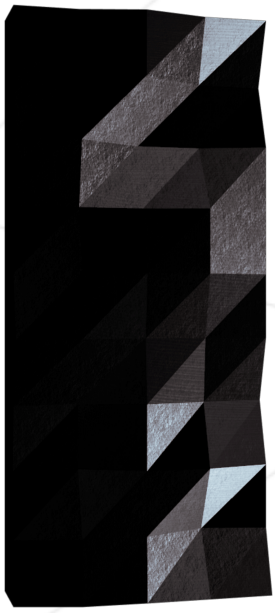
Made with Xodo PDF Reader and Editor



Structured Programming

sd00

Summary: In this module you will see the basics of structured programming

Version: 4.2

Made with Xodo PDF Reader and Editor

Contents

Made with Xodo PDF Reader and Editor

Chapter I

Preamble

Hello, and welcome to your first Structured Development workshop!

This workshop is part of a mini-course for *42* students curated by *Prof. Enrico Vicario* and *Dr. Nicolò Pollini*. Throughout this course, you'll be introduced to a programming methodology that originated in the 1960s and has since evolved in tandem with the C language
.

Structured Development is built on three key pillars:

- **Structured Programming** *(today's focus)*: This concerns the act of writing code in a clean, readable, and maintainable way. It emphasizes avoiding common pitfalls and code smells that lead to bugs and wasted time, especially during debugging.
- **Structured Design**: Good software design means organizing your codebase so that each function has a clear, well-defined purpose. The goal is to prevent situations where small changes ripple through the entire system, forcing you to rewrite large portions of code (yes, we're looking at you, *spaghetti code*).
- **Structured Analysis**: This is the most abstract level. It involves designing the architecture of an entire software system, especially when highly complex, based on its requirements and the flow of information needed to fulfill them.

`Remember: Code is not just written to work, it's written to be used. If it's worth using, it's worth writing for someone else to read, understand, and maintain. You never write code just for yourself.`

Made with Xodo PDF Reader and Editor

Chapter II

Introduction

Structured Programming is a paradigm aimed at improving the clarity, quality, and maintainability of code by using a well-defined set of control structures: sequence, selection, and repetition. Introduced in the late 1960s as a response to the chaotic and error-prone nature of unstructured code (often using goto statements), Structured Programming encourages:

- Breaking programs into small, modular functions.
- Writing code with one entry and one exit point per block (when possible).
- Designing a clear and predictable control flow, enabling *Axiomatic Reasoning*.

The goal is to make code easier to read, test, debug, and modify, laying a solid foundation for more advanced software design principles.
In the **June 13** lecture, you'll receive more in-depth information about Structured Programming. For now, this brief introduction is all you need to get started. You will get a chance to tackle them again after the lecture.
Approach the following exercises using **your own coding style**, without restrictions. You're encouraged to explore: discuss with your classmates, search online, and even use AI tools if you wish.
However, a word of caution about **Large Language Models (LLMs)** like ChatGPT, Gemini, and similar tools:
Researchers (including us) have studied the impact of over-relying on AI in learning-driven development, and the findings suggest that, while LLMs can boost your short-term performance, **overusing them can hinder your long-term growth**. The knowledge and skills you build now are what your future success depends on.
Here's our recommended approach:

- **Start by solving the problem on your own.**
- **If you get stuck, ask your peers or look for help online.**
- **If you're still stuck, and no one can help, use LLMs to understand *how* to approach the problem**, but don't let them solve it for you, unless you're certain that solving this particular problem is not important to your learning journey.

Made with Xodo PDF Reader and Editor

Chapter III

General instructions

- This document contains **10 C programming exercises**, arranged in **increasing order of difficulty**. While you're free to tackle them in any order, we recommend progressing sequentially to get the most out of the experience.
- These exercises are **not meant to be completed in full by everyone**, so don't be discouraged if you struggle early on. This course brings together students with a wide range of experience, and the more advanced tasks are designed to challenge even seasoned programmers.
- **Tomorrow, we'll release the second set of exercises (Structured Design)**. So focus on doing your best with today's set, don't get stuck trying to perfect a solution that's already good enough.

Made with Xodo PDF Reader and Editor

Chapter IV

Evaluation criteria

Unlike in previous experiences, you will be evaluated not only on whether your solution works and meets the requirements, but also on the **reasoning behind your choices**. In fact, this is the primary focus of the evaluation. *(No pressure, though)*
Here are a few important notes:

- **Your code must compile.** Minor oversights or non-critical errors may be tolerated, as long as the core functionality is preserved.
- **You're not required to follow the 42 Norm**, but it's a good starting point if you're unsure about what constitutes clean and readable code.

Note: This is the **first version** of this document, so it may contain errors or inaccuracies. If you spot any issues, please let the staff know as soon as possible. Thank you!

Made with Xodo PDF Reader and Editor

Chapter V

Exercise 00 : Average

	Exercise 00
Exercise 00 : Average	
Turn-in directory : $ex00/$	
Files to turn in : `average.c, average.h`	
Allowed functions : `None`	

Write a function that takes an array of integers and its size, and returns the average as a float. The elements must be:

- Validated (must be in range 0-100 inclusive).
- Ignored if invalid (must not stop the loop).

