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

Trabalho Prático nº 1
(Lex e Yacc)

Ano lectivo 2004/2005

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 7 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 **18 de Abril (2ªfeira)**, 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 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.

Sugestão: para geração da tabela classificativa em HTML recomenda-se o uso da biblioteca de combinadores em C a que se faz referência mais abaixo, na questão 2.5.

2.2 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.

2.3 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/curpl104.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.

2.4 Modelação de sistemas com base na linguagem FDL

FDL (Feature Description Language) é uma linguagem de modelação de sistemas que permite identificar as entidades envolvidas num determinado domínio e seus relacionamentos. Considere o seguinte exemplo:

```
carro: all(carroçaria, transmissão, motor, potência, atrelado?)
transmissão: one-of(automático, manual)
motor: one-or-both(eléctrico, gasolina)
potência: one-of(baixa, média, alta)
```

Neste exemplo o operador `all` indica que a classe especificada, `carro`, é formada ao todo por 5 subclasses que têm de co-existir. Cada uma dessas subclasses pode ser obrigatória ou opcional (`?`) e pode ela própria ser composta por outras subclasses (elemento composto) ou não ter subclasses associadas (elemento atómico). Quando as subclasses estão em alternativa, a escolha pode ser exclusiva (apenas uma delas), o que é indicado pelo operador `one-of` como sucede com `transmissao`, ou não-exclusiva (uma ou outra ou ambas), o que é indicado pelo operador `one-or-both`, como é o caso da classe `motor`. Existem outros operadores que não constam no exemplo tais como: `one-or-more` (composição de um ou mais elementos) e `zero-or-more` (composição de zero ou mais elementos).

Desenvolva um processador que analise o texto fonte e gere instruções de desenho em GraphViz—documentação completa sobre esta ferramenta para desenho de Grafos e respectiva linguagem de descrição de diagramas, pode ser obtida no sítio W3 da disciplina (`www.di.uminho.pt/~prh/curpl104.html`), na rubrica "`Notas Pedagógicas e Material de Apoio Diverso`". Essas instruções de desenho serão posteriormente usadas pelo interpretador de GraphViz para visualização de um grafo que representa o Diagrama de Entidades e Relações (DER) implícito na especificação FDL expressa no texto fonte. Para a respectiva tradução, considere que as entidades serão representadas por nodos do grafo e as relações entre elas serão os ramos etiquetados pelo tipo de dependência (dentro das 5 hipóteses acima identificadas). Elementos opcionais serão marcados por alguma simbologia gráfica à sua escolha.

2.5 Classificações do Paris-Dakar

Considere que possui uma base de dados que armazena informação relativa às classificações obtidas pelos participantes no rally Paris-Dakar numa dada etapa. Esta informação está armazenada textualmente em registos em que cada registo tem os seguintes campos:

`Nome`	:	$< nome >$
`Numero`	:	$< numero >$
`Tempo`	:	$< tempo >$
`Penalizações`	:	$< penalizacao^* >$

Os registos são separados por uma linha em branco. Nesta notação $< nome >$ e $< numero >$ indicam o nome e número do concorrente, $< tempo >$ denota o tempo despendido para concluir a etapa. O tempo tem o seguinte formato `hora:minutos:segundos`. As penalizações formam uma lista, possivelmente vazia, de penalidades separadas pelo caracter `;`. Uma penalização pode ser uma de três coisas: `"condução perigosa"`, `"falha de controlo"`, `"atalho"`.

Segue-se um exemplo de registo

`Nome`	:	"Carlos Sousa"
`Numero`	:	1
`Tempo`	:	1:50:24
`Penalizações`	:	`"condução perigosa"` ; `"atalho"` ; `"condução perigosa"`

que indica que o condutor com nome Carlos Sousa, com o número 1, gastou o tempo de 1:50:24 horas para concluir a etapa. Porém, este condutor foi penalizado por 3 razões, pois foi visto por duas vezes a conduzir perigosamente e uma vez a atalhar caminho.

