Practical Work No. 1
Introduction to C Programming

Goals :

- Understanding the structure of a C program
- Discuss the main functions of drop-down menus (File, edit, run, compile, etc.)
- Study the compilation, execution and testing steps
- Write and run a program that displays a string of characters on the screen.
- Detect and correct syntactic errors in a program

I. Introduction

The C language is an advanced and structured language, quite close to machine language intended for process control applications (input/output management, real-time applications, etc.).

The C language has a few instructions. However, it uses libraries, which are provided in greater or lesser numbers with the compiler.

Example: math.h : library of mathematical functions
stdio.h: Standard I/O library

II. Skeleton of a C program

The skeleton of a C program can be defined as follows:

```
Libraries Declaration
Declaration of constants
main ( ){
        Declaration of variables
        Body of the program
}
```

Remarks :

It is best to comment out difficult instructions. This helps explain their meaning to readers even if they are unfamiliar with the programming language. Comments in C can be placed anywhere in the program. They begin with /* and end with */ on the same line.

C is case-sensitive. Reserved words in C must be written in lowercase.

It is best to get into the habit of framing the body of the program in the following way:

```
#include<stdio.h> /*standard I/O library declaration */
void main ( ) {
   /*variable declaration*/
   /*program body*/
}
```

III. Identifiers and keywords

An identifier is the name given to a variable, constant, function, etc. This name is composed of unaccented letters, numbers, and the character in any order except for the first character, which cannot be a number. Some words cannot be used as identifiers because they are reserved; they are keywords of the language. These won't mean anything to you right now; all you need to know is that you should not use these words to name your variables and functions: **auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while**.

IV. Variables

A variable is a memory space that the program reserves during its execution at the time of its declaration. It is data whose value can be modified.

V. Variable Declaration

When programming, you must always specify the type of variables before using them. This is called declaration.

Syntaxes:

```
type_name variable_name;
type_name variable_name_1, variable_name_2, ..., variable_name_N;
```

Examples:

```
int A ; /*declaration of an integer variable*/
float X, Y ; /*declaration of two real type variables*/
```

Variable assignment

The symbol ← that we used in algorithms corresponds to the = sign in the C language.

Example :

```
int x, y ;
x=8 ;
y=x ;
```

VI. Constants

A constant is an identifier that contains a value that will never be changed during the program. For example, the constant PI can be set to 3.14.

Constants can be integers (12), real numbers (20.6), a character ('a'), or a string ("hello"). They are often written in uppercase to differentiate them from other identifiers.

Constant declaration

There are two ways to declare a constant:

Syntax 1:

```
#define constant_name constant_value
```

Examples:

```
#define PI 3.14
#define VAT 0.16
```

This declaration is made in the preprocessors part (after the inclusion of the libraries) and it allows the declaration of a constant that will be visible throughout the program and its lifetime is that of the entire application.

Syntax 2:

```
const type constant_name = constant_value;
```

Example :

```
const float Pi = 3.14 ;
```

This declaration is made in the block and it allows the declaration of a constant that will be visible only in the block and it will be destroyed when leaving the block.

VII. Displaying and entering variables

VII.1. The printf ( ) display function

This is a function implemented in the standard library stdio.h, it allows you to display data on the screen.

Syntax:

printf("String Format" [, argument_1…argument_n]);

- String format: string with formatting characters if necessary
- An argument can be: a value or a variable or an expression or a function call.

The number of arguments after the String Format is variable and depends on the number of variables for which you want to display the values. Note that to display a variable, once again, it depends on its type. Indeed, even if once displayed, a number or a string of characters only appears stupidly on the screen, the way the operating system proceeds to display it to you differs depending on the type of the variable. There are many formats. The following table shows the most used formats.

format	meaning
%c	Character
%s	string
%d	whole number to decimal
%e	real number in mantissa/exponent form [-]m.nnnnnne[+|-]xx
%E	real number in uppercase mantissa/exponent form [-]m.nnnnnnE[+|-]xx
%f	real number in the form [-]mmm.nnnnnn
%g	real number in the shorter form between types %e and %f
%G	real number in the shorter form between types %E and %f
%o	octal integer
%p	pointer or address of the numeric value
%u	unsigned integer in decimal
%x	whole number in hexadecimal
%X	whole number in hexadecimal; letters displayed in uppercase

Example:

In the following example we used the printf function to display a text message on the screen. printf is not a C instruction, but a function in the stdio.h library.

```
#include <stdio.h>
void main(){
        int x=8 ;
        float y=5.6 ;
        clrscr() ;
        printf("Hello");
        printf("x= %d and y=%f", x, y);

        printf("\n to continue hitting a key");
        /* \n allows line breaks*/
}
```

VII.2. The scanf() input function

This is a function implemented in the standard library stdio.h, it allows the entry of variables: it modifies their contents.

