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

Exame de 2ª Época (Recurso)

Data: 25 de Julho de 2006
Hora: 14:00

| Dispõe de 2:30 horas para realizar este exame |
|---|

Determinada linguagem de programação, chamada LIb—*Linguagem de Inteiros Básica*, permite trabalhar com variáveis dos tipos escalares pré-definidos: `inteiro`, `booleano` e `caracter`; para aumentar a legibilidade dos programas, o programador pode dar outros nomes a esses tipos através de uma declaração de tipo apropriada. Um valor de qualquer um desses três tipos vai ocupar o mesmo espaço de memória (1 bytes). Todas as variáveis que sejam usadas nas instruções do programa tem de ser previamente declaradas, associando-se-lhes o respectivo tipo. Essas variáveis serão alocadas, pelo compilador, em memória sequencialmente, a partir do endereço 0.
A nivel de instruções estarão disponiveis: a `atribuição` (o lado direito está restrito a uma constante ou variável), `leitura` e `escrita` (de inteiros ou caracteres); e instruções de `controlo condicional` ou `ciclico`.
A gramática independente de contexto $G$, abaixo apresentada, define a linguagem LIb. O Símbolo Inicial é `LIb`, os Símbolos Terminais são escritos em minúsculas (pseudo-terminais), ou em maiúscula (palavras-reservadas), ou entre apostrofes (sinais de pontuação) e a string nula é denotada por `&`; os restantes serão os Símbolos Não-Terminais.

```
p0:  LIb      -->  Decls Insts
p1:  Decls    -->  Tipos  Vars
p2:  Tipos    -->  TIPOS  Ts
p3:  Ts       -->  DclTipo
p4:             |  Ts  DclTipo
p5:  DclTipo  -->  IdT  '=' Tipo  ';'
p6:  Tipo     -->  INT
p7:             |  BOOL
p8:             |  CHAR
p9:  Vars     -->  VARIAVEIS  Vs
p10: Vs       -->  Ids  ':'  IdT  ';'
p11:            |  Vs   Ids  ':'  IdT ';'
p12: Ids      -->  IdV  RIds
p13: RIds     -->  &
p14:            |  ','  Ids
p15: IdT      -->  id
p16: IdV      -->  id

p17: Insts    -->  '{'  LstI  '}'
p18: LstI     -->  &
p19:            |  Inst ';' LstI
p20: Inst     -->  LeitEsc
p21:            |  Ctrl
p22:            |  Atrib
```

```
p23: Atrib    -->  IdVar  '='  Factor
p24: LeitEsc  -->  LER  IdVar
p25:            |  ESCREVER IdVar
p26: Ctrl     -->  Cond
p27:            |  Ciclo
p28: Cond     -->  SE '(' IdVar ')'  ENTAO  Insts  SENAO Insts
p29: Ciclo    -->  REPETIR Insts ATE '(' IdVar ')'
p30: Factor   -->  IdVar
p31:            |  TRUE
p32:            |  FALSE
p33:            |  num
p34:            |  caract
```

As questões deste exame serão colocadas neste contexto.

## Tradução Dirigida pela Semântica

### Questão 1: gramática de atributos

Escreva uma **Gramática de Atributos**, $GA$, para construir e manter a **Tabela de Identificadores** com todos os identificadores de Tipo ou Variável do programa (incluindo os pré-definidos). Pretende-se: que as variáveis sejam alocadas em memória (a partir do endereço `0` como se disse acima), registando na dita Tabela o seu endereço; e que se guarde ainda o número de vezes que cada uma é atribuida (leituras são consideradas como atribuições) ou usada (em atribuições, condições, ou escritas).

A sua $GA$, além das *regras para o cálculo dos atributos necessários*, deve incluir *condições de contexto* para garantir que: (a) não haja re-declaração de identificadores (note que os identificadores das variáveis também não podem ser iguais aos identificadores dos tipos); (b) o tipo de todas as variáveis declaradas exista.
Para facilitar a leitura da sua resposta, reúna numa tabela (no início ou no fim) os **atributos herdados e sintetizados, ou intrínsecos** de cada símbolo (NT ou T) de $GA$.

