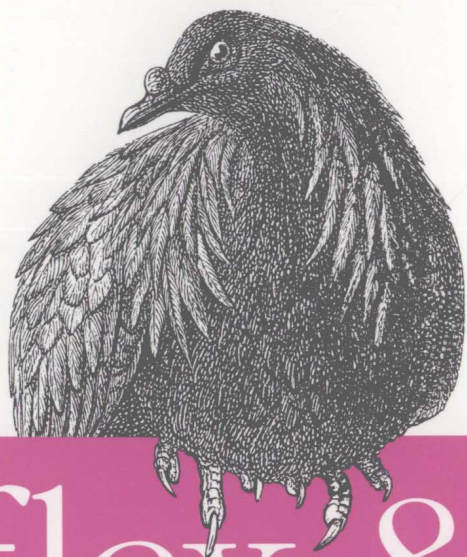
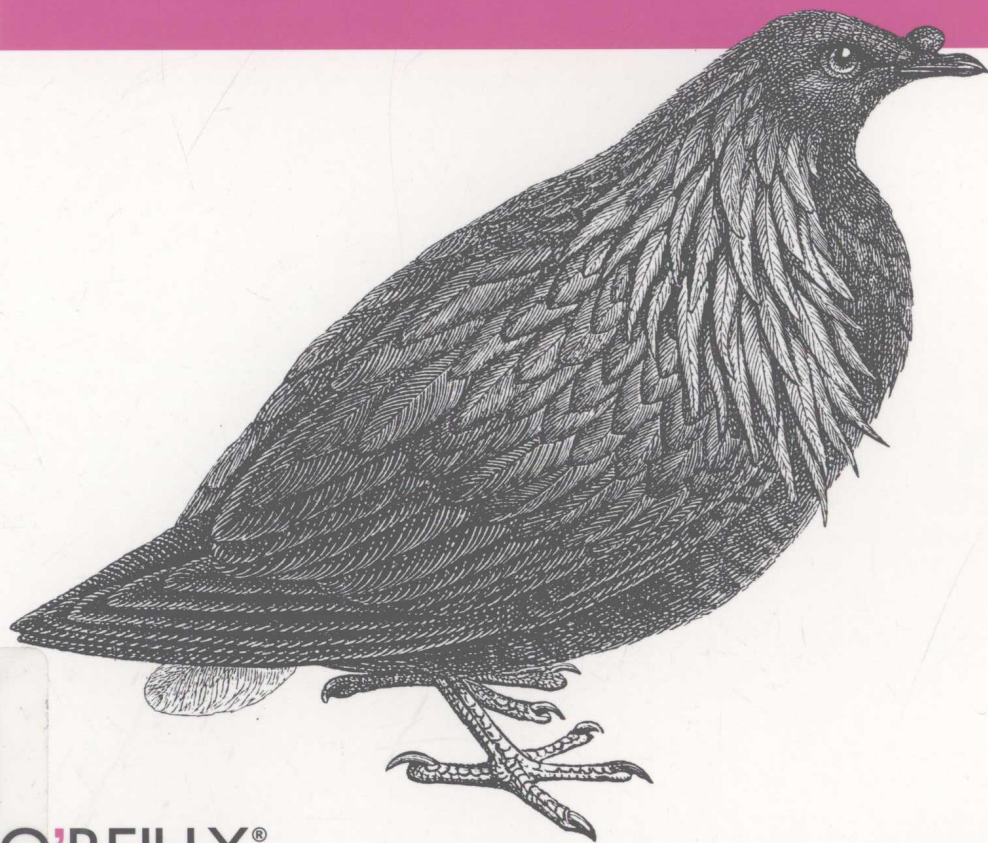


*flex*与*bison* (影印版)



flex & bison



O'REILLY®

東南大學出版社

John Levine 著

flex与bison (影印版)

flex and bison

John R. Levine

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc. 授权东南大学出版社出版

东南大学出版社

图书在版编目 (CIP) 数据

flex 与 bison: 英文 / (美) 莱文 (Levine, J.) 著.
影印本. — 南京: 东南大学出版社, 2010.1

书名原文: flex & bison

ISBN 978-7-5641-1932-4

I . F… II . 莱… III . 软件工具—程序设计—英文
IV . TP311.56

中国版本图书馆 CIP 数据核字 (2009) 第 205644 号

江苏省版权局著作权合同登记

图字: 10-2009-251 号

©2009 by O'Reilly Media, Inc.

