Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex` Coding Advice

Bibliography

Lexical Analysis

Ettore Speziale

Politecnico di Milano

Contents

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

1 Introduction

2 Unix Tools
- Automating Tedious Tasks

3 Scanner Generators
- A Simple Example
- The `flex` Input File
- Handling Words
- The Internals of `flex`
- Advanced `flex`
- Coding Advice

4 Bibliography

Contents

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

1 Introduction

2 Unix Tools
- Automating Tedious Tasks

3 Scanner Generators
- A Simple Example
- The `flex` Input File
- Handling Words
- The Internals of `flex`
- Advanced `flex`
- Coding Advice

4 Bibliography

Lexical

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

"Relating to words or the vocabulary of a language as distinguished from its grammar and construction."

Webster's Dictionary

Words

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

Words are *simple constructs*:

- on a natural language we can simply *enumerate* them
- not possible with technical languages (too many words!)

But technical words are simpler than natural words:

C identifier rules

- a sequence of non-digit characters (including the underscore _, the lowercase and uppercase Latin letters, and other characters) and digits
- cannot starts with a digit

Regular expression can *describe* their structure!

Analysis

Lexical analysis must:

- *recognize* tokens in a stream of characters (e.g. identifiers)
- possibly *decorate* tokens with additional info (e.g. the name of the identifier)

Performed by mean of a scanner:

- coding by hand is both tedious and error-prone
- usually automatically generated from a regular expressions-based description

No surprises: the scanner is just a big Finite State Automaton.

Contents

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

1 Introduction

2 Unix Tools
- Automating Tedious Tasks

3 Scanner Generators
- A Simple Example
- The `flex` Input File
- Handling Words
- The Internals of `flex`
- Advanced `flex`
- Coding Advice

4 Bibliography

A Simple Lexical Analysis

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

The simplest lexical analysis is *recognize* words:

- many UNIX tools provide a regular expression interface to match words
- each tool is specialized on doing something with matched words

Good knowledge of these tools can speedup your work.

Finding Words

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Given a file with a list of names, one per line, find all names starting with a vowel:

Using grep

```
$ cat names.txt
ettore
chiara
michela
antonio
$ grep '^[aeiou]' names.txt
ettore
antonio
```

Delete Patterns

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Given a file with a list of names, one per line, delete all names whose second character is a vowel:

Using sed

```
$ cat names.txt
ettore
chiara
michela
antonio
$ sed '/^.[aeiou]/d' names.txt
ettore
chiara
antonio
```

CSV Processing

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Given CSV file, where each line is the pair (person,field), print all people working on the astrophysics field:

Using awk

```
$ cat bindings.txt
ettore,compilers
chiara,automotive
michela,astrophysics
antonio,compilers
$ awk -F , '/,astrophysics$/ {print $1}' \
  bindings.txt
michela
```

Contents

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

1 Introduction

2 Unix Tools
- Automating Tedious Tasks

3 Scanner Generators
- A Simple Example
- The `flex` Input File
- Handling Words
- The Internals of `flex`
- Advanced `flex`
- Coding Advice

4 Bibliography

Back to Compilers

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

We need a scanner to recognize language words:

- the `flex` tool generates scanners

Getting `flex`

Available in your distribution repositories:

Debian `aptitude install flex`

Fedora `yum install flex`

The Scanning Problem

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex` Coding Advice

Bibliography

For some applications, a scanner is enough:

- it is used to both detecting words and applying semantic actions

But language translation is not a simple task, thus the scanner *prepares input* for semantic analysis:

- detect words (e.g. identifiers)
- clean input (e.g. removes comments)
- add info to words (e.g. identifier names)

We will see these aspects later, now we will use only the scanner.

A Case Lowering Tool

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Given a string, build the lower case equivalent string.

