

Programming in C/C++

Exercises set four: lexical scanners

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

February 22, 2017

Exercise 23, Embedded patterns

We were tasked to construct a program that uses a scanner for printing all the sorted words found in a piece of text.

Code listings

Listing 1: main.ih

```
1 #include "Scanner.h"
2 #include <set>
3 #include <fstream>
4
5 using namespace std;
```

Listing 2: main.cc

```
1 #include "main.ih"
2
3 int main(int argc, char **argv)
4 {
5     set<string> myset;
6     if (argc == 1)
7     {
8         Scanner scanner(cin, cout);
9
10        while (scanner.lex())
```

```

11         {
12             myset.insert(scanner.matched());
13         }
14     }
15     else
16     {
17         for (int idx = 1; idx != argc; ++idx)
18         {
19             ifstream input;
20             input.open(argv[idx]);
21             Scanner scanner(input, std::cout);
22
23             //read
24             while (scanner.lex())
25             {
26                 myset.insert(scanner.matched());
27             }
28         }
29     }
30     //print
31     for(string const &idx : myset)
32     {
33         cout << idx << '\n';
34     }
35 }

```

Listing 3: lexer.ll

```

1 NONBLANK  [a-zA-Z]
2
3 %%
4
5 {NONBLANK}+ return WORD;
6
7 .|\n      // ignore

```

Listing 4: Scanner.h

```

1 // Generated by Flexc++ V1.08.00 on Thu,
2 //16 Feb 2017 13:53:05 +0100
3

```

```

4  #ifndef Scanner_H_INCLUDED_
5  #define Scanner_H_INCLUDED_
6
7  enum { WORD = 256 };
8
9  // $insert baseclass_h
10 #include "Scannerbase.h"
11
12
13 // $insert classHead
14 class Scanner: public ScannerBase
15 {
16     public:
17         explicit Scanner(std::istream &in = std::cin,
18                         std::ostream &out = std::cout)
19     ;
20
21     Scanner(std::string const &infile,
22            std::string const &outfile);
23
24     // $insert lexFunctionDecl
25     int lex();
26
27     private:
28         int lex__();
29         int executeAction__(size_t ruleNr);
30
31         void print();
32         void preCode();           // re-implement this
33                                   // function for code
34                                   // that must be
35                                   // executed before the
36                                   // patternmatching starts
37
38         void postCode(PostEnum__ type);
39         // re-implement this function for code that
40         // must be exec'ed after the rules's actions.
41 };
42 // $insert scannerConstructors

```

```

43 inline Scanner::Scanner(std::istream &in,
44                          std::ostream &out)
45 :
46     ScannerBase(in, out)
47 {}
48
49 inline Scanner::Scanner(std::string const &infile,
50                          std::string const &outfile)
51 :
52     ScannerBase(infile, outfile)
53 {}
54
55 // $insert inlineLexFunction
56 inline int Scanner::lex()
57 {
58     return lex__();
59 }
60
61 inline void Scanner::preCode()
62 {
63     // optionally replace by your own code
64 }
65
66 inline void Scanner::postCode(PostEnum__ type)
67 {
68     // optionally replace by your own code
69 }
70
71 inline void Scanner::print()
72 {
73     print__();
74 }
75
76
77 #endif // Scanner_H_INCLUDED_

```

Exercise 24, Non-greedy matching

We made a lexical scanners that performs non-greedy matching.

Code listings

Listing 5: main.ih

```
1 #include "Scanner.ih"
2 #include "Scannerbase.h"
```

Listing 6: main.cc

```
1 #include "main.ih"
2
3 int main()
4 try
5 {
6     Scanner scanner;
7
8     while (scanner.lex())
9         ;
10 }
11 catch (...)
12 {
13     return 1;
14 }
```

Listing 7: lexer.ll

```
1 %x house
2 %x post
3
4 NONBLANK      [a-zA-Z\ -]
5
6 %%
7 house/(hold|boat)  {
8     begin(StartCondition__::house);
9     std::cout << matched() << '\n';
10    return WORD;
11    }
12
13 house          {
14     begin(StartCondition__::house);
15     more();
16 }
```

```

17
18 <house>
19 {
20     hold|boat    {
21         begin(StartCondition__::INITIAL);
22         std::cout << matched() << '\n';
23         return WORD;
24     }
25
26     {NONBLANK}*    {
27         begin(StartCondition__::INITIAL);
28         std::cout << matched() << '\n';
29         return WORD;
30     }
31 }
32
33 {NONBLANK}*house    {
34     begin(StartCondition__::post);
35     more();
36 }
37
38 <post>
39 {
40     {NONBLANK}*    {
41         begin(StartCondition__::INITIAL);
42         std::cout << matched() << '\n';
43         return WORD;
44     }
45 }
46
47 \n                // ignore