Reprint of the English Edition, jointly published by O'Reilly Media, Inc. and Southeast University Press, 2009. Authorized reprint of the original English edition, 2009 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly Media, Inc. 出版 2009。

英文影印版由东南大学出版社出版 2009。此影印版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有, 未得书面许可, 本书的任何部分和全部不得以任何形式重制。

flex 与 bison (影印版)

出版发行: 东南大学出版社

地 址: 南京四牌楼 2 号 邮编: 210096

出 版 人: 江 汉

网 址: <http://press.seu.edu.cn>

电子邮件: press@seu.edu.cn

印 刷: 扬中市印刷有限公司

开 本: 787 毫米 × 980 毫米 16 开本

印 张: 18.5 印张

字 数: 311 千字

版 次: 2010 年 1 月第 1 版

印 次: 2010 年 1 月第 1 次印刷

书 号: ISBN 978-7-5641-1932-4

印 数: 1~1600 册

定 价: 46.00 元 (册)

本社图书若有印装质量问题, 请直接与读者服务部联系。电话 (传真): 025-83792328

Preface

Flex and bison are tools designed for writers of compilers and interpreters, although they are also useful for many applications that will interest noncompiler writers. Any application that looks for patterns in its input or has an input or command language is a good candidate for flex and bison. Furthermore, they allow for rapid application prototyping, easy modification, and simple maintenance of programs. To stimulate your imagination, here are a few things people have used flex and bison, or their predecessors lex and yacc, to develop:

- The desktop calculator *bc*
- The tools *eqn* and *pic*, typesetting preprocessors for mathematical equations and complex pictures
- Many other “domain-specific languages” targeted for a particular application
- *PCC*, the Portable C Compiler used with many Unix systems
- Flex itself
- A SQL database language translator

Scope of This Book

Chapter 1, *Introducing Flex and Bison*, gives an overview of how and why flex and bison are used to create compilers and interpreters and demonstrates some simple applications including a calculator built in flex and bison. It also introduces basic terms we use throughout the book.

Chapter 2, *Using Flex*, describes how to use flex. It develops flex applications that count words in files, handle multiple and nested input files, and compute statistics on C programs.

Chapter 3, *Using Bison*, gives a full example using flex and bison to develop a fully functional desktop calculator with variables, procedures, loops, and conditional expressions. It shows the use of abstract syntax trees (ASTs), powerful and easy-to-use data structures for representing parsed input.

Chapter 4, *Parsing SQL*, develops a parser for the MySQL dialect of the SQL relational database language. The parser checks the syntax of SQL statements and translates them

into an internal form suitable for an interpreter. It shows the use of Reverse Polish Notation (RPN), another powerful form used to represent and interpret parsed input.

Chapter 5, *A Reference for Flex Specifications*, and Chapter 6, *A Reference for Bison Specifications*, provide detailed descriptions of the features and options available to flex and bison programmers. These chapters and the two that follow provide technical information for the now-experienced flex and bison programmer to use while developing flex and bison applications.

Chapter 7, *Ambiguities and Conflicts*, explains bison ambiguities and conflicts, which are grammar problems that keep bison from creating a parser from a grammar. It then develops methods that can be used to locate and correct such problems.

Chapter 8, *Error Reporting and Recovery*, discusses techniques that compiler or interpreter designers can use to locate, recognize, and report errors in the compiler input.

