

<<编译器构造>>

图书基本信息

书名：<<编译器构造>>

13位ISBN编号：9787302227205

10位ISBN编号：7302227209

出版时间：2010-6

出版时间：清华大学出版社

作者：（美）费希尔，塞特朗，勒布兰 著

页数：683

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

前言

Much has changed since *Crafting a Compiler*, by Fischer and LeBlanc, was published in 1988. While instructors may remember the 5.25-inch floppy disk of software that accompanied that text, most students today have neither seen nor held such a disk. Many changes have occurred in the programming languages that students experience in class and in the marketplace. In 1991 the book was available in two forms, with algorithms presented in either C or Ada. While C remains a popular language, Ada has become relatively obscure and did not achieve its predicted popularity. The C++ language evolved from C with the addition of object-oriented features. Java™ was developed as a simpler object-oriented language, gaining popularity because of its security and ability to be run within a Web browser. The College Board Advanced Placement curriculum moved from Pascal to C++ to Java.

<<编译器构造>>

内容概要

本书是一本面向计算机系本科生的编译器教材。

作者在三所美国大学拥有长达25年的编译器教学经验，在本书中对编译器构造的基本知识与关键技术进行了全新的讲解。

本书的主要内容包括：编译器历史和概述、词法分析（扫描）、语法分析（包括自顶向下和自底向上的分析）、语法制导翻译、符号表和声明处理、语义分析、中间表示形式、虚拟机上的代码生成、运行时支持、目标代码生成和程序优化等。

