コンパイラ理論 7
Racc その2

櫻井彰人

変数の導入

- 変数宣言なしとしよう
- 変数は、式の左辺（代入される）と右辺（引用される）に現れる。
- 同じものだが、左辺に現れるときと右辺に現れるときとでは、扱いが全く異なる
  - 右辺で参照されたときは、その値が使われる
  - 左辺で参照されたときは、代入先である「場所」が使われる
  - インタープリタのときは、常識に従えば、まぁ、大丈夫
- とりあえず、代入式を式（exp）とは別に定義する

変数名表

- 変数の型は考えない（宣言もないし）
- 必要な表は、変数名と値の対応を記録する表だけ
  - 関数定義を許すには、この表を拡張する必要がある。変数名の表ではなく、識別子の表となる。
- Rubyにはハッシュ表がある。これは便利。

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

  exp: exp '+' exp { result += val[2] }
     | exp '-' exp { result -= val[2] }
     | exp '*' exp { result *= val[2] }
     | exp '/' exp { result /= val[2] }
     | exp '^' exp { result **= val[2] }
     | '(' exp ')' { result = val[1] }
     | '-' exp  =UMINUS { result = -val[1] }
     | NUMBER
     | IDENT       { result = do_varref( val[0] ) }

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

end
```

変数に入っている値を変数表からもってくる。未定義のときは、exception発生予定

右辺の値を、変数表に入れる

Defaultのアクションは { result = result }、{result = val[0] }（ともに同語反復）である

```
---- inner

  def initialize
    @vtable={}
  end

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

  def do_varref( vname )
    @vtable[ vname ] or
       raise( ParseError, "unknown variable #{vname}" )
  end
```

innerの先頭に入れる

Hashにするつもりで初期化

Rubyのシンボル型のデータがくるはず

値がnilでも未定義になるがご勘弁

Exceptionを発生させる

```
def parse(str)
   @q = []
   until str.empty?
     case str
     when /¥A¥s+/
     when
       @q.push [:NUMBER,
     when /¥A¥d+/
       @q.push [:NUMBER, $&.to_i]
     when /¥A¥w+/
       @q.push [:IDENT, $&.intern]
     when /¥A.|¥n/o
       s = $&
       @q.push [s, s]
     end
     str = $'
   end
   @q.push [false, '$end']
   do_parse
end
def next_token
   @q.shift
end
```

```
---- footer

parser = Calcp.new
puts
puts 'type "Q" to quit.'
puts
while true
  puts
  print '? '
  str = gets.chop!
  break if /q/i =~ str
  begin
    puts "= #{parser.parse(str)}"
  rescue ParseError
    puts $!
  end
end
```

Exception ParseErrorを受取る

Exceptionの内容

Ruby 補足

- **配列: 要素の追加・削除**
- push メソッドを使用して、配列に要素を追加することができる。

```
irb(main):001:0> a = [1,2,3,4]
=> [1, 2, 3, 4]
irb(main):002:0> a.push(10)
=> [1, 2, 3, 4, 10]
irb(main):003:0>
```

- pop メソッドを使用して、配列の最後の要素を取り出すことができる

```
irb(main):003:0> a.pop
=> 10
irb(main):004:0> a
=> [1, 2, 3, 4]
irb(main):005:0>
```

Ruby 補足: ハッシュ

- 配列では、インデックスを用いて要素を指定する。ハッシュでは、キーと呼ばれるものを用いて要素を指定する。

```
irb(main):001:0> h = Hash::new
=> {}
irb(main):002:0> h['apple'] = 150
=> 150
irb(main):003:0> h['banana'] = 200
=> 200
irb(main):004:0> h['lemon'] = 300
=> 300
irb(main):005:0> h
=> {"apple"=>150, "banana"=>200, "lemon"=>300}
irb(main):006:0> h['apple']
=> 150
irb(main):007:0>
```