

# Programming in C/C++

## Exercises set six: parsers II

Christiaan Steenkist  
Jaime Betancor Valado  
Remco Bos

March 8, 2017

### Exercise 36, expand new grammar

We were tasked to add some functionalities to the demo parser.

#### Code listings

##### Scanner

Listing 1: lexer.ll

```
1 %filenames scanner
2 %interactive
3
4 digits      [0-9]+
5 optdigits   [0-9]*
6 exp         [eE] [-+]? {digits}
7
8 %%
9
10 [ \t]+
11
12 {digits}    |
13 {digits} "." {optdigits} {exp}? |
14 {optdigits} "." {digits} {exp}? return Parser::NUMBER;
15
16 QUIT        return ParserBase::
    Tokens__::QUIT;
```

```

17 EXIT                                     return ParserBase::
    Tokens__::QUIT;
18 LIST                                     return ParserBase::
    Tokens__::LIST;
19 [[:alpha:]]_ [[:alnum:]]_*               return Parser::IDENT;
20 \n|.                                     return matched()[0];

```

## Scanner

Listing 2: grammar.gr

```

1 %class-name Parser
2
3 %filenames parser
4 %parsefun-source parse.cc
5
6 %baseclass-preinclude rulevalue.h
7 %stype RuleValue
8
9 %scanner ../scanner/scanner.h
10
11 %token  NUMBER
12         IDENT
13         QUIT
14         EXIT
15         LIST
16
17 %right  '='
18 %left  '+' '-'
19 %right  uMinus
20
21 // %debug
22
23 %%
24
25 lines:
26     lines line
27 |
28     line
29 ;
30

```

```

31
32 line:
33     expr '\n'
34     {
35         display($1);
36     }
37 |
38     error '\n'
39     {
40         prompt();
41     }
42 |
43     '\n'
44     {
45         prompt();
46     }
47 |
48     LIST
49     {
50         list();
51     }
52 |
53     EXIT
54     {
55         quit();
56     }
57 |
58     QUIT
59     {
60         quit();
61     }
62 ;
63
64
65 expr:
66     NUMBER
67     {
68         $$ = value();
69     }
70 |

```

```

71     IDENT
72     {
73         $$ = variable();
74     }
75 |
76     '-' expr    %prec uMinus
77     {
78         $$ = negate($2);
79     }
80 |
81     expr '+' expr
82     {
83         $$ = add($1, $3);
84     }
85 |
86     expr '-' expr
87     {
88         $$ = sub($1, $3);
89     }
90 |
91     '(' expr ')'
92     {
93         $$ = $2;
94     }
95 |
96     expr '=' expr
97     {
98         $$ = assign($1, $3);
99     }
100 ;

```

Listing 3: parser.h

```

1  ...
2  #include <vector>
3  #include <unordered_map>
4  #include <string>
5  ...
6
7  ...
8      std::vector<double> d_value;

```

```

9      std::unordered_map
10         <std::string, unsigned> d_symtab;
11 ...
12
13 ...
14     // added functions for the calculator:
15     void display(RuleValue &e);
16     void done();
17     void prompt();
18
19     RuleValue &add(RuleValue &lvalue,
20                   RuleValue &rvalue);
21     RuleValue &assign(RuleValue &lvalue,
22                      RuleValue &rvalue);
23     RuleValue &negate(RuleValue &e);
24     RuleValue &sub(RuleValue &lvalue,
25                  RuleValue &rvalue);
26     RuleValue value();
27     RuleValue variable();
28
29     double valueOf(RuleValue const &e);
30
31     void quit();
32     void list();
33 };
34
35
36 #endif

```

Listing 4: list.cc

```

1  #include "Parser.ih"
2
3  void Parser::list()
4  {
5      std::map<std::string, unsigned>
6          ordered_map(d_symtab.begin(), d_symtab.end());
7
8      for (auto it = ordered_map.begin();
9           it != ordered_map.end(); ++it)
10

```

```

11         std::cout << it->first
12             << " ==> "
13             << it->second
14             << '\n';
15     }

```

Listing 5: quit.cc

```

1  #include "Parser.ih"
2
3  void Parser::quit()
4  {
5      ACCEPT();
6  }

```

## Exercise 37, substantial grammar extension

All these operators.

