INSTITUTO DE INFORMÁTICA UFG

LINGUAGENS E PARADIGMAS DE PROGRAMAÇÃO
TRABALHO EM GRUPO
2019/2

UFG

O trabalho se baseia na implementação de um interpretador para uma linguagem (bpl – *bruno's programming language*). O interpretador deve ser construído utilizando a linguagem Lua e só é permitido o uso dos módulos que são oferecidos por padrão (string, math, io, os).

1. BNF da Linguagem

```
<program> → <funcs>

  <funcs> → <func>
          | <func> <funcs>

   <func> → <header> <body>
          | <header> <vardefs> <body>

 <header> → function <name>() '\n'
          | function <name>( <params> ) '\n'

 <params> → <name>
          | <name>, <params>

<vardefs> → <vardef>
          | <vardef> <vardefs>

 <vardef> → var <name> '\n'
          | var <name>[ <size> ] '\n'

   <body> → begin '\n' <cmds> end '\n'

   <cmds> → <cmd>
          | <cmd> <cmds>

    <cmd> → <attr>
          | <funcall>
          | <if>

   <attr> → <var> = <arg> '\n'
          | <var> = <arg> <op> <arg> '\n'

    <var> → <name>
          | <name>[ <number> ]

    <arg> → <value>
          | <funcall>

<funcall> → <name>() '\n'
          | <name>(<values>) '\n'

<values> → <value>
         | <value>, <values>

 <value> → <name>
         | <name>[ <number> ]
         | <number>

    <op> → + | - | * | /

    <if> → if <value> <cmp> <value> then '\n' <attr> fi '\n'
         | if <value> <cmp> <value> then '\n' <attr> else '\n' <attr> fi '\n'

   <cmp> → < | <= | > | >= | == | !=

  <name> → sequência de letras minúsculas
<number> → números inteiros
  <size> → números maiores que zero (positivos)
```

2. Considerações sobre a sintaxe da linguagem

- Considere que todos os programas estão sintaticamente corretos.
- As palavras reservadas são: “function”, “var”, “if”, “then”, “else”, “fi”, “begin”, “end”.
- Espaços no início devem ser desconsiderados.
- Espaços antes do ‘\n’ devem ser desconsiderados.
- Nas posições que o espaço é permitido, podem ocorrer mais de um espaço.
- Não haverá espaço entre o nome da função e o parêntese, ou seja, não ocorrerá os casos abaixo, por exemplo:
  - f ()
  - f (a,b,d)
- Não haverá espaço dentro dos parâmetros da função, ou seja, não irá ocorrer os casos abaixo, por exemplo:
  - f( )
  - f(a, b, c)
  - f(a,b,c)
- Todas as chamadas de função passam a quantidade correta de parâmetros.
- Não haverá espaço entre o nome do vetor e o colchete, nem dentro dos colchetes, ou seja, não ocorrerá os casos abaixo, por exemplo:
  - “v [1]” ou “v[1]” ou “v[1]”
- Sempre haverá pelo menos um espaço entre os elementos de uma atribuição, ou seja, entre a variável e o ‘=’, entre o ‘=’ e o primeiro argumento, entre o primeiro argumento e a operação (se houver), entre a operação e o segundo argumento (se houver). Então, são exemplos válidos:
  - a = b + -1
  - b = f(8,vx[1])
  - vx[0] = voutro[-1] / zerar()
- Sempre haverá pelo menos um espaço entre os elementos da expressão de teste do “if”, ou seja, entre o “if” e o primeiro argumento, entre o primeiro argumento e o operador de comparação, entre o operador de comparação e o segundo argumento, entre o segundo argumento e o “then”. Então, são exemplos válidos:
  - if a > b then
  - if a[1] <= -3 then

3. Considerações sobre a semântica da linguagem

- Todas as funções terão no máximo 3 parâmetros.
- As variáveis podem ser escalares ou arrays.
- As variáveis e parâmetros só armazenam números inteiros.
- As operações são sobre inteiros. Destaque para divisão inteira, onde os valores são trucados (e não arredondados).
- Dentro de uma função, não é permitido definir uma variável com o mesmo nome de um parâmetro.
- Toda variável escalar definida possui valor inicial como 0 (zero). No caso de arrays, todas as posições são inicializadas com 0 (zero).
- Toda função possui uma variável implícita “ret” (iniciada com zero). O valor final desta variável será o retorno da função. Não é permitido definir outra variável ou parâmetro com esse nome.

