Programming in C/C++
Exercises set six: parsers II

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

March 2, 2017

Exercise 37, substantial grammar extension

All these operators.

Code listings

Scanner

Listing 1: 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::LSH;
19 ">>"                              return Parser::RSH;
20
21 \n|.                              return matched()[0];
```

Parser

Listing 2: 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 39, functions

This was actually made before 36-38.

Code listings

Listing 3: 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  ;
```

parser.h snippet

```
1   // arithmetic functions:
2
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;
```

Implementations

Listing 4: abs.cc

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

Listing 5: asin.cc

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

```
4  {
5    if (value.d_number <= 1 || value.d_number >= -1)
6      return RuleValue(asin(value.d_number));
7    else
8      error("Value (radians) out of interval -1 < value
       < 1");
9  }
```

Listing 6: deg.cc

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

Listing 7: done.cc

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

Listing 8: exp.cc

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

Listing 9: grad.cc

```
1  #include "parser.ih"
2
3  RuleValue &Parser::grad(RuleValue &value)
4  {
```

```
5   return RuleValue(2 * Parser::pi * value.d_number /
     400);
6 }
```

Listing 10: ln.cc

```
1 #include "parser.ih"
2
3 RuleValue &Parser::log(RuleValue &value)
4 {
5   if (value.d_number >= 0)
6     return RuleValue(log(value.d_number));
7   else
8     error("Value may not be negative");
9 }
```

Listing 11: raddeg.cc

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

Listing 12: radgrad.cc

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

Listing 13: sin.cc

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

Listing 14: sqrt.cc

```
#include "parser.ih"

RuleValue &Parser::sqrt(RuleValue &value)
{
  if (value.d_number >= 0)
    return RuleValue(sqrt(value.d_number));
  else
    error("Value may not be negative");
}
```

Exercise 40, polymorphic value type class

We attempted to make a polymorphic value type class.

Code listings

Listing 15: grammar.gr

```
%token INT STRING DOUBLE QUIT

%baseclass-preinclude polytype.h
%stype std::shared_ptr<BaseType>

%scanner Scanner.h

%%

lines:
  lines '\n' line
|
  line
;

line:
  INT
  {
    $$ = getInt();
    showInt($$);
  }
```

```
22  |
23    STRING
24    {
25      $$ = getString();
26      showString($0);
27    }
28  |
29    DOUBLE
30    {
31      $$ = getDouble();
32      showDouble($0);
33    }
34  |
35    QUIT
36    {
37      quit();
38    }
39  ;
```

Listing 16: Parser.ih

```
1  // Generated by Bisonc++ V4.13.01 on Mon, 27 Feb 2017
       15:39:49 +0100
2
3       // Include this file in the sources of the class
       Parser.
4
5  // $insert class.h
6  #include "Parser.h"
7
8  #include <cstdlib>
9
10 // $insert STYPE
11 typedef std::shared_ptr<BaseType> STYPE__;
12
13 inline void Parser::error(char const *msg)
14 {
15      std::cerr << msg << '\n';
16 }
17
18 // $insert lex
```

```
19 inline int Parser::lex()
20 {
21     return d_scanner.lex();
22 }
23
24 inline void Parser::print()
25 {
26     print__();           // displays tokens if --print
       was specified
27 }
28
29 inline void Parser::exceptionHandler__(std::exception
     const &exc)
30 {
31     throw;               // re-implement to handle
     exceptions thrown by actions
32 }
33
34
35     // Add here includes that are only required for
     the compilation
36     // of Parser's sources.
37
38
39
40     // UN-comment the next using-declaration if you
     want to use
41     // int Parser's sources symbols from the namespace
       std without
42     // specifying std::
43
44 //using namespace std;
```

