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

```
#include "Scanner.h"
#include <set>
#include <fstream>

using namespace std;
```

Listing 2: main.cc

```
#include "main.ih"

int main(int argc, char **argv)
{
    set<string> myset;
    if (argc == 1)
    {
        Scanner scanner(cin, cout);

        while (scanner.lex())
        {
            myset.insert(scanner.matched());
        }
  }
    else
    {
        for (int idx = 1; idx != argc; ++idx)
        {
            ifstream input;
            input.open(argv[idx]);
            Scanner scanner(input, std::cout);

            //read
            while (scanner.lex())
            {
                myset.insert(scanner.matched());
            }
        }
    }
    //print
    for(string const &idx : myset)
    {
        cout << idx << '\n';
    }
}
```

Listing 3: lexer.ll

```
NONBLANK  [a-zA-Z]

%%

{NONBLANK}+ return WORD;

.|\n    // ignore
```

Listing 4: Scanner.h

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

```

```
#ifndef Scanner_H_INCLUDED_
#define Scanner_H_INCLUDED_

enum { WORD = 256 };

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


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

        Scanner(std::string const &infile,
                std::string const &outfile);

        // $insert lexFunctionDecl
        int lex();

    private:
        int lex__();
        int executeAction__(size_t ruleNr);

        void print();
        void preCode();     // re-implement this
                            // function for code
                            // that must be
                            // executed before the
                            // patternmatching starts

        void postCode(PostEnum__ type);
        // re-implement this function for code that
        // must be exec'ed after the rules's actions.
};

// $insert scannerConstructors
```

```
inline Scanner::Scanner(std::istream &in,
                        std::ostream &out)
:
    ScannerBase(in, out)
{}

inline Scanner::Scanner(std::string const &infile,
                        std::string const &outfile)
:
    ScannerBase(infile, outfile)
{}

// $insert inlineLexFunction
inline int Scanner::lex()
{
    return lex__();
}

inline void Scanner::preCode()
{
    // optionally replace by your own code
}

inline void Scanner::postCode(PostEnum__ type)
{
    // optionally replace by your own code
}

inline void Scanner::print()
{
    print__();
}


#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

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

Listing 6: main.cc

```
#include "main.ih"

int main()
try
{
    Scanner scanner;

    while (scanner.lex())
        ;
}
catch (...)
{
    return 1;
}
```

Listing 7: lexer.ll

```
%x house
%x post

NONBLANK      [a-zA-Z\-]

%%
house/(hold|boat)   {
            begin(StartCondition__::house);
            std::cout << matched() << '\n';
            return WORD;
          }

house          {
            begin(StartCondition__::house);
            more();
          }
```

```

<house>
{
  hold|boat   {
             begin(StartCondition__::INITIAL);
             std::cout << matched() << '\n';
             return WORD;
           }

  {NONBLANK}*   {
             begin(StartCondition__::INITIAL);
             std::cout << matched() << '\n';
             return WORD;
           }
}

{NONBLANK}*house    {
             begin(StartCondition__::post);
             more();
           }

<post>
{
  {NONBLANK}*   {
             begin(StartCondition__::INITIAL);
             std::cout << matched() << '\n';
             return WORD;
           }
}

\n          // ignore
```

Listing 8: Scanner.h

```
// Generated by Flexc++ V2.03.04 on Thu,
// 16 Feb 2017 12:00:34 +0100

#ifndef Scanner_H_INCLUDED_
#define Scanner_H_INCLUDED_

// $insert baseclass_h
```

```
#include "Scannerbase.h"

enum Tokens
{
  WORD = 1
};

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

        Scanner(std::string const &infile,
        std::string const &outfile);

        // $insert lexFunctionDecl
        int lex();

    private:
        int lex__();
        int executeAction__(size_t ruleNr);

        void print();
        void preCode();     // re-implement this
                  // function for code
                  // that must be
                            // executed before the
                  // patternmatching starts

        void postCode(PostEnum__ type);
        // re-implement this function for code that
        // must be exec'ed after the rules's actions.
};

// $insert scannerConstructors
inline Scanner::Scanner(std::istream &in, std::ostream
    &out)
:
    ScannerBase(in, out)
{}

inline Scanner::Scanner(std::string const &infile, std
   ::string const &outfile)
:
    ScannerBase(infile, outfile)
{}

// $insert inlineLexFunction
inline int Scanner::lex()
{
    return lex__();
}

inline void Scanner::preCode()
{
    // optionally replace by your own code
}

inline void Scanner::postCode(PostEnum__ type)
{
    // optionally replace by your own code
}

inline void Scanner::print()
{
    print__();
}


#endif // Scanner_H_INCLUDED_
```

Exercise 26

See exercise 28.

