1. Record Nr. UNISA996499859403316

Autore	Ægidius Mogensen Torben
Titolo	Programming language design and implementation / / Torben Æ. Mogensen
Pubbl/distr/stampa	Cham, Switzerland : , : Springer, , [2022] ©2022
ISBN	9783031118067 9783031118050
Descrizione fisica	1 online resource (333 pages)
Collana	Texts in computer science
Disciplina	001.642
Soggetti	Computer programming Programming languages (Electronic computers)
Lingua di pubblicazione	Inglese
Formato	Materiale a stampa
Livello bibliografico	Monografia
Note generali	Includes index.
Nota di contenuto	Intro -- Preface -- Do We Need New Programming Languages? -- Weak Languages -- General Design Principles -- To the Reader -- Contents -- List of Figures -- 1 A Brief History of Programming Languages -- 1.1 Before Computers: Turing Machines and Lambda Calculus -- 1.2 Programmable Electronic Computers -- 1.3 Early and Influential Programming Languages -- 1.3.1 Plankalkül -- 1.3.2 FORTRAN -- 1.3.3 LISP -- 1.3.4 COBOL -- 1.3.5 ALGOL 60 -- 1.3.6 APL -- 1.3.7 PL/I -- 1.3.8 BASIC -- 1.3.9 Simula -- 1.3.10 Pascal -- 1.3.11 C -- 1.3.12 Prolog -- 1.3.13 ISWIM and ML -- 1.4 Further Reading -- 1.5 Exercises -- 2 Implementation Strategies -- 2.1 Compilation and Interpretation -- 2.2 REPLs and IDEs -- 2.3 Intermediate Code and Virtual Machines -- 2.4 Hybrid Methods -- 2.5 Cross Compilers, Reverse Compilers, and Obfuscation -- 2.6 Bootstrapping -- 2.6.1 Notation -- 2.6.2 Compiling Compilers -- 2.6.3 Full Bootstrap -- 2.6.4 Choosing the Language in Which to Write a Compiler -- 2.7 How Implementation Techniques can Influence Language Design -- 2.8 Further Reading -- 2.9 Exercises -- 3 Syntax -- 3.1 Lexical Elements -- 3.1.1 Character Sets -- 3.1.2 Case Sensitivity -- 3.1.3 Identifiers -- 3.1.4 Whitespace -- 3.1.5 Comments -- 3.1.6 Reserved Symbols -- 3.1.7 Separation of Tokens -- 3.1.8 Summary -- 3.2 Grammatical Elements -- 3.2.1 Line-Based Syntax -- 3.2.2 Multi-line Syntax --

3.2.3 Syntax that Looks Like a Natural Language -- 3.2.4 Bracketed Syntax -- 3.2.5 Prefix, Post Fix and Operator-Precedence Syntax -- 3.2.6 Context-Free Syntax -- 3.2.7 Stronger Grammar Formalisms -- 3.2.8 Other Syntactic Considerations -- 3.2.9 Bracketing Symbols -- 3.3 Concerns that Span Both Lexing and Grammar -- 3.3.1 Macros -- 3.3.2 Visual Languages -- 3.4 Considerations When Designing Syntax -- 3.5 Further Reading -- 3.6 Exercises -- 4 Memory Management -- 4.1 Introduction.
