Concepts in Programming Languages

This textbook for undergraduate and beginning graduate students explains and examines the central concepts used in modern programming languages, such as functions, types, memory management, and control. The book is unique in its comprehensive presentation and comparison of major object-oriented programming languages. Separate chapters examine the history of objects, Simula and Smalltalk, and the prominent languages C++ and Java.

The author presents foundational topics, such as lambda calculus and denotational semantics, in an easy-to-read, informal style, focusing on the main insights provided by these theories. Advanced topics include concurrency and concurrent object-oriented programming. A chapter on logic programming illustrates the importance of specialized programming methods for certain kinds of problems.

This book will give the reader a better understanding of the issues and trade-offs that arise in programming language design and a better appreciation of the advantages and pitfalls of the programming languages they use.

John C. Mitchell is Professor of Computer Science at Stanford University, where he has been a popular teacher for more than a decade. Many of his former students are successful in research and private industry. He received his Ph.D. from MIT in 1984 and was a Member of Technical Staff at AT&T Bell Laboratories before joining the faculty at Stanford. Over the past twenty years, Mitchell has been a featured speaker at international conferences; has led research projects on a variety of topics, including programming language design and analysis, computer security, and applications of mathematical logic to computer science; and has written more than 100 research articles. His previous textbook, Foundations for Programming Languages (MIT Press, 1996), covers lambda calculus, type systems, logic for program verification, and mathematical semantics of programming languages. Professor Mitchell was a member of the programming language subcommittee of the ACM/IEEE Curriculum 2001 standardization effort and the 2002 Program Chair of the ACM Principles of Programming Languages conference.
CONCEPTS IN

PROGRAMMING

LANGUAGES

John C. Mitchell
Stanford University
## Contents

**Preface**  
ix

### Part 1  Functions and Foundations

1  Introduction  
1.1  Programming Languages  
1.2  Goals  
1.3  Programming Language History  
1.4  Organization: Concepts and Languages  
3

2  Computability  
2.1  Partial Functions and Computability  
2.2  Chapter Summary  
10

3  Lisp: Functions, Recursion, and Lists  
3.1  Lisp History  
3.2  Good Language Design  
3.3  Brief Language Overview  
3.4  Innovations in the Design of Lisp  
3.5  Chapter Summary: Contributions of Lisp  
18

4  Fundamentals  
4.1  Compilers and Syntax  
4.2  Lambda Calculus  
4.3  Denotational Semantics  
4.4  Functional and Imperative Languages  
4.5  Chapter Summary  
48
## Contents

### Part 2 Procedures, Types, Memory Management, and Control

5 The Algol Family and ML  
5.1 The Algol Family of Programming Languages 93  
5.2 The Development of C 99  
5.3 The LCF System and ML 101  
5.4 The ML Programming Language 103  
5.5 Chapter Summary Exercises 121  

6 Type Systems and Type Inference 129  
6.1 Types in Programming 129  
6.2 Type Safety and Type Checking 132  
6.3 Type Inference 135  
6.4 Polymorphism and Overloading 145  
6.5 Type Declarations and Type Equality 151  
6.6 Chapter Summary Exercises 155  

7 Scope, Functions, and Storage Management 162  
7.1 Block-Structured Languages 162  
7.2 In-Line Blocks 165  
7.3 Functions and Procedures 170  
7.4 Higher-Order Functions 182  
7.5 Chapter Summary Exercises 190  

8 Control in Sequential Languages 204  
8.1 Structured Control 204  
8.2 Exceptions 207  
8.3 Continuations 218  
8.4 Functions and Evaluation Order 223  
8.5 Chapter Summary Exercises 227  

### Part 3 Modularity, Abstraction, and Object-Oriented Programming

9 Data Abstraction and Modularity 235  
9.1 Structured Programming 235  
9.2 Language Support for Abstraction 242  
9.3 Modules 252  
9.4 Generic Abstractions 259  
9.5 Chapter Summary Exercises 269  

10 Concepts in Object-Oriented Languages 277  
10.1 Object-Oriented Design 277  
10.2 Four Basic Concepts in Object-Oriented Languages 278
11 History of Objects: Simula and Smalltalk  
11.1 Origin of Objects in Simula  
11.2 Objects in Simula  
11.3 Subclasses and Subtypes in Simula  
11.4 Development of Smalltalk  
11.5 Smalltalk Language Features  
11.6 Smalltalk Flexibility  
11.7 Relationship between Subtyping and Inheritance  
11.8 Chapter Summary  
Exercises

12 Objects and Run-Time Efficiency: C++  
12.1 Design Goals and Constraints  
12.2 Overview of C++  
12.3 Classes, Inheritance, and Virtual Functions  
12.4 Subtyping  
12.5 Multiple Inheritance  
12.6 Chapter Summary  
Exercises

