# Processamento de Linguagens
# Curso de Especialização em Informática

Alberto André Leite Gonçalves

Junho de 2007

Trabalho Prático para Avaliação do 1º Módulo

## 1 Contextualização

O objectivo deste trabalho é estudar e dar a conhecer uma ferramenta de software existente para criação e teste de gramáticas para *parsers*, compiladores ou interpretadores.

Para isso foram realizados alguns testes de modo a conhecer melhor a ferramenta e explorar as suas potencialidades. Como caso de estudo, faz parte dos requisitos a implementação do exemplo Lavanda apresentado numa das aulas da disciplina.

## 2 Síntese

A ferramenta a estudar é o UltraGram, uma ferramenta disponibilizada pela UST Solutions.

O software possui um ambiente gráfico de fácil utilização. A estrutura de funcionamento é baseada em projectos, que são conjuntos de ficheiros que podem ser de dois tipos, ficheiros que implementam as gramáticas e ficheiros de teste para essas mesmas gramáticas.

O ambiente gráfico pode ser dividido em quatro áreas distintas. A área principal, que permite visualizar o conteúdo dos ficheiros do projecto, muito semelhante às janelas de edição dos ambientes de programação, incluindo a funcionalidade de destaque da sintaxe (*syntax highlight*).

Existe ainda uma janela para visualizar a estrutura do projecto, permitindo uma visão geral dos ficheiros componentes do projecto. Uma outra para visualizar a árvore de *parsing* criada para cada exemplo de teste. E finalmente a janela de saída (*Output*) que permite verificar as mensagens do software, incluindo os resultados das acções que se forem realizando.

Para a implementação da gramática, é necessário criar um ficheiro com extensão **.grm**, neste ficheiro é utilizada uma sintaxe muito parecida com a

apresentada nas aulas. Existem 3 marcadores de secção principais *pragma*, *tokens* e o *production*. Existe ainda um marcador opcional, o *precedence*.

O primeiro marcador, ***pragma***, define directivas para o parser, especialmente para tratamento de erros e resolução de conflitos.

A secção ***token***, permite definir os símbolos terminais, utilizando normalmente a seguinte estrutura:

```
'expressão' [identificador[ alias]] %ignore;
```

Em que expressão significa exactamente a expressão que permitirá reconhecer o símbolo terminal. O identificador, dá o nome ao símbolo e só é facultativo no caso de se utilizar a entrada %ignore (que indica ao *parser* para ignorar o símbolo terminal encontrado). Finalmente o *alias* que sendo facultativo, permite dar um nome mais curto ao símbolo.

Neste ponto poderá ser criada a secção ***precedence***, que serve basicamente para indicar as regras de precedência. São atribuídas aos operadores as respectivas regras de precedência, no caso de o operador ser associativo ou indicar explicitamente que o operador não será associativo.

Finalmente, a secção ***production***, onde se definem as produções da gramática. Aqui imediatamente a seguir ao marcador deverá ser indicado o nome da gramática que deverá ser igual ao nome da primeira produção. A sua estrutura é bastante simples:

```
nome: componente\_regra1 | componente\_regra2... ;
```

Os componentes da regra podem estar vazios, não sendo necessário indicar qualquer símbolo para o efeito.

Com a gramática completamente definida, é necessário compilar e verificar eventualmente os erros de compilação.

Para testar a gramática, podem ser definidos ficheiros de teste, escritos obviamente com as regras gramaticais definidas. A esses ficheiros de teste, será aplicado o *parser* criado e reportados eventuais erros ocorridos. Uma das funcionalidades a destacar é a possibilidade de realizar o *parsing* passo a passo, observando o comportamento da respectiva árvore.

Existe ainda a possibilidade de gerar automáticamente o autómato (DFA Graph) correspondente à gramática, assim como gerar também o código fonte em quatro linguagens de programação distintas, Java, VB.NET, C++ e C#, código este que poderá ser facilmente incluído noutros projectos de maiores dimensões.

# 3 Caso de Estudo

Como caso de estudo foi utilizado o exemplo Lavanda apresentado numa das aulas. Para além de ser necessário definir os identificadores para os símbo-

los terminais, foi também necessário definir os identificadores para todos os símbolos que deverão ser ignorados pelo parser, por exemplo, os espaços.