Exercise 27, tokens

Why are there so many operators?

Lexer

Listing 9: lexer.ll

```
%x cComment
%x string
%x header

SPACE        [ \t\n]
ALPHA        [a-zA-Z]
NUM          [0-9]
ALPHANUM     {ALPHA}|{NUM}

// Anything but ALPHANUM tab and newline
SYMBOL       [^{ALPHANUM}\t\n]

%%

({ALPHA}|"_")({ALPHANUM}|"_")*  return IDENT;
{NUM}+                          return INT;
({NUM}+"."{NUM}+)(E-?{NUM})?    return DOUBLE;

// Singular operators taken care off at the end
"<<"                            return SHL_OP;
">>"                            return SHR_OP;
"&&"                            return AND;
"||"                            return OR;
"<="                            return LESS_EQUALS;
">="                            return GREATER_EQUALS;
"=="                            return EQUALS;
"!="                            return NOT_EUQALS;
"+="|"-="|"<<="                 return ASSIGNMENT_OP;
">>="|"*="|"/="                 return ASSIGNMENT_OP;
"->"                            return POINTER;

"'"("\")?({ALPHANUM}|{SYMBOL}|"\")+"'"
{
    return CONSTANT;
}

```

```
\\ EOL comment is ignored (but not the newline)
"//".*\n                       redo(1);

"/*"                {
                        begin(StartCondition__::cComment);
                        more();
                    }
<cComment>"*/"      {
                        begin(StartCondition__::INITIAL);
                        // ignore
                    }
<cComment>{SPACE}|.     more();

R?\"                {
                        begin(StartCondition__::string);
                        more();
                    }
<string>\"          {
                        begin(StartCondition__::INITIAL);
                        return STRING;
                    }

// Needs stitching as seen in exercise 26/28
<string>\"{SPACE}*\"           return STRING_SEG;
<string>\n                     more();
<string>.                      more();

"#include <"|"#include \""
{
    begin(StartCondition__::header);
    more();
}

<header>\"|\>       {
                        begin(StartCondition__::INITIAL);
                        return HEADER;
                    }
<header>.           more();

\n                  return ENDLINE;
```

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

Exercise 28, scanner that scans text

We were asked to design a scanner thats scan a piece of text.

Code listings

Listing 10: main.h

```
#ifndef MAIN_H
#define MAIN_H

#include <vector>
#include <algorithm>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fstream>

#include "Scanner.h"

std::string dequote(std::string const &str);

void makeRaw(std::string &str);
char escapeChar(char rawChar);
bool isEscape(char rawChar);
void deEscape(std::string &str);

class Writer
{
  std::vector<std::string> d_literals;

  std::string d_path;
  std::string d_tempPath;
  std::string d_gslPath;

  int d_tempFD;
  int d_gslFD;

  public:
    Writer(std::string const &path);
    ~Writer();

    void addString(std::string const &str);
    void writeCode(std::string const &str);

  private:
    void grab(std::string const &str,
      std::size_t index);
};

bool checkDelimeter(std::string const &delim,
  std::string const &str);
void processString(std::string &str, Writer &writer,
  void (*func)(std::string &) = [](std::string &){});

#endif
```

Listing 11: main.ih

```
#include "main.h"

using namespace std;
```

Listing 12: main.cc

```
#include "main.ih"

int main(int argc, char **argv)
{
  if (argc <= 1)
  {
    cerr << "Please give your c/c++ file path.\n";
    return 1;
  }

  ifstream input;
  input.open(argv[1]);
  Scanner scanner(input, cout);
```

```cpp
  Writer writer(argv[1]);

  string str;
  string delimeter;
  while (int token = scanner.lex())
  {
    switch (token)
    {
      // Case for normal string
      case (STRING):
        str += scanner.matched();
        processString(str, writer);
        break;

      // Case for raw string
      case (RSTRING):
        str += scanner.matched();
        if (!checkDelimeter(delimeter, str))
          break;
        str = str.substr(0,
          str.find_last_of(')')) + '\"';
        processString(str, writer, makeRaw);
        delimeter = string{};
        break;

      case (RSTRING_START):
        str += scanner.matched();
        delimeter = str.substr(2,str.length() - 1);
        str = "\"";
        break;

      case (STRING_SEG):
        {
          string temp(scanner.matched());
          // remove second starting quote
          temp = temp.substr(0, temp.length() - 1);
          // add until first closing quote
          str += temp.substr(0,
            temp.find_last_of('\"'));
        }
        break;

      // Anything else
      default:
        writer.writeCode(scanner.matched());
        break;
    }
  }
}
```

