People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University of Mostaganem Abdelhamid Ibn Badis

Faculty of Science and Technology

Department of Science and Technology

Module: Computer Structure and Applications

Practical Work Manual: Introduction to C Programming

Prepared by:

Dr. ROUBA Baroudi

Associate Professor (Grade A)

Reviewed by:

Dr. Dejbbara Mohamed Redha

Dr. Laredj Mohamed Adnane

Academic Year 2025-2026

UNIVERSITE
Abdelhamid Ibn Badis
MOSTAGANEM

FST

Abstract

Programming is one of the most important means of automating information processing. Among the various programming languages used in computer science, the C language remains one of the most widely adopted in education. This practical work manual is intended for first-year students in Science and Technology and complements the *Computer Structure and Applications* course. It provides a step-by-step introduction to program development using the C language with the CodeBlocks environment. The manual covers fundamental concepts such as data types, input/output operations, conditional structures, loops, arrays and structures. Through structured practical sessions, exercises, and detailed solutions, this manual aims to build a solid foundation in C programming and develop students' basic programming skills.

Keywords

C language, Program, Algorithm, Compile, Execute, CodeBlocks.

Résumé

La programmation est l'un des moyens les plus importants pour automatiser le traitement de l'information. Parmi les différents langages de programmation utilisés en informatique, le langage C demeure l'un des plus largement adoptés dans le domaine de l'enseignement. Ce manuel de travaux pratiques est destiné aux étudiants de première année en Sciences et Technologie et complète le module *Structure de l'Ordinateur et Applications*. Il propose une introduction progressive au développement de programmes en langage C à l'aide de l'environnement CodeBlocks. Le manuel couvre les notions fondamentales telles que les types de données, les opérations d'entrée/sortie, les structures conditionnelles et les boucles, les tableaux, les matrices et les structures. À travers des séances pratiques structurées, des exercices et des solutions détaillées, ce manuel vise à fournir aux étudiants une base solide en programmation C et à développer leurs compétences fondamentales en programmation.

Mots clés

Langage C, Programme, Algorithme, Compiler, Exécuter, CodeBlocks

الملخص

تُعَدّ البرمجة من أهم الوسائل لأتمتة معالجة المعلومات. ومن بين مختلف لغات البرمجة المستخدمة في مجال الحاسوب، تظلّ لغة C واحدة من أكثر اللغات اعتمادًا، خاصة في مجال التعليم.
يستهدف هذا الدليل الخاص بالأعمال التطبيقية طلبة السنة الأولى علوم وتكنولوجيا، ويُعدّ مكمّلًا لمقياس بنية الحاسوب وتطبيقاته. ويقدّم هذا الدليل مدخلًا تدريجيًا لتطوير البرامج بلغة C باستخدام بيئة التطوير CodeBlocks. يغطي الدليل المفاهيم الأساسية مثل أنواع البيانات، وعمليات الإدخال والإخراج، والبُنى الشرطية والحلقات، والمصفوفات، والمصفوفات ثنائية الأبعاد، والبُنى (Structures). ومن خلال حصص تطبيقية منظّمة، وتمارين، وحلول مفصّلة، يهدف هذا الدليل إلى تزويد الطلبة بأساس متين في برمجة لغة C وتنمية مهاراتهم الأساسية في البرمجة.

الكلمات المفتاحية

لغةC ، برنامج، خوارزمية، مُصرِّف(Compiler) ، تنفيذ، CodeBlocks

Foreword

This practical work manual is intended for first-year undergraduate students enrolled in the Science and Technology program. It is designed as a complementary resource to the *Computer Structure and Applications* course, in which the fundamental concepts of algorithmic thinking and structured programming are introduced.

The primary objective of this manual is to guide students through the learning of the C programming language, which is one of the most widely used languages in computer science and engineering. The manual adopts a practical and progressive approach, enabling students to translate theoretical concepts into concrete implementations. All practical work is carried out using the CodeBlocks integrated development environment (IDE), version 25.03. For this reason, the first session of the manual is dedicated to presenting this development tool, including detailed instructions for its installation, configuration, and basic usage.

To ensure a gradual and effective learning process, the manual is organized into a series of practical sessions. Each session focuses on a specific programming concept, such as variables and data types, input/output operations, conditional structures, loops, arrays, matrices and structures. Concepts are introduced in a clear and progressive manner, starting from basic notions and advancing toward more complex programming constructs. At the end of each session, students are expected to be able to apply the newly acquired concepts to develop simple yet functional C programs.

In order to facilitate understanding and reinforce learning, each session includes a set of practical exercises designed to encourage active student participation. These exercises allow students to practice problem-solving skills and gain confidence in writing C programs. Furthermore, detailed solutions and explanations are provided to help students analyze their work, understand common mistakes, and consolidate their knowledge.

Overall, this manual aims to support students in developing a solid foundation in C programming, promoting both autonomy and methodological rigor, while preparing them for more advanced courses in computer science and engineering.

Table of contents

Abstract 1
Résumé 1
الملخص 2
Foreword 3
List of Figures 9
List of tables 11
1. Introduction to C programming 6
1.1. What is a programming language? 6
1.2. The C language 6
1.3. CodeBlocks Editor 6
1.4. Installation of CodeBlocks 6
1.5. Overview of the CodeBlocks Main Window 10
a. Menu Bar 10
b. Toolbar 10
c. Workspace Panel 10
d. Editor Window 10
e. Logs & Output Panel 11
f. Status Bar 11
1.6. Structure of C program 11
a. Preprocessor Directives 11
b. Global Declarations (optional) 11
c. The main() Function 11
1.7. Example of a Basic C Program 12
2. PW N°1- Getting familiar with CodeBlocks- 14
2.1. Objectives 14
2.2. Launch the CodeBlocks editor 14
2.3. Create a new project 14
2.4. Open an existing project 17
2.5. Edit a program file 17
2.6. Save a program file 18
2.8. Exit the CodeBlocks editor 18

2.9. Application exercise 18
3. PW N°2- Edit, compile and executer a program- 20
3.1. Objectives 20
3.2. Compile a program 20
3.3. Run a program 21
3.4. Exercice d'application 21
4. PW N°3-Basic data types- 23
4.1. Objectives 23
4.2. Basic Structure of a C Program 23
a. Preprocessor Directives 23
b. Main Function 23
c. Declarations 23
d. Statements / Instructions 24
e. Return Statement 24
4.3. Declaration of Constants 24
a. Using the *const* keyword 24
b. Using *#define* preprocessor directive 24
4.4. Declaration of variables 24
4.5. Basic data types 25
a. Int type 25
b. Float type 25
c. Double type 26
d. Char type 26
e. Boolean type 26
f. Predefined functions 26
g. The Precedence of Operators 27
4.6. Practical Exercise 28
4.7. Practical exercise solution 28
5. PW N°4-Basic instructions- 30
5.1. Objectives 30
5.2. The assignment statement 30
5.3. The output statement (printf) 30
5.4. Reading statement (scanf) 31
5.5. Practical exercise 32

5.6. Practical exercise solution 32
5.7. Exercises 34
6. PW N°5- Conditions- 37
6.1. Objectives 37
6.2. Express simple and complex conditions 37
a. Simple if Statement 37
b. if...else Statement 37
c. Nested if Statement 38
6.3. Illustrative Example1 38
6.4. Illustrative Example2 39
6.5. Multiple choice statement (Switch/Case) 40
6.6. Illustrative Example3 41
6.7. Application exercise 43
6.8. Application exercise solution 43
6.9. Exercises 44
7. PW N°6- Repetitive structures (Loops)- 47
7.1. Objectives 47
7.2. For loop 47
a. Illustrative Example1 48
b. Illustrative Example2 48
7.3. While Loop 48
a. Illustrative Example 49
7.4. Do-While Loop 49
a. Illustrative Example 50
7.5. Practical Exercise 1 51
7.6. Practical Exercise Solution 51
a. With For Loop 51
b. With While Loop 52
c. With Do-While Loop 52
7.7. Practical Exercise 2 52
7.8. Practical Exercise Solution 52
a. With While Loop 53
b. With Do-While Loop 53
7.9. Exercise 54

8. PW N°7- One Dimensional Arrays (Vectors)- 56
8.1. Objectives 56
8.2. Declare a one-dimensional array 56
8.3. Filling a one-Dimensional array 56
8.4. Displaying the Contents of a One-Dimensional Array 57
8.5. Application Exercise 1 58
8.6. Application Exercise Solution 58
8.7. Application Exercise 2 58
8.8. Application Exercise Solution 58
8.9. Exercises 59
9. PW N°8- Two-Dimensional Arrays (Matrices)- 62
9.1. Objectives 62
9.2. Declare a two-dimensional array 62
9.3. Filling a matrix 62
9.4. Displaying the content of a matrix 63
9.5. Application Exercise 1 63
9.6. Application Exercise 1 solution 63
9.7. Application Exercise 2 64
9.8. Application Exercise 2 solution 64
9.9. Exercises 65
10. PW N°9-Structures- 68
10.1. Objectives 68
10.2. Structures in C language 68
10.3. Declaring a structure 68
10.4. Accessing Structure Members 69
10.5. Application Exercise 69
10.6. Application Exercise solution 70
10.7. A structure as a member of another structure 70
10.8. Exercises 71
11. PW N°10-String manipulation- 74
11.1. Objectives 74
11.2. String Declaration 74
11.3. Reading and Displaying Strings 74
11.4. Manual String Manipulation 74

11.5. Standard String Functions 75
11.6. Illustrative examples.. 76
11.7. Exercises 77
12. Solution of the Exercises 79
12.1. PW N°4-Basic instructions-.. 79
12.2. PW N° 5-Conditions-.. 82
12.3. PW N°6- Repetitive structures (Loops)- 85
12.4. PW N°7- One dimensional arrays 88
12.5. PW N° 8-Two-dimensional arrays (matrices) 92
12.6. PW N°10- Structures.. 97
12.7. PW N°10-String manipulation- 101
References.. 104
Websites / Online Tutorials.. 104

List of Figures

Figure 1 The installation welcome screen ... 7
Figure 2 License agreement window ... 7
Figure 3 The components selection screen ... 8
Figure 4 Install location screen ... 8
Figure 5 Installation process screen ... 9
Figure 6 The installation complete screen ... 9
Figure 7 The workspace window ... 10
Figure 8 Launch CodeBlocks ... 14
Figure 9 Create a new project ... 15
Figure 10 Select the language to use in the new project. ... 15
Figure 11 Project information ... 16
Figure 12 Select the compiler ... 16
Figure 13 The first program ... 17
Figure 14 Open an existing project ... 17
Figure 15 Edit a program file. ... 17
Figure 16 Compilation error ... 21
Figure 17 program execution ... 21
Figure 18 Basic structure of a C program ... 23
Figure 19 First execution of the program ... 34
Figure 20 Second execution of the program ... 34
Figure 21 Program output: Admitted or Failed based on student's average ... 39
Figure 22 program output: sign of the product of two numbers ... 40
Figure 23 switch-case statement: calculator program output ... 42
Figure 24 Nested if-then-else : program output. ... 44
Figure 25 For Loop : Example 1 ... 48
Figure 26 For Loop : Example 2 ... 48
Figure 27 While Loop: Example 1 (ascending order) ... 49
Figure 28 While Loop: Example 1 (descending order). ... 49
Figure 29 Do-While Loop: Example 1 (ascending order). ... 50
Figure 30 Do-While Loop: Example 1 (descending order). ... 51
Figure 31 Average of n numbers with For Loop ... 51
Figure 32 Average of n numbers with While Loop ... 52
Figure 33 Average of n numbers with Do-While Loop ... 52
Figure 34 Number of numbers read with While Loop ... 53
Figure 35 Number of numbers read with Do-While Loop. ... 53
Figure 36 Filling a one-Dimensional array ... 57
Figure 37 Displaying the content of one-Dimensional array ... 57
Figure 38 Average Weekly Temperature Program ... 58
Figure 39 Highest and Lowest Scores program ... 59
Figure 40 Filling a matrix ... 63
Figure 41 Displaying the content of a matrix. ... 63
Figure 42 sum of the elements on the diagonal ... 64
Figure 43 Transpose of a matrix ... 65

Figure 44 Reading and displaying structure member values 69
Figure 45 Product Structure 70
Figure 46 A structure as a member of another structure 71
Figure 47 strlen() function 76
Figure 48 strcat() function 76
Figure 49 strcmp() 76
Figure 50 strcpy() 77

List of tables

Table 1 Example of a basic C program 12
Table 2 Basic Data Types 25
Table 3 Variants of type int 25
Table 4 Operations performed on int type 25
Table 5 Operations performed on float type 26
Table 6 Some predefined functions in C language 27
Table 7 Common printf Format Specifiers in C 31
Table 8 Common *scanf* Format Specifiers in C 32
Table 9 Standard string functions 75

Introduction to Programming

&

CodeBlocks Installation

1. Introduction to C programming

1.1. What is a programming language?

The computer only executes what the programmer asks it to do. To achieve this, the programmer must formulate instructions in a language understandable by the computer; however, since the computer only understands instructions encoded in binary, so-called high-level languages were created to facilitate programming.

There are several programming languages such as C, Fortran, Pascal, Python, Java, etc. This manual focuses on the C language.

1.2. The C language

The C language was developed in the early 1970s by **Dennis Ritchie** at **Bell Labs** (United States). It was initially designed for the development of the **UNIX operating system**, to replace assembly language and provide better portability of programs.

C originated from another language called the **B language**. Thanks to its simple syntax, high efficiency, and close relationship with hardware, C quickly established itself as a reference programming language in computer science.

Over time, C has evolved and been standardized to ensure program portability across different platforms. The first official standard, **ANSI C (C89/C90)**, was published at the end of the 1980s, followed by other standards such as **C99**, **C11**, and **C18**.

Today, C is widely used in the development of operating systems, embedded systems, compilers, and serves as a fundamental foundation for learning programming.

1.3. CodeBlocks Editor

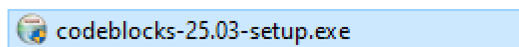
Several C language compilers and editors are available. In this manual, we will use **Code::Blocks 25.03**, which can be downloaded for free from the following URL:

https://code-blocks.fr.download.it/

1.4. Installation of CodeBlocks

To install CodeBlocks, follow the steps outlined below.

1. Double-click on the installation file “codeblocks-25.03-setup.exe”.



2. Click on “Next” to start the installation process.

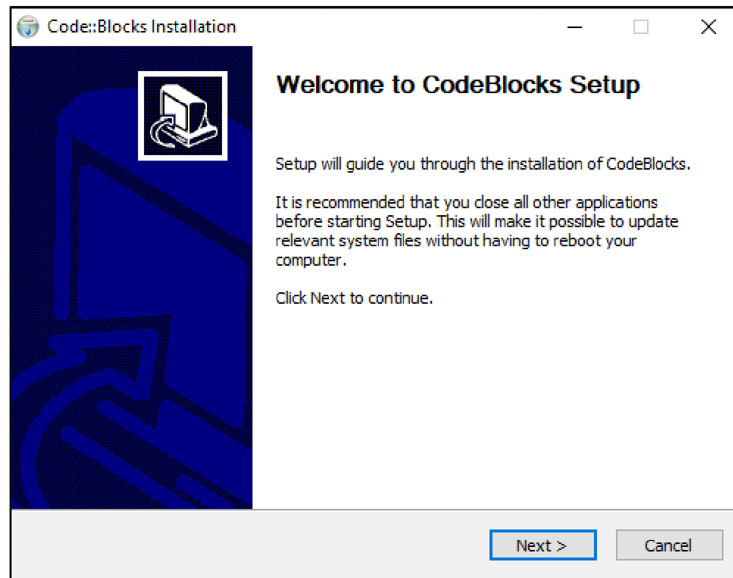


Figure 1 The installation welcome screen

3. Click on « I Agree » to accept the license agreement

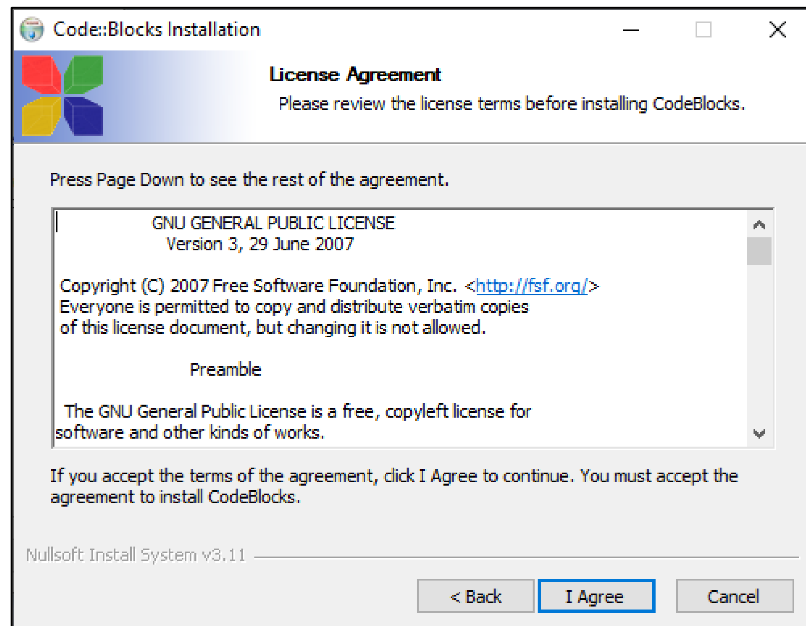


Figure 2 License agreement window

4. Select the components to install. To install all available components, choose ***"Full, all plugins, all tools, everything"***, then click on ***"Next"*** to proceed to the next step.

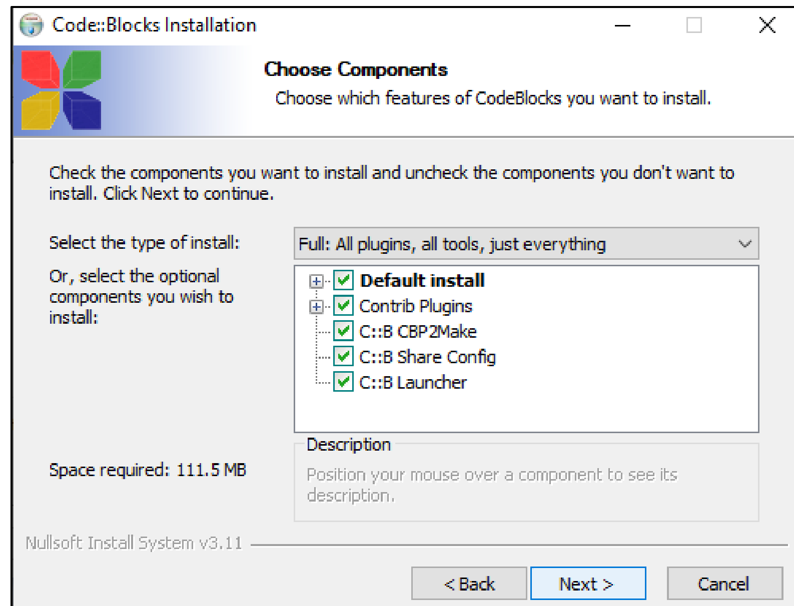


Figure 3 The components selection screen

5. Choose the folder where Code::Blocks will be installed. The default location is ***"C:\Program Files\CodeBlocks"***. Click on ***"Install"*** to start the installation.