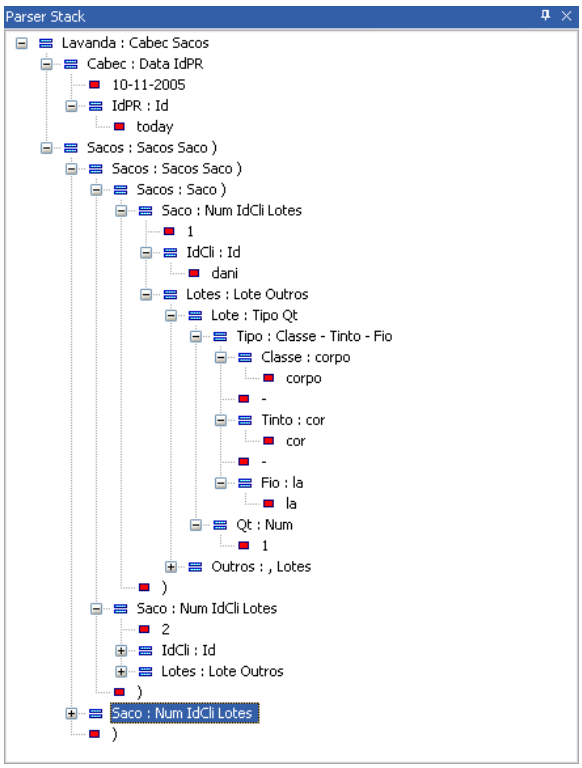
```
// ********************************************************
// *** Gramática LAVANDA
// ********************************************************

%pragma

dkey( off );   // detectar palavras-chave
rtc( off, 4 ); // resolver conflitos de token

%tokens // Declaração de símbolos terminais

'[ \n\t\r]+'  wspace,  %ignore;
'(//)[^\n]*'  cppComment,  %ignore;
'[1-9][0-9]*'  Num;
'([a-z_A-Z])([a-z_A-Z]|[0-9])*'  Id;
'[0-3][0-9]\-[0-1][0-9]\-[0-9][0-9][0-9][0-9]'  Data;
'\('  openPar,  %ignore;
'\)'  closePar,  ')';
'\-'  hiphen  '-';
'\,'  comma  ',';
'corpo'  corpo;
'casa'  casa;
'br'  br;
'cor'  cor;
'alg'  alg;
'la'  la;
'fib'  fib;

%production Lavanda // Produções

Lavanda :  Cabec Sacos;
Cabec : Data IdPR;
Sacos : Saco ')'| Sacos Saco ')';
Saco : Num IdCli Lotes;
Lotes : Lote Outros;
Lote : Tipo Qt;
Tipo : Classe '-' Tinto '-' Fio;
Outros : | ',' Lotes;
IdPR : Id;
IdCli : Id;
Qt : Num;
```

```
Classe : corpo | casa;
Tinto : br | cor;
Fio : alg | la | fib;
```

Para validar o correcto funcionamento do parser, foi utilizado o seguinte exemplo:

```
10-11-2005 today 1 dani (corpo-cor-la 1, casa-cor-alg 2)
                2 pedro (casa-br-fib 4)
                3 celina (corpo-cor-alg 2, corpo-cor-la 3, corpo-cor-fib 1,
                casa-cor-alg 2, casa-cor-la 3, casa-cor-fib 1)
```

Como resultado da aplicação do *parser* ao exemplo, foi possível visualizar no software a pilha construida.




# 4 Conclusão

Este software revelou-se bastante intuitivo e fácil de utilizar. Existem apenas disponíveis as opções mínimas necessárias para a criação e teste de uma gramática.

Uma das funcionalidades que revelou também algum potencial foi teste passo a passo, que permitindo visualizar o comportamento da árvore de

parsing, facilita a compreensão deste tipo de parsers. Uma boa solução para demonstrações aplicadas no ensino da disciplina.

Apesar de não ter sido testado, os criadores garantem que o software gerado, dado o seu nível de optimização, pode ser incluído em projectos de maiores dimensões e com âmbito profissional.