13 Portability and Safety: Java  
13.1 Java Language Overview  
13.2 Java Classes and Inheritance  
13.3 Java Types and Subtyping  
13.4 Java System Architecture  
13.5 Security Features  
13.6 Java Summary  
Exercises

Part 4 Concurrency and Logic Programming  
14 Concurrent and Distributed Programming  
14.1 Basic Concepts in Concurrency  
14.2 The Actor Model  
14.3 Concurrent ML  
14.4 Java Concurrency  
14.5 Chapter Summary  
Exercises
## Contents

15 The Logic Programming Paradigm and Prolog  
  15.1 History of Logic Programming  
  15.2 Brief Overview of the Logic Programming Paradigm  
  15.3 Equations Solved by Unification as Atomic Actions  
  15.4 Clauses as Parts of Procedure Declarations  
  15.5 Prolog’s Approach to Programming  
  15.6 Arithmetic in Prolog  
  15.7 Control, Ambivalent Syntax, and Meta-Variables  
  15.8 Assessment of Prolog  
  15.9 Bibliographic Remarks  
  15.10 Chapter Summary

Appendix A Additional Program Examples  
  A.1 Procedural and Object-Oriented Organization

Glossary  
Index
Preface

A good programming language is a conceptual universe for thinking about programming.
Alan Perlis, NATO Conference on Software Engineering Techniques, Rome, 1969

Programming languages provide the abstractions, organizing principles, and control structures that programmers use to write good programs. This book is about the concepts that appear in programming languages, issues that arise in their implementation, and the way that language design affects program development. The text is divided into four parts:

- **Part 1:** Functions and Foundations
- **Part 2:** Procedures, Types, Memory Management, and Control
- **Part 3:** Modularity, Abstraction, and Object-Oriented Programming
- **Part 4:** Concurrency and Logic Programming

Part 1 contains a short study of Lisp as a worked example of programming language analysis and covers compiler structure, parsing, lambda calculus, and denotational semantics. A short Computability chapter provides information about the limits of compile-time program analysis and optimization.

Part 2 uses procedural Algol family languages and ML to study types, memory management, and control structures.

In Part 3 we look at program organization using abstract data types, modules, and objects. Because object-oriented programming is the most prominent paradigm in current practice, several different object-oriented languages are compared. Separate chapters explore and compare Simula, Smalltalk, C++, and Java.

Part 4 contains chapters on language mechanisms for concurrency and on logic programming.

The book is intended for upper-level undergraduate students and beginning graduate students with some knowledge of basic programming. Students are expected to have some knowledge of C or some other procedural language and some
Preface

acquaintance with C++ or some form of object-oriented language. Some experience with Lisp, Scheme, or ML is helpful in Parts 1 and 2, although many students have successfully completed the course based on this book without this background. It is also helpful if students have some experience with simple analysis of algorithms and data structures. For example, in comparing implementations of certain constructs, it will be useful to distinguish between algorithms of constant-, polynomial-, and exponential-time complexity.

After reading this book, students will have a better understanding of the range of programming languages that have been used over the past 40 years, a better understanding of the issues and trade-offs that arise in programming language design, and a better appreciation of the advantages and pitfalls of the programming languages they use. Because different languages present different programming concepts, students will be able to improve their programming by importing ideas from other languages into the programs they write.

Acknowledgments

This book developed as a set of notes for Stanford CS 242, a course in programming languages that I have taught since 1993. Each year, energetic teaching assistants have helped debug example programs for lectures, formulate homework problems, and prepare model solutions. The organization and content of the course have been improved greatly by their suggestions. Special thanks go to Kathleen Fisher, who was a teaching assistant in 1993 and 1994 and taught the course in my absence in 1995. Kathleen helped me organize the material in the early years and, in 1995, transcribed my handwritten notes into online form. Thanks to Amit Patel for his initiative in organizing homework assignments and solutions and to Vitaly Shmatikov for persevering with the glossary of programming language terms. Anne Bracy, Dan Bentley, and Stephen Freund thoughtfully proofread many chapters.

Lauren Cowles, Alan Harvey, and David Tranah of Cambridge University Press were encouraging and helpful. I particularly appreciate Lauren’s careful reading and detailed comments of twelve full chapters in draft form. Thanks also are due to the reviewers they enlisted, who made a number of helpful suggestions on early versions of the book. Zena Ariola taught from book drafts at the University of Oregon several years in a row and sent many helpful suggestions; other test instructors also provided helpful feedback.

Finally, special thanks to Krzysztof Apt for contributing a chapter on logic programming.

John Mitchell