### Code listings

#### Scanner for both 37 and 38

Listing 6: lexer

```

1  %filenames scanner
2  %interactive
3
4  digits      [0-9]+
5  optdigits   [0-9]*
6  exp         [eE][+-]?{digits}
7
8  %%
9
10 [ \t]+
11
12 {digits} |
13 {digits}"."{optdigits}{exp}? |
14 {optdigits}"."{digits}{exp}? return Parser::NUMBER;
15
16 [[:alpha:]]_ [[:alnum:]]_* return Parser::IDENT;
17

```

```

18 "+"      return Parser::COMPADD;
19 "-="     return Parser::COMPSUB;
20 "*="     return Parser::COMPMUL;
21 "/="     return Parser::COMPDIV;
22 "<=<="    return Parser::COMPLSH;
23 ">=>="    return Parser::COMPRSH;
24 "<<="    return Parser::LSH;
25 ">>="    return Parser::RSH;
26
27 \n|.      return matched()[0];

```

## Parser

Listing 7: grammar

```

1 %class-name Parser
2
3 %filenames parser
4 %parsefun-source parse.cc
5
6 %baseclass-preinclude rulevalue.h
7 %stype RuleValue
8
9 %scanner ../scanner/scanner.h
10
11 %token  NUMBER
12         IDENT
13
14 %right  '=' COMPADD COMPSUB COMPMUL
15         COMPDIV COMPLSH COMPRSH
16 %left   '+' '-'
17 %left   '*' '/'
18 %right  '%' LSH RSH
19 %right  uMinus
20
21 // %debug
22
23 %%
24
25 lines:
26     lines line

```

```

27 |
28     line
29 ;
30
31
32 line:
33     expr '\n'
34     {
35         display($1);
36     }
37 |
38     error '\n'
39     {
40         prompt();
41     }
42 |
43     '\n'
44     {
45         prompt();
46     }
47 ;
48
49
50 expr:
51     NUMBER
52     {
53         $$ = value();
54     }
55 |
56     IDENT
57     {
58         $$ = variable();
59     }
60 |
61     '-' expr %prec uMinus
62     {
63         $$ = negate($2);
64     }
65 |
66     expr '*' expr

```



```

67     {
68         $$ = mul($1, $3);
69     }
70 |
71     expr '/' expr
72     {
73         $$ = div($1, $3);
74     }
75 |
76     expr '+' expr
77     {
78         $$ = add($1, $3);
79     }
80 |
81     expr '-' expr
82     {
83         $$ = sub($1, $3);
84     }
85 |
86 |
87     '(' expr ')'
88     {
89         $$ = $2;
90     }
91 |
92     expr '=' expr
93     {
94         $$ = assign($1, $3);
95     }
96 |
97     expr COMPADD expr
98     {
99         $$ = assign($1, add($1, $3));
100    }
101 |
102     expr COMPSUB expr
103     {
104         $$ = assign($1, sub($1, $3));
105     }
106 |

```

```

107     expr COMPMUL expr
108     {
109         $$ = assign($1, mul($1, $3));
110     }
111 |
112     expr COMPDIV expr
113     {
114         $$ = assign($1, div($1, $3));
115     }
116 |
117     expr COMPLSH expr
118     {
119         $$ = assign($1, lsh($1, toInt($3)));
120     }
121 |
122     expr COMPRSH expr
123     {
124         $$ = assign($1, rsh($1, toInt($3)));
125     }
126 |
127     expr '%' expr
128     {
129         $$ = mod($1, toInt($3));
130     }
131 |
132     expr LSH expr
133     {
134         $$ = lsh($1, toInt($3));
135     }
136 |
137     expr RSH expr
138     {
139         $$ = lsh($1, toInt($3));
140     }
141 ;

```

Listing 8: parser.h

```

1  ...
2      int toInt(RuleValue const &rv);
3  ...

```

Listing 9: toint.cc

```
1 #include "parser.ih"
2 #include <cmath>
3
4 int Parser::toInt(RuleValue const &rv)
5 {
6     return std::round(valueOf(rv));
7 }
```

## Exercise 38, calculator

We were tasked with making a calculator that behaves differently than a user would expect.

