Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares

Trabalho Prático I

1. Objetivo

O objetivo desse trabalho é desenvolver um interpretador para uma linguagem de programação fictícia chamada *MiniLab*. Essa linguagem é capaz de executar operações sobre matrizes de forma mais fácil e prática.

2. Contextualização

Um exemplo de aplicação do uso de matrizes é o cálculo do determinante. A seguir é dado um programa para esse propósito.

```
# read the random matrix
dim = input("Entre com a dimensao: ");
matrix = [].rand(dim, dim);

# show the matrix
show(matrix);

# calculate the first part of the determinant
det1 = 0;
for c = [].seq(0, dim-1)
  j = c;
  tmp = 1;
  for i = [].seq(0, dim-1)
    tmp = tmp * matriz.value(i, j);
    j = (j + 1) % dim;
  end
  det1 = det1 + tmp;
end

# calculate the second part of the determinant
det2 = 0;
for c = [].iseq(dim-1, 0)
  j = c;
  tmp = 1;
  for i = [].seq(0, dim-1)
    tmp = tmp * matriz.value(i, j);
    j = (j - 1) % dim;
    if j < 0
      j = j + dim;
    end
  end
  det2 = det2 + tmp;
end

# print the determinant
show(" = " (det1 - det2));
```

det.lab

Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares

Um programa em *MiniLab* possui uma memória indexada através de nomes de variáveis que armazenam inteiros ou matrizes. Todas as variáveis do sistema possuem visibilidade global. Operações aritméticas só podem ser feitas em inteiros. A linguagem possui comentários de uma linha onde são ignorados qualquer sequência de caracteres após o símbolo # (hashtag). A linguagem possui as seguintes características:

- **Declarações:**
  - `input`: ler um inteiro do usuário pelo teclado.
  - `show`: imprimir uma sequência de texto ou matriz.
  - `assign`: guardar um inteiro ou matriz em uma variável.
  - `if`: executar comandos baseado em expressões condicionais.
  - `while`: repetir comandos enquanto a expressão condicional for verdadeira.
  - `for`: percorrer os elementos de uma matriz.
- **Valores:**
  String: uma sequência de caracteres entre aspas duplas.
  Variável: começa com letra seguido de letras e dígitos.
  Inteiro: constante inteiras formadas por dígitos.
  Lógico: operações de comparações que obtém um valor lógico.
  Matriz: uma matriz de inteiros com dimensões.
- **Operadores:**
  - **Inteiro:** `+` (adição), `-` (subtração), `*` (multiplicação), `/` (divisão), `%` (resto inteiro)
  - **Lógico:** == (igual), != (diferença), < (menor), > (maior), <= (menor igual), >= (maior igual)
  - **Conectores**: & (E lógico) | (OU lógico)
- **Funções de matrizes**:
  - **Geração:**
    - `null(i, j)`: cria uma matriz `i x j` preenchida com zeros .
    - `fill(i, j, v)`: cria uma matriz `i x j` preenchida com `v`.
    - `rand(i, j)`: cria uma matriz `i x j` com valores aleatórios entre 0 e 100.
    - `id(i, j)`: cria uma matriz identidade `i x j` (diagonal com valor 1, outras com 0).
    - `seq(x, y)`: cria uma vetor com os elementos de `x` a `y` incrementados por uma unidade. Ex.: seq(2, 5) = [2, 3, 4, 5].
    - `iseq(x, y)`: cria uma vetor com os elementos de `x` a `y` decrementados por uma unidade. Ex.: seq(8, 5) = [8, 7, 6, 5].
  - **Operações:**
    - `opposed()`: obtém uma (nova) matriz oposta (inverte o sinal dos elementos).
    - `transposed()`: obtém uma (nova) matriz transposta (inverte linhas com colunas e vice-versa).

- `sum(x, y)`: soma a matriz `x` com matriz `y` e retorna uma nova matriz resultante. As matrizes devem ter a mesma dimensão.
- `mul(x, y)`: multiplica a matriz `x` com matriz `y` e retorna uma nova matriz resultante. A linha da matriz `x` tem que casar com a coluna da matriz `y` e a coluna da matriz `x` com a linha da matriz `y`.
- `mul(x, v)`: multiplica todos os elementos da matriz `x` pelo escalar `v` e retorna uma nova matriz resultante.

- Informação:
  - `size()`: obtém o tamanho da matriz (linhas x colunas).
  - `rows()`: obtém o número de linhas.
  - `cols()`: obtém o número de colunas.
  - `value(i, j)`: obtém o valor na posição `(i, j)` da matriz.

