INSTITUTO DE INFORMÁTICA UFG

# CIÊNCIAS DA COMPUTAÇÃO
# LINGUAGENS DE PROGRAMAÇÃO

## 2016 / 2

UFG

Prof. Bruno Silvestre

## PARTE I – DESCRIÇÃO DO TRABALHO

O trabalho consiste em construir um programa em **Lua** que recebe como parâmetro o nome de um arquivo contendo um programa em DOOL (vide PARTE II) e deve traduzir esse programa para C. Esse programa em C deve manter a mesma semântica que o programa DOOL original, ou seja, quando compilado e executado, o programa C deve produzir o mesmo resultado esperado do programa DOOL.

Exemplo esperado de uso do tradutor (comandos no terminal do Linux):

```
lua tradutor.lua prog.dool prog.c
gcc -o prog prog.c
./prog
```

Observações:

- Assuma que todos os programas DOOL estão corretos.
  - Não é necessário fazer verificação de sintaxe, tipo, etc.
- Ver o livro do Sebesta (seção 12.11 na 10ª. edição) sobre a implementação de orientação a objeto.
- Ver o texto sobre ponteiro para função em C, disponível no Moodle.

### I.1. REGRAS DO TRABALHO

- Trabalho deve ser feito em grupo de 2 alunos.
- Trabalho só será considerado válido após o grupo realizar a apresentação ao professor.
  - O integrante do grupo que não comparecer à apresentação terá nota zero.
- O grupo deve utilizar o Github para desenvolvimento e entrega do trabalho.
  - O professor irá criar e compartilhar com cada grupo um repositório.
  - *Commits* fora do prazo serão ignorados.
- O grupo deve produzir um relatório descrevendo como os principais pontos do trabalho foram realizados, ou seja, abordagens utilizadas para atingir os objetivos do trabalho.
  - Ex: comentários, qualidade e organização do código, etc.
  - O documento deve ser em formato PDF e o arquivo deve estar no repositório.
  - Dica: criar os diretórios “src” e “doc” para separar o código da documentação.
- Não será aceito o uso de módulos extras de Lua.
  - Todo o programa deve ser feito utilizando apenas os módulos:
    - io, os, math, string, coroutine e table.
- Boas práticas de programação também serão consideradas na avaliação do trabalho:
  - Ex: comentários, qualidade e organização do código, etc.
- Plágio (“cola”) total ou parcial implica em nota zero para todos os envolvidos.

# PARTE II – DEFINIÇÃO DA LINGUAGEM

A seguir daremos uma definição da estrutura da linguagem DOOL (*Defective Object Oriented Language*)

## II.1. BNF

```
DOOL → <PROGRAM>
     | <CLASSES_DEF> <PROGRAM>

CLASSES_DEF → <CLASS_DEF>
            | <CLASS_DEF> <CLASSES_DEF>

CLASS_DEF → class <ID> as <eol> <CLASS_BODY> end <eol>
          | class <ID> extends <ID> as <eol> <CLASS_BODY> end <eol>

CLASS_BODY → <ATTRIBUTE> <CLASS_BODY>
           | <METHOD>    <CLASS_BODY>
           |  ε

ATTRIBUTE → attribute <ID> : number <eol>

METHOD → def <METH_TYPE> <ID> as <eol> <METH_PARAMS> begin <eol> <METH_BODY> return <ID> <eol> end <eol>

METH_TYPE → static
          | dynamic

METH_PARAMS → <ID> : <TYPE> <eol>
            | <ID> : <TYPE> <eol> <ID> : <TYPE> <eol>
            | <ID> : <TYPE> <eol> <ID> : <TYPE> <eol> <ID> : <TYPE> <eol>
            |  ε

METH_BODY → <VAR_DEF> <eol> <METH_BODY>
          | <EXPRESSION> <eol> <METH_BODY>
          |  ε

VAR_DEF → var <ID> : <TYPE>

EXPRESSION → <ATTRIBUTION>
           | <CALL>

ATTRIBUTION → <ID> = new <ID>
            | <ID> = <CALL>
            | <ID> = <ID> + <ID>
            | <ID> = <ID> - <ID>
            | <ID> = <NUMERO>
            | <ID>.<ID> = <ID>
            | <ID> = <ID>.<ID>

CALL → <ID>.<ID> ( <CALL_PARAMS> )

CALL_PARAMS → <ID>
            | <ID> , <ID>
            | <ID> , <ID> , <ID>
            | ε

PPROGRAM → program <eol> <PROGRAM_BODY> end

PROGRAM_BODY → <VAR_DEF> <eol> <PROGRAM_BODY>
             | <EXPRESSION> <eol> <PROGRAM_BODY>
             | ε
```