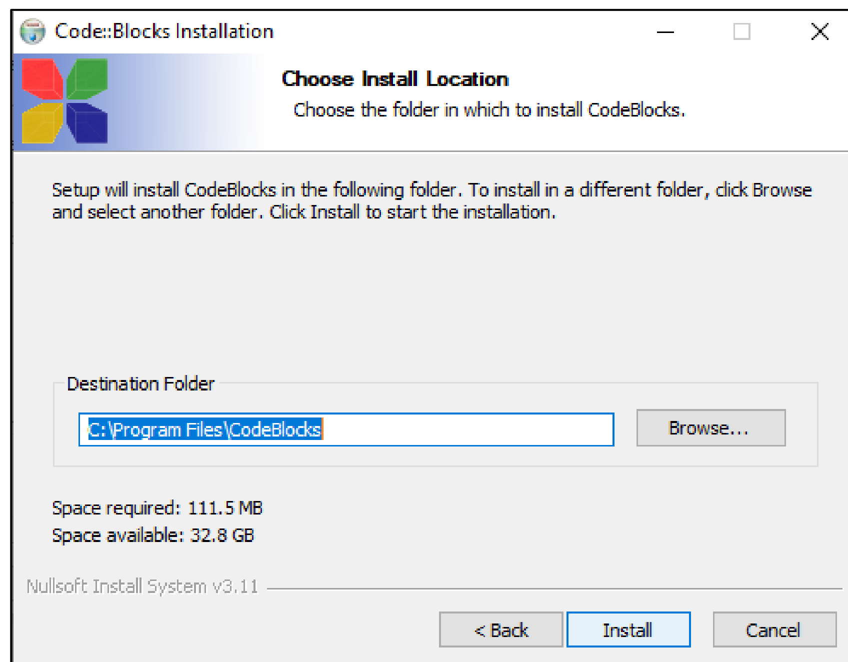


Figure 4 Install location screen

6. Click on ***« Yes »*** to run CodeBlocks.

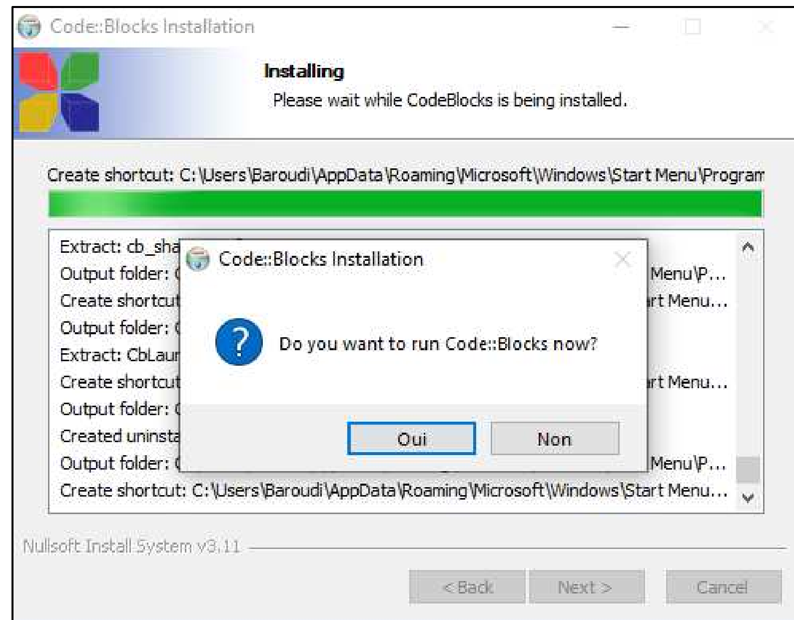


Figure 5 Installation process screen

7. Click on ***"Finish"*** to complete the installation process.

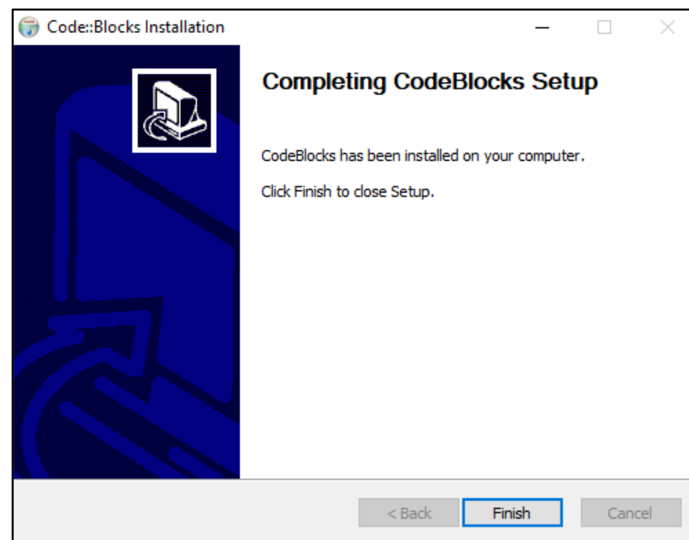


Figure 6 The installation complete screen

8. At the end of the installation process, CodeBlocks is launched

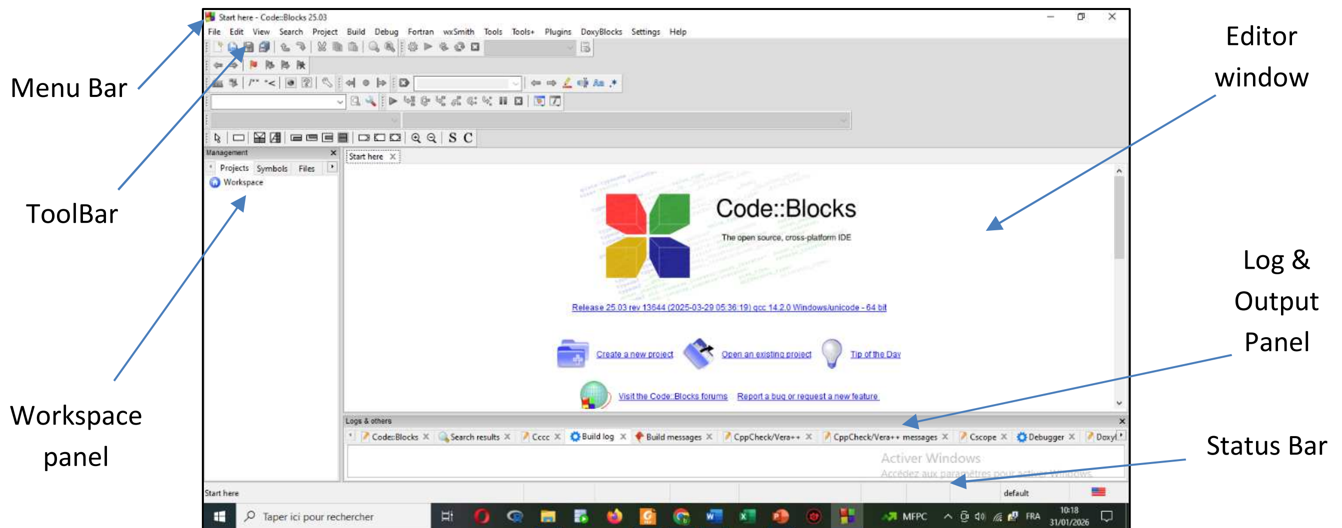


Figure 7 The workspace window

1.5.Overview of the CodeBlocks Main Window

When you open CodeBlocks, the main window provides all the tools you need to write, compile, and run your programs. The key components are:

a. Menu Bar

Located at the top of the window, it contains all the main menus for performing tasks:

- **File** – create, open, save, or close files and projects.
- **Edit** – undo, redo, copy, paste, and find/replace code.
- **Build** – compile, build, and run your programs.
- **Debug** – start debugging, set breakpoints, and step through code.
- **Settings** – customize the IDE and compiler options.

b. Toolbar

Directly below the menu bar, the toolbar provides quick-access buttons for common actions:

- New file/project, open file/project, save.
- Build, rebuild, run, and stop programs.
- Debugging controls like start/stop debug and step through code.

c. Workspace Panel

The workspace panel organizes your projects and files:

- Shows the structure of your project: folders, files, and source code.
- Allows quick navigation between files and functions.

d. Editor Window

The main area where you write and edit your code:

- Supports syntax highlighting for easier reading.
- Each file opens in a separate tab.

- The default welcome screen offers shortcuts to create a new project or open an existing project.

e. Logs & Output Panel

Displays important messages while building and running programs:

- Build messages: shows errors and warnings during compilation.
- Search results: shows results when searching in files.
- Debugger output : displays messages during debugging.

f. Status Bar

Shows useful information:

- Current line and column number.
- Active project and compiler target.

1.6.Structure of C program

A C program is composed of several fundamental components that together define how the program executes. Understanding these components is crucial for writing clear, organized, and functional programs. The main parts of a C program are described below:

a. Preprocessor Directives

Preprocessor directives are lines that begin with the # symbol. These directives are instructions to the compiler that are executed before actual compilation begins. They typically include:

- **#include**: This directive is used to include header files that provide access to standard library functions. For example, ***#include <stdio.h>*** allows the program to use input/output functions like ***printf()*** and ***scanf().***
- **#define**: This directive is used to define constants or macros that can simplify the code. For example, ***#define PI 3.14159*** allows the use of PI instead of writing 3.14159 everywhere.

Preprocessor directives make the program more modular and easier to maintain.

b. Global Declarations (optional)

Global declarations are variables or constants that are declared. These variables exist for the entire duration of the program execution. While global variables can be useful, they should be used sparingly, as excessive use can make programs harder to debug and maintain.

c. The main() Function

The ***main()*** function is the entry point of every C program. When a C program is executed, the execution starts from the first statement in the ***main()*** function. The ***main()*** function has a defined structure:

- Local Declarations: Variables that are needed only within the ***main()*** function are declared here. These are called local variables and their scope is limited to the function.
- Statements and Expressions: This section contains the instructions that perform the tasks of the program, such as calculations, input/output operations, or decision-making using control structures like ***if, for***, and ***while***.
- Return Statement: Typically, return 0; is used to indicate that the program has executed successfully. Some systems use other return values to indicate different types of termination.

1.7.Example of a Basic C Program

Below is an example of a simple C program illustrating the structure described:

Instruction	**Description**
#include <stdio.h>	Preprocessor directive for input/output functions
int counter = 0;	Global variable (optional)
int main() {	
int number;	Local variable declaration
printf("Enter a number: "); scanf("%d", &number);	Input from user
counter = number + 10; printf("The result after adding 10 is: %d\n", counter);	Processing and output
return 0; }	Indicating successful program termination

Table 1 Example of a basic C program

In the following chapters, we will examine each of these components in detail, exploring their purpose, syntax, and usage in practical C programs.

Practical Work N°1: Getting familiar with CodeBlocks

2. PW N°1- Getting familiar with CodeBlocks-

2.1.Objectives

At the end of this session, the student will be able to:

- Launch the CodeBlocks editor.
- Create a new project.
- Open an existing project
- Edit a program file.
- Save a program file.
- Close a file or project.
- Exit the CodeBlocks editor.

2.2.Launch the CodeBlocks editor

To start CodeBlocks, go to Start → Programs → CodeBlocks → CodeBlocks, or simply click the desktop shortcut

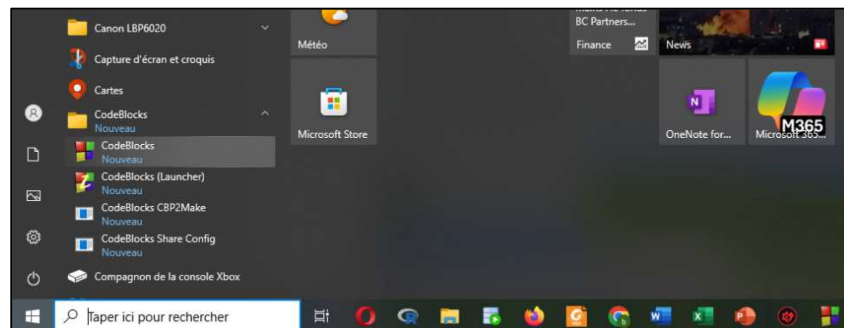


Figure 8 Launch CodeBlocks

2.3.Create a new project

To create a project, click on File → New → Project. In the displayed window, choose 'Console Application', then click on “Go”.

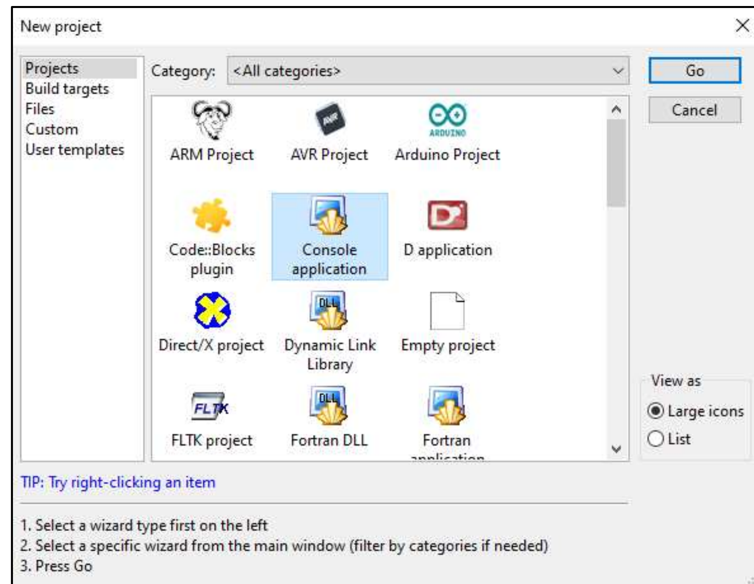


Figure 9 Create a new project

In the displayed window, choose “C” language, then click on “Next”.

Figure 10 Select the language to use in the new project.

In the next window, enter the project title and select the folder where the project will be created, then click on “Next”.

Figure 11 Project information

The last step consists of selecting the compiler to be used, then click on “Finish”.

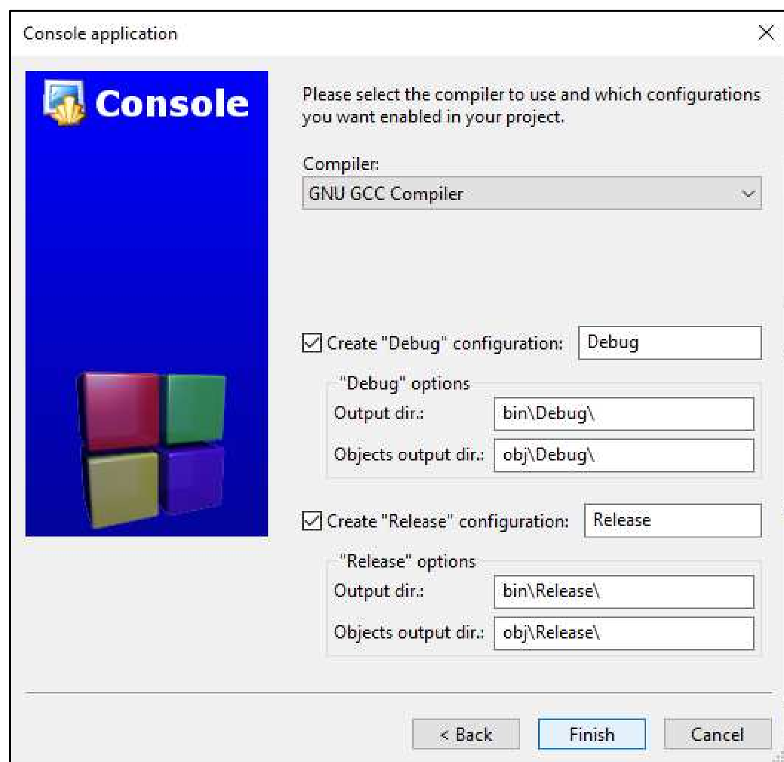


Figure 12 Select the compiler

At the end of the process, the project is created in the selected directory. Within this directory, a subfolder named “sources” is generated, containing a file named “main.c”. The main.c file includes the main function of the program, which contains a basic instruction that displays the message “Hello World” on the screen.

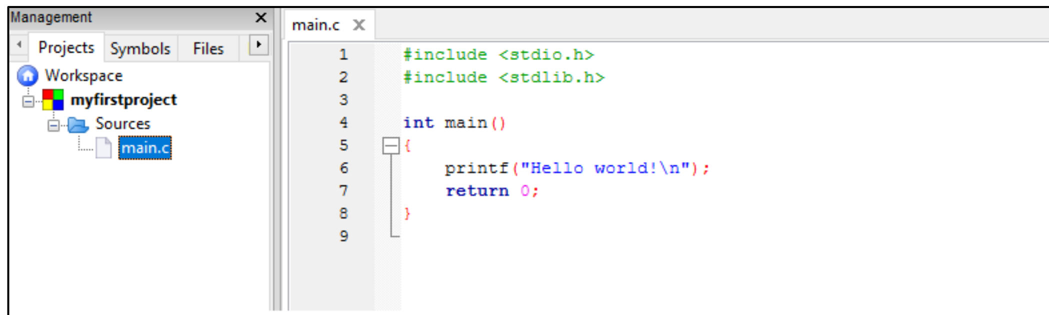


Figure 13 The first program

2.4.Open an existing project

To open an existing project, click *File → Open* from the main menu, then select the project you want to open.

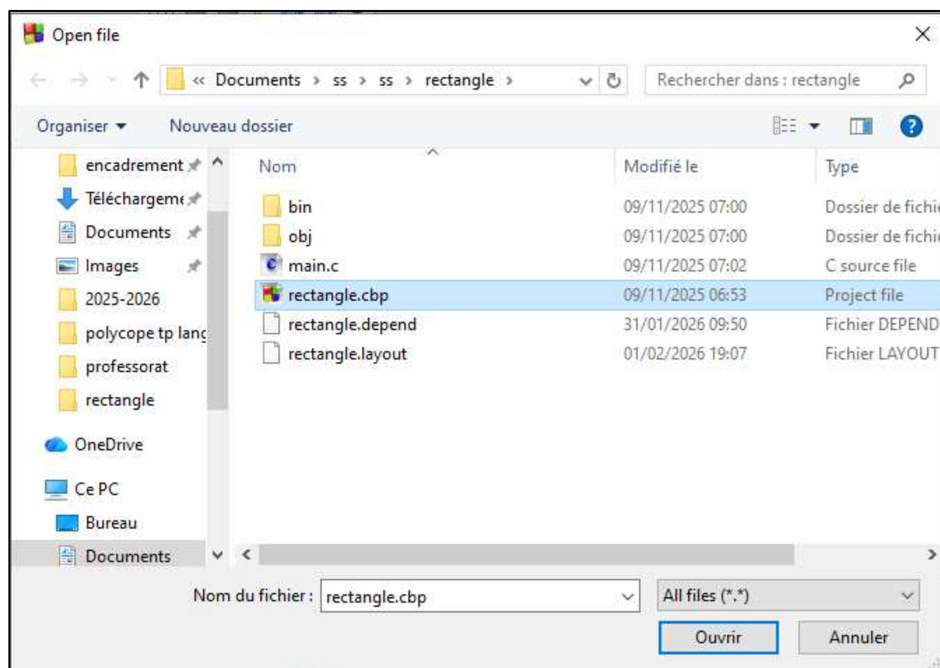


Figure 14 Open an existing project

2.5.Edit a program file

To edit a program file, first open the project containing the file, then double-click its name to display it in the editor.

