

## コンパイラ理論 課題

櫻井彰人

講義での説明を聞いて下さい

## 提出

- ◆ 締め切りは 8月1日とします。
- ◆ 順不同で、できるものをできるだけ多く回答して下さい。
- ◆ 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。ただし、プログラムはテキストファイルで。

## レポート課題1

それぞれ1問として考えます

- ◆ 「電卓」タイプのプログラムで、浮動小数点数を実装して下さい。
  - 字句解析部分だけの改造ですみます。
- ◆ 「電卓」タイプのプログラムで、大小・等不等を判別する式と論理式を実装してください。
- ◆ 「電卓」タイプのプログラムで、代入式を実装して下さい。

## 補足 論理式

- ◆ 論理式の導入は、直線的。
- ◆ 優先順位に注意。
  - 算術演算子より弱い
- ◆ ただ、論理式は限られた場所だけで用いることにする。

```
rule
target:
| exp
| !exp
```

```
!exp: exp '==' exp { result = (val [0]==val [2]) }
| exp '!=' exp { result = (val [0]!=val [2]) }
| exp '>=' exp { result = (val [0]>=val [2]) }
| exp '<=' exp { result = (val [0]<=val [2]) }
| exp '>' exp { result = (val [0]> val [2]) }
| exp '<' exp { result = (val [0]< val [2]) }
| exp '&&' !exp { result = (val [0] && val [2]) }
| exp '||' !exp { result = (val [0] || val [2]) }
| '!' !exp { result = !( val [1]) }
| '(' !exp ')' { result = val [1] }
| TRUE {result=true}
| FALSE {result=false}
```

忘れてはいけない、、、

prehigh

```
nonassoc '==' '!=' '>=' '<=' '>' '<'
nonassoc '!'
left '&&' '||'
prelow
```

最も弱いところに追加

```
when /¥A(==|>=|<=|!=)/
@q. push [ $&, $& ]
when /¥A(&&|¥|¥&)/
@q. push [ $&, $& ]
```

勿論、まとめても結構

エスケープ記号の右なので、  
文字としての縦棒

少々無理ですが、print ぐらいしたいので

```
args :      { result = [] }
| exp      { result = val } # result = [val [0]]
| l exp    { result = val }
| args ' ', exp
|          { result.push( val [2] ) }
```

## ヒント: 代入式

下記のようなものがどこかに書かれています

```
| IDENT '=' exp { result = do_assign( val [0], val [2] ) }
```

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

## レポート課題2

### 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
```

```
?
```

## ヒント: 直前結果の参照

```
when /%%%/
  @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)

## レポート課題3

- ◆ calc.y で文字列を扱えるようにして下さい。
  - intp.y の文字列部分をコピーすればよい

スキナーの "NUMBER" のあとに

```
when /%A" (? : [^"¥¥]+ | ¥¥. ) *"/, /%A' (? : [^' ¥¥]+ | ¥¥. ) *' /
  @q.push [ : STRING, eval ($&) ]
```

exp の定義中 ( "NUMBER" のあとに )

```
| STRING
```

- ◆ 上記の正規表現は何を行っているのか説明して下さい

## レポート課題4

ファイルが読めるようにしよう。行番号も実装しよう

```
---- footer
```

```
parser = Calc.new
begin
  if ARGV.length >= 1 then
    fname = ARGV[0]
  else
    fname = 'test'
  end
  File.open(fname) { |f|
    parser.parse(f)
  }
  rescue NoMethodError, ArgumentError, ParseError
    print $!, " at line #{parser.lineno}", "\n"
end
```

コマンドライン引数の1番目  
デフォルトのファイル名  
ファイルを、一行ずつ処理します  
行番号関連

```
def parse(str)
  @q = []
  lineno = 1
  # 色々
```

## ファイルが読めるようにしよう(2)

```
def parse(f)
  @q = []
  @lineno = 1

  f.each do |line|
    line.strip!
    print( ">>", line, "\n" )
    until line.empty?
      case line
        # 色々
      end
      line = $'
    end
    @q.push [false, $end]
    do_parse
    puts( "=>"+@result.to_s )
    @lineno += 1
  end
end
```

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

```
def lineno
  @lineno
end
# 下記も可
attr_reader :lineno
```

## 行番号部分

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

%%用です

```
def parse(f)
  @q = []
  @lineno = 0
  f.each do |line|
```

```
def lineno
  @lineno
end
```

```
rescue NoMethodError, ArgumentError, ParseError
  print $!, " at line #{@parser.lineno}", "\n"
end
```

## レポート課題5

### 練習問題4: if then else end

- ◆ Ruby の if then else end と if then end 相当の文が使えるようにしてください。
  - if 部分には数式を書き、その値が0ならばelse部分の値を、0以外であればthen部分の値を if 式の値とする
- ◆ 何となくできるようにするのは、簡単です。
  - 例えば、then部分、else部分には、"代入のない"式しか現れないとする
  - 無駄な計算をしてもよいとする
    - ・ 結果には不要な式 (例えば、条件が成立しているときのelse部分) の計算をしてもよいとする

### 例解

- ◆ 例えば exp の定義に次のようなものをつけ加えればよい

```
| IF exp THEN exp END
{ if val[1] !=0 then result=val[3] else result=nil end}
| IF exp THEN exp ELSE exp END
{ if val[1] !=0 then result=val[3] else result=val[5] end}
```

```
a = 0
if a then 1 else 2 end
```

