コンパイラ理論 11
Racc その6 (ファイルと論理式)

櫻井彰人

ファイルが読めるようにしよう

```
---- footer

parser = Calcp.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

デフォールトのファイル名

ファイルを、一行ずつ処理します

今は、関係なし

```
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
```

Ruby「文字」に関する確認

¥t	水平タブ	horizontal tab	(0x09)
¥v	垂直タブ	vertical tab	(0x0B)
¥n	改行	newline	(0x0A)
¥r	復帰	return	(0x0D)
¥b	バックスペース	back space	(0x08)
¥f	改ページ	form feed	(0x0C)
¥a	ベル	bell	(0x07)
¥e	エスケープ文字	escape	(0x1B)
¥nnn	符号化バイト値の8進数表現（nnn の8進数3文字で表現）		
¥xHH	符号化バイト値の16進数表現（HH の16進数2文字で表現）		
¥cx, ¥C-x	制御文字（x は a から z までのいずれかの文字）		
¥M-x	メタ（x|0x80）		
¥M-¥C-x	メタ制御文字		
¥uHHHH	ユニコード文字（HHHH の16進数4桁）		
¥u{HHHHHHH HHHHHHH ....}	ユニコード文字列（HHHHHHH は16進数1桁から6桁まで指定可能）		

¥b は文字クラス内でのみ有効な表現です。文字クラスの外では 単語の区切りを表すメタ文字列と解釈されます。
「¥s」は文字列では空白(0x20)を意味しますが、正規表現では タブなどを含む空白文字全般にマッチするメタ文字列です。

文字クラス(character class) とは角括弧 [と] で囲まれ、1個以上の文字を列挙したもので、 いずれかの1文字にマッチします。
例: /W[aeiou]rd/

動作の確認

```
a=0
b=0
if a then if b then c=0 else c=1 end else if b then c=2 else c=3 end end
c
a=0
b=1
if a then if b then c=0 else c=1 end else if b then c=2 else c=3 end end
c
a=1
b=0
if a then if b then c=0 else c=1 end else if b then c=2 else c=3 end end
c
a=1
b=1
if a then if b then c=0 else c=1 end else if b then c=2 else c=3 end end
c
```

↓

```
=>0          =>1
=>0          =>0
=>3          =>1
=>3          =>1
=>0          =>1
=>1          =>1
=>2          =>0
=>2          =>0
```

結果のみを持ってきた

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
        | lexp


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

忘れてはいけない、、、

```
prechigh
   nonassoc '==' '!=' '>=' '<=' '>' '<'
   nonassoc '!'
   left '&&' '||'
 preclow
```

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]]
      | lexp      { result = val }
      | args ',' exp
                  { result.push( val[2] ) }
```

```
$ ruby calcl1.rb iftest4
a= 1 b= 1
a==b
a= 1 b= 0
a!=b
a= 1 b= 1
a==b and b==1
a= 1 b= 1
not !a==b
a= 1 b= 1
not !(a==b and b==1)
true
```

「なし」です

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(^)を導入してください。
    - 注意: FuncallNode.newを呼ぶときの、べき乗の関数名は、^ではなく**です。