8

1

41 8
42 0
43 0

48 0

FB.

C++ Exercises
Set 6

Author(s): Pau Lopez, Peter Versluis

10:56

March 21, 2025

41

A program that prints the words found in a piece of text in a sorted order.

Listing 1: main.ih

```
#include "scanner/Scanner.h"
#include <fstream>
using namespace std;
```

Listing 2: main.cc

```
#include "main.ih"

int main(int argc, char *argv[])
{
    Scanner scanner;

    // if no file names are specified, read from cin
    if (argc == 1)
        scanner.lex();

    // otherwise, read from each file
    else for (int idx = 1; idx != argc; ++idx)
    {
        // open the file and switch scanner's input stream
        ifstream file{ argv[idx] };
        scanner.switchStreams(file);

        // read the file
        scanner.lex();
    }

    // print the readed words
    scanner.printWords();
}
```

Listing 3: scanner/lexer

```
%%
[ \t\n]+                                // skip white space chars.
[[:alpha:]]+                            d_words.push(matched());
```

Listing 4: scanner/Scanner.h

```
// Generated by Flexc++ V2.15.00 on Wed, 19 Mar 2025 15:18:50 +0100

#ifndef Scanner_H_INCLUDED_
#define Scanner_H_INCLUDED_

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

```
#include <queue>

// $insert classHead
class Scanner: public ScannerBase
{
    // using std::greater to reverse the order of the queue
    std::priority_queue<std::string,
                        std::vector<std::string>,
                        std::greater<std::string>> d_words;

    public:
        explicit Scanner(std::istream &in = std::cin,
                         std::ostream &out = std::cout, bool keepCwd = true);

        Scanner(std::string const &infile,
                std::string const &outfile, bool keepCwd = true);

        // $insert lexFunctionDecl
        int lex();

        void printWords();

    private:
        int lex_();
        int executeAction_(size_t ruleNr);

        void print();
        void preCode();     // re-implement this function for code that must
                            // be exec'ed 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, bool keepCwd)
:
    ScannerBase(in, out, keepCwd)
{}

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

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

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

inline void Scanner::postCode([[maybe_unused]] PostEnum_ type)
{
    // optionally replace by your own code
}

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


#endif // Scanner_H_INCLUDED_
```

Listing 5: scanner/Scanner.ih

```
// Generated by Flexc++ V2.15.00 on Wed, 19 Mar 2025 15:18:50 +0100

// $insert class_h
#include "Scanner.h"
```

Listing 6: scanner/printwords.cc

```
#include "Scanner.h"

void Scanner::printWords()
{
    while(d_words.size() != 0)
    {
        std::cout << d_words.top() << ' ';
        d_words.pop();
    }
}
```

Now the words have gone. It matches the exercise, but, now you can't use em twice...

42

A program that tokenizes it~~'~~s input.

Listing 7: main.cc

```
#include "scanner/Scanner.h"
#include <iostream>

int main()
{
    Scanner scanner;

    // process the lines according to the rules we defined
    std::cout << "Line 1\n";
    scanner.lex();
}
```

not like a C++ identifier

Listing 8: scanner/lexer

```
%%
[ \t]                        // skip white spaces
[\n]+                        std::cout << "Line " << lineNr() << '\n';

[[:alpha:]]+                 std::cout << "word: "      << matched() << '\n';
[0-9]+                       std::cout << "integer: "   << matched() << '\n';
[0-9]+\.[0-9]+               std::cout << "float: "     << matched() << '\n';
\".+\"                       std::cout << "string: "    << matched() << '\n';
\'[[:alpha:]]\'              std::cout << "character "
                                       << static_cast<int>(matched()[0]) << ": "
                                       << matched() << '\n';


// Operators
"++"
"--"
"<<"
">>"
"<="
">="
"=="
"!="
"&&"
"||"
"->"
"::"
".*"
"->*"
"+="
"-="
```

what about the e-notations of doubles?

what about strings like "" and "\""?

avoid

hard to read. (forget the [ ], also ( ) { } aren't operators

weird operator...

```
"*="
"/="
"%="
"&="
"|="
"^="
"<<="
">>="
[+\-*/%=<>!&|^~?:;.,\[\](){}#]                std::cout << "operator: "
                                                        << matched() << '\n';
```

Listing 9: scanner/Scanner.h

```
// Generated by Flexc++ V2.15.00 on Thu, 20 Mar 2025 11:30:00 +0100

#ifndef Scanner_H_INCLUDED_
#define Scanner_H_INCLUDED_

// $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, bool keepCwd = true);

        Scanner(std::string const &infile, std::string const &outfile, bool
           keepCwd = true);

        // $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 exec'ed 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, bool keepCwd)
:
    ScannerBase(in, out, keepCwd)
{}

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

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

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

inline void Scanner::postCode([[maybe_unused]] PostEnum_ type)
```

```
{
    // optionally replace by your own code
}

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


#endif // Scanner_H_INCLUDED_
```

what about string concatenation on 1 line?

Listing 10: scanner/Scanner.ih

```
// Generated by Flexc++ V2.15.00 on Thu, 20 Mar 2025 11:30:00 +0100

// $insert class_h
#include "Scanner.h"
```

43

A program that replaces strings in a source file with function calls.

Listing 11: main.ih