String lowering

HELLO Flex $\longrightarrow$ *hello flex*

We can describe our language with a regular expression:

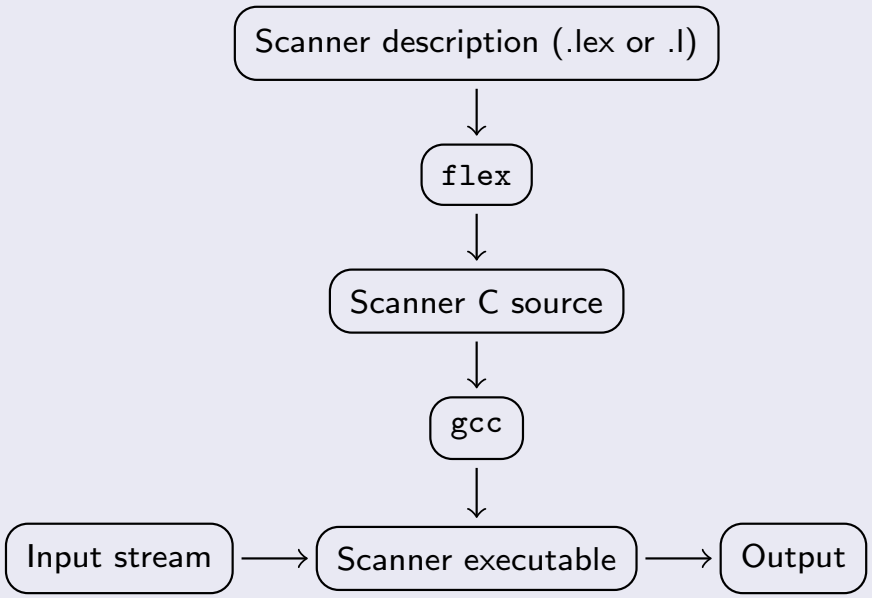
$$
\begin{aligned}
STRING &\longrightarrow WORD(' ', WORD)^* \\
WORD &\longrightarrow (UPPER|LOWER)^+ \\
UPPER &\longrightarrow ('A'|'B'|\ldots|'Z') \\
LOWER &\longrightarrow ('a'|'b'|\ldots|'z')
\end{aligned}
$$

We must express the same things using `flex`.

Introducing flex

The flex tool must be used inside a tool-chain:

flex work-flow



Detecting Words

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Starts with two simple concepts:

- by default unmatched chars are copied to `stdout`
- thus we must add only rules to match uppercase letters

case-matching.l

```
%option noyywrap
UPPER [A-Z]
%%
{UPPER} { }
%%
int main (int argc,char* argv[]) {
  return yylex();
}
```

Automating Repetitive Tasks

Invoking `flex` is easy:

By-hand compilation

```
$ flex case-matching.l
$ gcc lex.yy.c
```

Better to use `make`:

Automated compilation

```
$ make case-matching
lex -t case-matching.l > case-matching.c
cc -c -o case-matching.o case-matching.c
cc case-matching.o -o case-matching
rm case-matching.o case-matching.c
```

Adding Semantic

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Semantic actions are added beside rules:

case-lowering.l

```
%option noyywrap
UPPER [A-Z]
%%
{UPPER} { printf("%c",tolower(*yytext)); }
%%
int main (int argc,char* argv[]) {
  return yylex();
}
```

File Format

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Three sections:

definitions	declare tokens
rules	bind token combinations to actions
user code	plain old C code

Input of `flex` [a]

```
/* Definitions */
%%
/* Rules */
%%
/* User code */
```

[a] Comments not allowed inside rules

Definitions

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex` Coding Advice

Bibliography

Allows to associate a name to a set of characters:

- you can use regular expression to define character sets
- usually used to define *simple* concepts (e.g digits)

Definitions

```
/* Lower case and upper case letters */
LETTER [a-zA-Z]
/* Numerical digits */
DIGITS [0-9]
```

Rules I

What to do when something is recognized:

- exploits definitions to define complex concepts (e.g. from DIGIT to NUMBER)
- can use regular expression as glue!

