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]+                          // ignore
11
12 {digits}                        |
13 {digits}"."{optdigits}{exp}?    |
14 {optdigits}"."{digits}{exp}?    return Parser::NUMBER;
15
16 [[:alpha:]_][[:alnum:]_]*       return Parser::IDENT;
17
```

```
"+="                return Parser::COMPADD;
"-="                return Parser::COMPSUB;
"*="                return Parser::COMPMUL;
"/="                return Parser::COMPDIV;
"<<="               return Parser::COMPLSH;
">>="               return Parser::COMPRSH;
"<<"                return Parser::LSH;
">>"                return Parser::RSH;

\n|.                return matched()[0];
```

Parser

Listing 7: grammar

```
%class-name Parser

%filenames parser
%parsefun-source parse.cc

%baseclass-preinclude rulevalue.h
%stype RuleValue

%scanner ../scanner/scanner.h

%token  NUMBER
        IDENT

%right  '=' COMPADD COMPSUB COMPMUL
        COMPDIV COMPLSH COMPRSH
%left   '+' '-'
%left   '*' '/'
%right  '%' LSH RSH
%right  uMinus

// %debug

%%

lines:
    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: lexer

```
1  digits    [0-9]+
2  ident0    [a-zA-Z_]
3  identx    [a-zA-Z_0-9]
4
5  %%
6
7  [ \t]+           // ignore
8
9  {digits}                    |
10 {digits}?.{digits}E-?{digits} |
11 {digits}?.{digits}         return Parser::NUMBER;
12
13 e|pi           return Parser::CONST;
14 ln             return Parser::LN;
15 sin            return Parser::SIN;
16 asin           return Parser::ASIN;
17 sqrt           return Parser::SQRT;
18 deg            return Parser::DEG;
19 grad           return Parser::GRAD;
20 rad            return Parser::RAD;
21
22 {ident0}{identx}       return Parser::IDENT;
23
24 // Includes all operators and parentheses
25 \n|.                    return matched()[0];
```

Listing 13: grammar

```
%class-name Parser

%filenames parser
%parsefun-source parse.cc

%baseclass-preinclude rulevalue.h
%stype RuleValue

%scanner ../scanner/scanner.h

%token NUMBER CONST IDENT DEG GRAD RAD

%right EXP LN SIN ASIN SQRT ABS

// %debug

%%

lines:
    lines line
|
    line
;


line:
    expr '\n'
    {
        display($1);
    }
|
    DEG '\n'
    {
        deg();
    }
|
    GRAD '\n'
    {
```

```
39          grad();
40      }
41  |
42      RAD '\n'
43      {
44          rad();
45      }
46  |
47      error '\n'
48      {
49          prompt();
50      }
51  |
52      '\n'
53      {
54          prompt();
55      }
56  ;
57
58  expr:
59    CONST
60    {
61      $$ = constant();
62    }
63  |
64      NUMBER
65      {
66          $$ = value();
67      }
68  |
69      IDENT
70      {
71          $$ = variable();
72      }
73  |
74    EXP expr
75    {
76      $$ = exp($2);
77    }
78  |
```

```
79    LN expr
80    {
81      $$ = ln($2);
82    }
83  |
84    SIN expr
85    {
86      $$ = sin($2);
87    }
88  |
89    ASIN expr
90    {
91      $$ = asin($2);
92    }
93  |
94    SQRT expr
95    {
96      $$ = sqrt($2);
97    }
98  |
99    '|' expr '|'
100   {
101     $$ = abs($2);
102   }
103 |
104   ABS expr
105   {
106     $$ = abs($2);
107   }
108 ;
```