- Não há garantias de que os índices usados nos arrays estão dentro da faixa válida. O interpretador deve verificar e, em caso de erro, parar a execução do programa com uma mensagem de erro.
- Existe uma função *built-in* chamada "*print*()" que recebe apenas um parâmetro e mostra o valor na tela, com um "\n" no final. O interpretador deve fornecer essa função. Não é permitido criar uma outra função com esse nome.
- A função "*main*()" – sem parâmetro – é o início do programa. É garantido que todo programa terá uma função *main*.

4. Regra de escopo da linguagem

- A linguagem utiliza escopo dinâmico.
- Apenas as variáveis entram na regra de escopo, ou seja, não é possível acessar parâmetro de outras funções via nomes não-locais.

5. Execução do interpretador

O seu interpretador receberá um arquivo como parâmetro contendo um programa na linguagem definida e deverá executá-lo. Por exemplo:

```
$ lua interpretador.lua  prog.bpl
```

6. Regra do Trabalho

- Entrega do trabalho: 08/nov, 23:55 (via Moodle)
  - Um arquivo ZIP contendo:
    - Relatório em PDF
    - Código fonte Lua
- Caso tenha dúvida sobre alguma parte, consulte o professor.
  - Não vá inventar da sua cabeça.
  - Pode ser que a especificação esteja incompleta ou inconsistente.
- Trabalho deve feito em grupo de 2 alunos.
- Plágio significa nota 0 (zero) para todos os envolvidos.
- A nota irá considerar boas práticas de programação: qualidade, organização do código, comentários (relevantes), etc.
- Um relatório deve ser entregue junto com o código fonte, descrevendo o que cada integrante do grupo fez.
  - Se o relatório estiver faltando o trabalho será desconsiderado.
- O trabalho, depois de entregue, deve ser apresentado ao professor, em horário que será agendado.
  - Não apresentar o trabalho significa nota 0 (zero) para o grupo.
- Se ficar evidente que um membro do grupo desenvolveu a grande maioria do trabalho, a nota do grupo será rebaixada ou até zerada.
  - Deve-se notificar o professor com antecedência se um dos membros não estiver contribuindo para que isso não ocorra.

7. Exemplos

Abaixo são colocados comentários para ajudar a entender os programas.

Esses comentários **não** existem na linguagem, são apenas para ajudar a entender os exemplos abaixo:

```
function main()
begin
   print(12)      // saída "12"
end
```

```
function main()
   var x
begin
    x = 10
    print(x)          // saída "10"
end
```

```
function foo()
begin
  print(x)         // saída "5"
  ret = 3
end

function main()
  var x
begin
  x = 5
  x = foo()
  print(x)          // saída "3"
end
```

```
function fat(i)
  var tmp
begin
   tmp = i - 1
   if i == 1 then
       ret = 1
   else
       ret = i * fat(tmp)
   fi
end

function main()
   var f
begin
   f = fat(3)
   print(f)            // saída "6"
end
```

```
function main()
   var a[2]
   var varnum
begin
   varnum = 10
   a[0] = -1 - -2
   a[1] = varnum * a[0]
   faznada()
end

function show()
begin
    print(a[-2])      // saída "1"
    print(a[-1])      // saída "10"
    print(a[2])       // ERRO!
end

function faznada()
begin
   show()
end
```

```
function foo()
begin
  print(x)          // saída "5"
  x = 20
end

function bar(x)
begin
    x = 10 * x  // 'x' é parâmetro,
    foo()       // não entra no
                // escopo.
    print(x)    // saída 50
end

function main()
   var x
begin
    x = 5
    bar(x)
    print(x)    // saída "20"
end
```

```
function zero(a,b,c)
begin
end

function main()
   var x
   var y
begin
   x = 10  *  zero(x,-21,y)
   if  x  ==  0  then
       x  =  20
   fi
   print(x)      // saída "20"
end
```

```
function prints(param)
begin
end

function main()
   var vars
   var begins
   var ifs
begin
   vars = 10
   begins = 20
   ifs = vars + begins
   prints(ifs)
end
```