Listing 13: checkdelimeter.cc

```
#include "main.ih"

bool checkDelimeter(string const &delim,
  string const &str)
{
  string temp(str.substr(
    str.find_last_of(')') + 1,
    str.length()));
  temp = temp.substr(0,
    str.find_last_of('"'));
  return temp == delim;
}
```

Listing 14: deescape.cc

```
#include "main.ih"

void deEscape(string &str)
{
  for (auto idx = str.begin();
    idx != str.end(); ++idx)
  {
    if (isEscape(*idx))
    {
      *idx = escapeChar(*idx);
      idx = str.insert(idx, '\\') + 1;
    }
  }
}
```

Listing 15: dequote.cc

```
#include "main.ih"

string dequote(string const &str)
{
  size_t start = str.find('\"') + 1;
  return str.substr(start,
    str.length() - (start + 1));
}
```

Listing 16: escapechar.cc

```
#include "main.ih"

char escapeChar(char rawChar)
{
  switch(rawChar)
  {
    case('\?'):
      return '?';
    case('\a'):
      return 'a';
    case('\b'):
      return 'b';
    case('\f'):
      return 'f';
    case('\n'):
      return 'n';
    case('\r'):
      return 'r';
    case('\t'):
      return 't';
    case('\v'):
      return 'v';
  }
  return rawChar;
}
```

Listing 17: isescape.cc

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

```

bool isEscape(char rawChar)
{
  switch(rawChar)
  {
    case('\''):
      return true;
    case('\"'):
      return true;
    case('\?'):
      return true;
    case('\\'):
      return true;
    case('\a'):
      return true;
    case('\b'):
      return true;
    case('\f'):
      return true;
    case('\n'):
      return true;
    case('\r'):
      return true;
    case('\t'):
      return true;
    case('\v'):
      return true;
  }
  return false;
}
```

Listing 18: makeraw.cc

```
#include "main.ih"

void makeRaw(string &str)
{
  for (auto idx = str.begin();
    idx != str.end(); ++idx)
  {
    if (*idx == '\\')
```

```
        idx = str.insert(idx, '\\') + 1;
    }
}
```

Listing 19: processstring.cc

```
#include "main.ih"

void processString(string &str, Writer &writer,
  void (*func)(std::string &))
{
  string temp(dequote(str));
  func(temp);
  deEscape(temp);
  writer.addString(temp);
  str = string{};
}
```

Lexer and scanner

Listing 20: lexer.ll

```
%x cComment
%x string
%x rawString
%x header

SPACE            [ \t\n]

%%

"/*"             {
                 begin(StartCondition__::cComment);
                 more();
               }
<cComment>"*/"          {
                 begin(StartCondition__::INITIAL);
                 return COMMENT;
               }
<cComment>{SPACE}|.     more();

"//".*\n           {
```

```
                redo(1);
                return COMMENT;
              }

\"              {
                begin(StartCondition__::string);
                more();
              }
<string>\"          {
                begin(StartCondition__::INITIAL);
                return STRING;
              }
<string>\"{SPACE}*\"    return STRING_SEG;
<string>\n          // ignore
<string>.         more();

"R\""([^()\"]{1,16})?"("      {
                  begin(StartCondition__::rawString);
                  return RSTRING_START;
                }
<rawString>")"([^()\"]{1,16})?"\""  {
                  begin(StartCondition__::INITIAL);
                  return RSTRING;
                }
<rawString>.|\n         more();

"#include <"|"#include \""  {
                begin(StartCondition__::header);
                more();
              }
<header>\"|\>       {
                begin(StartCondition__::INITIAL);
                return HEADER;
              }
<header>.         more();

{SPACE}           return OTHER;
.             return OTHER;
```