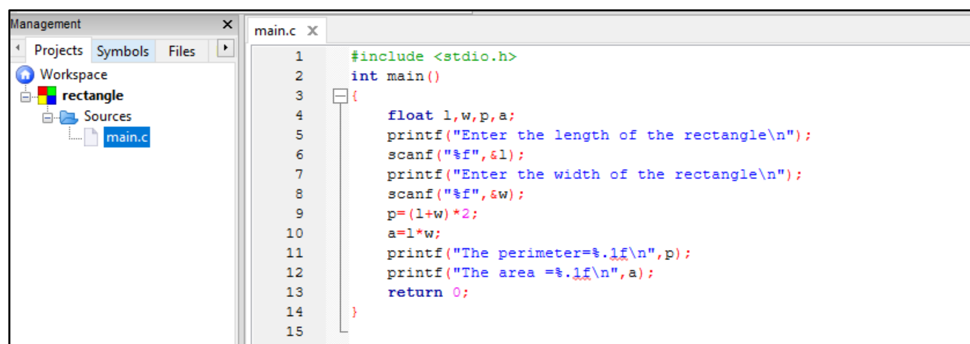
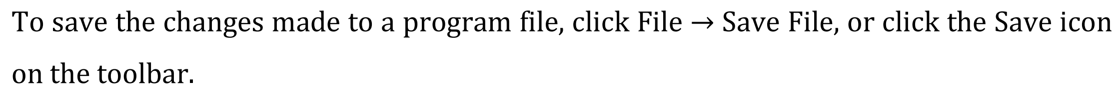


Figure 15 Edit a program file.

2.6.Save a program file

To save the changes made to a program file, click File → Save File, or click the Save icon  on the toolbar.

2.7.Close a file or project.

To close a file or a project, use the corresponding *Close File* or *Close Project* option from the *File* menu.

2.8.Exit the CodeBlocks editor

To close the CodeBlocks editor click File → Quit option from the *File* menu.

2.9.Application exercise

a. Launch the CodeBlocks editor and create a new project named "***My_first_project***".

b. Enter the following program in the main.c file:

```
#include <stdio.h>
int main()
 printf("Hello!\n")
printf("This is my first C program\n");
printf("I use CodeBlocks \n");
return 0;
}
```

c. Save the main.c file.

d. Close the project.

e. Exit CodeBlocks.

f. Relaunch CodeBlocks and open your project.

Practical Work N°2: Edit, compile and execute a program

3. PW N°2- Edit, compile and executer a program-

3.1.Objectives

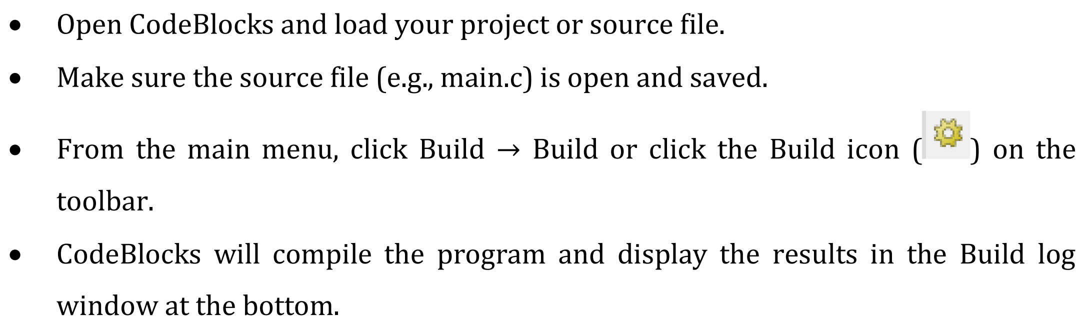
By the end of this session, the student will be able to:

- Compile a program.
- Identify and correct compilation errors.
- Run a program.

3.2.Compile a program

Compilation is the process by which a program written in a specific programming language (in this case, the C language) is translated into an executable form that can be processed by the computer. For this translation to occur successfully, the program must strictly conform to the syntactic rules of the language. Consequently, a program can only be executed if it is free of syntax errors.

To compile a program in CodeBlcoks :

- Open CodeBlocks and load your project or source file.
- Make sure the source file (e.g., main.c) is open and saved.
- From the main menu, click Build → Build or click the Build icon () on the toolbar.
- CodeBlocks will compile the program and display the results in the Build log window at the bottom.

If Compilation is successful

- The message "Build finished successfully" appears.
- No error messages are shown.

If Compilation errors occur

- Error messages are displayed in the Build log.
- Double-click an error message to go directly to the corresponding line in the source code.
- Correct the errors and compile again.

The figure below illustrates a compilation error occurring at line 6. The error message indicates that a semicolon (;) is expected before the ***printf*** instruction.

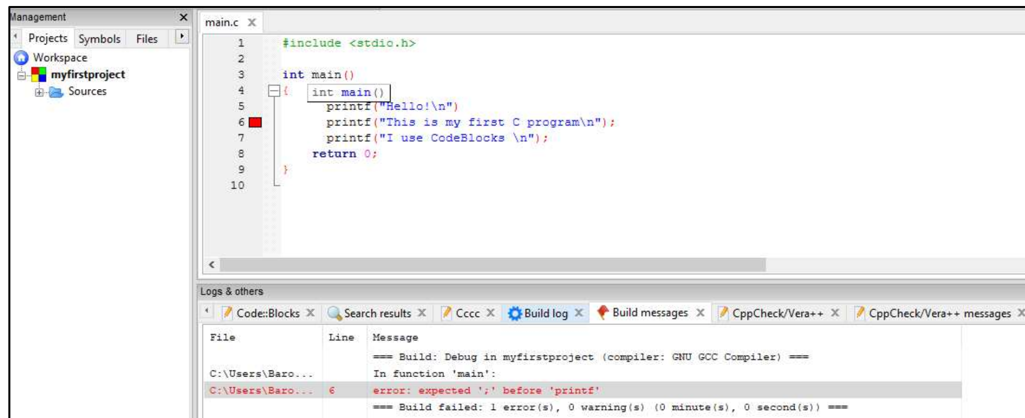


Figure 16 Compilation error

3.3.Run a program

Once the program has been successfully compiled, you can execute it by choosing Build → Run from the main menu or by clicking the **Run** button ( ) on the toolbar. The figure below shows an example of program execution.

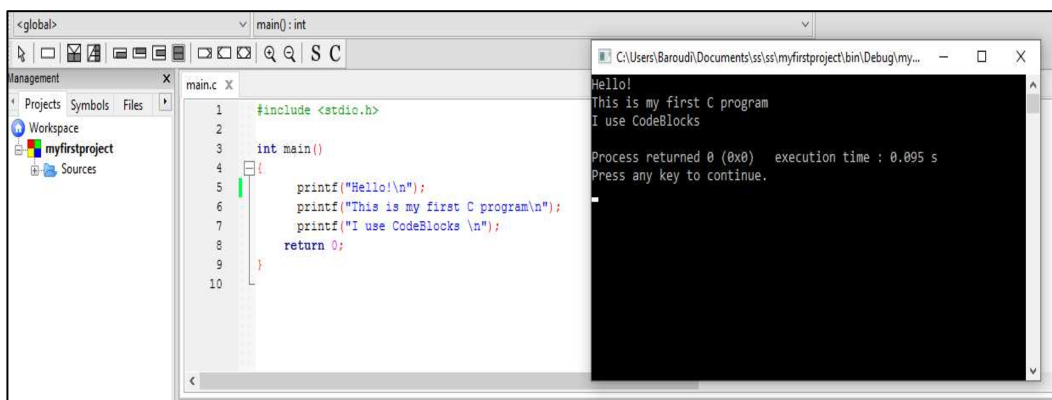


Figure 17 program execution

3.4. Exercice d'application

1. Launch CodeBlocks.
2. Open the project “my_first_project” created during the previous session.
3. Compile the program and correct any errors.
4. Run the program.

Practical Work N°3: Basic data types

4. PW N°3-Basic data types-

4.1.Objectives

A la fin de cette séance, l'étudiant sera capable de :

- Basic Structure of a C Program.
- Déclarer des constantes
- Déclarer des variables
- Identifier les types de données supportés en langage C.
- Identifier les fonctions standards applicables sur chaque type.

4.2.Basic Structure of a C Program

Before writing a C program, it is important to understand its basic structure, which defines how the different parts of the program are organized and executed. The main parts of a C program are illustrated in figure 18.

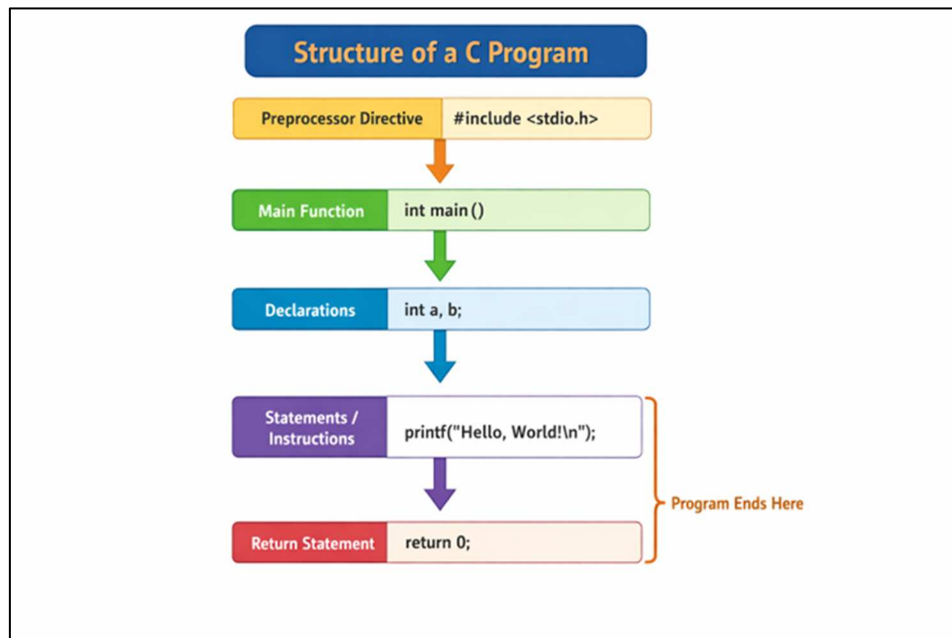


Figure 18 Basic structure of a C program

a. Preprocessor Directives

Lines that start with # (e.g., #include <stdio.h>) tell the compiler to include libraries or perform other preprocessing tasks before compilation.

b. Main Function

Every C program must have a main() function.

Execution of the program starts from this function.

c. Declarations

Variables are declared here (e.g., int a, b;) before they are used in the program.

d. Statements / Instructions

The commands that perform actions, such as ***printf()*** to display output.

Each statement ends with a semicolon (;).

e. Return Statement

return 0; indicates that the program has executed successfully.

4.3.Declaration of Constants

In C, a constant is a value that cannot be changed during the program execution. There are several ways to declare constants:

a. Using the *const* keyword

Examples:

const int DAYS_IN_WEEK = 7;

const float PI = 3.14159;

- const tells the compiler that the value cannot be modified.
- Attempting to change it later will produce a compilation error.

b. Using *#define* preprocessor directive

Examples:

#define MAX_SCORE 100

#define GREETING "Hello, World!"

4.4.Declaration of variables

A variable is a storage location that can hold a value for use in a program. Variables can have different types depending on the values they store.

In C, a variable must be declared before it is used. The general syntax is:

data_type variable_name;

- ***data_type***: Specifies the type of value the variable will hold (e.g., int, float, char).
- ***variable_name***: The name of the variable, chosen by the programmer.

Examples:

int age; ➔ Declares an integer variable named age

float salary; ➔ Declares a floating-point variable named salary

char grade; ➔ Declares a character variable named grade

You can also assign an initial value when declaring the variable:

Examples:

int age = 18;

float salary = 2500.50;

char grade = 'A';

4.5.Basic data types

C provides several fundamental data types to store different kinds of values:

Data Type	Description	Example
int	Stores integer numbers (whole numbers).	int age = 25;
float	Stores single-precision floating-point numbers (decimal numbers).	float price = 9.99;
double	Stores double-precision floating-point numbers (more precise decimals).	double pi = 3.14159;
char	Stores a single character.	char grade = 'A';

Table 2 Basic Data Types

a. Int type

Used to store whole numbers from -2147483648 to 2147483647.

Example : int age = 25;

There are several variants of this type.

Type	Domain
short int	From -32768 to 32767
unsigned int	From 0 to 4294967295
long int	From -9,223372036854775808 to 9223372036854775807
signed int	From -2147483648 to 2147483647

Table 3 Variants of type int

The operations that can be performed on the ***int*** type are

Operator	Description	Example	Result
+	Addition	a + b	Sum of a and b
-	Subtraction	a - b	Difference of a and b
*	Multiplication	a * b	Product of a and b
/	Division	a / b	Quotient of a divided by b
%	Modulus (remainder)	a % b	Remainder of a / b

Table 4 Operations performed on int type

b. Float type

The float type is used to store real numbers with decimal points. It represents single-precision floating-point values and is commonly used for calculations that require fractional results.

Example : float price = 9.99;

The operations that can be performed on the float type are

Operator	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division

Table 5 Operations performed on float type

c. Double type

The double type is used to store double-precision floating-point numbers, allowing the representation of real values with higher precision than float. It typically occupies 8 bytes of memory and is used when greater numerical accuracy is required.

Example: double pi = 3.14159;

The double type supports the same operations as the float type.

d. Char type

The char type is used to store a single character, such as a letter, digit, or symbol. It occupies 1 byte of memory and internally stores the ASCII code corresponding to the character.

Example: char grade = 'A';

e. Boolean type

The C language does not define a boolean type as a basic data type. Traditionally, boolean values are represented using integers, where 0 denotes false and any non-zero value denotes true.

Since the **C99 standard**, C provides a boolean type through the inclusion of the <stdbool.h> header. This header defines the type bool and the constants true and false. Internally, bool is implemented as an integer type.

Operations applicable to the bool type include:

- **Logical operations:** && (AND), || (OR), ! (NOT)
- **Relational operations:** == (equal), != (not equal), > (greater then), >= (greater then of equal), < (less then), <= (less then or qual).

The bool type is mainly used in conditional expressions and control structures to improve code readability.

f. Predefined functions

The C language provides a set of functions that can be used for each data type. A standard function allows the automatic computation of a value, which depends on the data type to which the function is applied. The table below presents the main functions used for each data type.

Function	**Description**	**Input data type**	**Output data type**
abs	The absolute value of an integer.	int	int
fabs	The absolute value of a real number.	float/double	float/double
sqr	The square of a number.	int/float	int/float
sqrt	The square root of a number	int/float	double
exp	The exponential of a number.	double	double
log	The natural logarithm of a number.	double	double
sin	The sine of a number.	double	double
cos	The cosine of a number.	double	double
atan	The arctangent of a number.	double	double
round	The nearest integer to a real number.	double	double

Table 6 Some predefined functions in C language

Examples :

- abs(-5) calculates the absolute value of -5, which is 5.
- sqr(3) calculates the square of 3, which is 9.
- sqrt(25) calculates the square root of 25, which is 5.
- round(26.9) returns 27, which is the nearest integer to 26.9.

g. The Precedence of Operators

Arithmetic and/or logical expressions that are not enclosed in parentheses must be evaluated according to the priority of operators. The operators are ordered as follows:

- Priority 1: ! (Not)
- Priority 2: *, /, % (modulus), && (And)
- Priority 3: +, -, || (Or)
- Priority 4: =, <, >, <=, >=, !=

In the absence of parentheses, operators with the same priority are evaluated from left to right.

Example:

The expression 2*A + 3 - B is evaluated as follows:

- 2*A → because the * operator has the highest priority.
- 2*A + 3 → + and - have the same priority, so evaluation starts from the left.
- 2*A + 3 - B → final result after evaluating left to right.

To enforce a specific order of evaluation, parentheses should be used.

4.6.Practical Exercise

1. Create a new project named **"Cercle"**.
2. Declare a constant named **Pi** with the value 3.14.
3. Declare three variables named **R**, **P**, and **S** of type **real** (floating-point).
4. Save the program file.

4.7. Practical exercise solution

The program is presented as follows:

```
#include <stdio.h>
int main() {
 const double Pi = 3.14;
double R, P, S;
 // The program will be completed in future exercises
 return 0;
}
```

The program will be completed during the exercises in the next sessions.

Practical Work N°4: Basic instructions

5. PW N°4-Basic instructions-

5.1.Objectives

At the end of this session, the student will be able to use:

- The assignment statement (=).
- The output instruction (printf).
- The input instruction (scanf).

5.2.The assignment statement

Assignment allows a value, described by an expression, to be stored in a variable. In C, assignment is represented by the = symbol.

A value can be assigned directly to a variable or as the result of a calculation.

Syntax:

A = E;

Where E represents an expression.

Examples:

- The statement A = 5; assigns the value 5 directly to the variable A.
- The statement A = B * C; assigns to A the result of multiplying B by C.

Type Compatibility

Assignment is valid only if the type of the expression is compatible with the type of the variable.

- If A is of type int, then A = "hello"; is not valid.
- If A is an int and B is a float, then A = B; is valid, but the fractional part of B will be truncated.
- Conversely, B = A; is valid because an int value can be stored in a float without loss.

It is possible to assign the result of a boolean expression to a variable of type bool (C99 and later).

#include <stdbool.h>

int a, b;

bool c;

c = a > b;

c=a>b returns true if a is greater than b, false otherwise.

5.3.The output statement (printf)

The *printf* function is used to display output on the screen. It can print text, numbers, characters, and formatted data.

Syntax:

printf("format string", variables);

Example:

int x = 5;

printf("The value of x is %d\n", x);

- %d is a **format specifier** indicating that an integer will be printed.
- \n represents a **new line**.

printf allows combining text and variables in a single output statement, making it very versatile for displaying results.

Common *printf* Format Specifiers in C

Specifier	**Type of Data**	**Example Code**	**Output**
%d	int	*printf("%d", 42);*	42
%i	int	*printf("%i", 42);*	42
%f	Float / Double	*printf("%f", 3.14);*	3.140000
%.2f	Float with 2 decimals	*printf("%.2f", 3.14);*	3.14
%c	Character	*printf("%c", 'A');*	A
%s	String	*printf("%s", "Hello");*	Hello
%e	Scientific notation	*printf("%e", 1234.56);*	1.234560e+03

Table 7 Common printf Format Specifiers in C

5.4.Reading statement (scanf)

The *scanf* function is used to read input from the user. It can read numbers, characters, and strings and store them in variables.