本书提供了详尽清晰的算法，主推在实践中学习编译器构造的相关技术，同时提供了配合教材使用的教学网站、参考资料以及源码下载。

不仅可以作为计算机专业本科生或研究生的参考教材，同时也适合相关领域的软件工程师、系统分析师等作为参考资料。

<<编译器构造>>

书籍目录

1 Introduction 1	1.1 History of Compilation 2	1.2 What Compilers Do 4	1.2.1 Machine Code Generated by Compilers 4	1.2.2 Target Code Formats 7	1.3 Interpreters 9	1.4 Syntax and Semantics 10	1.4.1 Static Semantics 11	1.4.2 Runtime Semantics 12	1.5 Organization of a Compiler 14	1.5.1 The Scanner 16	1.5.2 The Parser 16	1.5.3 The Type Checker (Semantic Analysis) 17	1.5.4 Translator (Program Synthesis) 17	1.5.5 Symbol Tables 18	1.5.6 The Optimizer 18	1.5.7 The Code Generator 19	1.5.8 Compiler Writing Tools 19	1.6 Programming Language and Compiler Design 20	1.7 Computer Architecture and Compiler Design 21	1.8 Compiler Design Considerations 22	1.8.1 Debugging (Development) Compilers 22	1.8.2 Optimizing Compilers 23	1.8.3 Retargetable Compilers 23	1.9 Integrated Development Environments 24	Exercises 26	2 A Simple Compiler 31	2.1 An Informal Definition of the a Language 32	2.2 Formal Definition of a 33	2.2.1 Syntax Specification 33	2.2.2 Token Specification 36	2.3 Phases of a Simple Compiler 37	2.4 Scanning 38	2.5 Parsing 39	2.5.1 Predicting a Parsing Procedure 41	2.5.2 Implementing the Production 43	2.6 Abstract Syntax Trees 45	2.7 Semantic Analysis 46	2.7.1 Symbol Tables 47	2.7.2 Type Checking 48	2.8 Code Generation 51	Exercises 54	3 Scanning—Theory and Practice 57	3.1 Overview of a Scanner 58	3.2 Regular Expressions 60	3.3 Examples 62	3.4 Finite Automata and Scanners 64	3.4.1 Deterministic Finite Automata 65	3.5 The Lex Scanner Generator 69	3.5.1 Defining Tokens in Lex 70	3.5.2 The Character Class 71	3.5.3 Using Regular Expressions to Define Tokens 73	3.5.4 Character Processing Using Lex 76	3.6 Other Scanner Generators 77	3.7 Practical Considerations of Building Scanners 79	3.7.1 Processing Identifiers and Literals 79	3.7.2 Using Compiler Directives and Listing Source Lines 83	3.7.3 Terminating the Scanner 85	3.7.4 Multicharacter Lookahead 86	3.7.5 Performance Considerations 87	3.7.6 Lexical Error Recovery 89	3.8 Regular Expressions and Finite Automata 92	3.8.1 Transforming a Regular Expression into an NFA 93	3.8.2 Creating the DFA 94	3.8.3 Optimizing Finite Automata 97	3.8.4 Translating Finite Automata into Regular Expressions 100	3.9 Summary 103	Exercises 106	4 Grammars and Parsing 113	4.1 Context-Free Grammars 114	4.1.1 Leftmost Derivations 116	4.1.2 Rightmost Derivations 116	4.1.3 Parse Trees 117	4.1.4 Other Types of Grammars 118	4.2 Properties of CFGs 120	4.2.1 Reduced Grammars 120	4.2.2 Ambiguity 121	4.2.3 Faulty Language Definition 122	4.3 Transforming Extended Grammars 122	4.4 Parsers and Recognizers 123	4.5 Grammar Analysis Algorithms 127	4.5.1 Grammar Representation 127	4.5.2 Deriving the Empty String 128	4.5.3 First Sets 130	4.5.4 Follow Sets 134	Exercises 138	5 Top-Down Parsing 143	5.1 Overview 144	5.2 LL(k) Grammars 145	5.3 Recursive-Descent LL(1) Parsers 149	5.4 Table-Driven LL(1) Parsers 150	5.5 Obtaining LL(1) Grammars 154	5.5.1 Common Prefixes 156	5.5.2 Left Recursion 157	5.6 A Non-LL(1) Language 159	5.7 Properties of LL(1) Parsers 161	5.8 Parse Table Representation 163	5.8.1 Compaction 164	5.8.2 Compression 165	5.9 Syntactic Error Recovery and Repair 168	5.9.1 Error Recovery 169	5.9.2 Error Repair 169	5.9.3 Error Detection in LL(1) Parsers 171	5.9.4 Error Recovery in LL(1) Parsers 171	Exercises 173	6 Bottom-Up Parsing 179	6.1 Overview 180	6.2 Shift-Reduce Parsers 181	6.2.1 LR Parsers and Rightmost Derivations 182	6.2.2 LR Parsing as Knitting 182	6.2.3 LR Parsing Engine 184	6.2.4 The LR Parse Table 185	6.2.5 LR(k) Parsing 187	6.3 LR(0) Table Construction 191	6.4 Conflict Diagnosis 197	6.4.1 Ambiguous Grammars 199	6.4.2 Grammars that are not LR(k) 202	6.5 Conflict Resolution and Table Construction 204	6.5.1 SLR(k) Table Construction 204	6.5.2 LALR(k) Table Construction 209	6.5.3 LALR Propagation Graph 211	6.5.4 LR(k) Table Construction 219	Exercises 224	7 Syntax-Directed Translation 235	7.1 Overview 235	7.1.1 Semantic Actions and Values 236	7.1.2 Synthesized and Inherited Attributes 237	7.2 Bottom-Up Syntax-Directed Translation 239	7.2.1 Example 239	7.2.2 Rule Cloning 243	7.2.3 Forcing Semantic Actions 244	7.2.4 Aggressive Grammar Restructuring 246	7.3 Top-Down Syntax-Directed Translation 247	7.4 Abstract Syntax Trees 250	7.4.1 Concrete and Abstract Trees 250	7.4.2 An Efficient AST Data Structure 251	7.4.3 Infrastructure for Creating ASTs 252	7.5 AST Design and Construction 254	7.5.1 Design 256	7.5.2 Construction 258	7.6 AST Structures for Left and Right Values 261	7.7 Design Patterns for ASTs 264	7.7.1
------------------	------------------------------	-------------------------	---	-----------------------------	--------------------	-----------------------------	---------------------------	----------------------------	-----------------------------------	----------------------	---------------------	---	---	------------------------	------------------------	-----------------------------	---------------------------------	---	--	---------------------------------------	--	-------------------------------	---------------------------------	--	--------------	------------------------	---	-------------------------------	-------------------------------	------------------------------	------------------------------------	-----------------	----------------	---	--------------------------------------	------------------------------	--------------------------	------------------------	------------------------	------------------------	--------------	-----------------------------------	------------------------------	----------------------------	-----------------	-------------------------------------	--	----------------------------------	---------------------------------	------------------------------	---	---	---------------------------------	--	--	---	----------------------------------	-----------------------------------	-------------------------------------	---------------------------------	--	--	---------------------------	-------------------------------------	--	-----------------	---------------	----------------------------	-------------------------------	--------------------------------	---------------------------------	-----------------------	-----------------------------------	----------------------------	----------------------------	---------------------	--------------------------------------	--	---------------------------------	-------------------------------------	----------------------------------	-------------------------------------	----------------------	-----------------------	---------------	------------------------	------------------	------------------------	---	------------------------------------	----------------------------------	---------------------------	--------------------------	------------------------------	-------------------------------------	------------------------------------	----------------------	-----------------------	---	--------------------------	------------------------	--	---	---------------	-------------------------	------------------	------------------------------	--	----------------------------------	-----------------------------	------------------------------	-------------------------	----------------------------------	----------------------------	------------------------------	---------------------------------------	--	-------------------------------------	--------------------------------------	----------------------------------	------------------------------------	---------------	-----------------------------------	------------------	---------------------------------------	--	---	-------------------	------------------------	------------------------------------	--	--	-------------------------------	---------------------------------------	---	--	-------------------------------------	------------------	------------------------	--	----------------------------------	-------

