

1. Record Nr.	UNINA9910132468303321
Autore	Abelson Harold
Titolo	Structure and interpretation of computer programs / / Harold Abelson and Gerald Jay Sussman, with Julie Sussman ; foreword by Alan J. Perlis
Pubbl/distr/stampa	London : , : MIT Press, , [1996] ©1996
Edizione	[Second edition.]
Descrizione fisica	1 online resource (xxiii, 657 pages) : illustrations
Disciplina	005.13/3
Soggetti	Computer programming LISP (Computer program language)
Lingua di pubblicazione	Inglese
Formato	Materiale a stampa
Livello bibliografico	Monografia
Nota di contenuto	Foreword xi -- Preface to the Second Edition xv -- Preface to the First Edition xvii -- Acknowledgments xxi -- 1 Building Abstractions with Procedures 1 (78) -- 1.1 The Elements of Programming 4 (27) -- 1.1.1 Expressions 5 (2) -- 1.1.2 Naming and the Environment 7 (2) -- 1.1.3 Evaluating Combinations 9 (2) -- 1.1.4 Compound Procedures 11 (2) -- 1.1.5 The Substitution Model for Procedure Application 13 (4) -- 1.1.6 Conditional Expressions and Predicates 17 (4) -- 1.1.7 Example: Square Roots by Newton's Method 21 (5) -- 1.1.8 Procedures as Black-Box Abstractions 26 (5) -- 1.2 Procedures and the Processes They Generate 31 (25) -- 1.2.1 Linear Recursion and Iteration 32 (5) -- 1.2.2 Tree Recursion 37 (5) -- 1.2.3 Orders of Growth 42 (2) -- 1.2.4 Exponentiation 44 (4) -- 1.2.5 Greatest Common Divisors 48 (2) -- 1.2.6 Example: Testing for Primality 50 (6) -- 1.3 Formulating Abstractions with Higher-Order Procedures 56 (23) -- 1.3.1 Procedures as Arguments 57 (5) -- 1.3.2 Constructing Procedures Using Lambda 62 (4) -- 1.3.3 Procedures as General Methods 66 (6) -- 1.3.4 Procedures as Returned Values 72 (7) -- 2 Building Abstractions with Data 79 (138) -- 2.1 Introduction to Data Abstraction 83 (14) -- 2.1.1 Example: Arithmetic Operations for Rational Numbers 83 (4) -- 2.1.2 Abstraction Barriers 87 (3) -- 2.1.3 What Is Meant by Data? 90 (3) -- 2.1.4 Extended Exercise: Interval Arithmetic 93 (4) -- 2.2 Hierarchical Data and the Closure Property 97 (45) -- 2.2.1 Representing Sequences