3. Gramática

A gramática da linguagem *MiniLab* é dada a seguir no formato de Backus-Naur estendida (EBNF):

```
<statements> ::= <statement> { < statement > }
<statement>  ::= <input> | <show> | <assign> | <if> | <while> | <for>

<input>      ::= input '(' <text> ')' ';'
<show>       ::= show '(' (<text> | <matrix>) ')' ';'
<assign>     ::= <var> = (<matrixexpr> | <intexpr>) ';'
<if>         ::= if <boolexpr> <commands> [ else <commands> ] end
<while>      ::= while <boolexpr> <commands> end
<for>        ::= for <matrixexpr> <commands> end

<text>       ::= { <string> | <intexpr> }

<boolexpr>   ::= <intexpr> <boolop> <intexpr> { ('&' | '|') <boolexpr> }
<boolop>     ::= '==' | '!=' | '<' | '>' | '<=' | '>='
<intexpr>    ::= <term> [ ('+' | '-') <term> ]
<term>       ::= <factor> [ ('*' | '/' | '%') <factor> ]
<factor>     ::= <number> | <intvalue> | '(' <intexpr> ')'

<matrixexpr> ::= (<var> | <gen>) { '.' (<opposed> | <transposed> | <sum> | <mul>) }
<opposed>    ::= opposed '(' ')'
<transposed> ::= transposed '(' ')'
<sum>        ::= sum '(' <matrixexpr> ')'
<mul>        ::= mul '(' (<matrixexpr> | <intexpr>) ')'
```

```
<gen>        ::= '[' ']' '.' (<null> | <fill> | <rand> | <id> | <seq> | <iseq>)
<null>       ::= null '(' <intexpr> ',' <intexpr> ')'
<fill>       ::= fill '(' <intexpr> ',' <intexpr> ',' <intexpr> ')'
<rand>       ::= rand '(' <intexpr> ',' <intexpr> ')'
<id>         ::= id '(' <intexpr> ',' <intexpr> ')'
<seq>        ::= seq '(' <intexpr> ',' <intexpr> ')'
<iseq>       ::= iseq '(' <intexpr> ',' <intexpr> ')'

<intvalue>   ::= <matrixexpr> [ . (<size> | <rows> | <cols> | <value>) ]
<size>       ::= size '(' ')'
<rows>       ::= rows '(' ')'
<cols>       ::= cols '(' ')'
<value>      ::= value '(' <intexpr> ',' <intexpr> ')'
```

4. Instruções

Deve ser desenvolvido um interpretador em linha de comando que recebe um programa-fonte na linguagem *MiniLab* como argumento e executa os comandos especificados pelo programa. Por exemplo, para o programa *det.lab* deve-se produzir uma saída semelhante a:

```
$ mlab
Usage: ./mlab [MiniLab File]
$ mlab det.lab
Entre com a dimensão: 3
[2, 3, 1]
[1, 2, 3]
[1, 0, 1]
= 8
```

O programa deverá abortar sua execução, em caso de qualquer erro léxico, sintático ou semântico, indicando uma mensagem de erro. As mensagens são padronizadas indicando o número da linha (2 dígitos) onde ocorreram:

Tipo de Erro	**Mensagem**
Léxico	Lexema inválido [*lexema*]
	Fim de arquivo inesperado
Sintático	Lexema não esperado [*lexema*]
	Fim de arquivo inesperado
Semântico	Operação inválida
	Tipos inválidos

Exemplo de mensagem de erro:

```
$ msi erro.msh
```

Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares
`03: Lexema não esperado [;]`

5. Avaliação

O trabalho deve ser feito em grupo de até dois alunos, sendo esse limite superior estrito. O trabalho será avaliado em 20 pontos, onde essa nota será multiplicada por um fator entre 0.0 e 1.0 para compor a nota de cada aluno individualmente. Esse fator poderá estar condicionado a apresentações presenciais a critério do professor.

Trabalhos copiados, parcialmente ou integralmente, serão avaliados com nota **ZERO**, sem direito a contestação. Você é responsável pela segurança de seu código, não podendo alegar que outro grupo o utilizou sem o seu consentimento.

6. Submissão

O trabalho deverá ser submetido até as 23:55 do dia 05/09/2016 (segunda-feira) via sistema acadêmico (Moodle) em pasta específica. Não serão aceitos, em hipótese alguma, trabalhos enviados por e-mail ou por quaisquer outras fontes.