Chapter 9, *Advanced Flex and Bison*, covers reentrant scanners and parsers, Generalized Left to Right (GLR) parsers that can handle grammars that regular bison parsers can't, and interfaces to C++.

The *appendix* provides the complete grammar and a cross-reference for the SQL parser discussed in Chapter 4.

The *glossary* lists technical terms from language and compiler theory.

We presume you are familiar with C, because most examples are in C, flex, or bison, with a few in C++ and the remainder in SQL or the special-purpose languages developed within the text.

Conventions Used in This Book

The following conventions are used in this book:

Italic

Used for new terms and concepts when they are introduced.

Constant Width

Used for program listings, as well as within paragraphs to refer to program elements such as statements, classes, macros, states, rules, all code terms, and files and directories.

Constant Bold

Shows commands or other text that should be typed literally by the user.

Constant width italic

Shows text that should be replaced with user-supplied values or by values determined by context.

\$

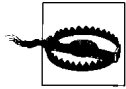
is the shell prompt.

[]

surround optional elements in a description of program syntax. (Don't type the brackets themselves.)



This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

Getting Flex and Bison

Flex and bison are modern replacements for the classic lex and yacc that were both developed at Bell Laboratories in the 1970s. Yacc was the first of the two, developed by Stephen C. Johnson. Lex was designed by Mike Lesk and Eric Schmidt (the same Eric Schmidt who now heads Google) to work with bison. Both lex and yacc have been standard Unix utilities since Seventh Edition Unix in the 1970s.

The GNU Project of the Free Software Foundation distributes bison, a forward-compatible replacement for yacc. It was originally written by Robert Corbett and Richard Stallman. The bison manual is excellent, especially for referencing specific features. Bison is included with all common distributions of BSD and Linux, but if you want the most up-to-date version, its home page is:

<http://www.gnu.org/software/bison/>

BSD and the GNU Project also distribute *flex* (Fast Lexical Analyzer Generator), “a rewrite of lex intended to fix some of that tool’s many bugs and deficiencies.” Flex was originally written by Jef Poskanzer; Vern Paxson and Van Jacobson have considerably improved it. Common distributions of BSD and Linux include a copy of flex, but if you want the latest version, it’s now hosted at SourceForge:

<http://flex.sourceforge.net/>

This Book’s Example Files

The programs in this book are available online as:

<ftp://ftp.iecc.com/pub/file/flexbison.zip>

They can be downloaded by any web browser or FTP client. The zip format file can be decoded by the popular freeware *unzip* utility on Unix-ish and Linux systems or opened as a compressed folder on Windows XP or newer.

The examples in the book were all tested with flex version 2.5.35 and bison 2.4.1.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Book Title* by Some Author. Copyright 2008 O'Reilly Media, Inc., 978-0-596-xxxx-x."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

Safari® Books Online

Safari® Books Online When you see a Safari® Books Online icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at <http://my.safaribooksonline.com>.

How to Contact Us

Despite all the help, errors remain the author's responsibility. When you find some, or if you have other comments, email them to fbook@iecc.com, being sure to include the name of the book in the subject line to alert the spam filters that you are a real person rather than a deceased kleptocrat from a developing country. Or drop by the Usenet group comp.compilers where questions about compiler tools are always on topic.

You can also address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)

707-829-0515 (international or local)
707 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

<http://www.oreilly.com/catalog/9780596155971>

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

<http://www.oreilly.com>

Acknowledgments

Every book is the work of many people, and this one is no exception. I thank Tony Mason and Doug Brown, my coauthors of *lex & yacc*, for permission to adapt parts of that book. Many people provided useful comments on the draft manuscript, including Lorenzo Bettini, Joel E. Denny, Danny Dubé, Ben Hanson, Jan Van Katwijk, Jeff Kenton, Timothy Knox, Chris Morley, Ken Rose, and Juha Vihavainen. I particularly thank Derek M. Jones, who provided a detailed page-by-page review in an unreasonably short time. Simon St. Laurent, my long-suffering editor, as always shepherded the book skillfully and without complaint through the editorial and production process.