Listing 17: Parser.h

```
1 // Generated by Bisonc++ V4.05.00 on Thu, 02 Mar 2017
     12:10:57 +0100
2
3 #ifndef Parser_h_included
4 #define Parser_h_included
5
```

```
6  // $insert baseclass
7  #include "Parserbase.h"
8  // $insert scanner.h
9  #include "Scanner.h"
10
11 #undef Parser
12 class Parser: public ParserBase
13 {
14     // $insert scannerobject
15     Scanner d_scanner;
16
17     public:
18         int parse();
19
20     private:
21         void error(char const *msg);     // called on (
       syntax) errors
22         int lex();                       // returns the
        next token from the
23                                          // lexical
       scanner.
24         void print();                    // use, e.g.,
       d_token, d_loc
25
26     // support functions for parse():
27         void executeAction(int ruleNr);
28         void errorRecovery();
29         int lookup(bool recovery);
30         void nextToken();
31         void print__();
32         void exceptionHandler__(std::exception const &
       exc);
33
34     // my own functions:
35     STYPE__ getInt();
36     STYPE__ getString();
37     STYPE__ getDouble();
38
39     void showInt(STYPE__ &ptr);
40     void showString(STYPE__ &ptr);
```

```
41      void showDouble(STYPE__ &ptr);
42      void quit();
43  };
44
45
46  #endif
```

Listing 18: getdouble.cc

```
1  #include "Parser.ih"
2
3  STYPE__ Parser::getDouble()
4  {
5    double ret = atof(d_scanner.matched().c_str());
6
7    return std::move(STYPE__{new DoubleType(ret)});
8  }
```

Listing 19: getint.cc

```
1  #include "Parser.ih"
2
3  STYPE__ Parser::getInt()
4  {
5    int ret = atol(d_scanner.matched().c_str());
6
7    return std::move(STYPE__{new IntType(ret)});
8  }
```

Listing 20: getstring.cc

```
1  #include "Parser.ih"
2
3  STYPE__ Parser::getString()
4  {
5    StringType *ptr = new StringType(d_scanner.matched()
       );
6    return std::move(STYPE__{ptr});
7  }
```

Listing 21: quit.cc

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

```
2
3  void Parser::quit()
4  {
5    ACCEPT();
6  }
```

Listing 22: showdouble.cc

```
1  #include "Parser.ih"
2
3  void Parser::showString(STYLE__ &ptr)
4  {
5    ptr->print(cout);
6  }
```

Listing 23: showint.cc

```
1  #include "Parser.ih"
2
3  void Parser::showInt(STYLE__ &ptr)
4  {
5    ptr->print(cout);
6  }
```

Listing 24: showstring.cc

```
1  #include "Parser.ih"
2
3  void Parser::showString(STYLE__ &ptr)
4  {
5    ptr->print(cout);
6  }
```

Polymorphic type

Listing 25: polytype.ih

```
1  #include "polytype.h"
2
3  using namespace std;
```

Listing 26: polytype.h

```
1  #ifndef POLYTYPE_H
```

```
2  #define POLYTYPE_H
3
4  #include <iostream>
5  #include <memory>
6
7  struct BaseType
8  {
9    virtual std::ostream &print(std::ostream &out) = 0;
10 };
11
12 class IntType : public BaseType
13 {
14   int d_value = 0;
15
16   public:
17     IntType(int value);
18
19     std::ostream &print(std::ostream &out) override;
20 };
21
22 class StringType : public BaseType
23 {
24   std::string d_value;
25
26   public:
27     StringType(std::string value);
28
29     std::ostream &print(std::ostream &out) override;
30 };
31
32 class DoubleType : public BaseType
33 {
34   double d_value = 0;
35
36   public:
37     DoubleType(double value);
38
39     std::ostream &print(std::ostream &out) override;
40 };
41
```

```
42  #endif
```

Listing 27: inttype_constr.cc

```
1  #include "polytype.ih"
2
3  IntType::IntType(int value)
4  :
5    d_value(value)
6  {}
```

Listing 28: inttype_print.cc

```
1  #include "polytype.ih"
2
3  ostream &IntType::print(ostream &out)
4  {
5    return out << d_value << '\n';
6  }
```