`eol` → *caractere de fim de linha, geralmente o* '\n'
`NUMERO` → *literal que representa um número inteiro 32 bits com sinal, por exemplo: 39, 536, -362, 46573*
`ID` → *identificador (sequência de letras – string) para nomear classes e variáveis*
`TYPE` → `number` *ou uma das classes definidas pelo usuário*

## II.2. DESCRIÇÃO INFORMAL DA LINGUAGEM DOOL

### II.2.1. Formato da Linguagem

Pelo BNF, percebemos que um programa em DOOL pode conter ou não definições de classes, mas deve ter obrigatoriamente uma seção *program … end* que seria como a função *main* de C, ou seja, é a primeira parte do programa a ser executada.

A linguagem utiliza o caractere de fim de linha (eol) como parte do separador de fim de sentenças. Isso vai auxiliar no *parser* da linguagem. No BNF, símbolo ε indica que aquela variável sintática pode ser vazia. Por exemplo, <METH_PARAMS> pode ser um conjunto de parâmetros ou pode simplesmente não existir, o que indica que um método pode ter ou não parâmetros.

### II.2.2. Classe

Uma classe pode estender outra, em caso de herança, ou ser uma definição independente (sem herança). A classe pode ter um conjunto de atributos, que no momento só podem ser to tipo *number*.

Vide mais sobre herança na seção I.3.

### II.2.3. Método

Os métodos devem ser considerados como funções que sempre retornam um valor do tipo *number*. Note que atualmente a sentença *return* só permite retornar o valor de uma variável.

Cada definição de método pode ser com vinculação estática ou dinâmica.

Métodos podem ter zero ou até três parâmetros de entrada, sendo que os tipos dos parâmetros podem ser *number* ou uma classe declarada pelo usuário. A passagem de objeto como parâmetro é por referência.

O corpo de um método pode ter definições de variáveis ou expressões. Todo método possui uma variável *this*, criada implicitamente, que faz referência ao objeto cujo método foi chamado. No exemplo da seção 2, podemos ver que o método soma utiliza *this.count*.

### II.2.4. Atributos

Todo acesso aos atributos de um objeto deve ser feito através de uma referência ao objeto, ou seja, através de uma variável. Assim, dentro de um método, deve-se sempre usar *this* para acessar os atributos do próprio objeto.

### II.2.5. Variáveis de Método

As variáveis de um método devem ser declaradas antes que possam ser usadas. O tipo da variável pode ser *number* ou uma classe definida pelo usuário. As variáveis são sempre referências para objetos.

### II.2.6. Expressões

As expressões podem ser atribuições ou chamadas de métodos.

As atribuições podem ser:

- Atribuição de variável: uma variável recebe o valor de outra variável, de um atributo de um objeto ou um número inteiro.
- Atribuição de atributo: um atributo de um objeto pode receber o valor de uma variável.
- Criação de um objeto: um objeto é criado com *new* e a variável recebe a referência desse objeto.
- Operação: uma variável recebe o resultado de uma soma ou subtração. Note que os operandos da operação só podem ser variáveis.
- Retorno de método: uma variável recebe o retorno de uma chamada de método.

### II.2.7. Chamada de Método

A chamada de método pode ser utilizada em uma atribuição ou isolada em uma linha. Um método pode ter zero ou até três parâmetros de entrada, sendo que todos os parâmetros só podem ser variáveis.

### II.2.8. Corpo de *program*

A seção *program* de um programa em DOOL pode conter definição de variáveis e expressões, de forma semelhante ao corpo dos métodos.

## II.3. HERANÇA

### II.3.1. Herança de Atributos

Quando uma classe herda de outra, ela automaticamente vai herdar todos os atributos da superclasse.