```

Listing 8: Scanner.h

```

1 // Generated by Flexc++ V2.03.04 on Thu,
2 // 16 Feb 2017 12:00:34 +0100
3
4 #ifndef Scanner_H_INCLUDED_
5 #define Scanner_H_INCLUDED_
6
7 // $insert baseclass_h

```

```

8  #include "Scannerbase.h"
9
10 enum Tokens
11 {
12     WORD = 1
13 };
14
15 // $insert classHead
16 class Scanner: public ScannerBase
17 {
18     public:
19         explicit Scanner(std::istream &in = std::cin,
20                         std::ostream &out = std::cout);
21
22         Scanner(std::string const &infile,
23               std::string const &outfile);
24
25         // $insert lexFunctionDecl
26         int lex();
27
28     private:
29         int lex__();
30         int executeAction__(size_t ruleNr);
31
32         void print();
33         void preCode();      // re-implement this
34                             // function for code
35                             // that must be
36                             // executed before the
37                             // patternmatching starts
38
39         void postCode(PostEnum__ type);
40         // re-implement this function for code that
41         // must be exec'ed after the rules's actions.
42 };
43
44 // $insert scannerConstructors
45 inline Scanner::Scanner(std::istream &in, std::ostream
46                        &out)
47 :

```

```

47     ScannerBase(in, out)
48 {}
49
50 inline Scanner::Scanner(std::string const &infile, std
    ::string const &outfile)
51 :
52     ScannerBase(infile, outfile)
53 {}
54
55 // $insert inlineLexFunction
56 inline int Scanner::lex()
57 {
58     return lex__();
59 }
60
61 inline void Scanner::preCode()
62 {
63     // optionally replace by your own code
64 }
65
66 inline void Scanner::postCode(PostEnum__ type)
67 {
68     // optionally replace by your own code
69 }
70
71 inline void Scanner::print()
72 {
73     print__();
74 }
75
76
77 #endif // Scanner_H_INCLUDED_

```

Exercise 26

See exercise 28.

Exercise 27, tokens

Why are there so many operators?

Lexer

Listing 9: lexer.ll

```
1  %x cComment
2  %x string
3  %x header
4
5  SPACE      [ \t\n]
6  ALPHA      [a-zA-Z]
7  NUM        [0-9]
8  ALPHANUM    {ALPHA} | {NUM}
9
10 // Anything but ALPHANUM tab and newline
11 SYMBOL      [ ^{ALPHANUM} \t\n]
12
13 %%
14
15 ({ALPHA} | "_" ) ({ALPHANUM} | "_") *  return IDENT;
16 {NUM} +      return INT;
17 ({NUM} + "." {NUM} +) (E-? {NUM}) ?    return DOUBLE;
18
19 // Singular operators taken care off at the end
20 "<<"          return SHL_OP;
21 ">>"          return SHR_OP;
22 "&&"          return AND;
23 "||"          return OR;
24 "<="          return LESS_EQUALS;
25 ">="          return GREATER_EQUALS;
26 "=="          return EQUALS;
27 "!="          return NOT_EQUALS;
28 "+=" | "-=" | "<="          return ASSIGNMENT_OP;
29 ">>=" | "*=" | "/="        return ASSIGNMENT_OP;
30 "->"          return POINTER;
31
32 "' " ("\" ) ? ( {ALPHANUM} | {SYMBOL} | "\" ) + "' "
33 {
34     return CONSTANT;
35 }
36
```

```

37  \\ EOL comment is ignored (but not the newline)
38  "//".*\n                redo(1);
39
40  "/*"                    {
41                          begin(StartCondition__::cComment);
42                          more();
43                          }
44  <cComment>"*/"          {
45                          begin(StartCondition__::INITIAL);
46                          // ignore
47                          }
48  <cComment>{SPACE}|.      more();
49
50  R?"                    {
51                          begin(StartCondition__::string);
52                          more();
53                          }
54  <string>"              {
55                          begin(StartCondition__::INITIAL);
56                          return STRING;
57                          }
58
59  // Needs stitching as seen in exercise 26/28
60  <string>\"{SPACE}*\"      return STRING_SEG;
61  <string>\n              more();
62  <string>.                more();
63
64  "#include <"|"#include \"\"
65  {
66      begin(StartCondition__::header);
67      more();
68  }
69
70  <header>\"|\">          {
71                          begin(StartCondition__::INITIAL);
72                          return HEADER;
73                          }
74  <header>.                more();
75
76  \n                      return ENDLINE;

```

```

77 [ \t]           // ignore
78 .               return matched()[0];

```

Exercise 28, scanner that scans text

We were asked to design a scanner that scans a piece of text.