Syntax:

scanf("format string", &variables);

Example:

int x;

printf("Enter a number: ");

scanf("%d", &x);

printf("You entered: %d\n", x);

Note:

- The **& (address-of) operator** is used to provide the memory location of the variable.

scanf allows the program to interactively get data from the user, which can then be used in calculations or displayed using *printf*.

Common scanf Format Specifiers in C

Specifier	Type of Data	Example Code	Input Example
%d	int	*scanf("%d", &x);*	42
%i	int	*scanf("%i", &x);*	42
%f	Float	*scanf("%f", &y);*	3.14
%lf	Double	*scanf("%lf", &z);*	3.141592
%c	Character	*scanf(" %c", &ch);*	A
%s	String (array)	*scanf("%s", str);*	Hello

Table 8 Common *scanf* Format Specifiers in C

5.5.Practical exercise

In this exercise, we will complete the program created during the practical exercise in the previous session.

1. Open the main.c file of the project "Cercle".
2. In the executable section (the instructions section), add the necessary statements to:
   a. Display the message: Enter the radius of the circle.
   b. Read the value of the radius and store it in the variable R.
   c. Calculate the perimeter of the circle and assign it to the variable P.
   d. Calculate the area of the circle and assign it to the variable S.
   e. Display the value of the perimeter in the form: The perimeter of the circle = value.
   f. Display the value of the area in the form: The area of the circle = value.
3. Compile and run the program with R = 2.5 and then with R = 3.

5.6. Practical exercise solution

The previously created program, stored in the project **“cercle”**, is as follows:

```
#include <stdio.h>
int main() {
 const double Pi = 3.14;
double R, P, S;
 // The program will be completed in future exercises
 return 0;
}
```

In this exercise, you are asked to modify the executable section by adding instructions.

a**.** To display the message **“Enter the radius of the circle”**, use the ***printf*** instruction:

printf("Enter the radius of the circle: \n");

b**.** To read the value of the radius and store it in the variable R, use the ***scanf*** instruction:

scanf("%lf", &R);

c. To calculate the perimeter of the circle and assign it to the variable P, use an assignment statement. The perimeter of a circle is 2 * π * R. Replace π with the constant Pi previously declared, and assign the result to P as follows:

P = 2 * Pi * R;

d. Similarly, calculate the **area** of the circle using:

S = Pi * R * R;

It is also possible to use the ***sqr*** function (square) if available in your context:

S = Pi * sqr(R);

e. To display the result, use ***printf*** with two parameters:

- The first parameter is the message to display, e.g., "The perimeter of the circle =".
- The second parameter is the calculated perimeter stored in the variable P.

printf("The perimeter of the circle = %.2f", P);

f. Display the area in the same way using:

printf("The area of the circle = %.2f", S);

Notes:

- C is **case-sensitive**. Variable names must respect case. For example, R is different from r.
- The format ***%.2f*** in ***printf*** displays the result with **two digits after the decimal point**.

The final version of the program is

```
#include <stdio.h>
int main() {
  const double Pi = 3.14;
  double R, P, S;
  printf("Enter the radius of the circle: \n");
  scanf("%lf", &R);
  P = 2 * Pi * R;
  S = Pi * R * R;
  printf("The perimeter of the circle = %.2f\n", P);
  printf("The area of the circle = %.2f\n", S);
  return 0;
}
```

To run the program, simply click on the Build → Run menu. The program displays the message "Enter the radius of the circle" and waits for the user to enter the radius value. We enter the value 2.5 and then press the Enter key. The program then displays the perimeter and the area of the circle.

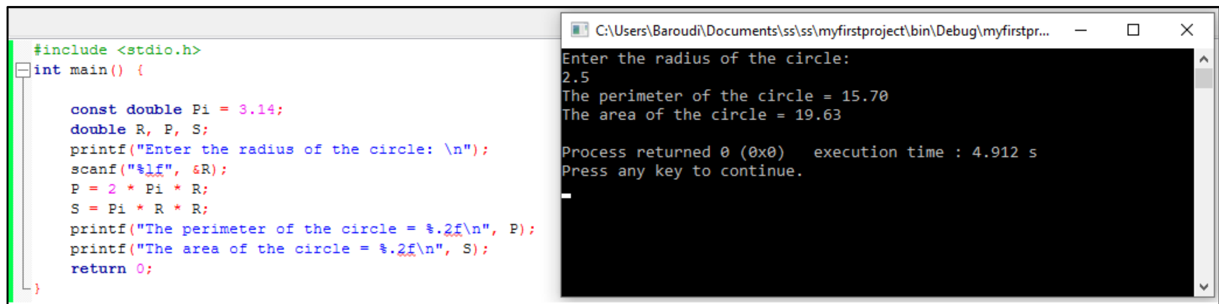


Figure 19 First execution of the program

We proceed in the same way to run the program with a radius value of 3.

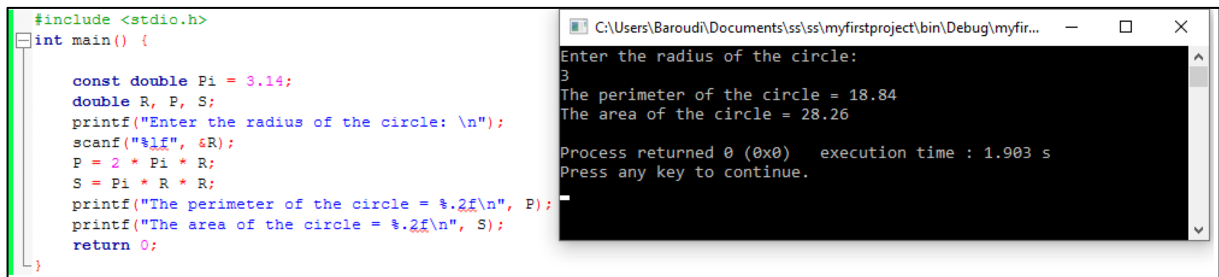


Figure 20 Second execution of the program

5.7.Exercises

Exercise 1

- Write a program that calculates the average of 3 numbers entered from the keyboard.

Exercise 2

- Write a program that allows the user to enter the parameters of a quadratic equation and calculates its discriminant Δ.

Exercise 3

- Write a program that calculates the Volume (V) and Surface Area (S) of a cylinder with radius R and height H, knowing that:

  Cylinder volume: $V=\Pi.R^2.H$

  Cylinder area: $S=2.\ \Pi\ .R.H$

  Π est une constante = 3.14159

Exercise 4

- Write a program that allows the user to:
  - Enter 3 marks of a student along with their respective coefficients.
  - Calculate and display the average.

Exercise 5

An olive oil merchant ships a quantity Q of bottles at a unit price P. Write a program that reads the number of bottles Q and the unit price P and displays the total amount of the order.

Exercise 6

- Write a program that calculates and displays the distance between two points, knowing their coordinates $P_1(x_1,y_1)$ and $P_2(x_2,y_2)$.

Exercise 7

- Write a program that swaps the values of two variables of type float.

Practical Work N°5: Conditions

6. PW N°5- Conditions-

6.1.Objectives

At the end of this session, the student will be able to:

- Express simple and complex conditions
- Express nested conditions
- Use the multiple-choice statement (switch / case)

6.2.Express simple and complex conditions

To express a condition in a C program, you need to use the **if statement**. Depending on the situation, three versions are available:

a. Simple if Statement

Syntax:

```
if (condition) {
   // instructions to execute if condition is true
}
```

Example:

```
int x = 10;
if (x > 5) {
   printf("x is greater than 5\n");
}
```

The code inside the braces { } is executed **only if the condition is true**.

b. if...else Statement

Syntax:

```
if (condition) {
   // instructions if condition is true
} else {
   // instructions if condition is false
}
```

Example:

```
int x = 3;
if (x > 5) {
   printf("x is greater than 5\n");
} else {
   printf("x is not greater than 5\n");
}
```

The else block is executed **only if the condition is false**.

c. Nested if Statement

Syntax:

```
if (condition1) {
   // instructions if condition1 is true
   if (condition2) {
      // instructions if condition2 is true
   }
} else {
   // instructions if condition1 is false
}
```

Example:

```
int x = 8;
int y = 12;
if (x > 5) {
   if (y > 10) {
      printf("x > 5 and y > 10\n");
   }
}
```

6.3.Illustrative Example1

The following program reads a student's average grade and displays **"Admitted"** if the average is greater than or equal to 10, and **"Failed"** otherwise.

```
#include <stdio.h>
int main() {
   float m;
  printf("Enter the student's average: \n");
  scanf("%f", &m);
   if (m >= 10) {
      printf("Admitted\n");
   } else {
      printf("Failed\n");
   }
   return 0;
       }
```

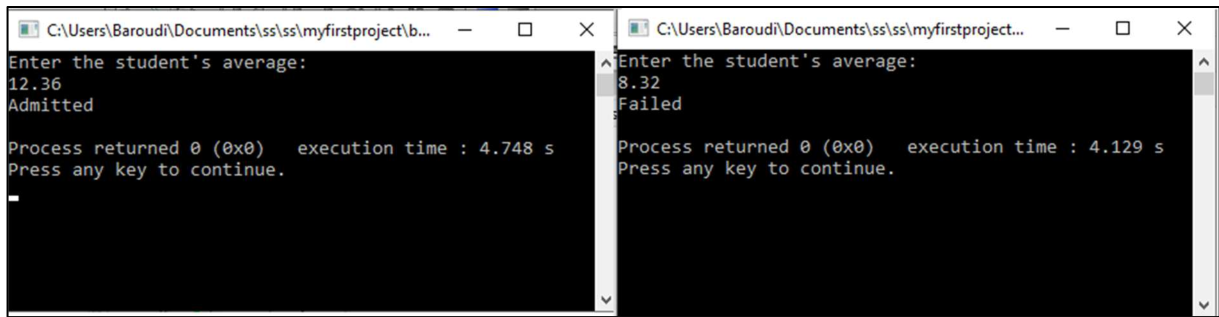


Figure 21 Program output: Admitted or Failed based on student's average.

Note :

It is possible to combine simple conditions to form a complex condition. The combination is done using logical operators (not, and, or, and xor).

Example 1

```
if (a > 0) && (b > 0) then
   inst1;
```

In this example, the instruction inst1 is executed only if both variables a and b are greater than zero.

Example 2

```
if (a > 0) || (b > 0) then
   inst1;
```

In this example, the instruction inst1 is executed if at least one of the two variables a or b is greater than zero.

6.4.Illustrative Example2

The following program displays the sign of the product of two numbers without actually calculating the product. There are three possible cases:

- The product is positive if the two numbers have the same sign (both are positive or both are negative).
- The product is zero if one of the two numbers is zero.
- The product is negative if it is neither zero nor positive.

```
#include <stdio.h>
int main() {
  float  a, b;
  printf("Enter a number: \n");
  scanf("%f", &a);
  printf("Enter a second number:\n ");
  scanf("%f", &b);
  if ((a > 0 && b > 0) || (a < 0 && b < 0)) {
```

```
    printf("The product is positive\n");
  } else if (a == 0 || b == 0) {
    printf("The product is zero\n");
  } else {
    printf("The product is negative\n");
  }
  return 0;
}
```

```
C:\Users\Baroudi\Documents\ss\ss\myfirstproject\bi...
Enter a number:
-3
Enter a second number:
4
The product is negative

Process returned 0 (0x0)   execution time : 7.760 s
Press any key to continue.
```

```
C:\Users\Baroudi\Documents\ss\ss\myfirstproject\bi...
Enter a number:
-5.2
Enter a second number:
-9.3
The product is positive

Process returned 0 (0x0)   execution time : 6.525 s
Press any key to continue.
```

```
C:\Users\Baroudi\Documents\ss\ss\myfirstproject\bi...
Enter a number:
8
Enter a second number:
3
The product is positive

Process returned 0 (0x0)   execution time : 3.778 s
Press any key to continue.
```

```
C:\Users\Baroudi\Documents\ss\ss\myfirstproject\bi...
Enter a number:
8
Enter a second number:
0
The product is zero

Process returned 0 (0x0)   execution time : 2.621 s
Press any key to continue.
```

Figure 22 program output: sign of the product of two numbers

6.5.Multiple choice statement (Switch/Case)

A switch-case statement is a control structure that allows a program to branch to different sections of code based on the value of a variable or expression. Instead of writing multiple *if-then-else* statements, a switch-case can make the code cleaner and easier to read when there are many discrete possible values. Writing switch-case statement is based on the following key points:

- The expression in a switch must evaluate to a value (usually an integer, character).
- Each case specifies a constant value. If the expression matches that value, the program executes the corresponding code.
- The default case (optional) runs if none of the specified values match.
- Switch-case cannot handle general conditions like x > 0; such cases need if-else statements.

Syntax

```
switch (expression) {
  case constant1:
    // code to execute if expression == constant1
    break;
```

```
    case constant2:
      // code to execute if expression == constant2
      break;
    // you can have as many cases as needed
    ...
    default:
      // code to execute if expression doesn't match any case
}
```

Where:

- **switch(expression)**: The variable or expression whose value is tested.
- **case constant:** Each possible value of the expression. If the expression matches this value, the code under this case runs.
- **break;** Ends the case. Without it, the program continues executing the next case(s) ("fall-through").
- **default:** Optional. Runs if none of the cases match the expression.

6.6.Illustrative Example3

The following program simulates a four-operation calculator. The user first enters two numbers, then enters the operator of the operation to be performed (+, -, *, or /). To implement this program, we need:

- Three variables (a, b, res) to represent, respectively, the numbers entered by the user and the result of the operation.
- One variable (op) of type character to represent the operation to be performed.

```
#include <stdio.h>
int main() {
  float a, b, res;
  char op;
  printf("Enter the first number: \n");
  scanf("%f", &a);
  printf("Enter the second number:\n ");
  scanf("%f", &b);
  printf("Enter the operator (+, -, *, /):\n ");
  scanf(" %c", &op);
  switch (op) {
    case '+':
      res = a + b;
      printf("Result = %.2f\n", res);
```

```
            break;
        case '-':
            res = a - b;
            printf("Result = %.2f\n", res);
            break;
        case '*':
            res = a * b;
            printf("Result = %.2f\n", res);
            break;
        case '/':
            if (b != 0) {
                res = a / b;
                printf("Result = %.2f\n", res);
            } else {
                printf("Error: division by zero\n");
            }
            break;
        default:
            printf("Invalid operator\n");
    }
    return 0;
}
```

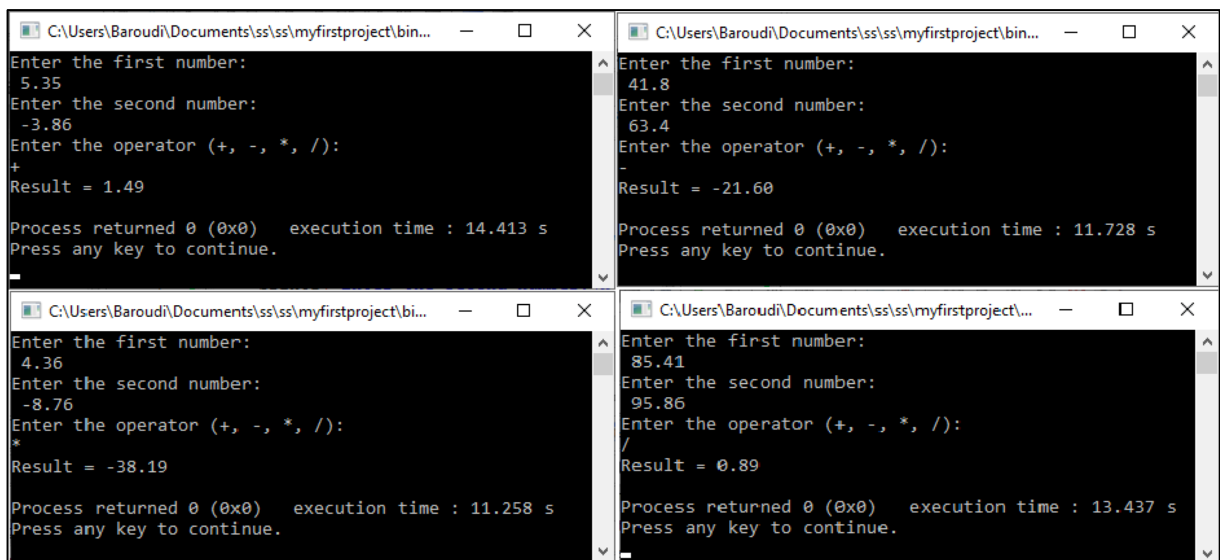


Figure 23 switch-case statement: calculator program output

6.7.Application exercise

Write a program that reads a student's average grade and then displays one of the following distinctions:

- "Very Good": if the student's average is greater than or equal to 16.
- "Good": if the student's average is between 14 and 16 (14 ≤ average < 16).
- "Fairly Good": if the average is between 12 and 14 (12 ≤ average < 14).
- "Passable": if the average is between 10 and 12 (10 ≤ average < 12).
- "Poor": if the average is less than 10.

6.8.Application exercise solution

To solve this problem, we need to use a variable m to store the student's average grade. To handle the five possible cases, it is necessary to use four nested if-then-else statements. The use of a case (or switch) statement is not possible in this exercise because the conditions are expressed using inequalities (>=, <, etc.).

The program is structured as follows:

```
#include <stdio.h>
int main() {
  float m;
  printf("Enter the student's average: \n");
  scanf("%f", &m);
  if (m >= 16)
    printf("Very Good\n");
  else if (m >= 14)
    printf("Good\n");
  else if (m >= 12)
    printf("Fairly Good\n");
  else if (m >= 10)
    printf("Passable\n");
  else
    printf("Poor\n");
  return 0;
}
```

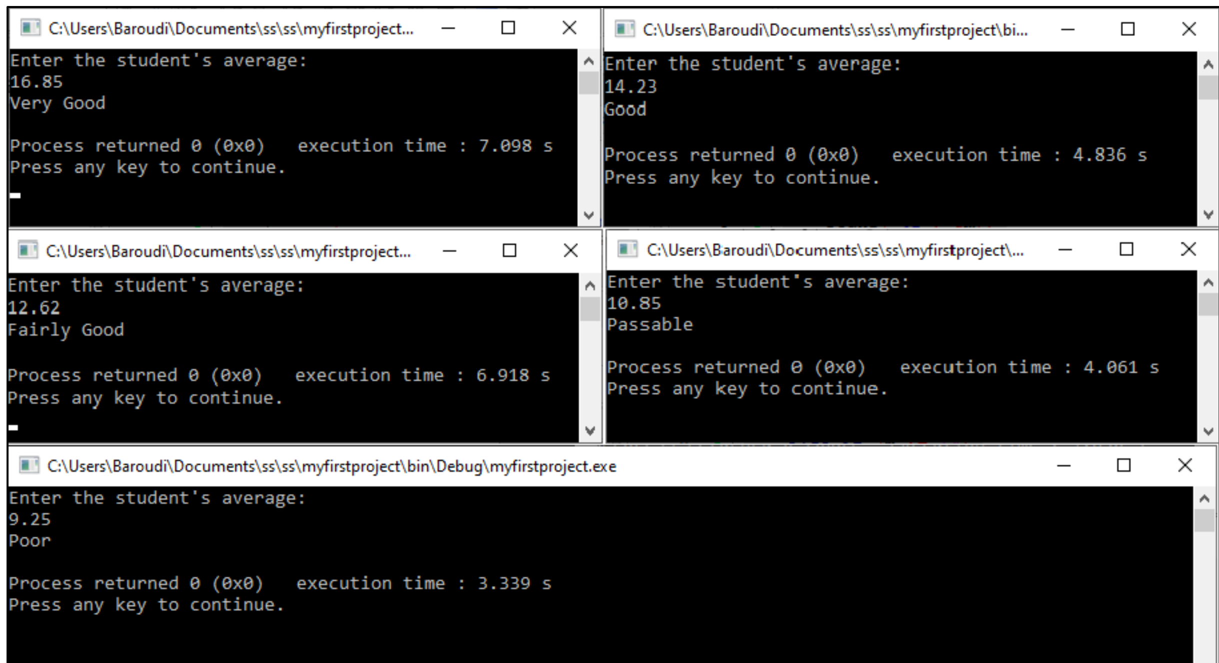


Figure 24 Nested if-then-else : program output.