<<编译器构造>>

Node ClassHierarchy 264 7.7.2 Visitor Pattern 265 7.7.3 Re.ectiveVisitorPattern 268 Exercises 272 8 Symbol
 Tables and Declaration Processing 279 8.1 Constructing aSymbolTable 280 8.1.1 Static Scoping 282 8.1.2 A
 Symbol Table Interface 282 8.2 Block-StructuredLanguages andScopes 284 8.2.1 Handling Scopes 284 8.2.2
 OneSymbolTable orMany? 285 8.3 Basic Implementation Techniques 286 8.3.1 Entering andFindingNames
 286 8.3.2 TheName Space 289 8.3.3 An Ef.cientSymbol Table Implementation 290 8.4 Advanced Features
 293 8.4.1 Records and Typenames 294 8.4.2 Overloading andTypeHierarchies 294 8.4.3 Implicit
 Declarations 296 8.4.4 Export andImportDirectives 296 8.4.5 Altered SearchRules 297 8.5 Declaration
 ProcessingFundamentals 298 8.5.1 Attributes in the Symbol Table 298 8.5.2 TypeDescriptorStructures 299
 8.5.3 TypeCheckingUsing anAbstractSyntaxTree 300 8.6 Variable andTypeDeclarations 303 8.6.1 Simple
 Variable Declarations 303 8.6.2 Handling TypeNames 304 8.6.3 TypeDeclarations 305 8.6.4 Variable
 DeclarationsRevisited 308 8.6.5 Static ArrayTypes 311 8.6.6 Struct and RecordTypes 312 8.6.7
 Enumeration Types 313 8.7 Class and Method Declarations 316 8.7.1 ProcessingClassDeclarations 317
 8.7.2 ProcessingMethod Declarations 321 8.8 An Introduction toTypeChecking 323 8.8.1 Simple Identifiers
 and Literals 327 8.8.2 Assignment Statements 328 8.8.3 Checking Expressions 328 8.8.4 Checking
 ComplexNames 329 8.9 Summary 334 Exercises 336 9 Semantic Analysis 343 9.1 Semantic
 AnalysisforControl Structures 343 9.1.1 Reachability and Termination Analysis 345 9.1.2 IfStatements 348
 9.1.3 While, Do, andRepeat Loops 350 9.1.4 ForLoops 353 9.1.5 Break,Continue, Return, andGoto
 Statements 356 9.1.6 Switch andCaseStatements 364 9.1.7 Exception Handling 369 9.2 Semantic Analysis
 ofCalls 376 9.3 Summary 384 Exercises 385 10 Intermediate Representations 391 10.1 Overview 392 10.1.1
 Examples 393 10.1.2 TheMiddle-End 395 10.2 Java Virtual Machine 397 10.2.1 Introduction
 andDesignPrinciples 398 10.2.2 Contents of aClassFile 399 10.2.3 JVMInstructions 401 10.3 Static Single
 Assignment Form 410 10.3.1 Renaming and -functions 411 Exercises 414 11 Code Generation for a Virtual
 Machine 417 11.1 Visitors forCode Generation 418 11.2 Class and Method Declarations 420 11.2.1
 ClassDeclarations 422 11.2.2 Method Declarations 424 11.3 The MethodBodyVisitor 425 11.3.1 Constants
 425 11.3.2 References to LocalStorage 426 11.3.3 Static References 427 11.3.4 Expressions 427 11.3.5
 Assignment 429 11.3.6 Method Calls 430 11.3.7 Field References 432 11.3.8 ArrayReferences 433 11.3.9
 Conditional Execution 435 11.3.10Loops 436 11.4 The LHSVisitor 437 11.4.1 Local References 437 11.4.2
 Static References 438 11.4.3 Field References 439 11.4.4 ArrayReferences 439 Exercises 441 12 Runtime
 Support 445 12.1 Static Allocation 446 12.2 Stack Allocation 447 12.2.1 Field AccessinClasses andStructs
 449 12.2.2 AccessingFrames at Runtime 450 12.2.3 Handling Classes and Objects 451 12.2.4 Handling
 Multiple Scopes 453 12.2.5 Block-LevelAllocation 455 12.2.6 MoreAbout Frames 457 12.3 Arrays 460
 12.3.1 Static One-Dimensional Arrays 460 12.3.2 Multidimensional Arrays 465 12.4 Heap Management
 468 12.4.1 Allocation Mechanisms 468 12.4.2 Deallocation Mechanisms 471 12.4.3 Automatic
 GarbageCollection 472 12.5 Region-Based MemoryManagement 479 Exercises 482 13 Target Code Generation
 489 13.1 Translating Bytecodes 490 13.1.1 Allocating memory addresses 493 13.1.2 Allocating Arrays
 andObjects 493 13.1.3 Method Calls 496 13.1.4 Example ofBytecodeTranslation 498 13.2 Translating
 ExpressionTrees 501 13.3 Register Allocation 505 13.3.1 On-the-FlyRegister Allocation 506 13.3.2
 RegisterAllocation Using GraphColoring 508 13.3.3 Priority-Based RegisterAllocation 516 13.3.4
 Interprocedural RegisterAllocation 517 13.4 Code Scheduling 519 13.4.1 ImprovingCode Scheduling 523
 13.4.2 Global andDynamicCodeScheduling 524 13.5 Automatic Instruction Selection 526 13.5.1
 InstructionSelection UsingBURS 529 13.5.2 InstructionSelection UsingTwig 531 13.5.3 OtherApproaches 532
 13.6 Peephole Optimization 532 13.6.1 Levels ofPeepholeOptimization 533 13.6.2 AutomaticGeneration
 ofPeepholeOptimizers 536 Exercises 538 14 Program Optimization 547 14.1 Overview 548 14.1.1
 WhyOptimize? 549 14.2 Control FlowAnalysis 555 14.2.1 Control FlowGraphs 556 14.2.2 Program
 andControlFlowStructure 559 14.2.3 DirectProcedureCall Graphs 560 14.2.4 Depth-FirstSpanning Tree 560
 14.2.5 Dominance 565 14.2.6 Simple Dominance Algorithm 567 14.2.7 FastDominanceAlgorithm 571
 14.2.8 Dominance Frontiers 581 14.2.9 Intervals 585 14.3 Introduction to DataFlowAnalysis 598 14.3.1