99 (8) -- 2.2.2 Hierarchical Structures 107 (6) -- 2.2.3 Sequences as Conventional Interfaces 113 (13) -- 2.2.4 Example: A Picture Language 126 (16) -- 2.3 Symbolic Data 142 (27) -- 2.3.1 Quotation 142 (3) -- 2.3.2 Example: Symbolic Differentiation 145 (6) -- 2.3.3 Example: Representing Sets 151 (10) -- 2.3.4 Example: Huffman Encoding Trees 161 (8) -- 2.4 Multiple Representations for Abstract Data 169 (18) -- 2.4.1 Representations for Complex Numbers 171 (4) -- 2.4.2 Tagged data 175 (4) -- 2.4.3 Data-Directed Programming and Additivity 179 (8) -- 2.5 Systems with Generic Operations 187 (30) -- 2.5.1 Generic Arithmetic Operations 189 (4) -- 2.5.2 Combining Data of Different Types 193 (9) -- 2.5.3 Example: Symbolic Algebra 202 (15) -- 3 Modularity, Objects, and State 217 (142) -- 3.1 Assignment and Local State 218 (18) -- 3.1.1 Local State Variables 219 (6) -- 3.1.2 The Benefits of Introducing Assignment 225 (4) -- 3.1.3 The Costs of Introducing Assignment 229 (7) -- 3.2 The Environment Model of Evaluation 236 (15) -- 3.2.1 The Rules for Evaluation 238 (3) -- 3.2.2 Applying Simple Procedures 241 (3) -- 3.2.3 Frames as the Repository of Local State 244 (5) -- 3.2.4 Internal Definitions 249 (2) -- 3.3 Modeling with Mutable Data 251 (46) -- 3.3.1 Mutable List Structure 252 (9) -- 3.3.2 Representing Queues 261 (5) -- 3.3.3 Representing Tables 266 (7) -- 3.3.4 A Simulator for Digital Circuits 273 (12) -- 3.3.5 Propagation of Constraints 285 (12) -- 3.4 Concurrency: Time Is of the Essence 297 (19) -- 3.4.1 The Nature of Time in Concurrent Systems 298 (5) -- 3.4.2 Mechanisms for Controlling Concurrency 303 (13) -- 3.5 Streams 316 (43) -- 3.5.1 Streams Are Delayed Lists 317 (9) -- 3.5.2 Infinite Streams 326 (8) -- 3.5.3 Exploiting the Stream Paradigm 334 (12) -- 3.5.4 Streams and Delayed Evaluation 346 (6) -- 3.5.5 Modularity of Functional Programs and Modularity of Objects 352 (7) -- 4 Metalinguistic Abstraction 359 (132) -- 4.1 The Metacircular Evaluator 362 (36) -- 4.1.1 The Core of the Evaluator 364 (4) -- 4.1.2 Representing Expressions 368 (8) -- 4.1.3 Evaluator Data Structures 376 (5) -- 4.1.4 Running the Evaluator as a Program 381 (3) -- 4.1.5 Data as Programs 384 (4) -- 4.1.6 Internal Definitions 388 (5) -- 4.1.7 Separating Syntactic Analysis from Execution 393 (5) -- 4.2 Variations on a Scheme Lazy Evaluation 398 (14) -- 4.2.1 Normal Order and Applicative Order 399 (2) -- 4.2.2 An Interpreter with Lazy Evaluation 401 (8) -- 4.2.3 Streams as Lazy Lists 409 (3) -- 4.3 Variations on a Scheme Nondeterministic Computing 412 (26) -- 4.3.1 Amb and Search 414 (4) -- 4.3.2 Examples of Nondeterministic Programs 418 (8) -- 4.3.3 Implementing the Amb Evaluator 426 (12) -- 4.4 Logic Programming 438 (53) -- 4.4.1 Deductive Information Retrieval 441 (12) -- 4.4.2 How the Query System Works 453 (9) -- 4.4.3 Is Logic Programming Mathematical Logic? 462 (6) -- 4.4.4 Implementing the Query System 468 (23) -- 5 Computing with Register Machines 491 (120) -- 5.1 Designing Register Machines 492 (21) -- 5.1.1 A Language for Describing Register Machines 494 (5) -- 5.1.2 Abstraction in Machine Design 499 (3) -- 5.1.3 Subroutines 502 (4) -- 5.1.4 Using a Stack to Implement Recursion 506 (6) -- 5.1.5 Instruction Summary 512 (1) -- 5.2 A Register-Machine Simulator 513 (20) -- 5.2.1 The Machine Model 515 (5) -- 5.2.2 The Assembler 520 (3) -- 5.2.3 Generating Execution Procedures for Instructions 523 (7) -- 5.2.4 Monitoring Machine Performance 530 (3) -- 5.3 Storage Allocation and Garbage Collection 533 (14) -- 5.3.1 Memory as Vectors 534 (6) -- 5.3.2 Maintaining the Illusion of Infinite Memory 540 (7) -- 5.4 The Explicit-Control Evaluator 547 (19) -- 5.4.1 The Core of the Explicit-Control Evaluator 549 (6) -- 5.4.2 Sequence Evaluation and Tail Recursion 555 (3) -- 5.4.3 Conditionals, Assignments, and Definitions 558 (2) -- 5.4.4 Running the Evaluator 560 (6) -- 5.5 Compilation 566

(45) -- 5.5.1 Structure of the Compiler 569 (5) -- 5.5.2 Compiling Expressions 574 (7) -- 5.5.3 Compiling Combinations 581 (6) -- 5.5.4 Combining Instruction Sequences 587 (4) -- 5.5.5 An Example of Compiled Code 591 (9) -- 5.5.6 Lexical Addressing 600 (3) -- 5.5.7 Interfacing Compiled Code to the Evaluator -- References 611 (8) -- List of Exercises 619 (2) -- Index 621.

Sommario/riassunto

Structure and Interpretation of Computer Programs has had a dramatic impact on computer science curricula over the past decade. This long-awaited revision contains changes throughout the text. There are new implementations of most of the major programming systems in the book, including the interpreters and compilers, and the authors have incorporated many small changes that reflect their experience teaching the course at MIT since the first edition was published. A new theme has been introduced that emphasizes the central role played by different approaches to dealing with time in computational models: objects with state, concurrent programming, functional programming and lazy evaluation, and nondeterministic programming. There are new example sections on higher-order procedures in graphics and on applications of stream processing in numerical programming, and many new exercises. In addition, all the programs have been reworked to run in any Scheme implementation that adheres to the IEEE standard.