Code listings

Listing 10: main.h

```

1  #ifndef MAIN_H
2  #define MAIN_H
3
4  #include <vector>
5  #include <algorithm>
6  #include <unistd.h>
7  #include <fcntl.h>
8  #include <sys/types.h>
9  #include <sys/stat.h>
10 #include <fstream>
11
12 #include "Scanner.h"
13
14 std::string dequote(std::string const &str);
15
16 void makeRaw(std::string &str);
17 char escapeChar(char rawChar);
18 bool isEscape(char rawChar);
19 void deEscape(std::string &str);
20
21 class Writer
22 {
23     std::vector<std::string> d_literals;
24
25     std::string d_path;
26     std::string d_tempPath;
27     std::string d_gslPath;
28
29     int d_tempFD;
30     int d_gslFD;

```

```

31
32     public:
33         Writer(std::string const &path);
34         ~Writer();
35
36         void addString(std::string const &str);
37         void writeCode(std::string const &str);
38
39     private:
40         void grab(std::string const &str,
41                 std::size_t index);
42 };
43
44 bool checkDelimiter(std::string const &delim,
45                    std::string const &str);
46 void processString(std::string &str, Writer &writer,
47                  void (*func)(std::string &) = [] (std::string &){});
48
49 #endif

```

Listing 11: main.ih

```

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

```

Listing 12: main.cc

```

1  #include "main.ih"
2
3  int main(int argc, char **argv)
4  {
5      if (argc <= 1)
6      {
7          cerr << "Please give your c/c++ file path.\n";
8          return 1;
9      }
10
11     ifstream input;
12     input.open(argv[1]);
13     Scanner scanner(input, cout);

```

```

14  Writer writer(argv[1]);
15
16  string str;
17  string delimiter;
18  while (int token = scanner.lex())
19  {
20      switch (token)
21      {
22          // Case for normal string
23          case (STRING):
24              str += scanner.matched();
25              processString(str, writer);
26              break;
27
28          // Case for raw string
29          case (RSTRING):
30              str += scanner.matched();
31              if (!checkDelimiter(delimiter, str))
32                  break;
33              str = str.substr(0,
34                  str.find_last_of('\\')) + "\\\"";
35              processString(str, writer, makeRaw);
36              delimiter = string{};
37              break;
38
39          case (RSTRING_START):
40              str += scanner.matched();
41              delimiter = str.substr(2, str.length() - 1);
42              str = "\\\"";
43              break;
44
45          case (STRING_SEG):
46              {
47                  string temp(scanner.matched());
48                  // remove second starting quote
49                  temp = temp.substr(0, temp.length() - 1);
50                  // add until first closing quote
51                  str += temp.substr(0,
52                      temp.find_last_of('\\\"'));
53              }

```

```

54         break;
55
56         // Anything else
57         default:
58             writer.writeCode(scanner.matched());
59             break;
60     }
61 }
62 }

```

Listing 13: checkdelimiter.cc

```

1  #include "main.ih"
2
3  bool checkDelimiter(string const &delim,
4      string const &str)
5  {
6      string temp(str.substr(
7          str.find_last_of(',') + 1,
8          str.length()));
9      temp = temp.substr(0,
10         str.find_last_of('"'));
11     return temp == delim;
12 }

```

Listing 14: deescape.cc

```

1  #include "main.ih"
2
3  void deEscape(string &str)
4  {
5      for (auto idx = str.begin();
6          idx != str.end(); ++idx)
7      {
8          if (isEscape(*idx))
9          {
10             *idx = escapeChar(*idx);
11             idx = str.insert(idx, '\\') + 1;
12         }
13     }
14 }

```

Listing 15: dequote.cc

```
1 #include "main.ih"
2
3 string dequote(string const &str)
4 {
5     size_t start = str.find('\0') + 1;
6     return str.substr(start,
7         str.length() - (start + 1));
8 }
```