Listing 14: parser.h

```
1  ...
2  enum AngleMode
3  {
4    DEGREES,
5    RADIANS,
6    GRAD
7  };
8
```

```
 9  #undef Parser
10  class Parser: public ParserBase
11  {
12      // $insert scannerobject
13      Scanner d_scanner;
14
15      AngleMode d_angleMode = RADIANS;
16  ...
17
18  ...
19    // arithmetic functions:
20      double angleTransform();
21
22      void display(RuleValue &value);
23          void done();
24          void prompt();
25      RuleValue exp(RuleValue &value);
26      RuleValue ln(RuleValue &value);
27      RuleValue sin(RuleValue &value);
28      RuleValue asin(RuleValue &value);
29      RuleValue sqrt(RuleValue &value);
30      RuleValue abs(RuleValue &value);
31
32      void deg();
33      void grad();
34      void rad();
35
36      double const pi = 3.14159;
37      double const e = 2.71828;
38      RuleValue constant();
39  };
40
41  #endif
```

Implementations

Listing 15: abs.cc

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

```
4  {
5    return RuleValue(abs(valueOf(value)));
6  }
```

Listing 16: angletransform.cc

```
1  #include "parser.ih"
2
3  double &Parser::angleTransform()
4  {
5    switch(d_angleMode):
6    {
7      case(DEGREES):
8        return Parser::pi / 180.0;
9      case(GRAD):
10       return Parser::pi / 200.0;
11     default:
12       return 1;
13   }
14 }
```

Listing 17: asin.cc

```
1  #include "parser.ih"
2
3  RuleValue &Parser::asin(RuleValue &value)
4  {
5    double transform = angleTransform();
6    if (valueOf(value) <= 1 || valueOf(value) >= -1)
7      return RuleValue(asin(valueOf(value)) / transform)
       ;
8    else
9      error("Value (radians) out of interval -1 < value
       < 1");
10 }
```

Listing 18: constant.cc

```
1  #include "parser.ih"
2
3  double Parser::constant()
4  {
```

```
5     string constant(parser.matched());
6     if (constant.compare("e") == 0)
7       return RuleValue(Parser::e);
8     if (constant.compare("pi") == 0)
9       return RuleValue(Parser::pi);
10    return 0;
11  }
```

Listing 19: deg.cc

```
1  #include "parser.ih"
2
3  void Parser::deg()
4  {
5    d_angleMode = DEGREES;
6  }
```

Listing 20: done.cc

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

Listing 21: exp.cc

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

Listing 22: grad.cc

```
1  #include "parser.ih"
2
3  void Parser::grad(double &value)
4  {
5    d_angleMode = GRAD;
6  }
```

Listing 23: 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 24: rad.cc

```
1  #include "parser.ih"
2
3  void Parser::rad(double &deg)
4  {
5    d_angleMode = RADIANS;
6  }
```

Listing 25: sin.cc

```
1  #include "parser.ih"
2
3  RuleValue &Parser::sin(RuleValue &value)
4  {
5    double transform = angleTransform();
6    return RuleValue(sin(valueOf(transform * value)));
7  }
```

Listing 26: 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 27: lexer.ll

```
NUM   [0-9]

%%

{NUM}+          return ParserBase::Tokens__::INT;
{NUM}*"."{NUM}+ return ParserBase::Tokens__::DOUBLE;
QUIT            return ParserBase::Tokens__::QUIT;
[\t ]           // ignore
\n              return matched()[0];
.+              return ParserBase::Tokens__::STRING;
```

Parser

Listing 28: grammar.gr

```
%token INT STRING DOUBLE QUIT

%polymorphic INT: int; STRING: std::string; DOUBLE:
   double;

%scanner Scanner.h

%type <INT> intline
%type <STRING> stringline
%type <DOUBLE> doubleline

%%

lines:
  lines '\n' line
|
  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 29: Parser.ih

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

Listing 30: 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 31: getdouble.cc

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

Listing 32: getint.cc

```
1 #include "Parser.ih"
2
3 int Parser::getInt()
4 {
```

```
5     return atol(d_scanner.matched().c_str());
6   }
```

Listing 33: getstring.cc

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

Listing 34: quit.cc

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

Listing 35: showdouble.cc

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

Listing 36: showint.cc

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

Listing 37: showstring.cc

```
1   #include "Parser.ih"
2
```

```
3  void Parser::showString(string &someString)
4  {
5    cout << someString << '\n';
6  }
```