Rules [1]

```
/* Identifiers: letters plus digits */
{LETTER}({LETTER}|{DIGIT})* { return ID; }
/* Number: a list of digits */
{DIGIT}+ { return NUMBER; }
/* The "if" keyword */
"if" { return IF; }
```

Rules II

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex` Coding Advice

Bibliography

Actions:

- are executed every time the rule is matched
- can access to matched data

Simple scanners executes directly the semantic action (e.g. case lowering).
Complex scanners (e.g. language tokenizer):

1. assign a value to the recognized token
2. return the token type

Rules III

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Here is a partial list of variables that can be accessed from inside an action:

Rule variables

Variable	Type	Meaning
`yytext`	`char*`	matched text
`yyleng`	`int`	matched text length

[1] Comments not allowed inside rules

User Code

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex` Coding Advice

Bibliography

User C code is copied to the generated scanner *as is*:

- the `main` function
- any other routine called by actions
- scanner-wrapping routines
- ...

Additional Code

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Arbitrary code can be put inside definitions and rules sections by *escaping* from `flex`:

- code copied *as is* into the generated scanner
- good place for header inclusions, globals, ...

Header inclusions

```
%{
#include <limits.h>
#include <string.h>
%}
```

Regular Expressions I

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

The following tables contains the regular expressions accepted by `flex`:

Basic regular expressions

Syntax	Matches
`x`	the `x` character
`.`	any character except newline
`[xyz]`	`x` or `y` or `z`
`[a-z]`	any character between `a` and `z`
`[^a-z]`	any character except those between `a` and `z`
`{X}`	expansion of `X` definition
`"hello"`	the `hello` string

Regular Expressions II

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Regular expression composition

Syntax	Matches
`R`	the `R` regular expression
`RS`	concatenation of `R` and `S`
`R|S`	either `R` or `S`
`R*`	zero or more occurrences of `R`
`R+`	one or more occurrences of `R`
`R?`	zero or one occurrence of `R`
`R{m,n}`	a number or `R` occurrences ranging from `n` to `m`
`R{n,}`	`n` or more occurrences of `n`
`R{n}`	exactly `n` occurrences of `R`

Regular Expressions III

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

Regular expression utilities

Syntax	Matches
`(R)`	override precedence
`^R`	`R` at beginning of a line
`R$`	`R` at the end of a line

Note that most of UNIX tools handling regular expression *accept* the same syntax.

Some Notes on Scanner Generation I

The scanner is just *a finite state automaton*! Look at the sources:

Scanner states of `case-lowering.l`

```
/* States */
static yyconst flex_int16_t yy_def[7] =
  { 0, 6, 1, 6, 6, 6, 0 };
/* Accepting states */
static yyconst flex_int16_t yy_accept[7] =
  { 0, 0, 0, 3, 2, 1, 0 };
/* Starting state */
static int yy_start = 0;
```

Some Notes on Scanner Generation II

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

Scanner transitions of `case-lowering.l`

```
/* Transitions */
static yyconst flex_int16_t yy_nxt[7] =
  { 0, 4, 5, 6, 3, 6, 6 };
```

Obviously states are encoded to allow fast matching.

Scanner Behaviour

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

The scanner applies the following:

longest matching rule if more than one matching string is found, the rule that generates the longest one is selected

first rule if more than one string with the same length are found, the rule listed first in the rules section is selected

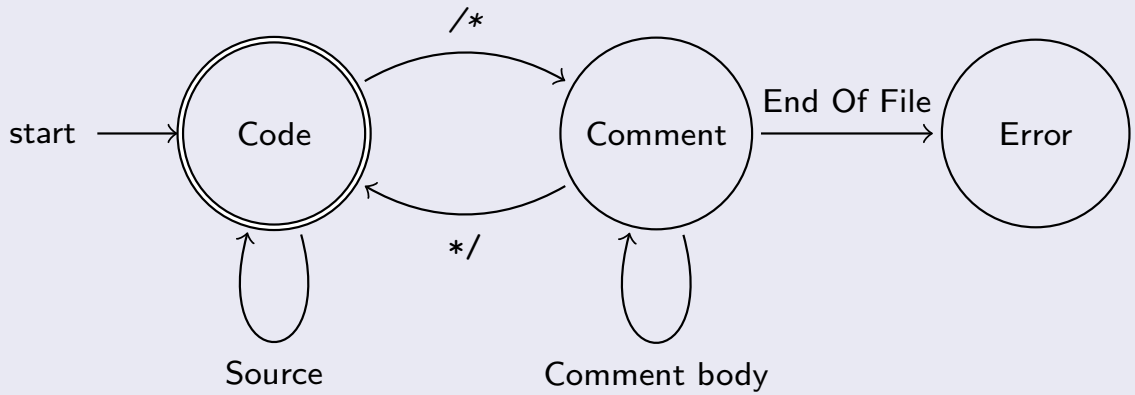
default action if no rules were found the next character in input is considered matched and it is copied to the output stream, then the scanner goes on