Table of Contents

Preface	xiii
1. Introducing Flex and Bison	1
Lexical Analysis and Parsing	1
Regular Expressions and Scanning	2
Our First Flex Program	2
Programs in Plain Flex	4
Putting Flex and Bison Together	5
The Scanner as Coroutine	6
Tokens and Values	7
Grammars and Parsing	9
BNF Grammars	10
Bison's Rule Input Language	11
Compiling Flex and Bison Programs Together	13
Ambiguous Grammars: Not Quite	14
Adding a Few More Rules	15
Flex and Bison vs. Handwritten Scanners and Parsers	16
Exercises	17
2. Using Flex	19
Regular Expressions	19
Regular Expression Examples	21
How Flex Handles Ambiguous Patterns	22
Context-Dependent Tokens	22
File I/O in Flex Scanners	23
Reading Several Files	24
The I/O Structure of a Flex Scanner	25
Input to a Flex Scanner	26
Flex Scanner Output	27
Start States and Nested Input Files	28
Symbol Tables and a Concordance Generator	32
Managing Symbol Tables	32

Using a Symbol Table	35
C Language Cross-Reference	38
Exercises	45
3. Using Bison	47
How a Bison Parser Matches Its Input	47
Shift/Reduce Parsing	48
What Bison's LALR(1) Parser Cannot Parse	50
A Bison Parser	51
Abstract Syntax Trees	51
An Improved Calculator That Creates ASTs	52
Literal Character Tokens	54
Building the AST Calculator	57
Shift/Reduce Conflicts and Operator Precedence	57
When Not to Use Precedence Rules	60
An Advanced Calculator	61
Advanced Calculator Parser	64
Calculator Statement Syntax	65
Calculator Expression Syntax	66
Top-Level Calculator Grammar	67
Basic Parser Error Recovery	67
The Advanced Calculator Lexer	68
Reserved Words	69
Building and Interpreting ASTs	70
Evaluating Functions in the Calculator	76
User-Defined Functions	76
Using the Advanced Calculator	78
Exercises	79
4. Parsing SQL	81
A Quick Overview of SQL	81
Relational Databases	82
Manipulating Relations	83
Three Ways to Use SQL	83
SQL to RPN	84
The Lexer	85
Scanning SQL Keywords	86
Scanning Numbers	90
Scanning Operators and Punctuation	91
Scanning Functions and Names	92
Comments and Miscellany	93
The Parser	94
The Top-Level Parsing Rules	96

SQL Expressions	96
Select Statements	101
Delete Statement	106
Insert and Replace Statements	107
Update Statement	110
Create Database	110
Create Table	111
User Variables	114
The Parser Routines	115
The Makefile for the SQL Parser	116
Exercises	117
5. A Reference for Flex Specifications	119
Structure of a Flex Specification	119
Definition Section	119
Rules Section	119
User Subroutines	120
BEGIN	120
C++ Scanners	121
Context Sensitivity	121
Left Context	121
Right Context	122
Definitions (Substitutions)	122
ECHO	123
Input Management	123
Stdio File Chaining	123
Input Buffers	123
Input from Strings	124
File Nesting	124
input()	124
YY_INPUT	125
Flex Library	125
Interactive and Batch Scanners	126
Line Numbers and yylineno	126
Literal Block	126
Multiple Lexers in One Program	127
Combined Lexers	127
Multiple Lexers	128
Options When Building a Scanner	128
Portability of Flex Lexers	129
Porting Generated C Lexers	129
Reentrant Scanners	130
Extra Data for Reentrant Scanners	130

