コンパイラ理論 10
Racc その3

櫻井彰人

関数呼び出し

- 関数呼び出しを可能にしよう
- といっても、Ruby のトップレベルのメソッドを呼べるようにするだけ
  - 関数定義は先送り。
- 実は、「Ruby 256倍 無道編」(p. 51以降)に従っても、Math.expなどは呼べない。
  - 引数が評価されてしまい、未定義！というエラーとなる
- まずは、Math モジュールをincludeして、exp(3) とか sin(3.14) といった形で使えるようにする。

```
exp: exp '+' exp { result += val[2] }
   | exp '-' exp { result -= val[2] }
   | exp '*' exp { result *= val[2] }
   | exp '/' exp { result /= val[2] }
   | exp '^' exp { result **= val[2] }
   | '(' exp ')' { result = val[1] }
   | '-' exp  =UMINUS { result = -val[1] }
   | NUMBER
   | IDENT       { result = do_varref( val[0] ) }
   | IDENT '(' args ')'
                 { result = do_funcall( val[0], val[2] ) }

args :           { result = [] }
     | exp       { result = val }  # result = [val[0]]
     | args ',' exp
                 { result.push( val[2] ) }
                  { result = val[0].push( val[2] ) }
                 引数列は、配列に入れる
```

inner に追加

```
def do_funcall( func, args )
  receiver = Object.new
  if receiver.respond_to?( func, true ) then
    ;
  elsif args[0] then
    receiver = args.shift
  else
    receiver = nil
  end

  receiver.send( func, *args )
end

---- footer

include Math
```

このOは大文字

呼べるかどうかを調べる

もくろみ(第一引数を対象オブジェクトとする)通りにいかない。args のパース時に、変数と思って評価してしまうからである

「最後の引数の直前に * がついている場合、その引数の値が展開されて 渡されます 」

メッセージ送信で、メソッドを起動する

Mathモジュールのメソッドを使うためのエ夫

respond_to? (Object)

標準クラス・モジュール > Object > respond_to?

obj.respond_to?(name, private = false)

respond_to? メソッドは、レシーバのオブジェクトに対してメソッドを呼び出せるかどうかを調べます。引数 name にはメソッド名をシンボルか文字列で指定します。メソッド name を持っていれば true、なければ false が返ります。

レシーバのクラスのメソッドだけでなく、親クラスやインクルードしているモジュールのメソッドも対象になります。デフォルトでは public なメソッドと protected なメソッドを調べますが、第2引数に true を指定すると private なメソッドも含めて調べます。

```
class Cat
  def hello
    "meow!"
  end

  private
  def sleep
    "zzz..."
  end
end

cat = Cat.new
p cat.respond_to?(:object_id)
p cat.respond_to?(:hello)
p cat.respond_to?(:sleep)
p cat.respond_to?(:sleep, true)
```

```
true
true
false
true
```

メソッド呼び出し

- 例えば、Math.exp(1.23) と書きたい。さらに、次も許したい
  - Math.exp(1.23).to_i
  - (Math.exp(1.23)).to_i
  - 2.3.to_i
  - x.to_f
  - モジュール名.メソッド名(引数1,..,引数n)
  - インスタンス名.メソッド名(引数1,..,引数n)
- 名前の解決(解決しないようにすること)が結構面倒

規則の追加

Math.exp(3) のとき、

[3]として

```
| IDENT '.' IDENT
              { result = do_methodinvoke( val[0], val[2], [] ) }
| IDENT '.' IDENT '(' args ')'
              { result = do_methodinvoke( val[0], val[2], val[4] ) }
| exp '.' IDENT
              { result = do_methodinvoke( val[0], val[2], [] ) }
| exp '.' IDENT '(' args ')'
              { result = do_methodinvoke( val[0], val[2], val[4] ) }
```

3.1.to_i のとき

前半2ルールを設けたのは、Mathなどの名前が第一のIDENTであるときに、それが評価されないようにするため
（exp からIDENTが生成されるが、そのときには、do_varref で値を取り出そうとしてしまう）

この結果、shift/reduce conflicts を起こすが、defaultであるshift優先で解決する

inner への追加

Math等のような名前は、シンボル（文字列の内部表現）で保存している

```
def do_methodinvoke( object, method, args )
   if object.class == Symbol then
     receiver = eval(object.id2name)
   else
     receiver = object
   end
   if receiver.respond_to?( method, true ) then
     ;
   else
     receiver = nil
   end

   if args==[] then
     receiver.send( method )
   else
     receiver.send( method, *args )
   end
 end
```

Math等の名前が入っているときの取扱い

3.5 等の値が入っているとき

わざわざエラーを起こすための処置

引数がないときの取扱い

footer を少し変更

```
while true
  puts
  print '? '
  str = gets.chop!
  break if /q/i =~ str
  begin
    puts "= #{parser.parse(str)}"
  rescue NoMethodError
    puts $!
  rescue ArgumentError
    puts $!
  rescue ParseError
    puts $!
  end
end
```

入っていたほうがよい

```
rescue NoMethodError, ArgumentError, ParseError
  print $!, "¥n"
```

練習問題3: 直前結果の参照

- 変数と関数呼び出しが使えるようにして下さい
- 直前の結果を参照することができると便利です。%%という特別な変数を用意してください。そして、

```
? Math.exp(3)
= 20.0855369231877

? x=%%*2
= 40.1710738463753

?
```

理解度の確認を兼ねて

- これまでの機能追加をすべて行ったインタープリタを作成して下さい。テスト結果を含めて、レポートとして下さい。
  - 来週の講義前に提出して下さい。
- 電子メールで、sakurai あっと ae どっと keio どっと ac どっと jp に送ってください。
  - ファイル形式は、pdfかMsWordで。

ヒント: 直前結果の参照

```
when /¥A%%/
      @q.push [$&, $&]
```

と

```
| '%%'       { result = @lastresult}
```

と

```
@lastresult = do_parse
```

をしかるべきところに入れればよい.
（3番目は、「入れる」ではなく「置き換える」です）