Reading Advice for EDAG01 Efficient C 2023

The relative importance of each part of the book for the exam is rated as follows:

3 Everyone should understand this.

4 Advanced.

5 More advanced.

R Reference: read it if you need it but I will not ask about details.

S Skip it for the EDAG01 exam.

M Skip it: covered by EDAN26 Multicore programming

- "What" refers to *Writing Efficient C Code: a thorough introduction*, ISBN 9781659599206.
- For example Chapter 1 is at level 3 but below it in the table there are more specific levels for parts of Chapter 1.
- The levels here and in the video titles are intended to be the same and if you spot a difference, please let Jonas know, thanks!

What	Level	Content
Chapter 1	3	the C programming language
Example 1.4.12	5	matrix allocation without VLA
Example 1.4.13	4	matrix allocation with variably modified type
Section 2.1-2.2	3	unsigned and signed integers
Section 2.3	4	fixed point numbers
Section 2.4	3	floating point numbers
Section 2.4.3	4	rounding
Chapter 3	3	Power CPU's but I will not ask about Power-specific details (see below)
Chapter 4	3	Ignore coherence misses and Exercises 4.7.1 and 4.7.2 about sequence of addresses 0x12, 0x13, 0x14, ...
Chapter 5	M	
Chapter 6	3	what you can do with them and what they are good at but not details such as command syntax
Section 7.1	3	only first sentence and skip rest
Section 7.2	3	implementation defined behavior
Section 7.3	3	unspecified behavior
Section 7.4	3	undefined behavior
Section 7.5	S	translation phases
Section 7.6	4	translation unit
Section 7.7	4	character sets
Section 7.8	4	scopes of identifiers
Section 7.9	4	linkage of identifiers
Section 7.10	4	name space of identifiers
Section 7.11	5	lvalues
Section 7.12	5	arrays and function designators
Section 7.13	3	storage duration of objects
Section 7.13.2	5	temporary lifetime

What	Level	Content
Section 7.13.4	M	thread storage duration
Section 7.14	3	types
Section 7.14.5	4	ANSI C aliasing rules
Section 7.15	3	only `CHAR_BIT`, i.e., *at least* eight bits
Section 7.16	4	conversions
Section 7.17	4	sequence points
Section 7.18	M	multithreaded execution
Chapter 8	3	lexical elements
Section 8.8	S	preprocessing numbers
Chapter 9	3	declarations
Section 9.2.4	M	`_Thread_local`
Section 9.4.2	5	`restrict`
Section 9.4.4	M	`_Atomic`
Section 9.5	5	function specifiers
Section 9.6	5	alignment specifier
Section 9.7	3	declarators
Section 9.8	5	type names (but ignore the grammar rules)
Section 9.9	4	type definitions
Section 9.10	3	initialization
Section 9.11	5	static assertions
Section 9.12	5	function definition
Section 10	3	expressions but skip grammar rules and generic selections also for 5
Section 10.3	5	contracted expressions
Section 10.12	4	equality expression
Section 10.21	5	constant expression
Chapter 11	3	statements

Chapter 12	4	the C preprocessor but for level 3 learn
		`#include, #define, #undef, #line, #error`
		`#if, #elif, #ifdef, #endif`
		`__FILE__, __LINE__, __DATE__, __TIME__`

Section 13	3-5 and R	C library is mostly R except the functions mentioned in the F11 youtube lectures which are at the level of the video title (they are identical for English and Swedish videos)
Section 14	R	common errors
Section 15	3	writing efficient C code
Section 15.4	4	cache optimization
Section 15.6.6	4	key idea but not details
Section 15.7	5	powers of two
Section 15.8	4	expression optimization
Chapter 16	4	optimizing compilers
Chapter 17	3	memory allocation
Section 17.5	5	sparse representations
Chapter 18-20	M	pthreads, OpenMP, transactional memory
Appendix A	S	Unix terminals
Appendix B	S 3	integer linear programming but I may ask questions about how you would optimize such code such as which tool would you use for what and why but nothing about the details of simplex or branch-and-bound or their pseudo code

A clarification to ”Power-specific details”: it is important to understand how CPU’s work in principle and especially the ideas with RISC. What is Power-specific?

- Power opcode types but learn the concept of fixed-witdh instruction formats,
- the purpose of the link-register on Power but learn that the return address somehow must be saved in a register at a function call (and then possibly in the stack frame of the called function if needed),
- which registers (e.g. R3..R10 for integers and pointers) are used for which paramter type is Power specific, but note that all RISC architectures put function parameters and the return value in certain registers.

Basically, you could instead learn a different modern architecture such as ARM, MIPS, RISC-V, or Sparc instead and know nothing about Power to pass the oral exam but for the labs and project you need to be able to read Power assembler code.