Access to Reentrant Scanner Data	131
Reentrant Scanners, Nested Files, and Multiple Scanners	131
Using Reentrant Scanners with Bison	132
Regular Expression Syntax	132
Metacharacters	132
REJECT	135
Returning Values from yylex()	135
Start States	135
unput()	137
yyinput() yyunput()	137
yylen	137
yyless()	137
yylex() and YY_DECL	138
yyomore()	138
yyrestart()	139
yy_scan_string and yy_scan_buffer	139
YY_USER_ACTION	139
yywrap()	139
 6. A Reference for Bison Specifications	 141
Structure of a Bison Grammar	141
Symbols	141
Definition Section	142
Rules Section	142
User Subroutines Section	142
Actions	142
Embedded Actions	143
Symbol Types for Embedded Actions	144
Ambiguity and Conflicts	144
Types of Conflicts	144
Shift/Reduce Conflicts	144
Reduce/Reduce Conflicts	145
%expect	145
GLR Parsers	145
Bugs in Bison Programs	146
Infinite Recursion	146
Interchanging Precedence	146
Embedded Actions	146
C++ Parsers	147
%code Blocks	147
End Marker	147
Error Token and Error Recovery	147
%destructor	148

Inherited Attributes (\$0)	148
Symbol Types for Inherited Attributes	149
%initial-action	149
Lexical Feedback	150
Literal Block	151
Literal Tokens	151
Locations	152
%parse-param	152
Portability of Bison Parsers	153
Porting Bison Grammars	153
Porting Generated C Parsers	153
Libraries	153
Character Codes	153
Precedence and Associativity Declarations	154
Precedence	154
Associativity	154
Precedence Declarations	154
Using Precedence and Associativity to Resolve Conflicts	155
Typical Uses of Precedence	155
Recursive Rules	155
Left and Right Recursion	156
Rules	157
Special Characters	158
%start Declaration	159
Symbol Values	160
Declaring Symbol Types	160
Explicit Symbol Types	160
Tokens	161
Token Numbers	161
Token Values	161
%type Declaration	162
%union Declaration	163
Variant and Multiple Grammars	163
Combined Parsers	163
Multiple Parsers	165
Using %name-prefix or the -p Flag	165
Lexers for Multiple Parsers	165
Pure Parsers	165
y.output Files	166
Bison Library	167
main()	167
yyerror()	167
YYABORT	168

YYACCEPT	168
YYBACKUP	168
yyclearin	169
yydebug and YYDEBUG	169
YYDEBUG	169
yydebug	169
yyerrok	170
YYERROR	170
yyerror()	171
yyparse()	171
YYRECOVERING()	171
 7. Ambiguities and Conflicts	 173
The Pointer Model and Conflicts	173
Kinds of Conflicts	175
Parser States	176
Contents of name.output	178
Reduce/Reduce Conflicts	178
Shift/Reduce Conflicts	180
Review of Conflicts in name.output	182
Common Examples of Conflicts	183
Expression Grammars	183
IF/THEN/ELSE	185
Nested List Grammar	186
How Do You Fix the Conflict?	187
IF/THEN/ELSE (Shift/Reduce)	188
Loop Within a Loop (Shift/Reduce)	190
Expression Precedence (Shift/Reduce)	191
Limited Lookahead (Shift/Reduce or Reduce/Reduce)	191
Overlap of Alternatives (Reduce/Reduce)	192
Summary	194
Exercises	194
 8. Error Reporting and Recovery	 197
Error Reporting	197
Locations	199
Adding Locations to the Parser	200
Adding Locations to the Lexer	201
More Sophisticated Locations with Filenames	202
Error Recovery	204
Bison Error Recovery	205
Freeing Discarded Symbols	206
Error Recovery in Interactive Parsers	206

Where to Put Error Tokens	207
Compiler Error Recovery	208
Exercises	208
9. Advanced Flex and Bison	209
Pure Scanners and Parsers	209
Pure Scanners in Flex	210
Pure Parsers in Bison	212
Using Pure Scanners and Parsers Together	213
A Reentrant Calculator	214
GLR Parsing	230
GLR Version of the SQL Parser	231
C++ Parsers	234
A C++ Calculator	235
C++ Parser Naming	235
A C++ Parser	236
Interfacing a Scanner with a C++ Parser	239
Should You Write Your Parser in C++?	241
Exercises	241
Appendix: SQL Parser Grammar and Cross-Reference	243
Glossary	259
Index	263

Introducing Flex and Bison

Flex and Bison are tools for building programs that handle structured input. They were originally tools for building compilers, but they have proven to be useful in many other areas. In this first chapter, we'll start by looking at a little (but not too much) of the theory behind them, and then we'll dive into some examples of their use.