Listing 16: escapechar.cc

```
1 #include "main.ih"
2
3 char escapeChar(char rawChar)
4 {
5     switch(rawChar)
6     {
7         case('\?'):
8             return '?';
9         case('\a'):
10            return 'a';
11        case('\b'):
12            return 'b';
13        case('\f'):
14            return 'f';
15        case('\n'):
16            return 'n';
17        case('\r'):
18            return 'r';
19        case('\t'):
20            return 't';
21        case('\v'):
22            return 'v';
23    }
24    return rawChar;
25 }
```

Listing 17: isescape.cc

```
1 #include "main.ih"
```

```

2
3 bool isEscape(char rawChar)
4 {
5     switch(rawChar)
6     {
7         case('\\'):
8             return true;
9         case('"'):
10            return true;
11         case('?'):
12            return true;
13         case('\\'):
14            return true;
15         case('\a'):
16            return true;
17         case('\b'):
18            return true;
19         case('\f'):
20            return true;
21         case('\n'):
22            return true;
23         case('\r'):
24            return true;
25         case('\t'):
26            return true;
27         case('\v'):
28            return true;
29     }
30     return false;
31 }

```

Listing 18: makeraw.cc

```

1 #include "main.ih"
2
3 void makeRaw(string &str)
4 {
5     for (auto idx = str.begin();
6         idx != str.end(); ++idx)
7     {
8         if (*idx == '\\')

```



```

9         idx = str.insert(idx, '\\') + 1;
10     }
11 }

```

Listing 19: processstring.cc

```

1  #include "main.ih"
2
3  void processString(string &str, Writer &writer,
4      void (*func)(std::string &))
5  {
6      string temp(dequote(str));
7      func(temp);
8      deEscape(temp);
9      writer.addString(temp);
10     str = string{};
11 }

```

Lexer and scanner

Listing 20: lexer.ll

```

1  %x cComment
2  %x string
3  %x rawString
4  %x header
5
6  SPACE          [ \t\n]
7
8  %%
9
10  "/*"          {
11                begin(StartCondition__::cComment);
12                more();
13            }
14  <cComment>"*/"      {
15                begin(StartCondition__::INITIAL);
16                return COMMENT;
17            }
18  <cComment>{SPACE}|.    more();
19
20  "//".*\n          {

```

```

21         redo(1);
22         return COMMENT;
23     }
24
25     \"          {
26         begin(StartCondition__::string);
27         more();
28     }
29     <string>\"          {
30         begin(StartCondition__::INITIAL);
31         return STRING;
32     }
33     <string>\"{SPACE}*\"          return STRING_SEG;
34     <string>\\n          // ignore
35     <string>.          more();
36
37     \"R\\\"([^(\\\"{1,16})?\"          {
38         begin(StartCondition__::rawString);
39         return RSTRING_START;
40     }
41     <rawString>\"([^(\\\"{1,16})?\"\\\"          {
42         begin(StartCondition__::INITIAL);
43         return RSTRING;
44     }
45     <rawString>.|\\n          more();
46
47     \"#include <\"|\"#include \\\"          {
48         begin(StartCondition__::header);
49         more();
50     }
51     <header>\\\"|\\>          {
52         begin(StartCondition__::INITIAL);
53         return HEADER;
54     }
55     <header>.          more();
56
57     {SPACE}          return OTHER;
58     .          return OTHER;

