コンパイラ理論 12
Racc その7 (ASTを作ろう)

櫻井彰人

intp.y を読もう

- プログラムを読んでみよう
- プログラム全体の流れは、
  - 構文木の作成
  - それを用いた翻訳実行
- 追加機能
  - 関数定義
- ほかは、基本機能に特化

intp.yの特徴であるAST生成は、「Ruby256倍」p.110以降に現れます
また、ASTを出力するのではなく、ASTを用いてインタープリタ動作をします

最初

```
program   : stmt_list
                {
                  result = RootNode.new( val[0] )
                }

stmt_list :
                {
                  result = []
                }
          | stmt_list stmt EOL
                {
                  result.push val[1]
                }
          | stmt_list EOL
```

ルートノード(木の根)を作る。
ノードの種類をいくつかつくり、
それごとに、メソッドを定義している。
それより細かい区分は、引数で行う

行の終わりを示す非終端記号

結果は配列に入れる

これは、前述のように、修正する

根っこ

```
class RootNode < Node

  def initialize(tree)
    super nil, nil
    @tree = tree
  end

  def evaluate
    exec_list Core.new, @tree
  end

end
```

親クラス
"new" のときに、実行される

使い方から分かるように、子木の配列が渡される

インスタンス変数に記憶する

インタプリタ動作用。今回は、各子ノードで必ず定義する。
コード生成が目的なら、ここをコード生成にする

親クラス Node のメソッドを使用する

大親 Node

```
class Node
  def initialize(fname, lineno)
    @filename = fname
    @lineno = lineno
  end

  attr_reader :filename
  attr_reader :lineno

  def exec_list(intp, nodes)
    v = nil
    nodes.each {|i| v = i.evaluate(intp) }
    v
  end

  def intp_error!(msg)
    raise IntpError, "in #{filename}:#{lineno}: #{msg}"
  end

  def inspect
    "#{self.class.name}/#{lineno}"
  end
end
```

http://ruby.kyoto-wu.ac.jp/documents/ruby-man-ja/Ruby_FAQ.html の 5.6 を参照
n = Node.new('intp', 1)
puts n.filename, n.lineno
などとできる

最後の値のみ残る

配列要素一個ずつ。
最後の値を返す

DefNode: 関数定義

```
class DefNode < Node

  def initialize(file, lineno, fname, func)
    super file, lineno
    @funcname = fname
    @funcobj = func
  end

  def evaluate(intp)
    intp.define_function @funcname, @funcobj
  end

end
```

Class Core

```
def define_function(fname, node)
    raise IntpError, "function #{fname} defined twice"
                              if @ftab.key?(fname)
    @ftab[fname] = node
  end
```

Function:関数本体定義

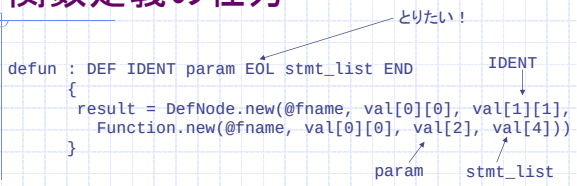
```
class Function < Node

   def initialize(file, lineno, params, body)
     super file, lineno
     @params = params
     @body = body
   end

   def call(intp, frame, args)
     unless args.size == @params.size
       raise IntpArgumentError,
         "wrong # of arg for #{frame.fname}()
              (#{args.size} for #{@params.size})"
     end
     args.each_with_index do |v,i|
       frame[@params[i]] = v
     end
     exec_list intp, @body
   end

 end
```

関数定義の仕方



FuncallNode: 関数呼び出し

```
class FuncallNode < Node
   def initialize(file, lineno, func, args)
     super file, lineno
     @funcname = func
     @args = args
   end
   def evaluate(intp)
     args = @args.map {|i| i.evaluate intp }
     begin
       intp.call_intp_function_or(@funcname, args) {
         if args.empty? or not args[0].respond_to?(@funcname)
           intp.call_ruby_toplevel_or(@funcname, args) {
             intp_error! "undefined function #{@funcname.id2name}"
           }
         else
           recv = args.shift
           recv.send @funcname, *args
         end
       }
     rescue IntpArgumentError, ArgumentError
       intp_error! $!.message
     end
   end

end
```

