Racc とは？

- パーサージェネレータ
  - 青木峰郎氏作 構文解析器（パーサー）のジェネレータ
- Racc = Ruby yACC
  - Yacc = Yet Another Compiler Compiler
- 入手方法
  - http://www.loveruby.net/ja/projects/racc/
- 使い方
  - http://www.loveruby.net/ja/projects/racc/doc/usage.html

まき「Raccでおてがる構文解析」より

Raccの基本的な使い方(1)

- 構文規則等のファイルを書く
  - 文法を記述
  - 「---- header」で require など
  - 「---- inner」でクラスの中身、字句解析処理など
  - 「---- footer」で後処理

j_parser.y

```
class JapaneseParser
rule
    文法の記述
end
---- header
require 'node'
---- inner
def parse
    ...
    do_parse
end
---- footer
....
```

Raccの基本的な使い方(2)

- パーサークラス生成

出来るパーサーのファイル名
Rubyのプログラムである
なくてもよい（下を参照）

```
>racc -o j_parser.rb j_parser.y
```

規則のファイル

  - 指定した名前(上記例だとj_parser.rb)のRubyソースファイルができる（中身はmodule_evalの山）

```
>racc  j_parser.y
```

  とした場合は、j_parser.tab.rb というファイルが作られる

文法ルール書き方(1)

非終端子: 一つの規則
　　{アクション}
　　| ほかの規則達

- 文法は Yacc そっくり. 終端記号は引用符(')で括る. アクションはRubyプログラム
- アクション
  - 文法の並びにマッチした時の処理を記述
  - valに、マッチした配列が渡される
  - resultにセットしたオブジェクトが、上位の木からvalで参照できる

文法ルール書き方(2)

（例）

```
program :
          {result = []}
          | program stmt
          { result ||= []
            result << val[1]}
stmt    : assign EOL
          {result = StmtNode.new(val[0])}
assign : IDENT '=' expr
          {result =  AssignNode.new(val[0],val[2])}
expr    : IDENT
          {result = VariableNode.new(val[0])}
          | expr '+' expr
          {result = ArithmeticNode.new(val[0],val[2])}
......
```