Multiple Scanners I

Sometimes is useful to have more than one scanner together:

- code scanner
- comment scanner

C scanners



Multiple Scanners II

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

To support multiple scanners:

- rules can be marked with the name of the associated scanner (*start condition*)
- special actions to switch between scanners

Multiple Scanners III

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

A start condition `S`:

- mark rules with a prefix `<S>RULE`
- activate marked rules only when the scanner is in `S`

Moreover:

- the `*` start condition matches every start condition
- the initial start condition is `INITIAL`
- start conditions are stored as integers (C `int`)
- the current start condition is stored in the `YY_START` variable

Multiple Scanners IV

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

Start conditions can be:

exclusive declared with `%x S`; disable unmarked rules when the scanner is in the `S` start condition

inclusive declared with `%s S`; unmarked rules active when scanner is in the `S` start condition

Multiple Scanners V

Here is a table with relevant special actions:

Special actions

Action	Meaning
`BEGIN(S)`	place scanner in start condition `S`
`ECHO`	copies `yytext` to output

Comment Eater I

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Eat C-99 style comment:

C-99 comment eater (counters)

```
%x COMMENT
%option noyywrap
%{
    #define MAX_DEPTH 10

    int nest = 0;
    int caller[MAX_DEPTH];
%}
```

Comment Eater II

C-99 comment eater (code rules)

```
<INITIAL>[^/]* {
                    ECHO;
                }
<INITIAL>"/"+[^*/]* {
                      ECHO;
                    }
<INITIAL>"/*" {
                caller[nest++] = YY_START;
                BEGIN(COMMENT);
              }
```

Comment Eater III

C-99 comment eater (comment rules)

```
<COMMENT>[^/*]*
<COMMENT>"/"+[^*/]*
<COMMENT>"/*"  {
                  caller[nest++] = YY_START;
                  BEGIN(COMMENT);
                }
<COMMENT>"*"+[^*/]*
<COMMENT>"*"+"/"  {
                    BEGIN(caller[--nest]);
                  }
%%
```

Comment Eater IV

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

C-99 comment eater (main)

```
int main(int argc, char* argv[]) {
  return yylex();
}
```

Clean Regular Expressions

Lexical Analysis
Ettore Speziale
Introduction
Unix Tools
Automating Tedious Tasks
Scanner Generators
A Simple Example
The flex Input File
Handling Words
The Internals of flex
Advanced flex
Coding Advice
Bibliography

Regular expression can describe `simple` concepts:

- complex structures are typically described by "encrypted" regular expression

Even with simple concepts is better to keep the regular expression as clean as possible:

- they becomes unreadable very quickly

Exploit tool features to simplify regular expressions (e.g. definitions).

Contents

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

1 Introduction

2 Unix Tools
- Automating Tedious Tasks

3 Scanner Generators
- A Simple Example
- The `flex` Input File
- Handling Words
- The Internals of `flex`
- Advanced `flex`
- Coding Advice

4 Bibliography

Bibliography

Lexical Analysis

Ettore Speziale

Introduction

Unix Tools

Automating Tedious Tasks

Scanner Generators

A Simple Example

The `flex` Input File

Handling Words

The Internals of `flex`

Advanced `flex`

Coding Advice

Bibliography

P. Hazel.
Perl Compatible Regular Expressions.
man 3 pcre, 2007.

Formal Languages and Compilers Group.
Software Compilers.
http://compilergroup.elet.polimi.it, 2010.

Linux man-pages project.
POSIX.2 Regular Expressions.
man 7 regex, 2007.

V. Paxson, W. Estes, and J. Millaway.
The `flex` Manual.
info flex, 2007.