6.9.Exercises

Exercise 1

Using the alternative structure, write a program that reads two numbers and displays the maximum of the two.

Write the same program using the conditional structure.

Exercise 2

Write a program that reads the temperature of water and displays its state:

- “Ice” if the temperature ≤ 0°C
- “Liquid” if 0°C < temperature < 100°C
- “Vapor” if the temperature ≥ 100°C

Exercise 3

Write a program that calculates and displays the Body Mass Index (BMI) of a person with weight P and height T, knowing that: BMI= P/T^2

The program should then display one of the following messages:

- “Underweight” if BMI < 16
- “Thin” if 16 ≤ BMI < 18
- “Normal weight” if 18 ≤ BMI ≤ 25
- “Overweight” if 25 < BMI < 30
- “Obesity” if BMI ≥ 30

Exercise 4

A large store offers its customers the possibility to receive a discount on the total amount of their purchases. The discount rate depends on the result of rolling a die and is determined as follows:

- If the result is 1, the discount rate = 5%
- If the result is 2, the discount rate = 7%
- If the result is 3, the discount rate = 8%
- If the result is 4, the discount rate = 15%
- If the result is 5, the discount rate = 20%
- If the result is 6, the discount rate = 50%

Using a multiple-choice statement (case), write a program that:

1. Reads the total purchase amount and the die result,
2. Calculates and displays the discount rate,
3. Calculates and displays the net amount to pay.

Practical Work N°6: Repetitive structures (Loops)

7. PW N°6- Repetitive structures (Loops)-

7.1. Objectives

A la fin de cette séance, l'étudiant sera capable d'utiliser :

- La boucle For
- La boucle While
- La boucle Do…While

7.2. For loop

The ***for*** loop in C is a control structure used to repeat a block of code a fixed number of times. It is especially useful when the number of iterations is known in advance.

Syntax

for (initialization; condition; update) {

// statements to be repeated

}

Where

- initialization: executed once at the beginning (e.g., int i = 0;)
- condition: checked before each iteration; the loop continues while it is true
- update: executed after each iteration (e.g., i++)

The variable i (also called the loop counter) must be of type int.

Examples

The statement for *(i=0 ;i<10;i++)* works as follows:

- i = 0 → initializes the variable i to 0. This happens once, at the start of the loop.
- i < 10→ This is the condition. The loop will continue as long as i is less than 10.
- i++→ This is the update. After each iteration, the value of i is increased by 1.
- Result: The loop runs **10 times**, with i taking the values 0, 1, 2, ..., 9.

The statement for *(i=10 ;i>0;i--)* works as follows:

- i = 0 → initializes the variable i to 10.
- i > 10→ The loop will continue as long as i is greater than 10.
- I--→ After each iteration, the value of i is decreased by 1.
- Result: The loop runs **10 times**, with i taking the values 9, 8, 7, ..., 0.

The statement for *(i=0 ;i<10;i+=2)* works as follows:

- i = 0 → initializes the variable i to 0.
- i < 10→. The loop will continue as long as i is less than 10.
- i+=2→. After each iteration, the value of i is increased by 2.
- Result: The loop runs **5 times**, with i taking the values 0, 2, 4, 6, 8.

a. Illustrative Example1

The following program displays the numbers from 0 to 10 in ascending order.

```
#include <stdio.h>
int main() {
    int i;

    for (i = 0; i <= 10; i++) {
        printf("%d\n", i);
    }

    return 0;
}
```

```
0
1
2
3
4
5
6
7
8
9
10

Process returned 0 (0x0)   execution time : 0.240 s
Press any key to continue.
```

Figure 25 For Loop : Example 1

b. Illustrative Example2

The following program displays the numbers from 0 to 10 in descending order.

```
#include <stdio.h>
int main() {
    int i;

    for (i = 10; i >= 0; i--) {
        printf("%d\n", i);
    }

    return 0;
}
```

```
10
9
8
7
6
5
4
3
2
1
0

Process returned 0 (0x0)   execution time : 0.130 s
Press any key to continue.
```

Figure 26 For Loop : Example 2

7.3.While Loop

This structure allows a process to be repeated **zero or more times** and stops when the execution condition is no longer satisfied. In other words, when the execution condition is true, the process is carried out; otherwise, the loop stops.

Syntax

while (condition) {

// statements to be executed repeatedly

}

Where

- condition: a boolean expression. The loop continues as long as this condition is true.
- The statements inside the braces { } are executed repeatedly while the condition holds.

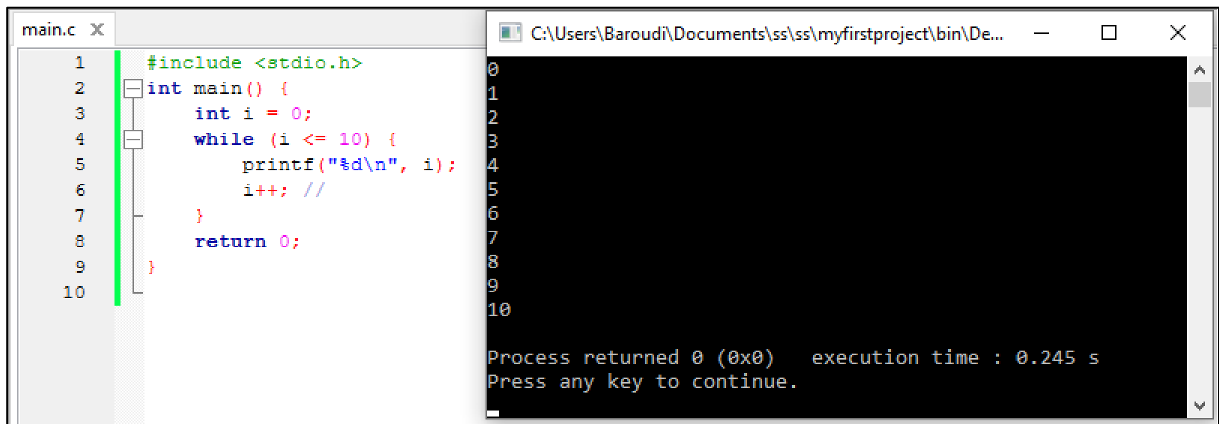
- If the condition is false at the beginning, the statements may not execute at all.

a. Illustrative Example

We take the example that displays the numbers from 0 to 10 in order, but this time using a **while loop**.

To display the numbers in **ascending order**, you simply need to:

- Initialize the counter i to 0.
- Repeat the execution of the process as long as i <= 10.
- Increase the counter by 1 after each iteration.



```c
#include <stdio.h>
int main() {
    int i = 0;
    while (i <= 10) {
        printf("%d\n", i);
        i++; //
    }
    return 0;
}
```

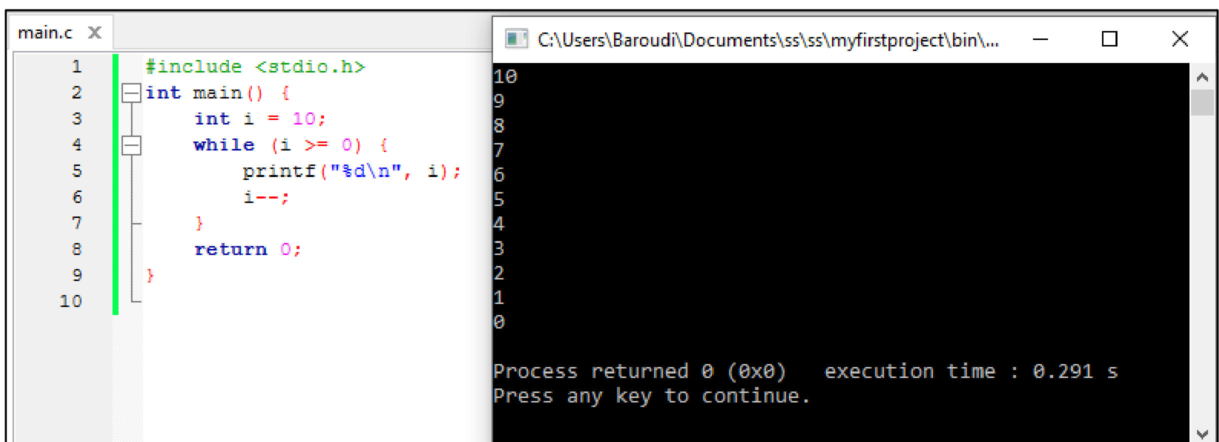
```
0
1
2
3
4
5
6
7
8
9
10

Process returned 0 (0x0)   execution time : 0.245 s
Press any key to continue.
```

Figure 27 While Loop: Example 1 (ascending order).

To display the numbers in **descending order**, you simply need to:

- Initialize the counter i to 10.
- Repeat the execution of the process as long as i >= 0.
- Decrease the counter by 1 after each iteration.



```c
#include <stdio.h>
int main() {
    int i = 10;
    while (i >= 0) {
        printf("%d\n", i);
        i--;
    }
    return 0;
}
```

```
10
9
8
7
6
5
4
3
2
1
0

Process returned 0 (0x0)   execution time : 0.291 s
Press any key to continue.
```

Figure 28 While Loop: Example 1 (descending order).

7.4.Do-While Loop

The do-while loop is a control structure that allows a block of code to be executed at least once, and then repeated as long as a given condition is true.

Unlike the while loop, where the condition is checked before the first execution, the do-while loop checks the condition after executing the code. This guarantees that the loop body runs at least once, even if the condition is initially false.

It's important to notice that:

- The loop continues only after the condition is evaluated.
- The do-while loop is often called a post-test loop, because the condition is tested after the loop body.

Think of it as a "repeat-until" loop (a type of loop that **does not exist** in standard C) — it always executes first, then checks whether to repeat.

Syntax

do {

// statements to execute

} while (condition);

It's important to notice that:

- The loop continues only after the condition is evaluated.
- The do-while loop is often called a post-test loop, because the condition is tested after the loop body.
- Think of it as a "repeat-until" loop (a type of loop that **does not exist** in standard C) — it always executes first, then checks whether to repeat.

a. Illustrative Example

We take the example that displays the numbers from 0 to 10 in order, but this time using a **do-while loop**.

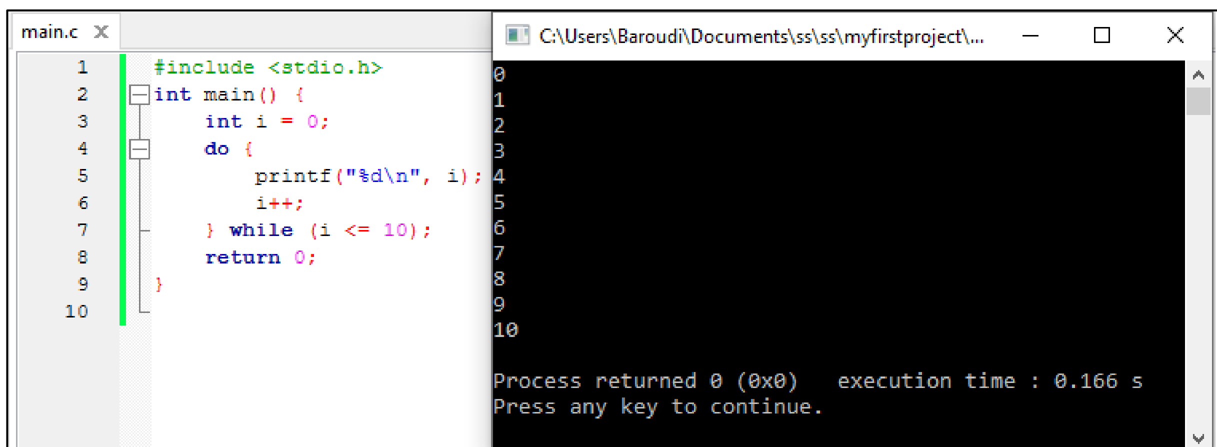


Figure 29 Do-While Loop: Example 1 (ascending order).

```c
#include <stdio.h>
int main() {
    int i = 10;
    do {
        printf("%d\n", i);
        i--;
    } while (i >= 0);
    return 0;
}
```

```
10
9
8
7
6
5
4
3
2
1
0

Process returned 0 (0x0)   execution time : 0.161 s
Press any key to continue.
```

Figure 30 Do-While Loop: Example 1 (descending order).

7.5.Practical Exercise 1

Write a program that calculates and displays the average of n numbers entered from the keyboard.

7.6.Practical Exercise Solution

In this program, we need to read n numbers, calculate their sum and their average. The reading operation must be repeated n times. After each input, the program must add the value of the entered number to the sum of the numbers read previously. Therefore, the process of reading a number and adding it to the sum must be repeated n times.

To solve this problem, we need to declare the following variables:

- n of type int, which represents the number of numbers to read.
- nb of type float, which represents the numbers to read.
- S of type float, which represents the sum of the numbers read.
- M of type float, which represents the average of the numbers read.
- i of type int, which represents the loop counter.

a. With For Loop

```c
#include <stdio.h>
int main() {
    int n, i;
    float nb, S, M;
    printf("Enter the number of values:\n ");
    scanf("%d", &n);
    S=0.0;
    for (i = 1; i <= n; i++) {
        printf("Enter number %d: \n", i);
        scanf("%f", &nb);
        S =S+ nb;
    }
    M = S / n;
    printf("Sum = %.2f\n", S);
    printf("Average = %.2f\n", M);
    return 0;
}
```

```
Enter the number of values:
 4
Enter number 1:
5
Enter number 2:
-8.7
Enter number 3:
19
Enter number 4:
25.9
Sum = 41.20
Average = 10.30

Process returned 0 (0x0)   execution time : 21.322 s
Press any key to continue.
```

Figure 31 Average of n numbers with For Loop

b. With While Loop

main.c

```c
#include <stdio.h>
int main() {
    int n, i;
    float nb, S, M;
    printf("Enter the number of values:\n ");
    scanf("%d", &n);
    S=0.0;
    i=1;
    while (i<=n) {
        printf("Enter number %d: \n", i);
        scanf("%f", &nb);
        S =S+ nb;
        i++;
    }
    M = S / n;
    printf("Sum = %.2f\n", S);
    printf("Average = %.2f\n", M);
    return 0;
}
```

C:\Users\Baroudi\Documents\ss\ss\myfirstproject\bin...

```
Enter the number of values:
 4
Enter number 1:
5
Enter number 2:
-8.7
Enter number 3:
19
Enter number 4:
25.9
Sum = 41.20
Average = 10.30

Process returned 0 (0x0)   execution time : 18.186 s
Press any key to continue.
```

Figure 32 Average of n numbers with While Loop

c. With Do-While Loop

main.c

```c
#include <stdio.h>
int main() {
    int n, i;
    float nb, S, M;
    printf("Enter the number of values:\n ");
    scanf("%d", &n);
    S=0.0;
    i=1;
    if (n > 0) {
        do {
            printf("Enter number %d: \n", i);
            scanf("%f", &nb);
            S =S+ nb;
            i++;
        } while (i <= n);
    }
    M = S / n;
    printf("Sum = %.2f\n", S);
    printf("Average = %.2f\n", M);
    return 0;
}
```

C:\Users\Baroudi\Documents\ss\ss\myfirstproject\bin\Debu...

```
Enter the number of values:
 4
Enter number 1:
5
Enter number 2:
-8.7
Enter number 3:
19
Enter number 4:
25.9
Sum = 41.20
Average = 10.30

Process returned 0 (0x0)   execution time : 12.700 s
Press any key to continue.
```

Figure 33 Average of n numbers with Do-While Loop

7.7.Practical Exercise 2

Write a program that reads a sequence of numbers. The input ends when the user enters 0. The program then displays the number of numbers read (the 0 is not counted).

7.8.Practical Exercise Solution

Unlike the previous exercises, in this exercise the number of numbers to read is not known in advance (the program must calculate it). Therefore, the number of repetitions is unknown, and it is not possible to use a for loop. We must write the program using either a while loop or a do-while loop. The loop stops when the number read is equal to 0.

To solve this problem, we need to declare the following variables:

- N of type int, which represents the number of numbers read.

- Nb of type float, which represents the numbers to read.

a. With While Loop

First, the variable N must be initialized to zero before the loop. The while loop executes as long as the number read is different from 0. Therefore, it is necessary to perform an initial read before the loop.

During each iteration of the loop, the counter N is incremented by 1 to keep track of the number of numbers read.

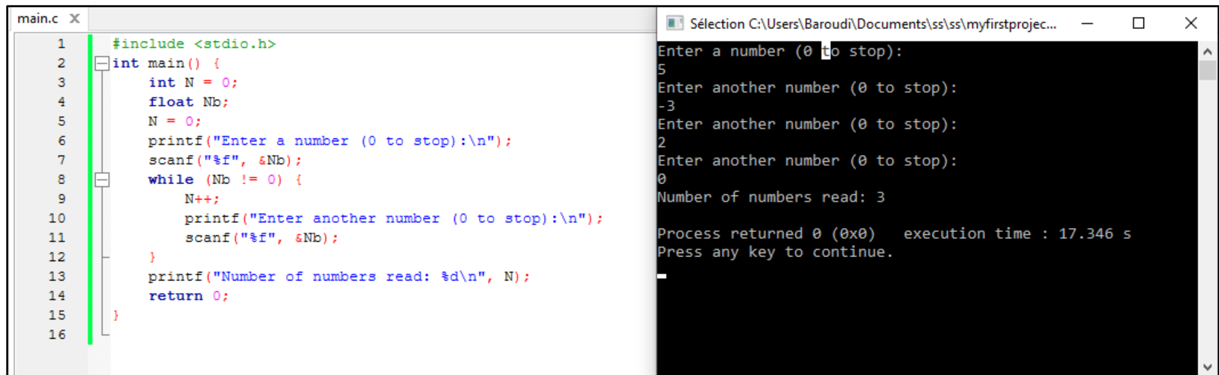


Figure 34 Number of numbers read with While Loop.

b. With Do-While Loop

In a ***do-while*** loop, the repetitions stop when the number read is equal to zero. In this type of loop, the condition is evaluated at the end of the loop. This behavior can cause a problem if the user enters 0 as the first number (this case does not cause any problem in a while loop, since the condition is checked before entering the loop).