## Tradução Dirigida pela Sintaxe

### Questão 2: parsing

Recordando os seus conhecimentos sobre análise sintáctica *Top-Down* e *Bottom-Up*, responda às alíneas seguintes:

**a)** Mostre que $G$ **não é LL(1)**, indicando todas as situações de conflito;
Diga como a poderia transformar numa gramática LL(1), mostrando como resolvia um desses conflitos.

**b)** Escreva as funções, pertencentes a um parser Recursivo-Descende Puro, para reconhecer os símbolos Não-Terminais `Ids` e `RIds`.

**c)** Supondo que apenas está a tratar a sub-linguagem de declarações (cujo axioma, ou símbolo inicial, seria `Decls`) e que vai analisar a frase

```
TIPOS idade = int; VARIAVEIS myage, yourage : idade;
```

com um parser *Bottom-Up LR(0)*, desenhe a respectiva *Árvore de Parsing* indicando a ordem de redução dos nodos.

**d)** Observando $G$, é evidente que a GIC dada **não é LR(0)**!
Diga o que nos permite tirar de imediato esta conclusão, explicando a sua resposta.

**e)** Construa completamente **apenas** o `estado 0` do autómato determinista LR(0) e os `estados` que dele se atingem (em relação a estes faça o `fecho` e indique as `transições`, mas não continue a construção a partir dos novos estado).

**f)** Diga qual a diferença entre a *stack de parsing* de um *parser LR* e a *stack valor* do respectivo *tradutor dirigido pela sintaxe*.

**Questão 3: gramática tradutora**

Transforme $G$ numa **gramática tradutora**, $GT$, reconhecível pelo yacc, para gerar código *Assembly* da Máquina de Stack Virtual MSP (instruções listadas em anexo) apenas para as Instruções de Atribuição e Controlo.
Explique qual a ideia de gerar código para uma *Máquina Virtual*, indicando se se trata de uma abordagem meramente académica ou se é algo que tem viabilidade prática; esclareça, ainda, quais as vantagens/desvantagens de conceber a máquina virtual como uma *Máquina de Stack*.

## Validação da Componente Prática

**Questão 4: sobre o Trabalhos Práticos**

Recorde o seu 1º Trabalho Prático e comece por indicar qual foi o problema que o seu grupo escolheu, justificando essa opção.
Na resolução desse TP usou o par de Geradores Flex/Yacc com que finalidade? explique qual a entrada fornecida a cada uma dessas ferramentas, o que foi produzido por cada uma e como utilizou essas partes para obter o processador de linguagens desejado.
Diga que tipos estruturados o seu grupo escolheu e implementou, referindo quais as maiores dificuldades enfrentadas na análise semântica e na geração de código.
Termine comentando se poderia ter reslvido esse mesmo problema recorrendo ao Gerador LISA; se sim, quais seriam as vantagens.

# Instruções Assembly da Máquina de Stack Virtual MSP

### Grupo 1: Manuseamento de Valores e Endereços

```
PUSH  valor  ;coloca o valor inteiro no Topo(Stack)
PSHA  ender  ;coloca o endereço no Topo(Stack)
LOAD         ;carrega para o Topo(Stack) o valor que está no endereço contido no Topo(Stack)
LDA          ;carrega um endereço no Topo(Stack)
STORE        ;armazena na memoria o valor que está no Topo(Stack), no endereço contido no Topo(Stack)-1
STRA         ;armazena na memória o endereço do Topo(Stack), no endereço contido no Topo(Stack)-1
IN           ;lê um inteiro para o Topo(Stack)
INC          ;lê um caracter para o Topo(Stack)
OUT          ;escreve um inteiro que está no Topo(Stack)
OUTC         ;escreve um caracter que está no Topo(Stack)
```

### Grupo 2: Operações Aritméticas e Lógicas

```
ADD          ;adiciona dois inteiros
SUB
MUL
DIV
ADDA         ;adiciona um inteiro a um endereço
AND          ;E lógico
OR           ;OU lógico
NOT          ;NEGAÇÃO lógica
EQ           ;igual
NE           ;diferente
LT
LE
GT
GE
```

### Grupo 3: Instruções de Controlo

```
JUMP  label  ;salta para a instrução de programa etiquetada pela ´label´
JMPF  label  ;salta para a ´label´, se Topo(Stack) for Falso
CALL  label  ;salta para a subrotina cujo código começa na 'label'
RET          ;retorna de uma subrotina
NOOP         ;não faz nada
HALT         ;termina a execução
```