

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
end

```

```

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

```

```

def lineno
  @lineno
end

```

下記も可
attr_reader :lineno

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、、、

```
prehigh
  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 calc11.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。