```

Listing 21: Scanner.h

```
1 // Generated by Flexc++ V2.03.04 on Tue,
2 // 14 Feb 2017 14:13:27 +0100
3
4 #ifndef Scanner_H_INCLUDED_
5 #define Scanner_H_INCLUDED_
6
7 // $insert baseclass_h
8 #include "Scannerbase.h"
9
10 enum Token
11 {
12     STRING = 1,
13     RSTRING_START = 2,
14     RSTRING = 3,
15     STRING_SEG = 4,
16     COMMENT = 5,
17     HEADER = 6,
18     OTHER = 7
19 };
20
21 // $insert classHead
22 class Scanner: public ScannerBase
23 {
24     public:
25         explicit Scanner(std::istream &in = std::cin,
26                         std::ostream &out = std::cout);
27
28         Scanner(std::string const &infile, std::string
29             const &outfile);
30
31         // $insert lexFunctionDecl
32         int lex();
33
34     private:
35         int lex__();
36         int executeAction__(size_t ruleNr);
37
38         void print();
```

```

38         void preCode();      // re-implement this
39             // function for code
40             // that must be
41                 // executed before the
42             // patternmatching starts
43
44         void postCode(PostEnum__ type);
45         // re-implement this function for code that
46         // must be exec'ed after the rules's actions.
47     };
48
49     // $insert scannerConstructors
50     inline Scanner::Scanner(std::istream &in,
51         std::ostream &out)
52     :
53         ScannerBase(in, out)
54     {}
55
56     inline Scanner::Scanner(std::string const &infile,
57         std::string const &outfile)
58     :
59         ScannerBase(infile, outfile)
60     {}
61
62     // $insert inlineLexFunction
63     inline int Scanner::lex()
64     {
65         return lex__();
66     }
67
68     inline void Scanner::preCode()
69     {
70         // optionally replace by your own code
71     }
72
73     inline void Scanner::postCode(PostEnum__ type)
74     {
75         // optionally replace by your own code
76     }
77

```

```

78 inline void Scanner::print()
79 {
80     print__();
81 }
82
83
84 #endif // Scanner_H_INCLUDED_

```

Writer

Listing 22: writer.addString.cc

```

1  #include "main.ih"
2
3  void Writer::addString(string const &str)
4  {
5      auto location = find(d_literals.begin(),
6          d_literals.end(), str);
7
8      size_t index;
9      if (location != d_literals.end())
10     {
11         index = location - d_literals.begin();
12         grab(str, index);
13     }
14     else
15     {
16         d_literals.push_back(str);
17         string temp = str + '\n';
18         write(d_gslFD, temp.c_str(),
19             sizeof(char) * temp.size());
20
21         index = d_literals.size();
22         grab(str, index);
23     }
24 }

```

Listing 23: writer.constr.cc

```

1  #include "main.ih"
2
3  Writer::Writer(string const &path)

```

```

4 :
5   d_path(path),
6   d_tempPath(path)
7 {
8   d_tempPath += ".tmp";
9   {
10    size_t idx = path.find_last_of('.');
11    if (idx == string::npos)
12        d_gslPath = path + ".gsl";
13    else
14        d_gslPath = path.substr(0,
15                                path.find_last_of('.') + ".gsl";
16    }
17
18    if ((d_tempFD = open(d_tempPath.c_str(),
19                        O_CREAT | O_WRONLY)) < 0)
20        cerr << "Couldn't open file " <<
21             d_tempPath << '\n';
22
23    if ((d_gslFD = open(d_gslPath.c_str(),
24                        O_CREAT | O_WRONLY)) < 0)
25        cerr << "Couldn't open file "
26             << d_gslPath << '\n';
27 }

```

Listing 24: writer.grab.cc

```

1 #include "main.ih"
2
3 Writer::~Writer()
4 {
5     struct stat stats;
6     stat(d_path.c_str(), &stats);
7     fchmod(d_tempFD, stats.st_mode);
8     remove(d_path.c_str());
9     rename(d_tempPath.c_str(), d_path.c_str());
10    close(d_tempFD);
11    close(d_gslFD);
12 }

```

Listing 25: writer.writecode.cc

```
1 #include "main.ih"
2 #include <errno.h>
3
4 void Writer::writeCode(string const &str)
5 {
6     write(d_tempFD, str.c_str(),
7         sizeof(char) * str.size());
8 }
```

Input

Listing 26: test.cc

```
1 #include <header1>
2 #include "header2"
3
4 R"dsfs(weird  chars
5 here)dsfs"
6
7 // Comment here
8
9 "string here"
10
11 /* cstyle
12 comment
13 here */
14
15 "compound strings\n"
16 "ahoy?"
17
18 // "STRING IN COMMENTS"
19
20 return of "string here"
21
22 and "compound strings\nahoy?" is back
```

Output

Listing 27: test.cc

```
1 #include <header1>
```

```

2  #include "header2"
3
4  grabbed(1, "test.gsl")
5
6  // Comment here
7
8  grabbed(2, "test.gsl")
9
10 /* cstyle
11 comment
12 here */
13
14 grabbed(3, "test.gsl")
15
16 // "STRING IN COMMENTS"
17
18 return of grabbed(1, "test.gsl")
19
20 and grabbed(2, "test.gsl") is back

```

Listing 28: test.gsl

```

1 weird\tchars\nhere
2 string here
3 compound strings\\nahoy\?

```