### Parser

Listing 10: grammar

```
1 %class-name Parser
2
3 %filenames parser
4 %parsefun-source parse.cc
5
6 %baseclass-preinclude rulevalue.h
7 %stype RuleValue
8
9 %scanner ../scanner/scanner.h
10
11 %token  NUMBER
12         IDENT
13
14 %right  '=' COMPADD COMPSUB COMPDIV
15         COMPMUL COMPLSH COMPRSH
16 %right  uMinus
17 %left   '+'
18 %right  '*'
19 %left   '/'
20 %right  '%' LSH RSH
21 %right  '-'
22
```

```

23 // %debug
24
25 %%
26
27 lines:
28     lines line
29 |
30     line
31 ;
32
33
34 line:
35     expr '\n'
36     {
37         display($1);
38     }
39 |
40     error '\n'
41     {
42         prompt();
43     }
44 |
45     '\n'
46     {
47         quit();
48     }
49 ;
50
51
52 expr:
53     NUMBER
54     {
55         $$ = value();
56     }
57 |
58     IDENT
59     {
60         $$ = variable();
61     }
62 |

```

```

63     expr '-' expr
64     {
65     $$ = add($1, $3);
66     }
67 |
68     expr '*' expr
69     {
70     $$ = div($1, $3);
71     }
72 |
73     expr '/' expr
74     {
75     $$ = sub($1, $3);
76     }
77 |
78     expr '+' expr
79     {
80     $$ = mul($1, $3);
81     }
82 |
83     '-' expr %prec uMinus
84     {
85     $$ = negate($2);
86     }
87 |
88     '(' expr ')'
89     {
90     //
91     }
92 |
93     expr '=' expr
94     {
95     $$ = assign($1, $3);
96     }
97 |
98     expr COMPADD expr
99     {
100     $$ = assign($1, add($1, $3));
101     }
102 |

```

```

103     expr COMPSUB expr
104     {
105         $$ = assign($1, sub($1, $3));
106     }
107 |
108     expr COMPMUL expr
109     {
110         $$ = assign($1, mul($1, $3));
111     }
112 |
113     expr COMPDIV expr
114     {
115         $$ = assign($1, div($1, $3));
116     }
117 |
118     expr COMPLSH expr
119     {
120         $$ = assign($1, lsh($1, toInt($3)));
121     }
122 |
123     expr COMPRSH expr
124     {
125         $$ = assign($1, rsh($1, toInt($3)));
126     }
127 |
128     expr '%' expr
129     {
130         $$ = mod($1, toInt($3));
131     }
132 |
133     expr LSH expr
134     {
135         $$ = lsh($1, toInt($3));
136     }
137 |
138     expr RSH expr
139     {
140         $$ = lsh($1, toInt($3));
141     }
142 ;

```

Listing 11: parser.h

```
1  ...
2  #include <vector>
3  #include <unordered_map>
4  #include <string>
5  ...
6
7  ...
8      std::vector<double> d_value;
9      std::unordered_map
10         <std::string, unsigned> d_symtab;
11  ...
12
13  ...
14      // added functions for the calculator:
15      void display(RuleValue &e);
16      void done();
17      void prompt();
18
19      RuleValue &add(RuleValue &lvalue,
20                    RuleValue &rvalue);
21      RuleValue &assign(RuleValue &lvalue,
22                       RuleValue &rvalue);
23      RuleValue &div(RuleValue &lvalue,
24                   RuleValue &rvalue);
25      RuleValue &lsh(RuleValue &lvalue,
26                   RuleValue &rvalue);
27      RuleValue &mul(RuleValue &lvalue,
28                   RuleValue &rvalue);
29      RuleValue &negate(RuleValue &e);
30      RuleValue &rsh(RuleValue &lvalue,
31                   RuleValue &rvalue);
32      RuleValue &sub(RuleValue &lvalue,
33                   RuleValue &rvalue);
34      RuleValue value();
35      RuleValue variable();
36
37      double valueOf(RuleValue const &e);
38      int toInt(RuleValue rv);
```

```

39
40         void quit();
41     };
42
43
44 #endif

```

## Exercise 39, functions

This was actually made before 36-38.

