

Racc とは？

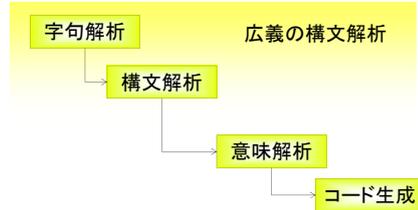
- パーサージェネレータ
 - 青木峰郎氏作 構文解析器(パーサ)のジェネレータ
- Racc = Ruby yACC
 - Yacc = Yet Another Compiler Compiler
- 入手方法
 - <http://www.loveruby.net/ja/projects/racc/>
- 使い方
 - <http://www.loveruby.net/ja/projects/racc/doc/usage.html>



まき「Raccでおてがる構文解析」より

コンパイラのしごと

- ソースコードをコンピュータやVMが実行できる命令語に変換すること



字句解析

下段が意味値

huge=(huge+1)/2

ひとつのトークン

huge	=	(huge	+	1)	/	2
識別子	-	-	識別子	-	整数	-	-	整数

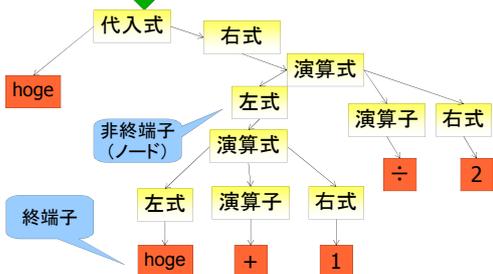
- ソースコードの文字列をトークンと呼ばれる単位に分割する
- トークンは意味値を持つ
 - 識別子、整数、文字列
 - 予約語やカッコに意味値はない

構文解析(狭義)(1)

- トークン列を解析して抽象構文木に変換する

構文解析(狭義)(2)

huge	=	(huge	+	1)	/	2
識別子	-	-	識別子	-	整数	-	-	整数



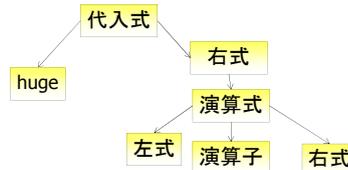
ここまでまとめ

$((huge+1) \div 2) \times 3$

↓ 字句解析

((huge + 1) ÷ 2) × 3

↓ 構文解析



Raccの基本的な使い方(1)

拡張子y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
  end
  ---- header
  require 'node'
  ---- inner
  def parse
    ...
    do_parse
  end
  ---- footer
  ....
```

文法の記述

Raccの基本的な使い方(2)

パーサークラス生成

- >racc -o j_parser.rb j_parser.y
- 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の嵐)
- >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

出来るパーサーのファイル名
Rubyのプログラムである

規則のファイル

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])}
.....
```