

Programming in C/C++

Exercises set four: lexical scanners

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

February 16, 2017

Exercise 24, Non-greedy matching

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

Code listings

Listing 1: main.ih

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

Listing 2: 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 3: 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 4: Scanner.h

```

1  // Generated by Flexc++ V2.03.04 on Thu, 16 Feb 2017
   12:00:34 +0100
2
3  #ifndef Scanner_H_INCLUDED_
4  #define Scanner_H_INCLUDED_
5
6  // $insert baseclass_h
7  #include "Scannerbase.h"
8
9  enum Tokens
10 {
11     WORD = 1
12 };
13
14 // $insert classHead
15 class Scanner: public ScannerBase
16 {
17     public:
18         explicit Scanner(std::istream &in = std::cin,
19                         std::ostream &out =
20                         std::cout);
21
22         Scanner(std::string const &infile, std::string
23         const &outfile);
24
25         // $insert lexFunctionDecl
26         int lex();
27
28     private:

```

```

27         int lex__();
28         int executeAction__(size_t ruleNr);
29
30         void print();
31         void preCode();      // re-implement this
function for code that must
32                             // be exec'ed before the
patternmatching starts
33
34         void postCode(PostEnum__ type);
35                             // re-implement this
function for code that must
36                             // be exec'ed after the
rules's actions.
37 };
38
39 // $insert scannerConstructors
40 inline Scanner::Scanner(std::istream &in, std::ostream
&out)
41 :
42     ScannerBase(in, out)
43 {}
44
45 inline Scanner::Scanner(std::string const &infile, std
::string const &outfile)
46 :
47     ScannerBase(infile, outfile)
48 {}
49
50 // $insert inlineLexFunction
51 inline int Scanner::lex()
52 {
53     return lex__();
54 }
55
56 inline void Scanner::preCode()
57 {
58     // optionally replace by your own code
59 }
60

```

```

61 inline void Scanner::postCode(PostEnum__ type)
62 {
63     // optionally replace by your own code
64 }
65
66 inline void Scanner::print()
67 {
68     print__();
69 }
70
71
72 #endif // Scanner_H_INCLUDED_

```

Exercise 26

See exercise 28.

Exercise 27, tokens

Why are there so many operators?

Lexer

Listing 5: lexer.ll

```

1  %x cComment
2  %x EOLcomment
3  %x string
4  %x header
5
6  SPACE    [ \t\n]
7  ALPHA    [a-zA-Z]
8  NUM      [0-9]
9  ALPHANUM {ALPHA} | {NUM}
10 ANY      [^ \t\n]
11
12 %%
13
14 ({ALPHA} | [_\ -]) ({ALPHANUM} | [_\ -]) *          return
    IDENT;

```

```

15 {NUM}+                                return INT;
16 ({NUM}+.{NUM}+)(E{NUM})?            return DOUBLE;
17
18 "+" | "-" | "*" | "/" | "." | "~" | "=" | "<" | ">" | "&" | "!" | "|" |
    return matched()[0];
19 "<<"                                return SHL_OP;
20 ">>"                                return SHR_OP;
21 "&&"                                return AND;
22 "||"                                return OR;
23 "<="                                return LESS_EQUALS;
24 ">="                                return GREATER_EQUALS;
25 "=="                                return EQUALS;
26 "!="                                return NOT_EQUALS;
27 "+=" | "-=" | "<=" | ">=" | "*=" | "/="    return
    ASSIGNMENT_OP;
28 "->"                                return POINTER;
29
30 \'\\?{ANY}+\'                        return CONSTANT;
31
32 "/*"                                {
33     begin(StartCondition__::cComment);
34     more();
35 }
36 <cComment>"*/"                        {
37     begin(StartCondition__::INITIAL);
38     // ignore
39 }
40 <cComment>{SPACE}|.                  more();
41
42 "//"                                {
43     begin(StartCondition__::EOLcomment);
44     more();
45 }
46 <EOLcomment>\n                      {
47     begin(StartCondition__::INITIAL);
48     redo(1);
49     // ignore
50 }
51 <EOLcomment>.                        more();
52

```

```

53 R?\"          {
54             begin(StartCondition__::string);
55             more();
56         }
57 <string>\"      {
58             begin(StartCondition__::INITIAL);
59             return STRING;
60         }
61 // Needs stitching as seen in exercise 26/28
62 <string>\"{SPACE}*\" return STRING_SEG;
63 <string>\n      more();
64 <string>.       more();
65
66 \"#include <\"|\"#include \"\" {
67             begin(StartCondition__::header);
68             more();
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 6: 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 char getRawChar(char rawChar);
17 void makeRaw(std::string &str);
18
19 class Writer
20 {
21     std::vector<std::string> d_literals;
22
23     std::string d_path;
24     std::string d_tempPath;
25     std::string d_gslPath;
26
27     int d_tempFD;
28     int d_gslFD;
29
30 public:
31     Writer(std::string const &path);
32     ~Writer();
33
34     void addString(std::string const &str);
35     void writeCode(std::string const &str);
36
37 private:
38     void grab(std::string const &str,
39             std::size_t index);
40 };
41
42 #endif

