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

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

March 2, 2017

Exercise 40, polymorphic value type class

We attempted to make a polymorphic value type class.

Code listings

Listing 1: grammar.gr

```
1  %token INT STRING DOUBLE QUIT
2
3  %baseclass-preinclude polytype.h
4  %stype std::shared_ptr<BaseType>
5
6  %scanner Scanner.h
7
8  %%
9
10 lines:
11   lines '\n' line
12 |
13   line
14 ;
15
16 line:
17   INT
18   {
```

```
19      $$ = getInt();
20      showInt($$);
21    }
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 2: 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 3: 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 4: 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 5: 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 6: 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 7: quit.cc

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

Listing 8: showdouble.cc

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

Listing 9: showint.cc

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

Listing 10: showstring.cc

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

Polymorphic type

Listing 11: polytype.ih

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

Listing 12: 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 13: inttype_constr.cc

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

Listing 14: inttype_print.cc

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