

Processamento de Linguagens

LEI (3ºano) + LCC (2ºano)

Trabalho Prático nº 2
(Yacc)

Ano lectivo 10/11

1 Objectivos e Organização

Este trabalho prático tem como principais **objectivos**:

- (genericamente) aumentar a experiência em *engenharia de linguagens* e em *programação generativa (gramatical)*;
- (especificamente) desenvolver processadores de linguagens segundo o método da *tradução dirigida pela sintaxe*, suportado numa gramática tradutora;
- (especificamente) desenvolver um **compilador** gerando código para uma **máquina de stack virtual**;
- (especificamente) utilizar *geradores de compiladores* baseados em *gramáticas tradutoras*, como o Yacc.

e como **objectivos** secundários:

- 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().

Para o efeito, esta folha contém apenas 1 enunciado.

O programa desenvolvido será apresentado aos membros da equipa docente, totalmente pronto e a funcionar (acompanhado do respectivo relatório de desenvolvimento) e será defendido por todos os elementos do grupo, em Junho, após o fim das aulas, em dia a combinar.

O **relatório** a elaborar, deve ser claro e, além do respectivo enunciado, da descrição do problema, das decisões que lideraram o desenho da linguagem/gramática e as regras de tradução para **Assembly** da VM (incluir as especificações Yacc), deverá conter exemplos de utilização (textos fontes diversos e respectivo resultado produzido). Como é de tradição, o relatório será escrito em \LaTeX .

2 Enunciado

A linguagem LogoLISS é uma simbiose da famosa linguagem de Seymour Papert, Logo—*a linguagem funcional da Tartaruga em movimento*, pensada por este conhecido pedagogo americano para ensino da programação, com uma versão simplificada da linguagem LISS—*Language of Integers, Sequences and Sets*, concebida há anos por Leonor Barroca e Pedro Henriques para ensino da compilação¹

Tendo por base a LISS, LogoLISS permite manusear *escalares* (valores atômicos) bem como *vetores* (*arrays*) do tipo **inteiro**, na forma de **constantes** e de **variáveis**. Como é da praxe neste tipo de linguagens, as variáveis deverão ser declaradas no início do programa e não pode haver re-declarações, nem utilizações sem declaração prévia; se nada for explicitado, o valor da variável após a declaração é indefinido. Sobre inteiros, estão disponíveis as habituais operações aritméticas e lógicas.

Além destas, a linguagem LISS permite ainda ler do *standard input* e escrever no *standard output*. As instruções vulgares para controlo do fluxo de execução—*condicional* e *cíclica*—estão também previstas.

Da Logo, a LogoLISS herda os movimentos da *Tartaruga* num plano de coordenadas cartesianas.

A gramática independente de contexto da dita linguagem LogoLISS é apresentada em Apêndice.

O que se pretende no contexto específico deste Trabalho Prático é implementar a funcionalidade base relativa ao comando da *Tartaruga* e sobre os *escalares* do tipo **inteiro**. A implementação de *arrays* com a operação habitual de *indexação* é opcional, sendo deixada ao critério de cada grupo.

Desenvolva, então, um compilador para LogoLISS, com base na GIC dada e recurso ao Gerador Yacc com auxílio do Flex.

O compilador de LogoLISS deve gerar pseudo-código, Assembly da Máquina Virtual VM cuja documentação completa se encontra em <http://ep1.di.uminho.pt/~gepl/LP/VirtualMachines.html>.

A LogoLISS - A Toy Language

```
/***** Program
```

```
Liss          --> "PROGRAM" identifier "{" Body  "}"
```

```
Body          --> "DECLARATIONS" Declarations "STATEMENTS"  Statements
```

```
/***** Declarations
```

```
Declarations  --> Declaration
```

¹LISS é uma linguagem de programação imperativa, simplificada, que combina conceitos e funcionalidades de várias outras linguagens; embora minimalista, a sua compilação levanta alguns desafios curiosos.

```

| Declarations Declaration

Declaration      --> Variable_Declaration

/***** Declarations: Variables *****/

Variable_Declaration --> Vars "->" Type ";"

Vars            --> Var
                | Vars "," Var

Var             --> identifier Value_Var

Value_Var       -->
                | "=" Inic_Var

Type            --> "INTEGER"
                | "BOOLEAN"
                | "ARRAY" "SIZE" number

Inic_Var        --> Constant
                | Array_Definition

Constant        --> number
                | string
                | "TRUE"
                | "FALSE"

/***** Declarations: Variables: Array_Definition *****/

Array_Definition --> "[" Array_Initialization "]"

Array_Initialization --> Elem
                    | Array_Initialization "," Elem

Elem            --> number

/***** Statements *****/

Statements      --> Statement ";"
                | Statements Statement ";"

Statement       --> Turtle_Commands
                | Assignment
                | Conditional_Statement
                | Iterative_Statement

```

/***** Turtle Statement

```
Turtle_Commands    --> Step
                    | Rotate
                    | Mode
                    | Dialogue
                    | Location

Step                --> "FORWARD" Expression
                    | "BACKWARD" Expression

Rotate              --> "RRIGHT"
                    | "RLEFT"

Mode                --> "PEN" "UP"
                    | "PEN" "DOWN"

Dialogue            --> Say_Statement
                    | Ask_Statement

Location            --> "GOTO" number "," number
                    | "WHERE" "?"
```

/***** Assignment Statement

```
Assignment          --> Variable "=" Expression

Variable            --> identifier Array_Acess

Array_Acess         -->
                    | "[" Single_Expression "]"
```

/***** Expression

```
Expression          --> Single_Expression
                    | Expression Rel_Oper Single_Expression
```

/***** Single_Expression

```
Single_Expression   --> Term
                    | Single_Expression Add_Op Term
```

/***** Term

```
Term                --> Factor
                    | Term Mul_Op Factor
```

/****** Factor

```
Factor          --> Constant
                  | Variable
                  | SuccOrPred
                  | "(" Expression ")"
```

/****** Operators

```
Add_Op         --> "+"
                  | "-"
                  | "||"
```

```
Mul_Op          --> "*"
                  | "/"
                  | "&&"
                  | "**"
```

```
Rel_Op          --> "=="
                  | "!="
                  | "<"
                  | ">"
                  | "<="
                  | ">="
                  | "in"
```

/****** SuccOrPred

```
SuccOrPred      --> SuccPred identifier
```

```
SuccPred        --> "SUCC"
                  | "PRED"
```

/****** IO Statements

```
Say_Statement   --> "SAY" "(" Expression ")"
```

```
Ask_Statement   --> "ASK" "(" string "," Variable ")"
```

/****** Conditional & Iterative Statements

```
Conditional_Statement --> IfThenElse_Stat
```

```
Iterative_Statement  --> While_Stat
```

/****** IfThenElse_Stat

```
IfThenElse_Stat   --> "IF" Expression "THEN" "{" Statements "}" Else_Expression
```

```
Else_Expression    -->
                    | "ELSE" "{" Statements "}"

/***** While_Stat

While_Stat          --> "WHILE" "(" Expression ")" "{" Statements "}"
```