To avoid this problem, we must add a condition before executing the loop to ensure that the first number read is different from 0. In other words, this condition ensures that the ***do-while*** loop is executed only if the first number read is not zero.

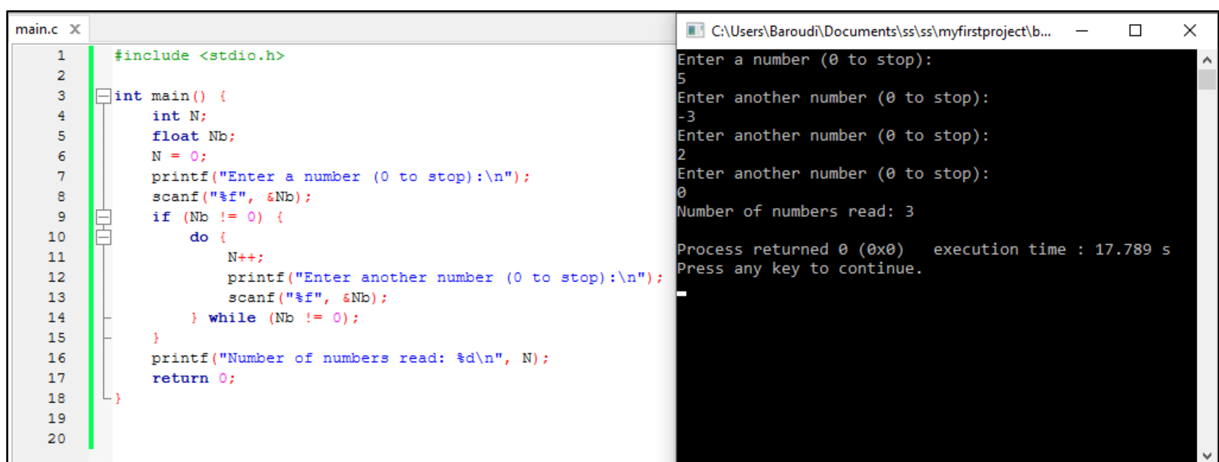


Figure 35 Number of numbers read with Do-While Loop.

7.9.Exercise

Exercise 1

Using “For” loop, write a program in C that reads a sequence of n numbers and displays the maximum among them.

Exercise 2

Using “For” loop, write a program in C that calculates and displays the following sum:

S = -1 + 1/2 - 1/3 + 1/4 - 1/5 + ⋯ + 1/n

Exercise 3

Rewrite the program of the first exercise with “while” loop.

Exercise 4

Rewrite the program of the second exercise with “Do….. while” loop.

Exercise 5

Write a program to obtain the following output using loops.

```
OO
OOO
OOOO
OOOOO
OOOOOO
OOOOOOO
OOOOOOOO
OOOOOOOOO
```

Exercise 6

Write a program to obtain the following output using loops.

```
0000000000
111111111
22222222
3333333
444444
55555
6666
777
88
9
```

Practical Work N°7:
One-Dimensional Arrays

8. PW N°7- One Dimensional Arrays (Vectors)-

8.1. Objectives

At the end of this session, the student will be able to:

- Declare a one-dimensional array.
- Fill a one-dimensional array.
- Display the contents of a one-dimensional array.

8.2. Declare a one-dimensional array

A one-dimensional array in C is a collection of elements of the same data type stored in consecutive memory locations and accessed using a single index. To declare an array in C language we use the following syntax.

Syntax :

data_type   array_name[size];

where

- **data_type** → type of elements (int, float, char, etc.)
- **array_name** → identifier you choose
- **size** → number of elements in the array (must be a positive integer)

Examples

int tab [10]: This code declares an array named **tab** of **10** *integers*.

char names [30]: This code declares an array named **names** of **30** *characters*.

In the C language, an array can be declared with initialization, and its size may be either explicitly specified or automatically inferred from the number of elements provided.

Examples

- **int a[5] = {1, 2, 3, 4, 5}:** This statement declares an array named ***a*** consisting of 5 integers and initializes its elements with the values 1 through 5.
- **int b[] = {10, 20, 30}:** This statement declares an array named ***b*** and initializes it with three values; therefore, the compiler automatically determines its size as 3.

8.3. Filling a one-Dimensional array

To fill the ***i*-th** element of an integer array **T** with a value entered by the user, the ***scanf*** statement should be used as follows:

scanf("%d", &T[i]);

To fill all the elements of an array, a loop is required. Since the array's size is known, a ***for*** loop is the most suitable for this task, although using other types of loops is also allowed.

Example

The program below declares and fills an array **T** with 5 elements of type **int**.

```
#include <stdio.h>
int main() {
    int T[5];  // Declare an array of 5 integers
    int i;
    // Fill the array with values entered by the user
    for(i = 0; i < 5; i++) {
        printf("Enter value for T[%d]: \n", i);
        scanf("%d", &T[i]);
    }
    return 0;
}
```

```
C:\Users\Baroudi\Documents\ss\ss\ee\bin\Debug\ee.exe
Enter value for T[0]:
5
Enter value for T[1]:
8
Enter value for T[2]:
20
Enter value for T[3]:
19
Enter value for T[4]:
0

Process returned 0 (0x0)   execution time : 11.753 s
Press any key to continue.
```

Figure 36 Filling a one-Dimensional array

8.4.Displaying the Contents of a One-Dimensional Array

To display the ***i*-th** element of an integer array **T**, the ***printf*** instruction can be used as follows: :

printf("%d", T[i]);

To display all the elements of an array, a loop is required. Since the array size is known, the **for** loop is the most appropriate for this task; however, other loops may also be used.

Example

The following program displays the elements of the array filled previously.

```
#include <stdio.h>

int main() {
    int T[5];  // Declare an array of 5 integers
    int i;
    // Fill the array with values entered by the user
    for(i = 0; i < 5; i++) {
        printf("Enter value for T[%d]: \n", i);
        scanf("%d", &T[i]);
    }
    // Display the array contents
    printf("The elements of the array are:\n");
    for(i = 0; i < 5; i++) {
        printf("T[%d]=%d \n",i, T[i]);
    }
    return 0;
}
```

```
C:\Users\Baroudi\Documents\ss\ss\ee\bin\Debug\ee....
Enter value for T[0]:
5
Enter value for T[1]:
8
Enter value for T[2]:
20
Enter value for T[3]:
19
Enter value for T[4]:
0
The elements of the array are:
T[0]=5
T[1]=8
T[2]=20
T[3]=19
T[4]=0

Process returned 0 (0x0)   execution time : 13.015 s
Press any key to continue.
```

Figure 37 Displaying the content of one-Dimensional array

8.5.Application Exercise 1

Write a program that reads the daily temperatures for one week and stores them in an array. The program must then calculate the average temperature for the week.

8.6.Application Exercise Solution

To solve this problem, we need to declare:

- An array **T** of size 7 and type float to store the temperature of each day of the week.
- A counter ***i*** of type integer to traverse the array.
- Two variables ***s*** and ***m*** of type float, representing respectively the sum of the temperatures and their average.

The program will calculate the sum of the temperatures progressively as they are read. The average is computed once all the temperatures have been entered.

```c
#include <stdio.h>
int main() {
    float T[7];   // array to store daily temperatures
    float s , m;
    int i;
    s=0;   //iInitialize the sum
    // Read temperatures and compute the sum
    for(i = 0; i < 7; i++) {
    //i + 1 is used so that the day numbers
    //range from 1 to 7 instead of starting at 0
        printf("Enter temperature for day %d: \n", i + 1);
        scanf("%f", &T[i]);
        s =s+ T[i];
    }
    // Calculate the average
    m = s / 7;
    // Display the average temperature
    printf("Average temperature of the week = %.2f\n", m);
    return 0;
}
```

```
Sélection C:\Users\Baroudi\Documents\ss\ss\ee\bin\Debug\ee.exe
Enter temperature for day 1:
12.5
Enter temperature for day 2:
10
Enter temperature for day 3:
7.9
Enter temperature for day 4:
8.3
Enter temperature for day 5:
12
Enter temperature for day 6:
-2.9
Enter temperature for day 7:
3.1
Average temperature of the week = 7.27

Process returned 0 (0x0)   execution time : 72.517 s
Press any key to continue.
```

Figure 38 Average Weekly Temperature Program

8.7.Application Exercise 2

Write a C program that reads the scores of 10 students and stores them in a one-dimensional array. Calculates and displays:

- The highest score
- The lowest score

8.8.Application Exercise Solution

The program first reads the scores from the user using a ***for loop*** and stores them in an array. It then initializes two variables, ***max*** and ***min***, with the first element of the array. Next, it traverses the array with a for loop, updating max if a score is greater than the current maximum and updating min if a score is smaller than the current minimum. Finally, the program displays the highest and lowest scores using ***printf***.

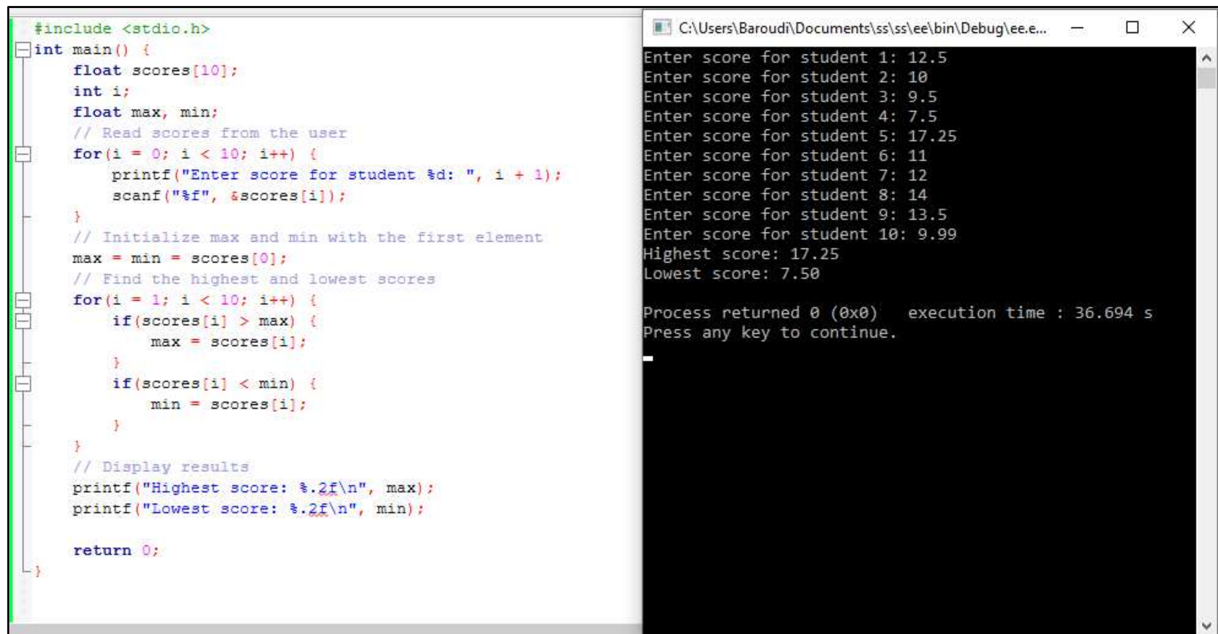


Figure 39 Highest and Lowest Scores program

8.9.Exercises

Exercise 1

Write a program that:

- Fills two integer arrays, T1 and T2, each with a capacity of 5.
- Calculates and stores the sum of the two arrays in a third array, T3.
- Displays the contents of T3.

Exercise 2

Write a program that calculates and displays the dot product of two integer arrays of size 4.

Exercise 3

Write a program that:

- Fills an integer array of size N.
- Searches for a number in this array. If the number exists, the program should display its index.

Exercise 4

Write a program that:

1. Reads 15 integers and stores them in an array.
2. Counts and displays:
   - The number of even numbers
   - The number of odd numbers

Exercise 5

Write a program that:

1. Fills an integer array of size 10.

2. Calculates and displays:
    - The number of positive elements
    - The number of negative elements
    - The number of zero elements

Practical Work N°8: Two-Dimensional Arrays (matrices)

9. PW N°8- Two-Dimensional Arrays (Matrices)-

9.1.Objectives

At the end of this session, the student will be able to:

- Declare a two-dimensional array.
- Fill a two-dimensional array.
- Display the contents of a two-dimensional array.

9.2.Declare a two-dimensional array

A two-dimensional array in C is like a table or matrix with rows and columns, where each element is accessed using two indices: one for the row and one for the column. It is used to store data in a structured way, such as a grid of numbers, characters, or other data types.

Syntax:

data_type array_name[rows][columns];

- **data_type** → type of elements (int, float, char, etc.)
- **array_name** → identifier you choose
- **rows** → number of rows in the matrix
- **columns** → number of columns in the matrix.

Example

- **int Mat [3][4]:** This code declares a matrix **Mat** with 3 rows and 4 columns of type **integer**
- **char Mat [5][6] :** This code declares a matrix **Mat** with 5 rows and 6 columns of type **caracter**.

9.3.Filling a matrix

To fill the element located in the *i*-th row and *j*-th column of an integer matrix **Mat** with a value entered by the user, the **scanf** instruction is used as follows:

scanf("%d", &Mat[i][j]);

Example

The program below declares and fills a matrix Mat with 2 rows and 3 columns of type integer. The program uses nested for loops to fill the matrix: the outer loop iterates over the rows, while the inner loop iterates over the columns. The ***scanf*** function is used inside the inner loop to read each element from the user.

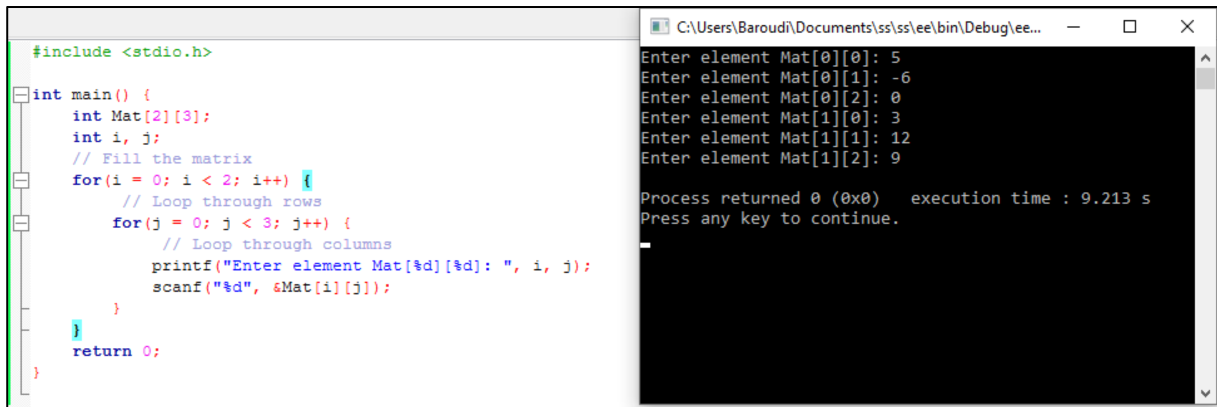


Figure 40 Filling a matrix

9.4.Displaying the content of a matrix

To display the element located in the *i*-th row and *j*-th column of an integer matrix **Mat**, the **printf** instruction is used as follows:

printf("%d", Mat[i][j]);

Example

The following program displays the elements of the matrix filled previously.

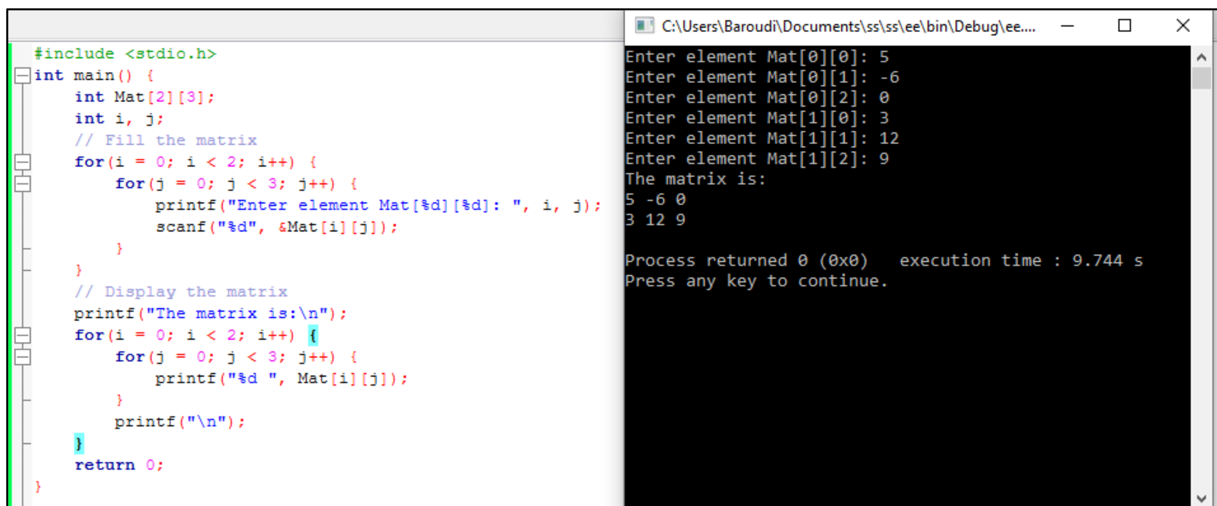


Figure 41 Displaying the content of a matrix.

9.5.Application Exercise 1

Write a program that fills a square matrix of size 3 and type integer. The program then displays the sum of the elements on the diagonal.

9.6.Application Exercise 1 solution

To solve this problem, we need to declare:

- A matrix **Mat** of 3 rows and 3 columns of type **integer**.
- Two indices (**i** and **j**) to traverse the rows and columns of the matrix.
- A variable **S** of type **integer** to store the sum of the elements on the diagonal of the matrix.

To identify the diagonal elements, we check that the row number is equal to the column number (**i = j**).

For example, in the execution window below, the diagonal elements are Mat[1][1], Mat[2][2], and Mat[3][3]. The sum is S = 1 - 4 + 10 = 7.

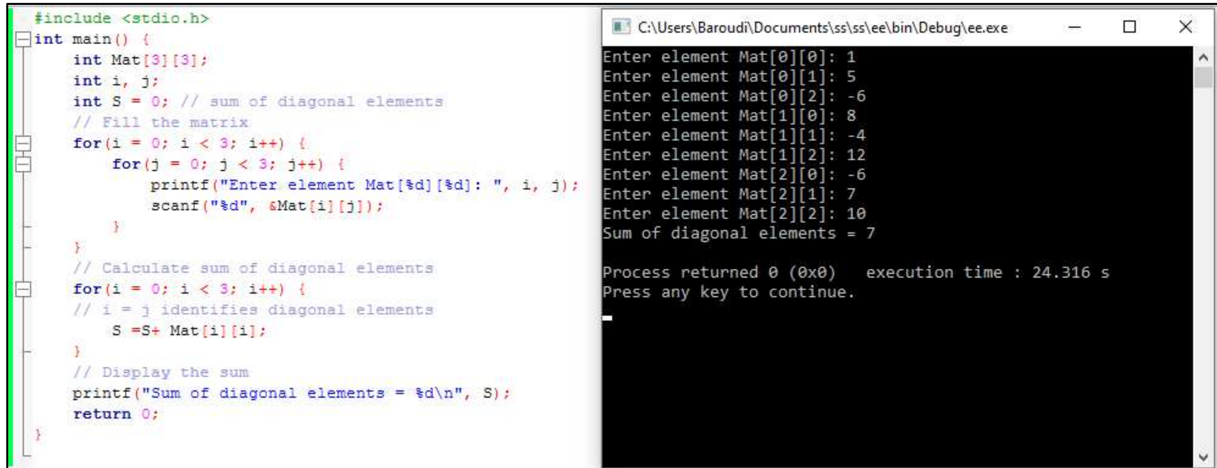


Figure 42 sum of the elements on the diagonal

9.7.Application Exercise 2

Write a program that reads the elements of a 3-row, 4-column integer matrix and then prints its transposed matrix.