```

Listing 7: main.ih

```

1  #include "main.h"
2

```



```
3 using namespace std;
```

Listing 8: 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     while (int token = scanner.lex())
18     {
19         switch (token)
20         {
21             // Case for normal string
22             case (STRING):
23                 str += scanner.matched();
24                 str = dequote(str);
25                 // cout << str;
26                 writer.addString(str);
27                 str = string{};
28                 break;
29
30             // Case for raw string
31             case (RSTRING):
32                 str += scanner.matched();
33                 str = dequote(str);
34                 makeRaw(str);
35                 // cout << str;
36                 writer.addString(str);
37                 str = string{};
```

```

38         break;
39
40     case (STRING_SEG):
41     {
42         string temp(scanner.matched());
43         temp = temp.substr(0, temp.length() - 1);
44         str += temp.substr(0,
45             temp.find_last_of('\\\"'));
46     }
47     break;
48
49     // Anything else
50     default:
51         writer.writeCode(scanner.matched());
52         break;
53     }
54 }
55 }

```

Listing 9: getrawchar.cc

```

1  #include "main.ih"
2
3  char getRawChar(char rawChar)
4  {
5      switch(rawChar)
6      {
7          case('n'):
8              return '\n';
9          case('t'):
10             return '\t';
11         case('\\'):
12             return '\\';
13         case('\\"'):
14             return '\"';
15         case('\'''):
16             return '\'';
17     }
18     return ' ';
19 }

```

Listing 10: dequote.cc

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

Listing 11: 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 }
```

Lexer and scanner

Listing 12: lexer.ll

```
1 %x cComment
2 %x EOLcomment
3 %x string
4 %x rawString
5 %x header
6
7 SPACE          [ \t\n]
8
9 %%
10
11 "/*"          {
12                begin(StartCondition__::cComment);
13                more();
14            }
```

```

15 <cComment>"*/"      {
16                     begin(StartCondition__::INITIAL);
17                     return COMMENT;
18                     }
19 <cComment>{SPACE}|.  more();
20
21 "/*"               {
22                     begin(StartCondition__::EOLcomment);
23                     more();
24                     }
25 <EOLcomment>\n      {
26                     begin(StartCondition__::INITIAL);
27                     redo(1);
28                     return COMMENT;
29                     }
30 <EOLcomment>|.      more();
31
32 \"                 {
33                     begin(StartCondition__::string);
34                     more();
35                     }
36 <string>\"          {
37                     begin(StartCondition__::INITIAL);
38                     return STRING;
39                     }
40 <string>\"{SPACE}*\"  return STRING_SEG;
41 <string>\n          more();
42 <string>|.          more();
43
44 "R\"               {
45                     begin(StartCondition__::rawString);
46                     more();
47                     }
48 <rawString>\"      {
49                     begin(StartCondition__::INITIAL);
50                     return RSTRING;
51                     }
52 <rawString>\"{SPACE}*\"  return STRING_SEG;
53 <rawString>\n          more();
54 <rawString>|.          more();

```

```

55
56 "#include <"| "#include \"\" {
57     begin(StartCondition__::header);
58     more();
59 }
60 <header>\"|\"> {
61     begin(StartCondition__::INITIAL);
62     return HEADER;
63 }
64 <header>. more();
65
66 {SPACE} return OTHER;
67 . return OTHER;

```

Listing 13: Scanner.h

```

1 // Generated by Flexc++ V2.03.04 on Tue, 14 Feb 2017
  14:13:27 +0100
2
3 #ifndef Scanner_H_INCLUDED_
4 #define Scanner_H_INCLUDED_
5
6 // $insert baseclass_h
7 #include "Scannerbase.h"
8
9 enum Token
10 {
11     STRING = 1,
12     RSTRING = 2,
13     STRING_SEG = 3,
14     COMMENT = 4,
15     HEADER = 5,
16     OTHER = 6
17 };
18
19 // $insert classHead
20 class Scanner: public ScannerBase
21 {
22     public:
23     explicit Scanner(std::istream &in = std::cin,

```

```

24                                     std::ostream &out =
std::cout);
25
26     Scanner(std::string const &infile, std::string
const &outfile);
27
28     // $insert lexFunctionDecl
29     int lex();
30
31     private:
32         int lex__();
33         int executeAction__(size_t ruleNr);
34
35         void print();
36         void preCode();      // re-implement this
function for code that must
37                             // be exec'ed before the
patternmatching starts
38
39         void postCode(PostEnum__ type);
40                             // re-implement this
function for code that must
41                             // be exec'ed after the
rules's actions.
42 };
43
44 // $insert scannerConstructors
45 inline Scanner::Scanner(std::istream &in, std::ostream
&out)
46 :
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_

```

Writer

Listing 14: 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 15: 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 16: 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 17: 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 18: test.cc

```
1  #include <header1>
2  #include "header2"
3
4  R"weird\tchars\n"
5
6  // Comment here
7
8  "string here"
9
10 /* cstyle
11 comment
12 here */
13
```

```

14 "compound strings\n"
15 "ahoy?"
16
17 // "STRING IN COMMENTS"
18
19 return of "string here"
20
21 and "compound strings\nahoy?" is back

```

Output

Listing 19: test.cc

```

1 #include <header1>
2 #include "header2"
3
4 grabbed(1, "testfile/test.gsl")
5
6 // Comment here
7
8 grabbed(2, "testfile/test.gsl")
9
10 /* cstyle
11 comment
12 here */
13
14 grabbed(3, "testfile/test.gsl")
15
16 // "STRING IN COMMENTS"
17
18 return of grabbed(1, "testfile/test.gsl")
19
20 and grabbed(2, "testfile/test.gsl") is back

```

Listing 20: test.gsl

```

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

```