4.2 Static Allocation -- 4.2.1 Limitations -- 4.3 Stack Allocation -- 4.4 Heap Allocation -- 4.5 Manual Memory Management -- 4.5.1 A Simple Implementation of malloc() and free() -- 4.5.2 Joining Freed Blocks -- 4.5.3 Sorting by Block Size -- 4.5.4 Large Objects -- 4.5.5 Summary of Manual Memory Management -- 4.6 Automatic Memory Management -- 4.7 Reference Counting -- 4.8 Tracing Garbage Collectors -- 4.8.1 Mark-Sweep Collection -- 4.8.2 Two-Space Collection -- 4.8.3 Generational and Concurrent Collectors -- 4.9 Summary of Automatic Memory Management -- 4.10 Memory Management and Language Design -- 4.11 Further Reading -- 4.12 Exercises -- 5 Scopes, Functions, and Parameter Passing -- 5.1 Scope Rules -- 5.1.1 Global Scoping -- 5.1.2 Local Variables Only -- 5.1.3 Block Structure -- 5.1.4 Nested Function Declarations -- 5.1.5 Recursion -- 5.1.6 Macros -- 5.1.7 Parameter-Passing Methods -- 5.2 Implementing Functions and Function Calls -- 5.2.1 Summary of Implementing Function Calls -- 5.2.2 C-Style Functions -- 5.2.3 Nested Function Declarations -- 5.3 Functions as Parameters -- 5.4 First-Class Functions -- 5.5 Functional Programming Languages -- 5.5.1 Impure Functional Languages -- 5.5.2 Defunctionalisation -- 5.5.3 Pure Functional Languages -- 5.5.4 Lazy Functional Languages -- 5.5.5 Strictness Analysis -- 5.6 Exceptions -- 5.6.1 Tagged Return -- 5.6.2 Stack Unwinding -- 5.6.3 Using a Handler Stack -- 5.7 Further Reading -- 5.8 Exercises -- 6 Control Structures -- 6.1 Jumps -- 6.2 Structured Control -- 6.2.1 Conditionals -- 6.2.2 Loops -- 6.2.3 Whole-Collection Operations -- 6.2.4 Break and Continue -- 6.2.5 Structured Versus Unstructured Control -- 6.3 Exceptions and Continuations -- 6.3.1 Continuations -- 6.4 Function Calls as Control -- 6.5 Multithreading -- 6.5.1 Coroutines -- 6.5.2 Threads -- 6.5.3 Message Passing -- 6.6 Exercises -- 7 Types.
7.1 Checking Types -- 7.2 Type Conversion -- 7.3 Atomic and Composite Types -- 7.3.1 Numbers -- 7.3.2 Characters -- 7.3.3 Boolean Values -- 7.3.4 Enumerated Types and Symbols -- 7.3.5 Product Types -- 7.3.6 Records and Structs -- 7.3.7 Collection Types -- 7.3.8 Union and Sum Types -- 7.3.9 Function Types -- 7.3.10 Recursive Types -- 7.3.11 Named Types and Type Equivalence -- 7.4 Polymorphism -- 7.4.1 Ad hoc Polymorphism -- 7.4.2 Interface Polymorphism -- 7.4.3 Subtype Polymorphism -- 7.4.4 Parametric Polymorphism -- 7.4.5 Polymorphic Type Inference -- 7.4.6 Polymorphism in Various Languages -- 7.5 Further Reading -- 7.6 Exercises -- 8 Modularisation -- 8.1 Simple Modules -- 8.1.1 Shared Modules -- 8.2 Modules with Abstraction -- 8.3 Name Spaces -- 8.3.1 Nested Modules -- 8.4 Modules and Classes -- 8.5 Modules as Parameters/Values -- 8.6 Further Reading -- 8.7 Exercises -- 9 Language Paradigms -- 9.1 What Is a Language Paradigm? -- 9.1.1 Data Flow -- 9.1.2 Execution Order -- 9.1.3 Structuring -- 9.1.4 Nomenclature -- 9.2 A Closer Look at Some Paradigms -- 9.3 Object-Oriented Languages -- 9.3.1 Classes and Objects -- 9.3.2 Single Inheritance -- 9.3.3 Multiple Inheritance -- 9.3.4 Prototype-Based Languages -- 9.4 Logic Languages -- 9.4.1 Pure Prolog -- 9.4.2 List Notation -- 9.4.3 Resolution -- 9.4.4 Full Prolog -- 9.4.5 Other Logic