Se uma classe for definir atributos, o nome desses atributos deve ser diferente de qualquer outro atributo herdado das superclasses. Por exemplo:

```
Class A as
    attribute count : number
end
```

```
Class B extends A as
   attribute value : number
end
```

A declaração de classe abaixo não é válida pois B, que estende de A, define um novo atributo com o mesmo nome de um atributo já existente em A.

```
Class A as
    attribute x : number
end
```

```
Class B extends A as
   attribute x : number
end
```

Neste novo exemplo, a declaração de classe também não é válida pois C define um atributo com o mesmo nome de um atributo de A.

```
class A as
    attribute value : number
end
```

```
class B extends A as
   attribute count : number
end
```

```
class C extends B as
   attribute value : number
end
```

### II.3.2. Herança de Métodos

Só existe herança de métodos com vinculação dinâmica. Por exemplo:

```
class A as
   def dynamic show as
   begin
     var x : number
     x = 10
     io.print(x)
     return x
   end
end
```

```
class B extends A as

end
```

```
program
  var b : B
  b = new B
  b.show()
end
```

Outro exemplo de herança com métodos dinâmicos. Neste caso, o método *show*() da classe B será executado, pois B sobrescreve o método de A:

```
Class A as
   def dynamic show as
   begin
     var x : number
     x = 10
     io.print(x)
     return x
   end
end
```

```
Class B extends A as
   def dynamic show as
   begin
     var y : number
     y = -21
     io.print(y)
     return y
   end
end
```

```
Class C extends B as

end
```

```
program
  var c : C
  c = new C
  c.show()
end
```

Os métodos com vinculação estática pertencem apenas às classes que foram definidas, e não dão suporte à herança.

O exemplo abaixo não é válido pois B não possui o método *show*():

```
Class A as
   def static show as
   begin
     var x : number
     x = 10
     io.print(x)
     return x
   end
end
```

```
Class B extends A as

end
```

```
program
  var b : B
  b = new B
  b.show()
end
```

No entanto, o exemplo abaixo é válido, pois o método está sendo chamado sobre a variável do tipo A:

```
Class A as
   def static show as
   begin
     var x : number
     x = 10
     io.print(x)
     return x
   end
end
```

```
Class B extends A as

end
```

```
Program
  var a : A
  var b : B
  b = new B
  a = b
  a.show()
end
```

O próximo exemplo também é válido pois tanto A como B definem o método *show*(), sendo que a linha "b.show()" invoca o método definido em B, e a linha "a.show()" invoca o método definido em A:

```
Class A as
   def static show as
   begin
     var x : number
     x = 10
     io.print(x)
     return x
   end
end
```

```
Class B extends A as
   def static show as
   begin
     var y : number
     y = -21
     io.print(y)
     return y
   end
end
```

```
Program
  var a : A
  var b : B
  b = new B
  b.show()
  a = b
  a.show()
end
```

Em resumo, só é permitido chamar um método estático se a variável for do mesmo tipo da classe em que o método foi definido.

### II.4. Biblioteca Padrão

A linguagem DOOL só possui uma função padrão *print()* que pertence ao objeto especial *io*. Essa função imprime o valor da variável na tela seguido do caractere de nova linha.

# PARTE III – EXEMPLOS DE PROGRAMAS EM DOOL

## Exemplo 1

```
program
  var x:number
  var p:number
  var q:number

  x = 10
  io.print(x)

  p = 10
  q = 30

  io.print(p)
  io.print(q)
end
```

## Exemplo 2

```
class Foo as
  attribute max : number

  def static show as
    count : number
  begin
    io.print(count)
    return count
  end

  attribute min : number

end

program
  var x : number
  var foo : Foo
  x = 20
  foo = new Foo
  foo.show(x)
end
```

## Exemplo 3

```
class A as

  attribute value : number

  def static soma as
    x : number
    b : B
  begin
    var r : number
    var t : number

    t = b.min
    r = this.count
    r = t + x

    return r
  end

  attribute count : number

end

class B extends A as

  attribute max : number

  def dynamic calc as
    x : number
    y : number
  begin
    var v : number
    v = 2 * x
    return v
  end

  attribute min : number

  def static show as
  begin
    var v : number
```

```
    v = this.max
    io.print(v)
    return v
  end
end

program
  var p : number
  var q : number

  p = 10
end
```

## Exemplo 4

```
class A as

  attribute value : number
  attribute count : number

  def static soma as
    x : number
    b : B
  begin
    var r : number
    var t : number

    t = b.min
    r = this.count
    r = t + x

    return r
  end

end

class B extends A as

  def dynamic calc as
    x : number
    y : number
  begin
    var v : number
    v = 2 * x
    return v
  end

  attribute max : number
  attribute min : number

end

program
  var p : number
  var q : number

  p = 10
  q = 30

  io.print(p)

  var b: B
  var a: A

  b = new B
  a = b
  b = a

  p = b.calc(p, q)
  io.print(x)

  q = a.soma(p, b)
  io.print(q)
end
```