Lexical Analysis and Parsing

The earliest compilers back in the 1950s used utterly ad hoc techniques to analyze the syntax of the source code of programs they were compiling. During the 1960s, the field got a lot of academic attention, and by the early 1970s, syntax analysis was a well-understood field.

One of the key insights was to break the job into two parts: *lexical analysis* (also called *lexing* or *scanning*) and *syntax analysis* (or *parsing*).

Roughly speaking, scanning divides the input into meaningful chunks, called *tokens*, and parsing figures out how the tokens relate to each other. For example, consider this snippet of C code:

```
alpha = beta + gamma ;
```

A scanner divides this into the tokens `alpha`, `equal` `sign`, `beta`, `plus` `sign`, `gamma`, and `semicolon`. Then the parser determines that `beta + gamma` is an expression, and that the expression is assigned to `alpha`.

Getting Flex and Bison

Most Linux and BSD systems come with flex and bison as part of the base system. If your system doesn't have them, or has out-of-date versions, they're both easy to install.

Flex is a Sourceforge project, at <http://flex.sourceforge.net/>. The current version as of early 2009 was 2.5.35. Changes from version to version are usually minor, so it's not essential to update your version if it's close to .35, but some systems still ship with version 2.5.4 or 2.5.4a, which is more than a decade old.

Bison is available from <http://www.gnu.org/software/bison/>. The current version as of early 2009 was 2.4.1. Bison is under fairly active development, so it's worth getting an up-to-date version to see what's new. Version 2.4 added support for parsers in Java, for example. BSD users can generally install a current version of flex or bison using the ports collection. Linux users may be able to find current RPMs. If not, flex and bison both use the standard GNU build process, so to install them, download and unpack the current flex and bison tarballs from the web sites, run `./configure` and then `make` to build each, then become superuser and `make install` to install them.

Flex and bison both depend on the GNU m4 macroprocessor. Linux and BSD should all have m4, but in case they don't, or they have an ancient version, the current GNU m4 is at <http://www.gnu.org/software/m4/>.

For Windows users, both bison and flex are included in the Cygwin Linux emulation environment available at <http://www.cygwin.com/>. You can use the C or C++ code they generate either with the Cygwin development tools or with native Windows development tools.

Regular Expressions and Scanning

Scanners generally work by looking for patterns of characters in the input. For example, in a C program, an integer constant is a string of one or more digits, a variable name is a letter followed by zero or more letters or digits, and the various operators are single characters or pairs of characters. A straightforward way to describe these patterns is *regular expressions*, often shortened to *regex* or *regexp*. These are the same kind of patterns that the editors `ed` and `vi` and the search program `egrep` use to describe text to search for. A flex program basically consists of a list of regexps with instructions about what to do when the input matches any of them, known as *actions*. A flex-generated scanner reads through its input, matching the input against all of the regexps and doing the appropriate action on each match. Flex translates all of the regexps into an efficient internal form that lets it match the input against all the patterns simultaneously, so it's just as fast for 100 patterns as for one.*

Our First Flex Program

Unix systems (by which I also mean Unix-ish systems including Linux and the BSDs) come with a word count program, which reads through a file and reports the number of lines, words, and characters in the file. Flex lets us write `wc` in a few dozen lines, shown in Example 1-1.

* The internal form is known as a deterministic finite automation (DFA). Fortunately, the only thing you really need to know about DFAs at this point is that they're fast, and the speed is independent of the number or complexity of the patterns.