```
#include "scanner/Scanner.h"
#include <sstream>
#include <fstream>
using namespace std;
```

Listing 12: main.cc

```
#include "main.ih"

int main(int argc, char *argv[])
{
    // a file name must be provided
    if (argc != 2)
    {
        cout << "Please provide an file name\n";
        return 1;
    }

    // initialize the scanner
    Scanner scanner;

    // write to a tmp stringstream and read from the specified file
    stringstream tmp;
    scanner.switchOstream(tmp);
    scanner.switchIstream(argv[1]);

    // process the file
    scanner.lex();

    // override the file
    ofstream outpf { argv[1] };
    outpf << tmp.str();
}
```

members to keep track of what's going on.

it's not that simple. you can't handle RSL with one Regex. Neither can you do that with strings and multi-line comment. You also need to recognize, e.g., octal, hexadecimal numbers, escape characters (in strings) etc., etc. Your Scanner class will need quie [illegible] support

Listing 13: scanner/lexer

```
%%

// don't replace strings in comments
"//".*$             out() << matched();

'R'?\"[^\"]*\"          {
                        ++d_num;
```

```
                    out() << "grabbed(" << d_num << ", \""
                          << filename() << "\")";
                };
```

Listing 14: scanner/Scanner.h

```
// Generated by Flexc++ V2.15.00 on Thu, 20 Mar 2025 12:39:27 +0100

#ifndef Scanner_H_INCLUDED_
#define Scanner_H_INCLUDED_

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


// $insert classHead
class Scanner: public ScannerBase
{
    size_t d_num = 0;

    public:
        explicit Scanner(std::istream &in = std::cin,
                                std::ostream &out = std::cout, bool keepCwd = true);

        Scanner(std::string const &infile, std::string const &outfile,
                bool keepCwd = true);

        // $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 exec'ed 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, bool keepCwd)
:
    ScannerBase(in, out, keepCwd)
{}

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

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

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

inline void Scanner::postCode([[maybe_unused]] PostEnum_ type)
{
    // optionally replace by your own code
}

inline void Scanner::print()
```

```
{
    print_();
}


#endif // Scanner_H_INCLUDED_
```

Listing 15: scanner/Scanner.ih

```
// Generated by Flexc++ V2.15.00 on Thu, 20 Mar 2025 12:39:27 +0100

// $insert class_h
#include "Scanner.h"
```

48

A program that replaces includes in a source file with it~~'~~s corresponding included content.

Listing 16: main.ih

```
#include "scanner/Scanner.h"
#include <stdexcept>
#include <iostream>
using namespace std;
```

Listing 17: main.cc

```
#include "main.ih"

int main(int argc, char *argv[])
try
{
    // a file name must be provided
    if (argc != 2)
        throw std::runtime_error("Please specify a file name\n");

    // initialize the scanner
    Scanner scanner;
    scanner.switchIstream(argv[1]);

    // read the input file
    scanner.lex();
}
catch(std::runtime_error exc)
{
    cerr << "Program ended after catching an exception: " << exc.what();
}
```

Listing 18: scanner/lexer

```
%x INCLUDE
%x PATH

%%

"#include "                 begin(StartCondition_::INCLUDE);

<INCLUDE>\"                 begin(StartCondition_::PATH);

<PATH>[^\"\n]+              {
                                // obtain the path
                                std::string pathName { matched() };
                                std::filesystem::path path{ pathName };

                                // open the file
                                std::ifstream includedFile{ path };

                                // check if file was opened correctly
```

(8)

```
                            if (!includedFile)
                                throw std::runtime_error("Cannot open '"
                                                         + pathName + "\'\n");

                            // check for recursive included
                            if (d_included.find(pathName) == d_included.end())
                                d_included.insert(pathName);
                            else
                            {
                                throw std::runtime_error("Recursive
                                                          inclusion\n");
                            }

                            // process the file
                            Scanner auxScanner{ includedFile, out() };
                            auxScanner.lex();

                            // erase the file name from the inclusions
                            d_included.erase(pathName);
                        }
// skip final double quote and return to previous condition
<PATH>\"                begin(StartCondition_::INITIAL);
```

(way) TC for an action.

there should be no need for a stack of scanners all the info you need is already available in the scanner. As a hint: our lexer file defines 1 mini-scanner and has 4 rules, each containing 1 expression as its action.

Listing 19: scanner/Scanner.h

```
// Generated by Flexc++ V2.15.00 on Thu, 20 Mar 2025 14:14:55 +0100

#ifndef Scanner_H_INCLUDED_
#define Scanner_H_INCLUDED_

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

#include <set>

// $insert classHead
class Scanner: public ScannerBase
{
    static std::set<std::string> d_included;

    public:
        explicit Scanner(std::istream &in = std::cin,
                         std::ostream &out = std::cout, bool keepCwd = true);

        Scanner(std::string const &infile, std::string const &outfile,
                bool keepCwd = true);

        // $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 exec'ed 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, bool keepCwd)
:
    ScannerBase(in, out, keepCwd)
{}

inline Scanner::Scanner(std::string const &infile, std::string const &outfile,
                        bool keepCwd)
```

(etc: not scanned)