Exercises week 5: Lexical Scanners - Revision

Klaas Isaac Bijlsma s2394480
David Vroom s2309939

March 21, 2018

Exercise 32

Learn to perform a non-greedy match

In onze eerste poging hebben we niet gebruik gemaakt van een mini-scanner om houseboat en household van de andere woorden te onderschijden. Nu wel.

De scanner print de langste aaneenschakeling van non-spaces, tenzij het woord begint met house en eindigt met house of boat, dan gaat de scanner de mini-scanner in.

lexer

```
1  %target-directory = "scanner"
2  %filenames = "scanner"
3
4  /* The exercise defines a word as any consecutive series of
5     non-blank (we interpreted this as non-space) characters */
6
7  WORD                    [[:^space:]]+
8
9  %x house
10
11 %%
12
13 house/boat|hold         {
14                             out() << matched();
15                             begin(StartCondition__::house);
```

```
16                          }
17
18  <house>boat|hold        {
19                              out() << '\n' << matched() << '\n';
20                              begin(StartCondition__::INITIAL);
21                          }
22
23  {WORD}                  out() << matched() << '\n';
24
25  .|\n                    // ignore
```

Exercise 34

Learn to design a scanner scanning a piece of text

In onze eerste poging werkte <<EOF>> niet. We zijn erachter gekomen waarom dit het geval was: de scanner strandde in de `eolComment` mini-scanner en keerde niet terug naar `INITIAL`. Nu werkt de code naar behoren.

Aan de scanner class hebben we een constructor toegevoegd, waaraan de naam van de te bewerken file mee kan worden gegeven, dit is de inputfile. Vervolgens wordt aan deze naam de extensie tmp toegevoegd, dit is de output file. Aan het einde van de bewerking wordt `finish` binnen lexer aangeroepen. De file met de extensie tmp krijgt de naam van de inputfile en vervangt hiermee deze file.

lexer

```
1  %target-directory = "scanner"
2  %filenames = "scanner"
3
4  %x      string
5  %x      eolComment
6  %x      CComment
7
8  %%
9
10 /* The following substitutes a (concatenated) string with a function call
11    to grabbed() and saves the string in <barefilename>.gsl */
12
13 \"                          begin(StartCondition__::string);
14
15 //%nowarn
16                                          // match everyting except quote
17 <string>[^\"]*              d_gsl << matched();
18
19 <string>\"[[:space:]]*\"    // ignore to support string concatenation
20
21 <string>\"                  {
22                                  d_gsl << '\n';
23                                  out() << "grabbed("
24                                        << ++d_counter
```

```
                                        << ", \""
                                        << d_gslFileName
                                        << "\")";
                                    begin(StartCondition__::INITIAL);
                                }

/* Everything within eol comment and c comment is inserted into the
   output file */

"//"                            {
                                    out() << matched();
                                    begin(StartCondition__::eolComment);
                                }

<eolComment>$                   begin(StartCondition__::INITIAL);

"/*"                            {
                                    out() << matched();
                                    begin(StartCondition__::CComment);
                                }

<CComment>"*/"                  {
                                    out() << matched();
                                    begin(StartCondition__::INITIAL);
                                }

<<EOF>>                         {
                                    finish();
                                    return 0;
                                }

/* All characters not matched are inserted implicitely into the
   output file */
```

scanner/scanner.h

```
// Generated by Flexc++ V2.06.04 on Mon, 19 Mar 2018 14:57:38 +0100

#ifndef Scanner_H_INCLUDED_
#define Scanner_H_INCLUDED_
```

```
5
6  // $insert baseclass_h
7  #include "scannerbase.h"
8
9  #include <fstream>   // toegevoegd
10 #include <cstdio>    // toegevoegd
11 #include <exception> // toegevoegd
12
13 // $insert classHead
14 class Scanner: public ScannerBase
15 {
16     std::string d_fileName;     // toegevoegd
17     std::string d_gslFileName;  // toegevoegd
18     std::ofstream d_gsl;        // toegevoegd
19     size_t d_counter = 0;       // toegevoegd
20
21     public:
22
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         explicit Scanner(std::string const &file);   // toegevoegd
29
30         // $insert lexFunctionDecl
31         int lex();
32
33     private:
34         int lex__();
35         int executeAction__(size_t ruleNr);
36
37         void print();
38         void preCode();     // re-implement this function for code that must
39                             // be exec'ed before the patternmatching starts
40
41         void postCode(PostEnum__ type);
42                             // re-implement this function for code that must
43                             // be exec'ed after the rules's actions.
44
45         void finish();      // toegevoegd
```

```
46 };
47
48 // $insert scannerConstructors
49 inline Scanner::Scanner(std::istream &in, std::ostream &out)
50 :
51     ScannerBase(in, out)
52 {}
53
54 inline Scanner::Scanner(
55     std::string const &infile, std::string const &outfile)
56 :
57     ScannerBase(infile, outfile)
58 {}
59
60 // toegevoegd
61 inline Scanner::Scanner(std::string const &file)
62 :
63     ScannerBase(file, std::string{file} + ".tmp"),
64     d_fileName(file),
65     d_gslFileName(file.substr(0, file.find_last_of(".")) + ".gsl"),
66     d_gsl(d_gslFileName)
67 {}
68
69 inline void Scanner::finish()
70 {
71     int result = std::rename(
72         (d_fileName + ".tmp").c_str(), d_fileName.c_str());
73     if (result)
74         throw std::runtime_error("Renaming temporary file failed");
75 }
76
77 // $insert inlineLexFunction
78 inline int Scanner::lex()
79 {
80     return lex__();
81 }
82
83 inline void Scanner::preCode()
84 {
85     // optionally replace by your own code
86 }
```

```
87
88 inline void Scanner::postCode([[maybe_unused]] PostEnum__ type)
89 {
90     // optionally replace by your own code
91 }
92
93 inline void Scanner::print()
94 {
95     print__();
96 }
97
98 #endif // Scanner_H_INCLUDED_
```

