

コンパイラ理論 8 Racc その3

櫻井彰人

Raccの練習問題2

- ◆ 浮動小数点数が扱えるようにする。
- ◆ Ruby は変数に型がないため、そして、今作っている電卓にも型がないため、数値表現のみを可能とすれば、よい
- ◆ scannerの変更ですむ

```
case str
  when /%A%+%/
  when /%A%d+%.%d+%/
    @q.push [:NUMBER, $&.to_f]
  when /%A%d+%/
    @q.push [:NUMBER, $&.to_i]
```

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」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 =UMI NUS { 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
```

この0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 * が付いている場合、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

---- footer Mathモジュールのメソッドを使うための工夫
include Math

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

```

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

```

Math.exp(3) のとき、
[3]として
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よう)

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

```

def do_method_invoke( 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
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
end

```

入っていたほうがよい

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う便利です。%%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

?

```

Scannerをファイル入力対応に

- ◆ 単に、対応した。つまり1行ずつ読むでは、インタプリタ動作(構文解析と実行)
- ◆ ただし、次の拡張をした
 - 代入式を式として扱う
 - 文字列が使える

```

rule
target:
  | 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
    STRING
    IDENT { result = do_varref( val [0] ) }
    IDENT (' args ')
      { result = do_funcall( val [0], val [2] ) }
    IDENT '.' IDENT
      { result = do_method_invoke( val [0], val [2], [] ) }
    IDENT '.' IDENT (' args ')
      { result = do_method_invoke( val [0], val [2], val [4] ) }
    exp '.' IDENT
      { result = do_method_invoke( val [0], val [2], [] ) }
    exp '.' IDENT (' args ')
      { result = do_method_invoke( val [0], val [2], val [4] ) }
    IDENT '=' exp { result = do_assign( val [0], val [2] ) }

```

%% { result = getlastresult }
をしかるべき場所に挿入して下さい。

```

inner への追加
      %%用です。
      今回使いませんが、「Ruby 256倍」をそのまま
      持ってきました。

def getlastresult
  @lastresult
end
def putlastresult(v)
  @lastresult = v
end
  今回は使っていませんが。

def initialize
  @vtable = {}
  @lineno = 0
  @lastresult = 0
end

def do_assign( vname, val )
  @vtable[ vname ] = val
end

RESERVED = {
  'if' => :IF,
  'else' => :ELSE,
  'while' => :WHILE,
  'then' => :THEN,
  'do' => :DO,
  'def' => :DEF,
  'true' => :TRUE,
  'false' => :FALSE,
  'nil' => :NIL,
  'end' => :END
}

RESERVED_V = {
  'true' => true,
  'false' => false,
  'nil' => nil
}

```

```

def parse(f)
  @q = []
  @lineno = 0
  f.each do |line|
    line.strip!
    until line.empty?
      case line
      when /%A[a-zA-Z]w*/
        word = $&
        @q.push [(RESERVED[word] || :IDENT),
                 RESERVED_V.key?(word) ? RESERVED_V[word] : word.intern ]
      when /%A[d+y.Yd+]/
        @q.push [:NUMBER, $&.to_f]
      when /%A[d+]/
        @q.push [:NUMBER, $&.to_i]
      when /%A"(?:"[^"¥]+|¥¥.)"/
        @q.push [:STRING, eval($&)]
      when /%A=/
        @q.push [:EQ, '=' ]
      when /%A./
        @q.push [$&, $& ]
      else
        raise RuntimeError, 'must not happen'
      end
    end
    line = $'
  end

  @lineno += 1
  @q.push [false, '$']
  @lastresult = do_parse
end

  %%用です

```

「Ruby 256倍」から持ってきたが
(1) linenoを扱っていない。
(2) 1行ずつの扱いにしている
(3) 浮動小数点数の扱いを追加している

when /%A%/
 @q.push [\$&, \$&]
 をしかるべき場所に挿入して下さい

```

---- footer
# include Math

parser = Calc.new
begin
  fname = 'src.inpt'
  File.open(fname) {|f|
    parser.parse(f)
  }
rescue NoMethodError, ArgumentError, ParseError
  print $!, " at line #{parser.lineno}", "\n"
end

  ファイル読み込みだけの対応となる。

```

```

もとは、calc.y ですヨ

# $Id: calc.y, v 1.4 2005/11/20 13:29:32 aamine Exp $
#
# Very simple calculator.

class Calc
  prehigh
  nonassoc UMINUS
  left '*' '/'
  left '+ '-'
  preclow
end

```

```

rule
  target: exp
  | /* none */ { result = 0 }

  exp: exp '+' exp { result += val[2] }
  | exp '-' exp { result -= val[2] }
  | exp '*' exp { result *= val[2] }
  | exp '/' exp { result /= val[2] }
  | '(' exp ')' { result = val[1] }
  | '-' NUMBER =UMINUS { result = -val[1] }
  | NUMBER
end

```

```

---- header
# $Id: calc.y, v 1.4 2005/11/20 13:29:32 aamine Exp $

---- inner
def parse(str)
  @q = []
  until str.empty?
    case str
    when /%A[s+]/
    when /%A[d+]/
      @q.push [:NUMBER, $&.to_i]
    when /%A.\n/o
      s = $&
      @q.push [s, s]
    end
    str = $'
  end
  @q.push [false, 'Send']
  do_parse
end

def next_token
  @q.shift
end

```

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 あつと ae どつと keio どつと ac どつと 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 = getlastresult}
```

と

```
@lastresult = do_parse
```

をしかるべきところに入れればよい
三番目のものを入れるべきところを考えるのは
ちょっと難しいので、前の方のスライドに実際に
入れておいた。