平成26年度 3回生前期学生実験SW
中間レポート1

松田貴大

学籍番号:1029-24-4015

提出日：平成26 年6 月5 日17:00
提出期限：平成26 年6 月13 日 17:00

1 はじめに

使用言語はRuby(2.1.0p0)で、パーサの作成にはracc(1.4.11)[1]を用いている。

2 課題3

2.1 ソースコード

パスは、~/2014sw/compiler/report1/compiler.yである。

```
1  # $Id: compiler.y
2  #
3  # TinyC compiler
4
5  class Tinyc
6    prechigh
7      right DATATYPE
8      right IF
9      right ELSE
10     right WHILE
11     right RETURN
12     left LE
13     left GE
14     left EQUAL
15     left NOTEQUAL
16     left LOGICALAND
17     left LOGICALOR
18     left '*' '/' '%'
19     left '+' '-'
20     left '<' '>'
21     right '='
22     right '(' '{'
23     left ')' '}'
24   preclow
25 rule
26   program
27       : external_declaration
28       | program external_declaration
29   external_declaration
```

[1] http://i.loveruby.net/ja/projects/racc/

```
30      : declaration
31      | function_definition
32  declaration
33      : DATATYPE declarator_list ';'
34  declarator_list
35      : declarator
36      | declarator_list ',' declarator
37  declarator
38      : IDENTIFIER
39  function_definition
40      : DATATYPE declarator '(' parameter_type_list_opt
            ')' compound_statement
41  parameter_type_list
42      : parameter_declaration
43      | parameter_type_list ',' parameter_declaration
44  parameter_type_list_opt
45      : parameter_type_list
46      | /* none */
47  parameter_declaration
48      : DATATYPE declarator
49  statement
50      : ';'
51      | expression ';'
52      | compound_statement
53      | IF '(' expression ')' statement
54      | IF '(' expression ')' statement ELSE statement
55      | WHILE '(' expression ')' statement
56      | RETURN expression ';'
57  compound_statement
58      : '{' declaration_list_opt statement_list_opt '}'
59  declaration_list
60      : declaration
61      | declaration_list declaration
62  declaration_list_opt
63      : declaration_list
64      | /* none */
65  statement_list
66      : statement
67      | statement_list statement
68  statement_list_opt
69      : statement_list
70      | /* none */
71  expression
```

```
72      : assign_expr
73      | expression ',' assign_expr
74  assign_expr
75      : logical_OR_expr
76      | IDENTIFIER '=' assign_expr
77  logical_OR_expr
78      : logical_AND_expr
79      | logical_OR_expr LOGICALOR logical_AND_expr
80  logical_AND_expr
81      : equality_expr
82      | logical_AND_expr LOGICALAND equality_expr
83  equality_expr
84      : relational_expr
85      | equality_expr EQUAL relational_expr
86      | equality_expr NOTEQUAL relational_expr
87  relational_expr
88      : add_expr
89      | relational_expr '<' add_expr
90      | relational_expr '>' add_expr
91      | relational_expr LE add_expr
92      | relational_expr GE add_expr
93  add_expr
94      : mult_expr
95      | add_expr '+' mult_expr
96      | add_expr '-' mult_expr
97  mult_expr
98      : unary_expr
99      | mult_expr '*' unary_expr
100     | mult_expr '/' unary_expr
101 unary_expr
102     : posifix_expr
103     | '-' unary_expr
104 posifix_expr
105     : primary_expr
106     | IDENTIFIER '(' argument_expression_list_opt ')'
107 primary_expr
108     : IDENTIFIER
109     | CONSTANT
110     | '(' expression ')'
111 argument_expression_list
112     : assign_expr
113     | argument_expression_list ',' assign_expr
114 argument_expression_list_opt
```

```
115         : argument_expression_list
116         | /* none */
117 end
118
119 ---- header
120 # $Id: calc.y,v 1.4 2005/11/20 13:29:32 aamine Exp $
121 ---- inner
122
123   def parse(str)
124     @q = []
125     until str.empty?
126       case str
127       when /\A\s+/
128       when /\A\d+/
129         @q.push [:CONSTANT, $&.to_i]
130       when /\A(&&)/
131         @q.push [:LOGICALAND, $&]
132       when /\A(\|\|)/
133         @q.push [:LOGICALOR, $&]
134       when /\A(int)/
135         @q.push [:DATATYPE, $&]
136       when /\A(if)/
137         @q.push [:IF, $&]
138       when /\A(else)/
139         @q.push [:ELSE, $&]
140       when /\A(while)/
141         @q.push [:WHILE, $&]
142       when /\A(<=)/
143         @q.push [:LE, $&]
144       when /\A(>=)/
145         @q.push [:GE, $&]
146       when /\A(==)/
147         @q.push [:EQUAL, $&]
148       when /\A(!=)/
149         @q.push [:NOTEQUAL, $&]
150       when /\A(return)/
151         @q.push [:RETURN, $&]
152       when /\A[a-zA-Z]\w*/
153         @q.push [:IDENTIFIER, $&]
154       when /\A.|\n/o
155         s = $&
156         @q.push [s, s]
157       end
```

```
158         str = $'
159       end
160       @q.push [false, '$end']
161       do_parse
162     end
163
164     def next_token
165       @q.shift
166     end
167
168 ---- footer
169
170 parser = Tinyc.new
171
172 str = ''
173 while true
174   add = gets
175   if add == nil
176     break
177   else
178     str += add
179   end
180 end
181 # puts str.inspect
182 if str != nil
183   str.chop!
184   begin
185     puts "success!!! \n result => #{parser.parse(str)}"
186   rescue ParseError
187     puts $!
188   end
189 end
```