= 2

```
a = 1
if a then 1 else 2 end
```

= 1

### プログラム作成上の注意

if, then, else, end といったキーワードが出てくる。  
変数として認識しては困る。  
予約語 (つまり、システムが使用することを予約している語) とする。

```
def parse(str)
  # 色々
  when /%A[a-zA-Z_]\w*/
    word = $&
    @q.push [(RESERVED[word] || :IDENT),
              RESERVED_V.key?(word) ? RESERVED_V[word] : word.intern ]
  when /%A[%d+%.]\d+%/
    @q.push [:NUMBER, $&.to_f]
  # 色々
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
}
```

## しかし、失敗？

- ◆ 代入式が書けるようにすると、結果が思わしくない

```
| IF exp THEN exp END
{ if val[1] !=0 then result=val[3] else result=nil end}
| IF exp THEN exp ELSE exp END
{ if val[1] !=0 then result=val[3] else result=val[5] end}
```

```
a=0
if a then b=1 else b=2 end
print( "b= ", b, "\n")
```

b= 2

```
a=1
if a then b=1 else b=2 end
print( "b= ", b, "\n")
```

b= 2

代入式を書けるようにする。  
式として、代入式も認める

```
prehigh
right '^'
nonassoc UMINUS
left '*' '/'
left '+' '-'
right '='
preclow
rule
target: exp
  + assign
  /* none */ { result = 0 }

exp: exp '+' exp { result += val[2] }
    exp '-' exp { result -= val[2] }

# 色々
| assign

assign : IDENT '=' exp { result = do_assign( val[0], val[2] ) }
end
```

この現象が起こることを確認してください。

## 思わしくない理由

- ◆ "then" 部分も "else" 部分も、必ず、構文解析を行い
- ◆ 構文解析に成功すれば、意味解析（インタープリターでは「実行」）もしてしまう。
- ◆ if 全体の値は正しいのだが、副作用がある部分（代入！）はおかしくなる。
  - また、効率が悪い（計算する必要のない計算を行ってしまう）

```
| IF exp THEN exp END
{ if val [1] !=0 then result=val [3] else result=nil end}
| IF exp THEN exp ELSE exp END
{ if val [1] !=0 then result=val [3] else result=val [5] end}
```

## レポート課題6

- ◆ calc.y つまり電卓形式のプログラムでは、ループ（whileループ、forループ、untilループ他）を実装することは難しい理由を説明して下さい。

## レポート課題7 練習問題6

- ◆ intp.y を動かしてみてください。そして次の確認と改善をして下さい。
  - コマンドラインからファイル名を読むようにして下さい。
  - if then else end の構文が複数行に渡ることを強制しています。一行内にも書いてもよいようにして下さい。
  - （山勘を働かせて）べき乗(^)を導入して下さい。
    - 注意: FuncallNode.newを呼ぶときの、べき乗の関数名は、^ではなく\*\*です。

## ファイルの読み込み

```
begin
  tree = nil
  if ARGV.length>=1 then
    fname = ARGV[0]
  else
    fname = 'src.intp'
  end
  File.open(fname) {|f|
    tree = Intp::Parser.new.parse(f, fname)
  }
  tree.evaluate
rescue Racc::ParseError, Intp::IntpError, Errno::ENOENT
  raise #####
  $stderr.puts "#{File.basename $0}: #{$_!}"
  exit 1
end
```

変更です

## べき乗

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

```
prehigh
nonassoc UMINUS
right '^'
left '*' '/'
left '+' '-'
nonassoc EQ
preclow
```

## If-then-else-end: 失敗ではないが

```
if_stmt : IF stmt THEN EOL stmt_list else_stmt END
        {
            result = IfNode.new( @fname, val [0][0],
                                val [1], val [4], val [5] )
        }
        | IF stmt THEN      stmt_list else_stmt END
        {
            result = IfNode.new( @fname, val [0][0],
                                val [1], val [3], val [4] )
        }
```

追加した

shift/reduce conflict がたくさん出る。というのは stmt\_list から EOL stmt EOL という形が導出され、追加した部分からも、もともたからある部分からも、同じものが生成される、すなわち、曖昧な文法になっているからである

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

## If-then-else-end: 失敗ではないが

```
else_stmt : ELSE EOL stmt_list
          {
              result = val [2]
          }
          |
          {
              result = nil
          }
          | ELSE stmt_list
          {
              result = val [1]
          }
```

追加した

shift/reduce conflict がたくさん出る。というのは stmt\_list から EOL stmt EOL という形が導出される、すなわち、曖昧な文法になっているからである

## If-then-else-end

- ◆ つまり、逆に(つまり、この問題の原因を逆手にとり)、stmt\_list が空文を許すように定義されているため、if\_stmt と else\_stmt で EOL を落とすだけでよい！

これで万事解決かというと、そうでもない:

- ◆ if ... then a=1 else ... というように、else の左側に (EOLなしに) 文を書くことはできない。
- ◆ これは、stmt\_list の定義によると、stmt\_list の最後は必ず EOL でなければならないからである。
- ◆ つまり、... a=1 else ... などとするには、stmt\_list の定義を書き換えて、最後にEOLが来る必要をなくせばよい。

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

```
stmt_list :
{
    result = []
}
| stmt
{
    result.push val [0]
}
| stmt_list EOL stmt
{
    result.push val [2]
}
| stmt_list EOL
```