<<编译器构造>>

Available Expressions 598 14.3.2 LiveVariables 601 14.4 Data FlowFrameworks 604 14.4.1 Data
FlowEvaluation Graph 604 14.4.2 Meet Lattice 606 14.4.3 TransferFunctions 608 14.5 Evaluation 611
14.5.1 Iteration 611 14.5.2 Initialization 615 14.5.3 Termination andRapidFrameworks 616 14.5.4
Distributive Frameworks 620 14.6 Constant Propagation 623 14.7 SSA Form 627 14.7.1 Placing
-Functions 629 14.7.2 Renaming 631 Exercises 636 Bibliography 651 Abbreviations 661 Pseudocode Guide
663 Index

章节摘录

An optimizing compiler is specially designed to produce efficient target code at the cost of increased compiler complexity and possibly increased compilation times. In practice, all production-quality compilers (those whose output will be used in everyday work) make some effort to generate reasonable target code. For example, no add instruction would normally be generated for the expression $i+0$. The term optimizing compiler is actually a misnomer. This is because no compiler of any sophistication can produce optimal code for all programs. The reason for this is twofold. First, theoretical computer science has shown that even so simple a question as whether two programs are equivalent is undecidable: such questions cannot generally be answered by any computer program. Thus finding the simplest (and most efficient) translation of a program cannot always be done. Second, many program optimizations require time proportional to an exponential function of the size of the program being compiled. Thus, optimal code, even when theoretically possible, is often infeasible in practice.

<<编译器构造>>

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>