

Buy in [print](#) and [eBook](#).

Table of Contents

Prologue

I. Language Concepts

II. Tools and Techniques

III. The Runtime System

19. Foreign Function Interface

20. Memory Representation of Values

21. Understanding the Garbage Collector

22. The Compiler Frontend: Parsing and Type Checking

23. The Compiler Backend: Bytecode and Native code

Index

Login with GitHub to view
and add comments

Part III. The Runtime System

19. Foreign Function Interface

Example: A Terminal Interface

Basic Scalar C Types

Pointers and Arrays

Allocating Typed Memory for Pointers

Using Views to Map Complex Values

Structs and Unions

Defining a Structure

Adding Fields to Structures

Incomplete Structure Definitions

Recap: A time-printing command

Defining Arrays

Passing Functions to C

Example: A Command-Line Quicksort

Learning More About C Bindings

Struct Memory Layout

20. Memory Representation of Values

OCaml Blocks and Values

Distinguishing Integers and Pointers at Runtime

Blocks and Values

Integers, Characters, and Other Basic Types

Tuples, Records, and Arrays

Floating-Point Numbers and Arrays

Variants and Lists

Polymorphic Variants

String Values

Custom Heap Blocks

Managing External Memory with Bigarray

21. Understanding the Garbage Collector

Mark and Sweep Garbage Collection

Generational Garbage Collection

The Fast Minor Heap

Allocating on the Minor Heap

The Long-Lived Major Heap

Allocating on the Major Heap

Memory Allocation Strategies

Next-fit allocation

First-fit allocation

Marking and Scanning the Heap

Heap Compaction

Intergenerational Pointers

The mutable write barrier

Attaching Finalizer Functions to Values

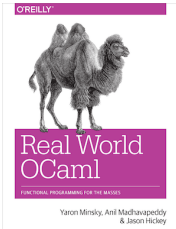
22. The Compiler Frontend: Parsing and Type Checking

An Overview of the Toolchain

Parsing Source Code

Syntax Errors

Automatically Indenting Source Code



Buy in [print](#) and [eBook](#).

Table of Contents

Prologue

I. Language Concepts

II. Tools and Techniques

III. The Runtime System

- 19. Foreign Function Interface
- 20. Memory Representation of Values
- 21. Understanding the Garbage Collector
- 22. The Compiler Frontend: Parsing and Type Checking
- 23. The Compiler Backend: Bytecode and Native code

Index

Login with GitHub to view
and add comments

Preprocessing Source Code

- Using Camlp4 Interactively
- Running Camlp4 from the Command Line
- Preprocessing Module Signatures
- Further Reading on Camlp4

Static Type Checking

- Displaying Inferred Types from the Compiler
- Type Inference
 - Adding type annotations to find errors
 - Enforcing principal typing
- Modules and Separate Compilation
 - The mapping between files and modules
 - Defining a module search path
- Packing Modules Together
- Shorter Module Paths in Type Errors

The Typed Syntax Tree

- Using ocp-index for Autocompletion
- Examining the Typed Syntax Tree Directly

23. The Compiler Backend: Bytecode and Native code

The Untyped Lambda Form

- Pattern Matching Optimization
- Benchmarking Pattern Matching

Generating Portable Bytecode

- Compiling and Linking Bytecode
- Executing Bytecode
- Embedding OCaml Bytecode in C

Compiling Fast Native Code

- Inspecting Assembly Output
 - The impact of polymorphic comparison
 - Benchmarking polymorphic comparison
- Debugging Native Code Binaries
 - Understanding name mangling
 - Interactive breakpoints with the GNU debugger
- Profiling Native Code
 - Gprof
 - Perf
- Embedding Native Code in C

Summarizing the File Extensions

[< Previous](#)

[Next >](#)