The function must be formatted as follows:

```
float average(const int *arr, int size);
```

Made with Xodo PDF Reader and Editor

Chapter VI

Exercise 01 : first_last

	Exercise 01
Exercise 01 : first_last	
Turn-in directory : $ex01/$	
Files to turn in : `first_last.c, first_last.h`	
Allowed functions : `None`	

Write a function that receives:

- An array of integers (‘arr[]‘)
- Its size (‘size‘)
- A target integer (‘target‘)

And computes:

- The index of the **first** occurrence of ‘target‘ in ‘arr‘, or ‘-1‘ if it doesn’t appear.
- The index of the **last** occurrence of ‘target‘ in ‘arr‘, or ‘-1‘ if it doesn’t appear.

the function must be formatted as follows:

```
void first_last(int arr[], int size, int target, int *first, int *last);
```

Made with Xodo PDF Reader and Editor

Chapter VII

Exercise 02 : segmented_runs

	Exercise 02
Exercise 02 : segmented_runs	
Turn-in directory : *ex*02/	
Files to turn in : `segmented_runs.c, segmented_runs.h`	
Allowed functions : `None`	

You have an array of integers divided into multiple segments. The boundaries of each segment are marked by a special sentinel value '-1' (which is **not** part of any segment).

```
Example:
`[2, 3, 4, -1, 5, 6, -1, 1, 2, 3, 4]`
contains 3 segments:

- Segment 1: `[2, 3, 4]`
- Segment 2: `[5, 6]`
- Segment 3: `[1, 2, 3, 4]`
```

Your task is to write a function that:

- Counts how many segments contain *at least one increasing sequence of length 3 or more* (strictly increasing consecutive numbers).
- Returns this count.

the function must be formatted as follows:

```
int count_segments(const int *arr, int size);
```

The input array may contain zero or more segments, separated by '-1'.
Segments can be empty (two consecutive '-1's).

Made with Xodo PDF Reader and Editor

Chapter VIII

Exercise 03 : critical_windows

	Exercise 03
Exercise 03 : critical_windows	
Turn-in directory : $ex03/$	
Files to turn in : `critical_windows.c, critical_windows.h`	
Allowed functions : `None`	

Write a function that takes an array of daily seismic tremor readings and its size, and returns how many **critical windows** occurred.

A **critical window** is any **5day** period that meets **all** the following conditions:

- At least 3 of the 5 days had readings $\geq$ **70** .
- No reading exceeded **150** .
- The **average** of the 5 values is $\geq$ **90** .

The function must analyze each 5-day window in the input (using a **sliding window**) and return how many of them qualify as "critical".

the function must be formatted as follows:

```
int count_critical_windows(const int *readings, int size);
```

Made with Xodo PDF Reader and Editor

Chapter IX

Exercise 04 : filesystem

	Exercise 04
Exercise 04 : filesystem	
Turn-in directory : $ex04/$	
Files to turn in : `filesystem.c, filesystem.h`	
Allowed functions : `malloc, strdup`	

Implement a basic file system manager that simulates the hierarchical structure of folders and files.

Each **folder** can contain:

- Files (each with a name and size).
- Other folders (size set to zero).

the functions must be formatted as follows:

```
FSNode *create_file(const char *name, int size);
FSNode *create_folder(const char *name);
void add_child(FSNode *parent, FSNode *child);
FSNode *get_children(const FSNode *parent);
FSNode *get_sibling(const FSNode *node);
```

Made with Xodo PDF Reader and Editor

Chapter X

Exercise 05 : password_validator

	Exercise 05
Exercise 05 : password_validator	
Turn-in directory : $ex05/$	
Files to turn in : `password_validator.c, password_validator.h`	
Allowed functions : `None`	

Implement a password validation system that enforces the following strength rules for a new password:

- At least 8 characters long
- At least one uppercase letter
- At least one lowercase letter
- At least one digit
- At least one special character (@#$%^&*)
- Must differ from current password

The validation function returns one of two states:

- '0 = VALID'
- '1 = INVALID' (fails any of the rules above)

The function must be formatted as follows:

```
PwStatus validate_password(const char *new_pw, const char *curr_pw);
```

Made with Xodo PDF Reader and Editor

Chapter XI

Exercise 06 : grade_mapping

	Exercise 06
Exercise 06 : grade_mapping	
Turn-in directory : *ex*06/	
Files to turn in : `grade_map.c, grade_map.h`	
Allowed functions : `None`	

Write a program that converts an array of integer scores into an array of grade strings using a dynamic mapping strategy.

