C# Chapters 1 to 6 – Ebook Study Summaries

Chapter 1: Introduction

- • Programming is writing instructions to automate tasks using logic.
- • The software development cycle includes: Requirements → Design → Implementation → Testing → Deployment → Maintenance.
- • C# is a modern, object-oriented, high-level language developed by Microsoft.
- • .NET Framework provides CLR, MSIL, assemblies, and JIT compiler support.
- • Visual Studio is a full-featured IDE with tools like IntelliSense and Debugger.
- • Alternatives to Visual Studio include MonoDevelop, SharpDevelop, and CLI tools like csc.

Chapter 2: Data Types

- • Data represents facts, values, or instructions in memory.
- • Primitive types include int, float, char, bool, decimal – stored directly in memory.
- • Variables must be declared with a type before use and can be initialized during declaration.
- • Literals are fixed values, e.g., 10, 'A', true. Use suffixes for long (L), float (f).
- • Type conversion can be implicit (safe) or explicit (casting). Use Convert and Parse.
- • Nullable types (e.g., int?) allow null values. Use .HasValue and .Value to check/access.
- • Type inference with 'var' allows the compiler to infer the type.
- • Constants are declared using 'const' and are immutable.
- • Naming conventions: camelCase for variables, PascalCase for methods/constants.

Chapter 3: Operators & Operands

- • Operators perform operations on data: arithmetic, assignment, comparison, logical, bitwise.

- • Arithmetic: +, -, *, /, %, follow precedence rules. Integer division truncates.
- • Assignment: =, +=, -=, etc. simplify operations.
- • Comparison: ==, !=, <, >, <=, >= return boolean values.
- • Logical: &&, ||, ! combine boolean expressions; used in conditions.
- • Special operators: ++, --, ?: (ternary), ?? (null-coalescing).
- • Expressions combine operators and operands to produce results.
- • Overflow occurs when values exceed type limits. Use 'checked' to detect.
- • Bitwise operators (&, |, ^, ~, <<, >>) work at the binary level.

Chapter 4: Output

- • Console.Write() prints without newline; Console.WriteLine() adds newline.
- • Use escape characters like \n (newline), \t (tab) for formatting.
- • String interpolation ($"{a} + {b} = {sum}") and format specifiers (F2, C, P) enhance output.
- • Console.ReadLine() reads input as string – requires conversion to int, float etc.
- • Use int.Parse(), double.Parse(), Convert.To<Type>() for type-safe conversions.
- • int.TryParse() avoids exceptions when parsing invalid input.
- • Unicode output requires setting Console.OutputEncoding = Encoding.UTF8.
- • Console.Read() reads ASCII code, Console.ReadKey() reads a keypress (use .KeyChar).
- • Practice programs: Input/output for sum, greetings, and basic interaction.

Chapter 5: Conditional Statements

- • Conditional logic directs code execution based on boolean expressions.
- • if statement executes a block if condition is true.
- • if-else provides alternate execution paths.

- • else-if ladder checks multiple conditions in sequence.
- • switch-case handles multiple fixed values; use break to avoid fall-through.
- • Common errors: using = instead of ==, nesting confusion, missing break.
- • Best practices: clear structure, avoid deep nesting, use switch when applicable.

Chapter 6: Loops

- • Loops repeat tasks and reduce code duplication.
- • while loop: pre-condition check; executes while condition is true.
- • do-while loop: post-condition; runs at least once.
- • for loop: fixed iterations with initializer, condition, and increment.
- • Nested loops are used for matrix or grid operations; increase complexity.
- • break exits loop immediately; continue skips to next iteration.
- • Infinite loops: no terminating condition; used in menu-driven programs.
- • Loop patterns: counting, sum/product, condition-based iteration.