main.cc

```
1 #include "scanner/scanner.h"
2
3 using namespace std;
4
5 int main(int argc, char **argv)
6 {
7     Scanner scanner{ argv[1] };
8     scanner.lex();
9 }
```

Exercise 35

Design a small tokenizer

In onze eerste poging accepteerde onze lexer niet de juiste character constanten en werd een double met een e-operator niet herkend. Dit is opgelost. Daarbij hebben we de code opgeschoond en leesbaarder gemaakt.

lexer

```
%target-directory = "scanner"
%filenames = "scanner"

unaryOp          \+\+|--|\(\)|\[\]
relationalOp     ==|!=|\<=|\>=|&&|\|\|
assignOp         \+=|-=|\*=|\/=|%=|\>\>=|&=|\^=|\|=
otherOp          ::|->|\.\*|->\*|\<\<|\>\>

int              [0-9]+
double           [0-9]*\.[0-9]+
eDouble          ({double}|{int})[Ee][\+-]?{int}

escapeSeq        '\\[abfnrtv'"?\\]'
octal            '\\[0-7]+'
hexadecimal      '\\x[0-9a-fA-F]+'

ident            [a-zA-Z_][a-zA-Z_0-9]*

%%

[[:blank:]]                     // ignore

/* Through greedy matching until end of line,
   string concatenation is applied */
\".+\"                          return STRING;

{escapeSeq}                     |
{octal}                         |
{hexadecimal}                   return CHARCONST;

{unaryOp}                       |
```

```
{relationalOp}                  |
{assignOp}                      |
{otherOp}                       return OPERATOR;

{double}                        |
{eDouble}                       return DOUBLE;

{ident}                         return IDENT;

{int}                           return INTEGRAL;

.                               return matched()[0];
```

scanner/scanner.h

```
// Generated by Flexc++ V2.06.04 on Mon, 19 Mar 2018 21:37:11 +0100

#ifndef Scanner_H_INCLUDED_
#define Scanner_H_INCLUDED_

// $insert baseclass_h
#include "scannerbase.h"

enum ScannerToken
{
    IDENT = 257,
    INTEGRAL,
    DOUBLE,
    OPERATOR,
    STRING,
    CHARCONST
};

// $insert classHead
class Scanner: public ScannerBase
{
    public:
        explicit Scanner(std::istream &in = std::cin,
                         std::ostream &out = std::cout);

```

```
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(
51      std::string const &infile, std::string const &outfile)
52  :
53      ScannerBase(infile, outfile)
54  {}
55
56  // $insert inlineLexFunction
57  inline int Scanner::lex()
58  {
59      return lex__();
60  }
61
62  inline void Scanner::preCode()
63  {
64      // optionally replace by your own code
65  }
66
```

```
67 inline void Scanner::postCode([[maybe_unused]] PostEnum__ type)
68 {
69     // optionally replace by your own code
70 }
71
72 inline void Scanner::print()
73 {
74     print__();
75 }
76
77
78 #endif // Scanner_H_INCLUDED_
```

main.cc

```
1  #include "scanner/scanner.h"
2  #include <fstream>
3
4  using namespace std;
5
6  int main(int argc, char **argv)
7  {
8      ifstream file{ argv[1] };
9      Scanner scanner{ file };
10     while (int TOKEN = scanner.lex())
11         cout << TOKEN << ' ' << scanner.matched() << '\n';
12 }
```