FuncallNode: 関数呼び出し2

Class Core

```
def call_intp_function_or(fname, args)
     if func = @ftab[fname]
       frame = Frame.new(fname)
       @stack.push frame
       func.call self, frame, args
       @stack.pop
     else
       yield
     end
   end

   def call_ruby_toplevel_or(fname, args)
     if @obj.respond_to? fname, true
       @obj.send fname, *args
     else
       yield
     end
   end
```

FuncallNode: 使い方

```
expr  : expr '+' expr
      {
          result = FuncallNode.new(@fname, val[0].lineno,
                                   '+', [val[0], val[2]])
      }
```

IfNode: if文

```
class IfNode < Node

   def initialize(fname, lineno, cond, tstmt, fstmt)
     super fname, lineno
     @condition = cond
     @tstmt = tstmt
     @fstmt = fstmt
   end

   def evaluate(intp)
     if @condition.evaluate(intp)
       exec_list intp, @tstmt
     else
       exec_list intp, @fstmt if @fstmt
     end
   end

 end
```

IfNode: 使い方

```
if_stmt : IF stmt THEN stmt_list else_stmt END
        {
         result = IfNode.new( @fname, val[0][0],
                              val[1], val[3], val[4] )
        }
```

stmt　stmt_list　else_stmt

WhileNode: while

```
class WhileNode < Node

   def initialize(fname, lineno, cond, body)
     super fname, lineno
     @condition = cond
     @body = body
   end

   def evaluate(intp)
     while @condition.evaluate(intp)
       exec_list intp, @body
     end
   end

end
```

WhileNode: 使い方

とりたい！

```
while_stmt: WHILE stmt DO EOL stmt_list END
            {
              result = WhileNode.new(@fname, val[0][0],
                                     val[1], val[4])
            }
```

AssignNode: 代入

```
class AssignNode < Node

   def initialize(fname, lineno, vname, val)
     super fname, lineno
     @vname = vname
     @val = val
   end

   def evaluate(intp)
     intp.frame[@vname] = @val.evaluate(intp)
   end

end
```

AssignNode: 使い方

```
assign   : IDENT '=' expr
    {
      result = AssignNode.new(@fname,
                              val[0][0], val[0][1], val[2])
    }
```

VarRefNode: 変数引用

```
class VarRefNode < Node

   def initialize(fname, lineno, vname)
     super fname, lineno
     @vname = vname
   end

   def evaluate(intp)
     if intp.frame.lvar?(@vname)
       intp.frame[@vname]
     else
       intp.call_function_or(@vname, []) {
         intp_error!
        "unknown method or local variable #{@vname.id2name}"
       }
     end
   end

end
```

```
class Frame

  def initialize(fname)
    @fname = fname
    @lvars = {}
  end

  attr :fname

  def lvar?(name)
    @lvars.key? name
  end
```

Rubyレファレンス「クラス/メソッドの定義」中「演算子式の定義」

使用例: frame[@params[i]] = v

```
  def [](key)
    @lvars[key]
  end

  def []=(key, val)
    @lvars[key] = val
  end

end
```

VarRefNode: 使い方

```
realprim :
      IDENT
        {
          result = VarRefNode.new(@fname, val[0][0],
                                  val[0][1])
        }
    | NUMBER
        {
          result = LiteralNode.new(@fname, *val[0])
        }
    | STRING
        {
          result = StringNode.new(@fname, *val[0])
        }
```

StringNode と LiteralNode

```
class StringNode < Node

  def initialize(fname, lineno, str)
    super fname, lineno
    @val = str
  end

  def evaluate(intp)
    @val.dup
  end

end
```

```
class LiteralNode < Node

  def initialize(fname, lineno, val)
    super fname, lineno
    @val = val
  end

  def evaluate(intp)
    @val
  end

end
```

StringNode/LiteralNode : 使い方

```
realprim :
      IDENT
        {
          result = VarRefNode.new(@fname, val[0][0],
                                  val[0][1])
        }
    | NUMBER
        {
          result = LiteralNode.new(@fname, *val[0])
        }
    | STRING
        {
          result = StringNode.new(@fname, *val[0])
        }
```

```
StringNode.new(@fname, val[0][0], val[0][1])
```