2.2 設計方針

実験ページの資料に書かれた仕様を持つ TinyC パーサを作成する。今回の課題においては、アクション部分の実装は行わない。構文チェックのみを行い、正しい構文の場合は"success" を返し、間違った構文の場合はエラーを返す。

2.3 各部の説明

2.3.1 prechigh〜preclow

この部分には、終端記号となる演算子と結合のルールを記述する。left は左結合、right は右結合を表す。

2.3.2 rule〜end

この部分には、文法規則を文脈自由文法の形式で記述する。実験ページの資料に書かれた仕様をそのまま書いていく。文法規則の後にアクションを書くことも出来るが、今回は省略する[2]。

工夫点 資料の文法規則には、省略可能の意味で opt と振られたルールが存在するが、それをそのまま記述することは出来ない。そこで、省略しない文法規則と、それを基にした省略可能な文法規則を書くことで、対応した。たとえば、以下のようなルールが該当する。

```
declaration_list
    : declaration
    | declaration_list declaration
declaration_list_opt
    : declaration_list
    | /* none */
```

上記ルールでは、declaration_list は省略不可能なルールだが、declaration_list_opt は省略可能なルールとなり、仕様を満たす。

2.3.3 inner

この部分がパーサの本体となる。関数 parse では、文字列を正規表現によって切り出しラベルを振っていく。たとえば、整数文字列は CONSTANT というラベルを振り、英字から始まり英数字とアンダーライン (_) からなる文字列には IDENTIFIER というラベルを振る。また、1 個以上のスペースとタブと改行は読み飛ばす。

[2] アクションを省略した場合、その部分のアクションは対応文法規則の最初の部分に現れる終端記号となる。

2.3.4 footer

この部分にはRubyプログラムの本体を書く。今回は、標準入力から入力の終わりまで文字列を読み、パーサによって構文解析を行い、正しい構文であれば”success!!!” の文字列とともにアクション部分で記述した結果を結合したもの[3]が出力され、間違った構文であればエラーを返す。

3 課題4

2.1節で述べたソースコードから生成されたパーサを使い、いくつかのTinyCプログラムの構文を解析した結果を示す。

3.1 例1

```
1  int fact(int x)
2  {
3      int z;
4      z = 1;
5      while(x >= 1)
6      {
7          z = z / x;
8          x = x - 1;
9      }
10     if(((x || z) == (x && z)) != z){
11         z = 1;
12     }else {
13         z = 2;
14     }
15     return z;
16 }
```

結果

```
success!!!
 result => int
```

[3]今回の場合は文字列の最初のルールで現れる最初の終端記号

3.2 例2

```
1  int hoge()
2  {
3    return 1;
4  }
5
6  int fuga()
7  {
8    if(x == 0)
9    {
10   }
11 }
```

結果

```
success!!!
 result => int
```

3.3 例3

```
1  a = 0;
```

結果

```
parse error on value "a" (IDENTIFIER)
```

3.4 例4

```
1  int a_52dax_255(int a, int b, int c){
2      int hsdr_3151s_;
3      hsdr_3151s_ = -c + ((1 + a) * 5 / (2 * b));
4
5      return hsdr_3151s_;
6  }
```

結果

```
success!!!
 result => int
```

3.5 例5

```
1 int foo(){
2     int 1dame;
3 }
```

結果

```
parse error on value 1 (CONSTANT)
```

4 感想

racc を使えば文脈自由文法形式の記述で簡単にパーサが生成できて便利だと思った。TinyC でなくても、構文解析器を作ることが出来るようになったと思う。