The grade mapping function must support the following mapping strategies:

Score	Grade
97–100	A+
93–96	A
90–92	A-
87–89	B+
83–86	B
80–82	B-
77–79	C+
73–76	C
70–72	C-
67–69	D+
63–66	D
60–62	D-
0–59	F

Table XI.1: plusminus mapper

Score	Grade
60–100	P
0–60	F

Table XI.2: passfail mapper

Score	Grade
90–100	A
80–89	B
70–79	C
60–69	D
0–59	F

Table XI.3: standard mapper

The function must be formatted as follows:

```
void map_scores(const int *scores, int size, GradeMapper mapper, const char *mapped_grades);
```

Made with Xodo PDF Reader and Editor

Chapter XII

Exercise 07 : filesystem manager (bonus)

	Exercise 07
Exercise 07 : filesystem manager (bonus)	
Turn-in directory : *ex*07/	
Files to turn in : `filesystem.c, filesystem.h, filesystem_bonus.c`	
Allowed functions : `malloc, strdup, free, printf`	

Develop new features for the file system manager you built previously.

- Recursively compute the total size of any given folder, which includes all files and nested contents.
- Implement a helper to print the structure in a tree-like format.
- Implement a recursive function to free the entire file system.

The function must be formatted as follows:

```
int compute_total_size(FSNode *node);
void print_structure(const FSNode *node, int indent);
void free_filesystem(FSNode *node);
```

Remember, include the functions you developed for the previous filesystem exercise

Made with Xodo PDF Reader and Editor

Chapter XIII

Exercise 08 : Password Validator (bonus)

	Exercise 08
Exercise 08 : Password Validator (bonus)	
Turn-in directory : *ex*08/	
Files to turn in : `password_validator.c, password_validator.h, password_validator_bonus.c`	
Allowed functions : `strncpy`	

Develop new features for the password validation system you built previously.
The system must now enforce the following strength rules for a new password:

Strength rules:

- Follow the same guidelines as the ones of **ex05**

Similarity rule:

- The new password must not have an **edit distance** less or equal than *one* from any of the last three chosen passwords.

Store the last three accepted passwords so that inserting a new valid password overwrites the oldest entry. The validation function returns one of three states:

- 0 = VALID
- 1 = INVALID_WEAK (fails strength rules)
- 2 = INVALID_SIMILAR (too similar to one of the last three **valid** passwords)

The function must be formatted as follows:

```
PwStatus validate_password(const char *new_pw, PasswordHistory *history);
```

Made with Xodo PDF Reader and Editor

Chapter XIV

Exercise 09 : Grade Mapping (bonus)

	Exercise 09
Exercise 09 : Grade Mapping (bonus)	
Turn-in directory : *ex*09/	
Files to turn in : `grade_map.c, grade_map.h, grade_map_bonus.c`	
Allowed functions : `strncpy, malloc, free, strcmp, strlen, printf`	

Develop new features for the grade mapping system you built previously.
You must now:

- Compute the distribution of grades (i.e., the count of each grade).
- Print the distribution of grades (must be understandable).
- Free the distribution of grades.

Compute the distribution by inserting each pair grade/count into a data structure of your choice. If a grade never appears in the mapping, it should not appear in the distribution either.

The functions must be formatted as follows:

```
GradeNode *compute_distribution(const char **mapped_grades, int size);
void print_distribution(GradeNode *head);
void free_distribution(GradeNode *head);
```

Remember, include the functions you developed for the previous Grade Mapping exercise

Made with Xodo PDF Reader and Editor

Chapter XV

Submission and peer evaluation

Create a personal repository named:
42xunifi-structured-development-2025-<your-intra-login>
Share it with the staff and organize the exercises as follows:

```
sd00/
 ex00/
 ex01/
 ex02/
 ...
sd01/
 ex00/
 ex01/
 ...
sd02/
 ex00/
 ...
```

- **Update your repository regularly**, and make sure to **push all completed exercises before the deadline**.
- **This workshop's deadline is June 13**. Any changes made after this date **will not be considered** for assessment purposes.
- This exercise follows the **42 methodology**, so **peer-to-peer collaboration is allowed and strongly encouraged**.
- **Formal peer evaluations via Intra will not be available**, but you are still welcome to ask your peers for feedback on your work.
- **Our feedback won't be immediate**, so to help you track your progress, we recommend using **SonarQube**, a free tool that detects common pitfalls and code smells, many of which will be considered during evaluation. Here is a guide on how to set up it their cloud version.

Made with Xodo PDF Reader and Editor

Chapter XVI

Setting Up SonarQube (SonarCloud) with GitHub

1. **Create a public GitHub repository** for the course, named: `42xunifi-structured-development-2025-<your-intra-login>`.
2. Go to SonarCloud and click **"Start now."**
3. Sign up using your **GitHub account**.
4. When prompted, **import an organization**. You can use `"42"` or your GitHub username.
5. **Authorize access only to selected repositories**, and choose the one you created in step 1.
6. Choose a **Name** and a **Key** for your organization (your username is usually fine for both).
7. Select the **Free Plan** when prompted.
8. Click **"Create Organization."**
9. Select the previously created repository and click **"Set Up."**
10. For code technology, choose **"Previous version."**
11. Click **"Create Project."**
12. *Profit.*