### Code listings

Listing 12: grammar

```

1  %class-name Parser
2
3  %filenames parser
4  %parsefun-source parse.cc
5
6  %baseclass-preinclude rulevalue.h
7  %stype RuleValue
8
9  %scanner ../scanner/scanner.h
10
11 %token NUMBER IDENT
12
13 %right 'e' ln sin asin sqrt deg grad rad
14 %left '^'
15
16 // %debug
17
18 %%
19
20 lines:
21     lines line
22 |
23     line
24 ;
25
26

```



```

27 line:
28     expr '\n'
29     {
30         display($1);
31     }
32 |
33     error '\n'
34     {
35         prompt();
36     }
37 |
38     '\n'
39     {
40         prompt();
41     }
42 ;
43
44 expr:
45     NUMBER
46     {
47         $$ = value();
48     }
49 |
50     IDENT
51     {
52         $$ = variable();
53     }
54 |
55     'e' '^' expr
56     {
57         $$ = exp($3);
58     }
59 |
60     ln expr
61     {
62         $$ = ln($2);
63     }
64 |
65     sin expr
66     {

```

```

67     $$ = sin($2);
68 }
69 |
70     asin expr
71 {
72     $$ = asin($2);
73 }
74 |
75     sqrt expr
76 {
77     $$ = sqrt($2);
78 }
79 |
80     '||' expr '||'
81 {
82     $$ = abs($2);
83 }
84 |
85     deg expr
86 {
87     $$ = deg($2);
88 }
89 |
90     rad expr
91 {
92     $$ = rad($2);
93 }
94 |
95     grad expr
96 {
97     $$ = grad($2);
98 }
99 ;

```

Listing 13: parser.h

```

1  ...
2  // arithmetic functions:
3      void display(double &value);
4      void done();
5      void prompt();

```

```

6     RuleValue &exp(RuleValue &value);
7     RuleValue &ln(RuleValue &value);
8     RuleValue &sin(RuleValue &value);
9     RuleValue &asin(RuleValue &value);
10    RuleValue &sqrt(RuleValue &value);
11    RuleValue &abs(RuleValue &value);
12
13    RuleValue &deg(RuleValue &value);
14    RuleValue &grad(RuleValue &value);
15    RuleValue &rad(RuleValue &deg);
16    RuleValue &rad(RuleValue &grad);
17
18    double const pi = 3.14159;
19    double const e = 2.71828;
20 };
21
22 #endif

```

### Implementations

Listing 14: abs.cc

```

1 #include "parser.ih"
2
3 RuleValue &Parser::abs(RuleValue &value)
4 {
5     return RuleValue(abs(valueOf(value)));
6 }

```

Listing 15: asin.cc

```

1 #include "parser.ih"
2
3 RuleValue &Parser::asin(RuleValue &value)
4 {
5     if (valueOf(value) <= 1 || valueOf(value) >= -1)
6         return RuleValue(asin(valueOf(value)));
7     else
8         error("Value (radians) out of interval -1 < value
9             < 1");
9 }

```

Listing 16: deg.cc

```
1 #include "parser.ih"
2
3 RuleValue &Parser::deg(RuleValue &value)
4 {
5     return RuleValue(2 * Parser::pi * valueOf(value) /
6         360);
7 }
```

Listing 17: done.cc

```
1 #include "parser.ih"
2
3 void Parser::done()
4 {
5     cout << "Bye\n";
6     ACCEPT();
7 }
```

Listing 18: exp.cc

```
1 #include "parser.ih"
2
3 RuleValue &Parser::exp(RuleValue &value)
4 {
5     return RuleValue(Parser::e ^ valueOf(value));
6 }
```

Listing 19: grad.cc

```
1 #include "parser.ih"
2
3 RuleValue &Parser::grad(RuleValue &value)
4 {
5     return RuleValue(2 * Parser::pi * valueOf(value) /
6         400);
7 }
```

Listing 20: ln.cc

```
1 #include "parser.ih"
2
3 RuleValue &Parser::log(RuleValue &value)
```

```

4 {
5     if (valueOf(value) >= 0)
6         return RuleValue(log(valueOf(value)));
7     else
8         error("Value may not be negative");
9 }