9.8.Application Exercise 2 solution

The program first fills a 3×4 matrix by using nested for loops: the outer loop iterates over the rows, and the inner loop iterates over the columns, with scanf used to read each element from the user. It then computes the transpose of the matrix by swapping the row and column indices, storing the result in a new 4×3 matrix. Finally, another set of nested loops is used to display the transposed matrix, printing its elements row by row.

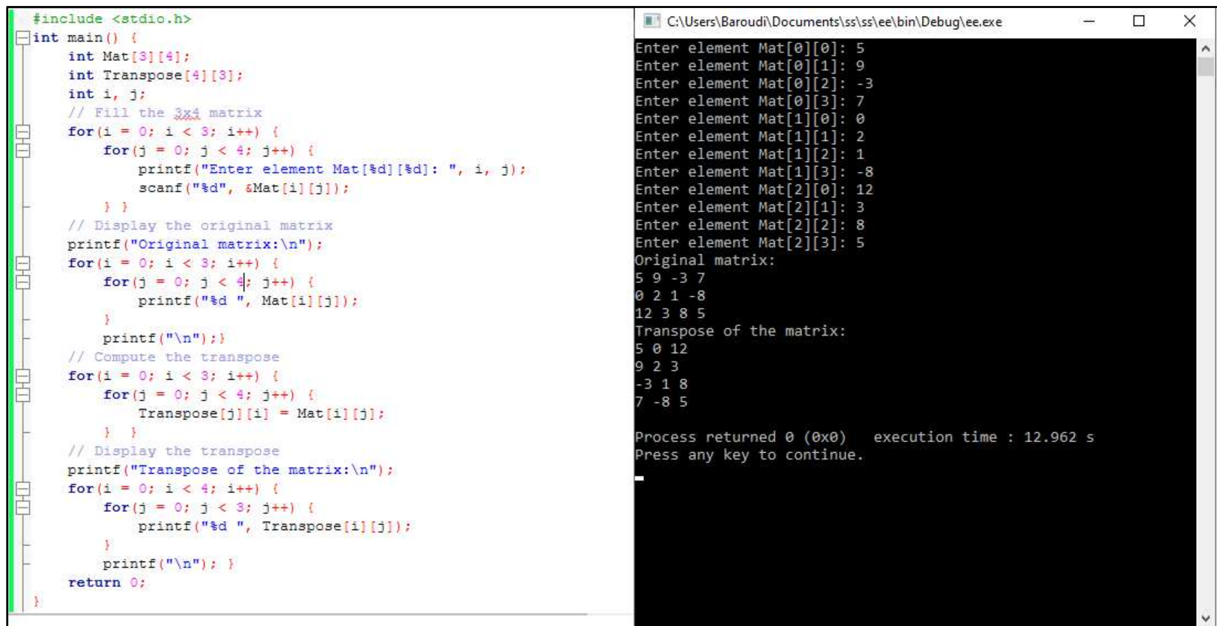


Figure 43 Transpose of a matrix

9.9.Exercises

Exercise 1

Write a program that:

- Fills two integer matrices, T1 and T2, each with 3 rows and 2 columns.
- Calculates and stores the sum of the two matrices in a matrix T3.
- Displays the contents of T3.

Exercise 2

Write a program that:

- Fills a 3×4 matrix of real numbers.
- Displays the maximum value in the matrix along with its position (row number and column number).

Exercise 3

Write a program that:

- Fills a 3×4 matrix of real numbers.
- Displays the maximum value of each row.

Exercise 4

Write a program that:

- Fills a 5×4 matrix of real numbers.
- Calculates and displays the average of all the elements in the matrix.
- Displays the number of elements greater than the average.

Exercise 5

Write a program that:

- Fills a 4×4 matrix of real numbers.
- Checks whether the matrix is symmetric.

Indication: A matrix is symmetric if it is equal to its transpose, i.e., Mat[i][j] == Mat[j][i] for all i and j.

Practical Work N°9: Structures

10. PW N°9-Structures-

10.1. Objectives

At the end of this session, the student will be able to:

- Define what a structure is.
- Declare a structure.
- Access the members of a structure.

10.2. Structures in C language

A structure is a user-defined data type that allows grouping different variables under a single name. These variables, called **members**, can have different data types such as int, float, or char.

Structures are useful when we want to represent a real-world entity composed of several attributes, for example: a student (name, age, grade), a book (title, author, price), or a point (x, y coordinates).

10.3. Declaring a structure

A structure is declared using the keyword ***struct*** followed by the structure name and its members.

Syntax

```
struct StructureName {
  dataType field1;
  dataType field2;
  dataType field3;
};
```

Example

```
struct Student {
  int id;
  char name[50];
  float average;
};
```

This structure defines a student using three attributes: **id**, **name**, and **average**. After defining the structure, variables of type **Student** must be declared in order to use it in the program.

Note

The structure must be declared before the main function.

Example

```
struct Student s1, s2;
```

This statement declares two variables s1 and s2 and both are of type struct Student. So each variable represents one student and has its own fields (for example: s1.id, s1.name, s1.average).

10.4. Accessing Structure Members

Members of a structure are accessed using the dot operator (.).

Examples

s1.id=20 ;

s1.average=10.23;

We can use ***scanf*** and ***printf*** to read and display member values. The following program illustrates how to read and display member values.

```
#include <stdio.h>
struct Student
{
    int id;
    char name[50];
    float average;
};
int main()
{
    struct Student s;
    /* Reading data */
    printf("Enter student ID: ");
    scanf("%d", &s.id);
    printf("Enter student name: ");
    scanf("%s", s.name);
    printf("Enter student average: ");
    scanf("%f", &s.average);
    /* Displaying data */
    printf("\n--- Student Information ---\n");
    printf("ID: %d\n", s.id);
    printf("Name: %s\n", s.name);
    printf("Average: %.2f\n", s.average);
    return 0;
}
```

```
C:\Users\Baroudi\Documents\ss\ss\ee\bin\Debug\ee.exe
Enter student ID: 20
Enter student name: Rabah
Enter student average: 12.3

--- Student Information ---
ID: 20
Name: Rabah
Average: 12.30

Process returned 0 (0x0)   execution time : 9.579 s
Press any key to continue.
```

Figure 44 Reading and displaying structure member values

10.5. Application Exercise

Write a program that allows you to:

1. Declare a variable of type struct, named prod1, representing a product composed of the following fields: Code of type integer, Name of type string, and Price of type float.
2. Fill the fields of the variable prod1 with values entered by the user.
3. Display the fields of the variable prod1 on the screen.

10.6. Application Exercise solution

```c
#include <stdio.h>
/* Structure declaration */
struct Product
{
    int code;
    char name[50];
    float price;
};
int main()
{
    struct Product prod1;
    /* Reading data */
    printf("Enter product code: ");
    scanf("%d", &prod1.code);
    printf("Enter product name: ");
    scanf("%s", prod1.name);
    printf("Enter product price: ");
    scanf("%f", &prod1.price);
    /* Displaying data */
    printf("\nProduct information:\n");
    printf("Code: %d\n", prod1.code);
    printf("Name: %s\n", prod1.name);
    printf("Price: %.2f\n", prod1.price);
    return 0;
}
```

Figure 45 Product Structure

10.7. A structure as a member of another structure

In C, it is possible for a structure to contain another structure as one of its fields. This allows you to model complex real-world entities that have nested attributes.

Example

Suppose we have a structure Student with the fields id, name, and average. We want to add an address field to the student, which itself is composed of multiple attributes: house number, street, and province. Instead of adding separate fields to Student, we can define a structure Address and include it as a member of Student.

```c
#include <stdio.h>
/* Inner structure for address */
struct Address {
  int num;
  char street[50];
  char province[30];
};
/* Outer structure for student */
```

struct Student {

int id;

char name[50];

float average;

struct Address addr;

};

The following program defines the Student structure with an Address structure as a member, and then reads and displays all of its fields.

```
#include <stdio.h>
struct Address {/* Structure for Address */
    int num;
    char street[50];
    char province[30];
};
struct Student {/* Structure for Student */
    int id;
    char name[50];
    float average;
    struct Address addr;  // Address as a member
};
int main() {
    struct Student s;
    printf("Enter student ID: ");
    scanf("%d", &s.id);
    printf("Enter student name: ");
    scanf("%s", s.name);
    printf("Enter student average: ");
    scanf("%f", &s.average);
    printf("Enter house number: ");
    scanf("%d", &s.addr.num);
    printf("Enter street: ");
    scanf("%s", s.addr.street);
    printf("Enter province: ");
    scanf("%s", s.addr.province);
    printf("\n--- Student Information ---\n");
    printf("ID: %d\n", s.id);
    printf("Name: %s\n", s.name);
    printf("Average: %.2f\n", s.average);
    printf("Address: %d, %s, %s\n", s.addr.num, s.addr.street, s.addr.province);
    return 0;}
```

Figure 46 A structure as a member of another structure

10.8. Exercises

Exercise 1

Write a program that:

1. Declares a struct Date with fields: day (int), month (int), year (int).
2. Declares a struct Person with fields: name (string), birthDate of type Date.
3. Reads and displays information for two people, including their dates of birth.

Exercise 2

Write a program that:

1. Declares a struct Complex with fields: real (float), imag (float).
2. Reads two complex numbers from the user.
3. Calculates and displays their sum

Exercise 3

Write a program that:

1. Declares a struct Book with fields: title (string), author (string), price (float).
2. Reads data for 4 books.
3. Displays all books that cost more than 20.0.

Exercise 4

Write a program that:

1. Declares a struct Employee with fields: id (int), name (string), salary (float).
2. Reads data for 3 employees.
3. Calculates and displays the average salary.
4. Displays the employee(s) with the highest salary.

Practical Work N°10: String manipulation

11. PW N°10-String manipulation-

11.1. Objectives

At the end of this session, the student will be able to:

- Declare strings
- Read and display strings
- Use the standard string functions

11.2. String Declaration

In C language, a string is an array of characters (char) terminated by a special character called the null character '\0'. This character marks the end of the string.

<u>Examples:</u>

char word[] = "Hello";

In memory, the string is represented as folllows: H e l l o \0

char name[20]; declares a string that can store up to 19 characters + the null character.

The following example declares 2 strings with initialization

char city[] = "Oran";

char country[10] = "Algeria";

A two-dimensional character array can be used to declare an array of 3 strings, where each string can store up to 19 characters:

char words[3][20] ;

11.3. Reading and Displaying Strings

a) Using scanf

```
char name[20];
scanf("%s", name);
```

This statement reads only a single word (no spaces).

b) Using fgets

```
fgets(name, 20, stdin);
```

This statement reads a full sentence including spaces.

c) Displaying a String

```
printf("%s", name);
```

This statement displays the content of the variable name.

11.4. Manual String Manipulation

Since a string is essentially a one-dimensional array, array operations can be applied to it. For instance, to traverse a string, we can simply use a loop to iterate through the array elements. The following code displays the content of a string character by character.

```
int i = 0;
while(name[i] != '\0')
{
  printf("%c\n", name[i]);
  i++;
}
```

We can also use a loop to determine manually the length of a string by the following code:

```
int len = 0;
while(name[len] != '\0')
{
  len++;
}
```

11.5. Standard String Functions

To use standard string functions, the <string.h> library must be included. The following table summarizes these functions.

Function	Purpose	Syntax	Explanation
strlen()	Find the length of a string	strlen(str);	Returns the number of characters in a string excluding the null character '\0'.
strcpy()	Copy one string into another	strcpy(destination, source);	Copies the content of source into destination. The destination array must be large enough to hold the copied string.
strcat()	Concatenate two strings	strcat(a, b);	Appends string b to the end of string a. The array a must have enough space to store the resulting string.
strcmp()	Compare two strings	strcmp(a, b);	Compares two strings lexicographically. Returns **0** if equal, **<0** if a is less than b, and **>0** if a is greater than b.

Table 9 Standard string functions

11.6. Illustrative examples

a. strlen ()

```
#include <stdio.h>
#include <string.h>

int main()
{
    char str[20];
    printf("Enter a string:\n");
    scanf("%s", str);

    printf("The Length of the string= %d", strlen(str));
    return 0;
}
```

```
C:\Users\Baroudi\Documents\ss\ss\ee\bin\Debug\ee.exe
Enter a string:
Programming
The Length of the string= 11
Process returned 0 (0x0)   execution time : 12.980 s
Press any key to continue.
```

Figure 47 strlen() function

b. strcat()

```
#include <stdio.h>
#include <string.h>

int main()
{
    char a[50] = "Hello ";
    char b[] = "Students";
    strcat(a, b);
    printf("%s", a);
    return 0;
}
```

```
C:\Users\Baroudi\Documents\ss\ss\ee\bin\Debug\e...
Hello Students
Process returned 0 (0x0)   execution time : 0.183 s
Press any key to continue.
```

Figure 48 strcat() function

c. strcmp()

```
#include <stdio.h>
#include <string.h>

int main()
{
    char a[20], b[20];
    printf("Enter 2 strings:\n");
    scanf("%s %s", a, b);
    if(strcmp(a,b)==0)
        printf("Equal");
    else
        printf("Different");
    return 0;
}
```

```
Sélection C:\Users\Baroudi\Documents\ss\ss\ee\bi...
Enter 2 strings:
Hello
World
Different
Process returned 0 (0x0)   execution time : 9.880 s
Press any key to continue.
```

```
C:\Users\Baroudi\Documents\ss\ss\ee\bin\Debug\...
Enter 2 strings:
Hello
Hello
Equal
Process returned 0 (0x0)   execution time : 9.441 s
Press any key to continue.
```

Figure 49 strcmp()

d. strcpy()

```c
#include <stdio.h>
#include <string.h>

int main() {
    char source[] = "Programming in C";
    char destination[30];
    strcpy(destination, source);
    // copy source into destination
    printf("Source string: %s\n", source);
    printf("Destination string: %s\n", destination);
    return 0;
}
```

```
C:\Users\Baroudi\Documents\ss\ss\ee\bin\Debug\ee.e...
Source string: Programming in C
Destination string: Programming in C

Process returned 0 (0x0)   execution time : 0.186 s
Press any key to continue.
```

Figure 50 strcpy()

11.7. Exercises

Exercise 1

Write a program that reads a string and displays the reversed string.

Example:

Input → hello

Output → olleh

Exercise 2

Write a program that reads a string and counts:

- number of vowels
- number of consonants
- number of digits

Exercise 3

Write a program that reads 5 words and identifies the longest and the shortest word.

Exercise 4

Write a program that reads five words and sorts them in alphabetical order.

Solution of the Exercises

12. Solution of the Exercises

12.1. PW N°4-Basic instructions-

Exercise 1

```
#include <stdio.h>
int main() {
   float a, b, c, avg;
   printf("Enter three numbers:\n");
   scanf("%f %f %f", &a, &b, &c);
   avg = (a + b + c) / 3;
   printf("Average = %.2f\n", avg);
   return 0;
}
```

Exercise 2

```
#include <stdio.h>
int main() {
   float a, b, c, delta;
   printf("Enter a, b and c:\n");
   scanf("%f %f %f", &a, &b, &c);
   delta = b*b - 4*a*c;
   printf("Discriminant = %.2f\n", delta);
   return 0;
}
```

Exercise 3

```
#include <stdio.h>
int main() {
   const float PI 3.14159;
   float R, H, V, S;
   printf("Enter radius and height:\n");
   scanf("%f %f", &R, &H);
   V = PI * R * R * H;
   S = 2 * PI * R * H;
   printf("Volume = %.2f\n", V);
   printf("Surface = %.2f\n", S);
   return 0;
}
```

Exercise 4

```
#include <stdio.h>
int main() {
   float n1, n2, n3, c1, c2, c3, avg;
   printf("Enter 3 grades:\n");
   scanf("%f %f %f", &n1, &n2, &n3);
   printf("Enter their coefficients:\n");
   scanf("%f %f %f", &c1, &c2, &c3);
   avg = (n1*c1 + n2*c2 + n3*c3) / (c1 + c2 + c3);
   printf("Average = %.2f\n", avg);
   return 0;
}
```

Exercise 5

```
#include <stdio.h>
int main() {
   int Q;
   float P, total;
   printf("Enter quantity and unit price:\n");
   scanf("%d %f", &Q, &P);
   total = Q * P;
   printf("Total amount = %.2f\n", total);
   return 0;
}
```

Exercise 6

```
#include <stdio.h>
#include <math.h>
int main() {
   float x1, y1, x2, y2, d;
   printf("Enter coordinates of P1:\n");
   scanf("%f %f", &x1, &y1);
   printf("Enter coordinates of P2:\n");
   scanf("%f %f", &x2, &y2);
   d = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
   printf("Distance = %.2f\n", d);
   return 0;
}
```

Note :

#include <math.h> has been added to use sqrt function.