As penalizações detectadas originam que o tempo final seja incrementado de acordo com uma tabela que é definida para a etapa em questão. Apresenta-se de seguida um exemplo de uma possível tabela de penalizações

`"condução perigosa"`	=	0:10:00
`"falha de controlo"`	=	1:00:00
`"atalho"`	=	1:30:00

De acordo com a tabela apresentada, nesta etapa e por cada atalho detectado a um piloto, o seu tempo final é aumentado de 1 hora e 30 minutos, enquanto que uma pena por condução perigosa apenas agrava o seu tempo em 10 minutos e um esquecimento de registo num ponto de controlo acrescenta 1 hora. *Esta tabela pode encontrar-se em qualquer sitio da base de dados textual. Isto é, pode estar no seu inicio, fim, ou mesmo encontrar-se no meio dos registos.*

Desenvolva um processador para esta base de dados textual que analise léxica e sintacticamente uma dada instância da base de dados. Considere ainda que o processador deve produzir uma listagem com a classificação final da etapa do tipo (nome do piloto, tempo final), ordenada por ordem decrescente de tempo final. Esta classificação será disponibilizada ao público numa pagina Web. Assim, produza esta classificação em formato HTML. Utilize para tal combinadores C para representar/produzir HTML tal como os combinadores incluídos na ferramenta Gram2C, apresentada em MPIII (acessível no sítio W3 da disciplina (`www.di.uminho.pt/~prh/curpl104.html`), na rubrica "`Notas Pedagógicas e Material de Apoio Diverso`").

2.6 Formatador de Tabelas em C

Neste trabalho prático pretende-se que desenvolva um processador para representar em texto puro (ASCII) tabelas definidas em HTML. Como sabe, a linguagem HTML tem marcas (ou etiquetas) pré-definidas para representar tabelas e seus elementos. De seguida apresenta-se um exemplo de uma tabela em HTML.

```
<TABLE>
<TR> <TD> This   </TD> </TR>
<TR> <TD> is  </TD> <TD> a </TD> </TR>
<TR> <TD> <TABLE>
          <TR> <TD> This </TD> <TD> is </TD> </TR>
          <TR> <TD> another </TD> </TR>
          <TR> <TD> table </TD> </TR>
          </TABLE>
</TD> <TD> table </TD> </TR>
</TABLE>
```

Apresenta-se agora a representação em ASCII que se pretende construir para as tabelas HTML.

```
|------------------|
|This        |     |
|------------------|
|is          |a    |
|------------------|
||----------||table|
||This   |is||     |
||----------||     |
||another|  ||     |
||----------||     |
||table  |  ||     |
||----------||     |
|------------------|
```

Note que em HTML não é necessário que todas as linhas tenham o mesmo numero de colunas (o que faz com que certos browsers produzam uma representação "feia" das tabelas).

Para desenvolver este processador considere o problema mais genérico de desenvolver um conjunto de combinadores de formatação de tabelas em C. Assim, defina uma linguagem abstracta para tabelas e utilize o sistema Gram2C, apresentado em MPIII (acessível no sítio W3 da disciplina (`www.di.uminho.pt/~prh/curpl104.html`), na rubrica "`Notas Pedagógicas e Material de Apoio Diverso`") para gerar o tipo (abstracto) e os construtores de tabelas. Utilize estes construtores nas acções semânticas do parser e desenvolva as funções de formatação como um catamorfismo sobre este tipo de dados (também ele produzido pelo Gram2C). Por ultimo, inclua a gramática abstracta e as funções que produzem a representação ASCII como mais um exemplo em Gram2C.

2.7 A Hierarquia de Imagens nos Laboratórios do DI

No Departamento de Informática da Universidade do Minho, utiliza-se 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 por parte de quase todo o corpo docente, o número de imagens proliferou de modo a satisfazer as exigências de software específico para cada disciplina, tornando 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 (haverá uma árvore hierárquica para cada SO), 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 constituída por uma descrição onde constarão vários 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 intermédia da árvore contém a indicação da imagem que ela especializa, ou seja, identifica a imagem *pai* (aquela que lhe está acima na estrutura hierárquica), tem o seu próprio identificador e uma descrição semelhante à da raiz, mas devido à herança já não se fará referência ao SO nem aos pacotes incluídos nas imagens ancestrais (ou antecessoras).

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:

- No início a página contém um índice remissivo de imagens.
- Cada imagem aparece individualizada das restantes (linhas horizontais, tipo de letra diferente, ...).
- 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 componentes próprios e os herdados.
- Cada componente deverá apresentar uma sub-lista com as notas que lhe estão associadas.
- Cada componente herdado deverá ter a ele associado a sua origem (numa segunda versão poderia conter um link para o seu *owner*).