```

Listing 21: raddeg.cc

```

1 #include "parser.ih"
2
3 RuleValue &Parser::rad(RuleValue &deg)
4 {
5     return RuleValue((360 * deg) / (2 * Parser::pi));
6 }

```

Listing 22: radgrad.cc

```

1 #include "parser.ih"
2
3 RuleValue &Parser::rad(RuleValue &grad)
4 {
5     return RuleValue((400 * grad) / (2 * pi));
6 }

```

Listing 23: sin.cc

```

1 #include "parser.ih"
2
3 RuleValue &Parser::sin(RuleValue &value)
4 {
5     return RuleValue(sin(valueOf(value)));
6 }

```

Listing 24: sqrt.cc

```

1 #include "parser.ih"
2
3 RuleValue &Parser::sqrt(RuleValue &value)
4 {
5     if (valueOf(value) >= 0)
6         return RuleValue(sqrt(valueOf(value)));
7     else

```

```

8     error("Value may not be negative");
9 }

```

## Exercise 40, polymorphic value type class

We attempted to make a polymorphic value type class.

### Code listings

#### Scanner

Listing 25: lexer.ll

```

1 NUM    [0-9]
2
3 %%
4
5 {NUM}+      return ParserBase::Tokens__::INT;
6 {NUM}*"."{NUM}+ return ParserBase::Tokens__::DOUBLE;
7 QUIT        return ParserBase::Tokens__::QUIT;
8 [\t ]       // ignore
9 \n          return matched()[0];
10 .+         return ParserBase::Tokens__::STRING;

```

#### Parser

Listing 26: grammar.gr

```

1 %token INT STRING DOUBLE QUIT
2
3 %polymorphic INT: int; STRING: std::string; DOUBLE:
4     double;
5
6 %scanner Scanner.h
7
8 %type <INT> intline
9 %type <STRING> stringline
10 %type <DOUBLE> doubleline
11
12 %%
13 lines:

```

```

14  lines '\n' line
15  |
16  line
17  ;
18
19  line:
20      intline
21      {
22          $$ = $1;
23      }
24  |
25      stringline
26      {
27          $$ = $1;
28      }
29  |
30      doubleline
31      {
32          $$ = $1;
33      }
34  |
35      QUIT
36      {
37          quit();
38      }
39  ;
40
41  intline:
42      INT
43      {
44          $$ = getInt();
45          showInt($$);
46      }
47  ;
48
49  stringline:
50      STRING
51      {
52          $$ = getString();
53          showString($$);

```

```

54     }
55 ;
56
57 doubleline:
58     DOUBLE
59     {
60         $$ = getDouble();
61         showDouble($$);
62     }
63 ;

```

Listing 27: Parser.ih

```

1  ...
2  #include <cstdlib>
3  ...

```

Listing 28: Parser.h

```

1  ...
2  // my own functions:
3  int getInt();
4  std::string getString();
5  double getDouble();
6
7  void showInt(int &someInt);
8  void showString(std::string &someString);
9  void showDouble(double &someDouble);
10 void quit();
11 ...

```

Listing 29: getdouble.cc

```

1  #include "Parser.ih"
2
3  double Parser::getDouble()
4  {
5      return atof(d_scanner.matched().c_str());
6  }

```

Listing 30: getint.cc

```

1  #include "Parser.ih"

```



```

2
3 int Parser::getInt()
4 {
5     return atol(d_scanner.matched().c_str());
6 }

```

Listing 31: getstring.cc

```

1 #include "Parser.ih"
2
3 string Parser::getString()
4 {
5     return d_scanner.matched();
6 }

```

Listing 32: quit.cc

```

1 #include "Parser.ih"
2
3 void Parser::quit()
4 {
5     ACCEPT();
6 }

```

Listing 33: showdouble.cc

```

1 #include "Parser.ih"
2
3 void Parser::showDouble(double &someDouble)
4 {
5     cout << someDouble << '\n';
6 }

```

Listing 34: showint.cc

```

1 #include "Parser.ih"
2
3 void Parser::showInt(int &someInt)
4 {
5     cout << someInt << '\n';
6 }

```

Listing 35: showstring.cc

```
1 #include "Parser.ih"
2
3 void Parser::showString(string &someString)
4 {
5     cout << someString << '\n';
6 }
```