Exercise 7

```
#include <stdio.h>
int main() {
   float a, b, temp;
   printf("Enter two numbers:\n");
   scanf("%f %f", &a, &b);
   temp = a;
   a = b;
   b = temp;
   printf("After swap: a = %.2f, b = %.2f\n", a, b);
   return 0;
}
```

12.2. PW N° 5-Conditions-

Exercise 1

a) Using the alternative structure (if–else)

```
#include <stdio.h>
int main() {
   float a, b;
   printf("Enter two numbers:\n");
   scanf("%f %f", &a, &b);
   if (a > b)
      printf("The maximum is: %.2f\n", a);
   else
      printf("The maximum is: %.2f\n", b);
   return 0;
}
```

b) Using the conditional structure

```
#include <stdio.h>
int main() {
   float a, b, max;
   printf("Enter two numbers:\n");
   scanf("%f %f", &a, &b);
   max = a;
   if (b>max) {
    max=b;
    }
   printf("The maximum is: %.2f\n", max);
   return 0;
}
```

Exercise 2

```
#include <stdio.h>
int main() {
   float temp;
   printf("Enter the water temperature:\n");
   scanf("%f", &temp);
   if (temp <= 0)
```

```
    printf("Ice\n");
  else if (temp < 100)
    printf("Liquid\n");
  else
    printf("Vapor\n");
  return 0;
}
```

Exercise 3

```
#include <stdio.h>
int main() {
  float P, T, BMI;
  printf("Enter weight (kg):\n");
  scanf("%f", &P);
  printf("Enter height (m):\n");
  scanf("%f", &T);
  BMI = P / (T * T);
  printf("BMI = %.2f\n", BMI);
  if (BMI < 16)
    printf("Underweight\n");
  else if (BMI < 18)
    printf("Thin\n");
  else if (BMI <= 25)
    printf("Normal weight\n");
  else if (BMI < 30)
    printf("Overweight\n");
  else
    printf("Obesity\n");
  return 0;
}
```

Exercise 4

```
#include <stdio.h>
int main() {
  float amount, discountRate = 0, discount, netAmount;
  int die;
```

```
    printf("Enter total purchase amount:\n");
    scanf("%f", &amount);
    printf("Enter die result (1 to 6):\n");
    scanf("%d", &die);
    switch (die) {
        case 1: discountRate = 0.05; break;
        case 2: discountRate = 0.07; break;
        case 3: discountRate = 0.08; break;
        case 4: discountRate = 0.15; break;
        case 5: discountRate = 0.20; break;
        case 6: discountRate = 0.50; break;
        default:
            printf("Invalid die result\n");
            return 0;
    }
    discount = amount * discountRate;
    netAmount = amount - discount;
    printf("Discount rate: %.0f%%\n", discountRate * 100);
    printf("Net amount to pay: %.2f\n", netAmount);
    return 0;
}
```

Note:

The format specifier %.0f is used to display the value of discountRate without any decimal places, while %% is used to print the percent sign (%) in the output.

12.3. PW N°6- Repetitive structures (Loops)-

Exercise 1

```
#include <stdio.h>
int main() {
  int n, i;
  float x, max;
  printf("Enter the number of values:\n ");
  scanf("%d", &n);
  printf("Enter value 1: \n");
  scanf("%f", &max);
  for (i = 2; i <= n; i++) {
    printf("Enter value %d: \n", i);
    scanf("%f", &x);
    if (x > max)
      max = x;
  }
  printf("Maximum value = %.2f\n", max);
  return 0;
}
```

Exercise 2

```
#include <stdio.h>
int main() {
  int n, i;
  float S = 0;
  printf("Enter n:\n ");
  scanf("%d", &n);
  for (i = 1; i <= n; i++) {
    if (i % 2 == 0)
      S =S+ 1.0 / i;
    else
      S =S- 1.0 / i;
  }
  printf("Sum S = %.4f\n", S);
  return 0;
}
```

Exercise 3

```
#include <stdio.h>
int main() {
   int n, i = 1;
   float x, max;
   printf("Enter the number of values: \n");
   scanf("%d", &n);
   printf("Enter value 1: \n");
   scanf("%f", &max);
   while (i < n) {
      printf("Enter next value:\n ");
      scanf("%f", &x);
      if (x > max)
         max = x;
      i++;
   }
   printf("Maximum value = %.2f\n", max);
   return 0;
}
```

Exercise 4

```
#include <stdio.h>
int main() {
   int n, i = 1;
   float S = 0;
   printf("Enter n: \n");
   scanf("%d", &n);
   do {
      if (i % 2 == 0)
         S =S+ 1.0 / i;
      else
         S =S- 1.0 / i;
      i++;
   } while (i <= n);
   printf("Sum S = %.4f\n", S);
```

```
   return 0;
}
```

Exercise 5

```
#include <stdio.h>
int main() {
   int i, j;
   for (i = 2; i <= 9; i++) {
      for (j = 1; j <= i; j++) {
         printf("O");
      }
      printf("\n");
   }
   return 0;
}
```

Exercise 6

```
#include <stdio.h>
int main() {
   int i, j;
   for (i = 0; i <= 9; i++) {
      for (j = 1; j <= 10 - i; j++) {
         printf("%d", i);
      }
      printf("\n");
   }
   return 0;
}
```

12.4. PW N°7- One dimensional arrays

Exercise 1

```
#include <stdio.h>
int main() {
  int T1[5], T2[5], T3[5];
  int i;
  // Read arrays T1 and T2
  for(i = 0; i < 5; i++) {
    printf("Enter T1[%d]: ", i);
    scanf("%d", &T1[i]);
  }
  for(i = 0; i < 5; i++) {
    printf("Enter T2[%d]: ", i);
    scanf("%d", &T2[i]);
  }
  // Calculate sum of arrays
  for(i = 0; i < 5; i++) {
    T3[i] = T1[i] + T2[i];
  }
  // Display T3
  printf("T3 (sum of T1 and T2) = ");
  for(i = 0; i < 5; i++) {
    printf("%d ", T3[i]);
  }
  printf("\n");
  return 0;
}
```

Exercise 2

```
#include <stdio.h>
int main() {
  int a[4], b[4];
  int c = 0; // This will store the dot product
  int i;
  // Read elements of vector a
  for(i = 0; i < 4; i++) {
```

```
    printf("Enter element %d of vector a: ", i + 1);
    scanf("%d", &a[i]);
  }
  // Read elements of vector b
  for(i = 0; i < 4; i++) {
    printf("Enter element %d of vector b: ", i + 1);
    scanf("%d", &b[i]);
  }
  // Calculate the dot product
  for(i = 0; i < 4; i++) {
    c =c+ a[i] * b[i];
  }
  // Display the result
  printf("Dot product of vectors a and b = %d\n", c);
  return 0;
}
```

Exercise 3

```
#include <stdio.h>
#include <stdbool.h>
int main() {
  int N, i, num;
  bool found = false;  // boolean variable to track if number exists
  printf("Enter the size of the array: ");
  scanf("%d", &N);
  int arr[N];
  // Read array elements
  for(i = 0; i < N; i++) {
    printf("Enter element %d: ", i + 1);
    scanf("%d", &arr[i]);
  }
  // Read the number to search
  printf("Enter number to search: ");
  scanf("%d", &num);
  // Search in the array
  for(i = 0; i < N; i++) {
```

```
    if(arr[i] == num) {
      printf("Number found at index %d\n", i);
      found = true;  // mark as found
      break;        // stop searching after first occurrence
    }
  }
  if(!found) {
    printf("Number not found in the array.\n");
  }
  return 0;
}
```

Exercise 4

```
#include <stdio.h>
int main() {
  int arr[15];
  int i, even = 0, odd = 0;
  // Read 15 integers
  for(i = 0; i < 15; i++) {
    printf("Enter number %d: ", i + 1);
    scanf("%d", &arr[i]);
  }
  // Count even and odd numbers
  for(i = 0; i < 15; i++) {
    if(arr[i] % 2 == 0) {
      even++;
    } else {
      odd++;
    }
  }
  // Display results
  printf("Number of even numbers: %d\n", even);
  printf("Number of odd numbers: %d\n", odd);
  return 0;
}
```

Exercise 5

```
#include <stdio.h>
int main() {
  int arr[10];
  int i;
  int positive = 0, negative = 0, zero = 0;
  // Read 10 integers
  for(i = 0; i < 10; i++) {
    printf("Enter element %d: ", i + 1);
    scanf("%d", &arr[i]);
  }
  // Count positive, negative, and zero elements
  for(i = 0; i < 10; i++) {
    if(arr[i] > 0) {
      positive++;
    } else if(arr[i] < 0) {
      negative++;
    } else {
      zero++;
    }
  }
  // Display results
  printf("Positive elements: %d\n", positive);
  printf("Negative elements: %d\n", negative);
  printf("Zero elements: %d\n", zero);
  return 0;
}
```

12.5. PW N° 8-Two-dimensional arrays (matrices)

Exercise 1

```
#include <stdio.h>
int main() {
  int T1[3][2], T2[3][2], T3[3][2];
  int i, j;
  // Fill T1
  for(i = 0; i < 3; i++) {
    for(j = 0; j < 2; j++) {
      printf("Enter T1[%d][%d]: ", i, j);
      scanf("%d", &T1[i][j]);
    }
  }
  // Fill T2
  for(i = 0; i < 3; i++) {
    for(j = 0; j < 2; j++) {
      printf("Enter T2[%d][%d]: ", i, j);
      scanf("%d", &T2[i][j]);
    }
  }
  // Calculate T3 = T1 + T2
  for(i = 0; i < 3; i++) {
    for(j = 0; j < 2; j++) {
      T3[i][j] = T1[i][j] + T2[i][j];
    }
  }
  // Display T3
  printf("Sum matrix T3:\n");
  for(i = 0; i < 3; i++) {
    for(j = 0; j < 2; j++) {
      printf("%d ", T3[i][j]);
    }
    printf("\n");
  }
  return 0;
}
```

Exercise 2

```
#include <stdio.h>
int main() {
   float Mat[3][4];
   float max;
   int i, j, maxRow = 0, maxCol = 0;
   // Fill the matrix
   for(i = 0; i < 3; i++) {
      for(j = 0; j < 4; j++) {
         printf("Enter element Mat[%d][%d]: ", i, j);
         scanf("%f", &Mat[i][j]);
      }
   }
   // Initialize max
   max = Mat[0][0];
   // Find maximum and its position
   for(i = 0; i < 3; i++) {
      for(j = 0; j < 4; j++) {
         if(Mat[i][j] > max) {
            max = Mat[i][j];
            maxRow = i;
            maxCol = j;
         }
      }
   }
   printf("Maximum value = %.2f at position (%d, %d)\n", max, maxRow,
maxCol);
   return 0;
}
```

Exercise 3

```
#include <stdio.h>

int main() {
   float Mat[3][4];
   float maxRow[3];  // array to store maximum of each row
   int i, j;
```

```
  // Fill the matrix
  for(i = 0; i < 3; i++) {
    for(j = 0; j < 4; j++) {
      printf("Enter element Mat[%d][%d]: ", i, j);
      scanf("%f", &Mat[i][j]);
    }
  }
  // Find maximum of each row and store in maxRow
  for(i = 0; i < 3; i++) {
    maxRow[i] = Mat[i][0]; // initialize with first element of the row
    for(j = 1; j < 4; j++) {
      if(Mat[i][j] > maxRow[i]) {
        maxRow[i] = Mat[i][j];
      }
    }
  }
  // Display maximums of each row
  for(i = 0; i < 3; i++) {
    printf("Maximum of row %d = %.2f\n", i, maxRow[i]);
  }
  return 0;
}
```

Exercise 4

```
#include <stdio.h>
int main() {
  float Mat[5][4];
  int i, j;
  float sum = 0, avg;
  int count = 0;
  // Fill the matrix
  for(i = 0; i < 5; i++) {
    for(j = 0; j < 4; j++) {
      printf("Enter element Mat[%d][%d]: ", i, j);
      scanf("%f", &Mat[i][j]);
```

```
        sum =sum+ Mat[i][j]; // calculate sum while reading
      }
    }
    // Calculate average
    avg = sum / (5 * 4);
    printf("Average of elements = %.2f\n", avg);
    // Count elements greater than average
    for(i = 0; i < 5; i++) {
      for(j = 0; j < 4; j++) {
        if(Mat[i][j] > avg) {
          count++;
        }
      }
    }
    printf("Number of elements greater than average = %d\n", count);
    return 0;
}
```

Exercise 5

```
#include <stdio.h>
#include <stdbool.h>
int main() {
    float Mat[4][4];
    int i, j;
    bool symmetric = true;
    // Fill the matrix
    for(i = 0; i < 4; i++) {
      for(j = 0; j < 4; j++) {
        printf("Enter element Mat[%d][%d]: ", i, j);
        scanf("%f", &Mat[i][j]);
      }
    }
    // Check symmetry
    for(i = 0; i < 4; i++) {
      for(j = 0; j < 4; j++) {
        if(Mat[i][j] != Mat[j][i]) {
```

```
            symmetric = false;
            break;
          }
        }
        if(!symmetric) break;
      }
      if(symmetric) {
        printf("The matrix is symmetric.\n");
      } else {
        printf("The matrix is not symmetric.\n");
      }
      return 0;
    }
```

12.6. PW N°10- Structures

Exercise 1

```
#include <stdio.h>
struct Date {
  int day;
  int month;
  int year;
};
struct Person {
  char name[50];
  struct Date birthDate; // Nested structure
};
int main() {
  struct Person p[2];
  int i;
  // Read person data
  for(i = 0; i < 2; i++) {
    printf("Enter name of person %d: ", i+1);
    scanf("%s", p[i].name);
    printf("Enter birth date (day month year): ");
    scanf("%d %d %d", &p[i].birthDate.day, &p[i].birthDate.month,
&p[i].birthDate.year);
  }
  // Display person data
  printf("\nPersons information:\n");
  for(i = 0; i < 2; i++) {
    printf("Name: %s, Birth Date: %02d/%02d/%04d\n", p[i].name,
        p[i].birthDate.day, p[i].birthDate.month, p[i].birthDate.year);
  }
  return 0;
}
```

Exercise 2

```
#include <stdio.h>
struct Complex {
   float real;
   float imag;
};
int main() {
   struct Complex c1, c2, sum;
   // Read first complex number
   printf("Enter real and imaginary part of first complex number: ");
   scanf("%f %f", &c1.real, &c1.imag);
   // Read second complex number
   printf("Enter real and imaginary part of second complex number: ");
   scanf("%f %f", &c2.real, &c2.imag);
   // Calculate sum
   sum.real = c1.real + c2.real;
   sum.imag = c1.imag + c2.imag;
   // Display sum
   printf("Sum = %.2f + %.2fi\n", sum.real, sum.imag);
   return 0;
}
```

Exercise 3

```
#include <stdio.h>
struct Book {
   char title[50];
   char author[50];
   float price;
};
int main() {
   struct Book books[4];
   int i;
   // Read book data
   for(i = 0; i < 4; i++) {
      printf("Enter title of book %d: ", i+1);
      scanf("%s", books[i].title);
```

```
        printf("Enter author: ");
        scanf("%s", books[i].author);
        printf("Enter price: ");
        scanf("%f", &books[i].price);
    }
    // Display books costing more than 20.0
    printf("\nBooks costing more than 20.0:\n");
    for(i = 0; i < 4; i++) {
        if(books[i].price > 20.0) {
            printf("Title: %s, Author: %s, Price: %.2f\n", books[i].title, books[i].author,
books[i].price);
        }
    }
    return 0;
}
```

Exercise 4

```
#include <stdio.h>
struct Employee {
    int id;
    char name[50];
    float salary;
};
int main() {
    struct Employee emp[3];
    int i;
    float total = 0, maxSalary;
    int maxIndex = 0;
    // Read employee data
    for(i = 0; i < 3; i++) {
        printf("Enter ID of employee %d: ", i+1);
        scanf("%d", &emp[i].id);
        printf("Enter name: ");
        scanf("%s", emp[i].name);
        printf("Enter salary: ");
```

```
        scanf("%f", &emp[i].salary);

        total += emp[i].salary;
    }
    // Calculate average salary
    printf("\nAverage salary = %.2f\n", total/3);
    // Find employee with highest salary
    maxSalary = emp[0].salary;
    for(i = 1; i < 3; i++) {
        if(emp[i].salary > maxSalary) {
            maxSalary = emp[i].salary;
            maxIndex = i;
        }
    }
    printf("Employee with highest salary:\n");
    printf("ID: %d, Name: %s, Salary: %.2f\n", emp[maxIndex].id,
emp[maxIndex].name, emp[maxIndex].salary);
    return 0;
}
```

12.7. PW N°10-String manipulation-

Exercise 1

```
#include <stdio.h>
#include <string.h>
int main() {
  char str[100];
  int i, len;
  printf("Enter a string: ");
  scanf("%s", str);
  len = strlen(str);
  printf("Reversed string: ");
  for (i = len - 1; i >= 0; i--) {
    printf("%c", str[i]);
  }
  return 0;
}
```

Exercise 2

```
#include <stdio.h>
#include <ctype.h>
int main() {
  char str[100];
  int i = 0, vowels = 0, consonants = 0, digits = 0;
  printf("Enter a string: ");
  fgets(str, sizeof(str), stdin);
  while (str[i] != '\0') {
    if (isdigit(str[i]))
      digits++;
    else if (isalpha(str[i])) {
      char c = tolower(str[i]);
      if (c=='a'||c=='e'||c=='i'||c=='o'||c=='u')
        vowels++;
      else
        consonants++;
    }
```

```
    i++;
  }
  printf("Vowels: %d\n", vowels);
  printf("Consonants: %d\n", consonants);
  printf("Digits: %d\n", digits);
  return 0;
}
```

Notes :

- *isdigit(): is a function that tests whether a character is a numeric digit (0–9).*
- *isalpha(): is a function that tests whether a character is an alphabetic letter (A–Z or a–z).*
- *tolower(): is a function that converts an uppercase letter to lowercase.*
- *sizeof() is function that determines the size of the array.*
- *All these functions are included in <ctype.h> library.*

Exercise 3

```
#include <stdio.h>
#include <string.h>
int main() {
 char words[5][50];
 int i, longest = 0, shortest = 0;
 for (i = 0; i < 5; i++) {
    printf("Enter word %d: ", i + 1);
    scanf("%s", words[i]);
  }
 for (i = 1; i < 5; i++) {
    if (strlen(words[i]) > strlen(words[longest]))
       longest = i;
    if (strlen(words[i]) < strlen(words[shortest]))
       shortest = i;
 }
 printf("Longest word: %s\n", words[longest]);
 printf("Shortest word: %s\n", words[shortest]);
 return 0;
}
```

Exercise 4

```
#include <stdio.h>
#include <string.h>
int main() {
  char words[5][50], temp[50];
  int i, j;
  for (i = 0; i < 5; i++) {
    printf("Enter word %d: ", i + 1);
    scanf("%s", words[i]);
  }
  for (i = 0; i < 4; i++) {
    for (j = i + 1; j < 5; j++) {
      if (strcmp(words[i], words[j]) > 0) {
        strcpy(temp, words[i]);
        strcpy(words[i], words[j]);
        strcpy(words[j], temp);
      }
    }
  }
  printf("\nWords in alphabetical order:\n");
  for (i = 0; i < 5; i++) {
    printf("%s\n", words[i]);
  }
  return 0;
}
```

References

1. Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language (2nd ed.)*. Prentice Hall.
2. King, K. N. (2008). *C Programming: A Modern Approach (2nd ed.).* W. W. Norton & Company.
3. Griffiths, D., & Griffiths, D. (2012). *Head First C*. O'Reilly Media.
4. Kochan, S. G. (2014). *Programming in C (4th ed.)*. Addison-Wesley.
5. Bouchiha, D. (2011). *Algorithmique et programmation en C*. Éditions Universitaires.
6. Malgouyres, R. (2005). *Initiation à l'algorithmique et à la programmation en C*.
7. Dmitrovic, S. (2017). *Modern C for Absolute Beginners: A Friendly Introduction to the C Programming Language*.

Websites / Online Tutorials

1. TutorialsPoint. C Programming Tutorial.
   Available at: https://www.tutorialspoint.com/cprogramming/index.htm
2. GeeksforGeeks. C Programming Language.
   Available at: https://www.geeksforgeeks.org/c-programming-language/
3. Programiz. Learn C Programming.
   Available at: https://www.programiz.com/c-programming
4. Learn-C.org. Interactive C Programming Tutorial.
   Available at: https://www.learn-c.org/