabalhos Práticos`".

Construa, então, um programa que interprete estes comandos e mantenha a informação actualizada numa estrutura em memória, ou ficheiro, e gere quando necessário um teste em LaTeX sobre um tema de uma determinada disciplina. Caso não haja em memória número suficiente de perguntas com o grau de dificuldade solicitado, será emitida uma mensagem de erro.

Além dos dois comandos (acima explicados para criação da base de dados de perguntas e para composição de testes), a linguagem que definir deve incluir pelo menos dois *interrogadores* que permitam ao professor obter alguma informação

sobre o estado do sistema. Por exemplo, para saber *quantas/quais perguntas* existem sobre determinado tema, ou de uma dada disciplina e determinado grau de dificuldade.

2.8 A Hierarquia de Imagens do CRI

Considere que no Centro de Recursos Informáticos (CRI) da ESTiG, se utiliza um sistema interessante para gerir o software que é instalado e utilizado nas máquinas laboratoriais. Há um servidor para cada laboratório onde são instaladas várias *imagens de possíveis sistemas*. Uma **imagem** é constituída por um sistema operativo (SO) base e vários pacotes de utilitários complementares. No início de cada utilização, o utilizador escolhe a imagem que quer carregar. No fim da sua sessão, quando o utilizador desliga a sua máquina, o conteúdo desta é apagado para na sessão seguinte carregar uma nova imagem. Desta maneira, garante-se que quando um novo utilizador selecciona uma dada imagem esta é-lhe oferecida tal como foi criada originalmente, sem vírus nem corrupções.

Com a adopção desta tecnologia tornou-se necessária a catalogação sistemática dessas imagens e a criação de uma ferramenta de gestão de imagens.

Para o efeito, pretende-se que defina uma linguagem para o registo de imagens, atendendo às seguintes considerações:

- As várias imagens formam uma ou mais hierarquias, de modo a usufruir do mecanismo de herança relativamente aos pacotes de software instalados.
- Uma imagem que está na raíz de uma hierarquia (marcada como *root*) é identificada por um nome ou código, e é descrita por um conjunto de atributos como, por exemplo, o espaço ocupado pela imagem, o autor da imagem, a data de criação, a indicação do sistema operativo de base, uma explicação que justifique a sua criação, a indicação dos vários pacotes de software instalados (cada um pode ter a ele associadas uma ou mais notas – versão instalada, acessórios, ...).
- Uma imagem pode ser definida à custa de uma imagem já existente indicando apenas o seu identificador, o nome da imagem *pai* (aquela que lhe está acima na estrutura hierárquica) e os pacotes (e respectivas notas) que lhe acrescenta.

Assim, uma imagem pode ser criada com um comando do tipo:

```
nome_imagem tam autor data so pacotes
```

ou usando uma outra imagem como *pai*:

```
nome_imagem imagem_pai tam autor data pacotes
```

Desenvolva, então, um processador para essa linguagem de descrição de imagens que, recebendo uma instância do tipo de documento definido, produza uma página HTML com o seguinte conteúdo:

- A página inicial (raíz do site) contém um índice remissivo de imagens.
- Cada imagem aparece individualizada das restantes, numa página HTML própria.

- Para cada imagem, é mostrada a descrição com os vários atributos e a lista de componentes de software que a compõem, dividida em duas partes: os pacotes próprios e os herdados.
- Cada pacote deverá apresentar uma sub-lista com as notas que lhe estão associadas.
- Cada imagem herdada deverá indicar a imagem *pai*, podendo, até, conter um link para essa imagem *pai*.