Listing 21: Scanner.h

```
// Generated by Flexc++ V2.03.04 on Tue,
// 14 Feb 2017 14:13:27 +0100

#ifndef Scanner_H_INCLUDED_
#define Scanner_H_INCLUDED_

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

enum Token
{
  STRING = 1,
  RSTRING_START = 2,
  RSTRING = 3,
  STRING_SEG = 4,
  COMMENT = 5,
  HEADER = 6,
  OTHER = 7
};

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

        Scanner(std::string const &infile, std::string
    const &outfile);

        // $insert lexFunctionDecl
        int lex();

    private:
        int lex__();
        int executeAction__(size_t ruleNr);

        void print();
        void preCode();     // re-implement this
              // function for code
              // that must be
                            // executed before the
              // patternmatching starts

        void postCode(PostEnum__ type);
        // re-implement this function for code that
        // must be exec'ed after the rules's actions.
};

// $insert scannerConstructors
inline Scanner::Scanner(std::istream &in,
            std::ostream &out)
:
    ScannerBase(in, out)
{}

inline Scanner::Scanner(std::string const &infile,
          std::string const &outfile)
:
    ScannerBase(infile, outfile)
{}

// $insert inlineLexFunction
inline int Scanner::lex()
{
    return lex__();
}

inline void Scanner::preCode()
{
    // optionally replace by your own code
}

inline void Scanner::postCode(PostEnum__ type)
{
    // optionally replace by your own code
}

```

```
inline void Scanner::print()
{
    print__();
}


#endif // Scanner_H_INCLUDED_
```

Writer

Listing 22: writer.addString.cc

```
#include "main.ih"

void Writer::addString(string const &str)
{
  auto location = find(d_literals.begin(),
    d_literals.end(), str);

  size_t index;
  if (location != d_literals.end())
  {
    index = location - d_literals.begin();
    grab(str, index);
  }
  else
  {
    d_literals.push_back(str);
    string temp = str + '\n';
    write(d_gslFD, temp.c_str(),
      sizeof(char) * temp.size());

    index = d_literals.size();
    grab(str, index);
  }
}
```

Listing 23: writer.constr.cc

```
#include "main.ih"

Writer::Writer(string const &path)
```

```
:
  d_path(path),
  d_tempPath(path)
{
  d_tempPath += ".tmp";
  {
    size_t idx = path.find_last_of('.');
    if (idx == string::npos)
      d_gslPath = path + ".gsl";
    else
      d_gslPath = path.substr(0,
        path.find_last_of('.')) + ".gsl";
  }

  if ((d_tempFD = open(d_tempPath.c_str(),
      O_CREAT | O_WRONLY)) < 0)
    cerr << "Couldn't open file " <<
      d_tempPath << '\n';

  if ((d_gslFD = open(d_gslPath.c_str(),
      O_CREAT | O_WRONLY)) < 0)
    cerr << "Couldn't open file "
      << d_gslPath << '\n';
}
```

Listing 24: writer.grab.cc

```
#include "main.ih"

Writer::~Writer()
{
  struct stat stats;
  stat(d_path.c_str(), &stats);
  fchmod(d_tempFD, stats.st_mode);
  remove(d_path.c_str());
  rename(d_tempPath.c_str(), d_path.c_str());
  close(d_tempFD);
  close(d_gslFD);
}
```

Listing 25: writer.writecode.cc

```
#include "main.ih"
#include <errno.h>

void Writer::writeCode(string const &str)
{
  write(d_tempFD, str.c_str(),
    sizeof(char) * str.size());
}
```

Input

Listing 26: test.cc

```
#include <header1>
#include "header2"

R"dsfs(weird  chars
here)dsfs"

// Comment here

"string here"

/* cstyle
comment
here */

"compound strings\n"
"ahoy?"

// "STRING IN COMMENTS"

return of "string here"

and "compound strings\nahoy?" is back
```

Output

Listing 27: test.cc

```
#include <header1>
```

```
#include "header2"

grabbed(1, "test.gsl")

// Comment here

grabbed(2, "test.gsl")

/* cstyle
comment
here */

grabbed(3, "test.gsl")

// "STRING IN COMMENTS"

return of grabbed(1, "test.gsl")

and grabbed(2, "test.gsl") is back
```

Listing 28: test.gsl

```
weird\tchars\nhere
string here
compound strings\\nahoy\?
```