Syntax:

```
        scanf ("Format1 […Formatn]", &argument1[, … &argumentn]);
```

- Format must be a format from the previous table (%d, %f…)
- Argument can only be a variable whose value will be entered by the user
- & is mandatory except for entering pointer type variables

Example:

```
        int x;
        float y;
        scanf ("%d %f", &x, &y);
```

VIII. Work requested

Exercise 1:
Write a program that takes two integers A and B and performs a permutation of these two variables.

Exercise 2:
Write a program that calculates the surface area of a disk of radius R.

Exercise 3:
Write a program that allows you to enter an integer n and display each of these three digits separately: One, Ten, and Hundred. Assume that the number is less than 1000.

Exercise 4:
Enter and run the following program:

```
# include <stdio.h>
void main ( ) {
        int i = 6 ;        /* Declaration and initialization of the variable i */
        printf("%d \n", i) ;
}
```

Modify the program by changing the structure of the printf function (according to the table below), then complete the table:

Instruction	**Execution result**
printf(”%d\n”, i) ;	
printf(”%d\n”, 4+2) ;	
printf(”the sum 4+2 gives %d\n”, 4+2) ;	
printf(”%d + %d gives : %d\n”, i, 3, i+3) ;	
printf(”%d gives in octal %o and in hexadecimal %x or %X\n”, 90, 90, 90, 90) ;	

Exercise 5:
Enter and run the following program:

```
# include <stdio.h>
void main() {
        char x = ’b’, y = ’A’ ;
        printf("%c et %c are characters", x, y) ;
}
```

Modify the program by changing the structure of the printf function (according to the table below), then complete the table.

Instruction	**Execution result**
printf(”%c and %c are characters \n”, x, y) ;	
printf(”%c and %c have the ASCII codes %d and %d\ n”, ’b’, ’A’, ’b’, ’A’) ;	
printf(”%c and %c have the ASCII codes %d and %d\n”, 98, 65, 98, 65)	

Exercise 6

Enter and run the following program:

```
# include <stdio.h>
void main() {
        float x ;
        printf("Enter x: ");
        scanf("%f", &x) ;
        printf("%f", x) ;
}
```

Test this program with the values of x given below and complete the table:

Value of x	Execution result	Value of x	Execution result
3.4567		0.000012345	
12.3456789		1e-10	

1. Replace the instruction printf("%f", x) by printf ("%.3 f", x)

Value of x	Execution result	Value of x	Execution result
3.4567		123.456789E8	
123.45		-123.456789E8	

Exercise 7:

Enter the following program:

```
# include <stdio.h>
void main ( ) {
        int x;
        float y ;
        printf("Enter x: ");
        scanf("%d", &x) ;
        printf("Enter y: ");
        scanf("%f", &y) ;
        printf("%3d\n", x) ;,
        printf("%10f\n", y) ;
}
```

Test this program with the values of x and y given below and complete the table:

	Value of x	Execution result	Value of y	Execution result
1	17		1.2345	
2	6		12.345	
3	5432		1.2345E5	

Exercise 8: Conversational I/O

What results does the following program provide:

```
#include <stdio.h>
#include <stdlib.h>
void main() {
        int n=543;
        int p=5;
        float x=34.5678;
        printf ( "A : %d %f \n" ,n , x );
        printf ( "B : %4d %10f \n" ,n , x );
        printf("C:%2d%3f\n",n,x);
        printf ( "D : %10.3 f %10.3 e \n" ,x , x );
        printf ( "E : %-5d %f \n" ,n , x );
        printf ( "F : %*d\n" ,p , n ) ;
        printf ( "G : %*.*f \n" , 1 2 , 5 , x );
        printf("H:%x:%8x:\n",n,n);
        printf("I:%o:%8o:\n",n,n);
}
```

Exercise 9: Conversational I/O

1) What will be the values of the two variables n and p (of type int), by the following instruction:

```
        scanf("%d %d",&n, &p) ;
```

When provided with the following data (the ^ symbol represents a space and the @ symbol represents an end of line, i.e. a "commit").

(a) 253^45@
(b) ^253^@ ^^4^5@

2) What will be the values of the two variables n and p (of type int), by the following instruction:

```
        scanf ( "%4d %2d",&n,&p ) ;
```

When provided with the following data:

(a) 12^45@
(b) 123456@
(c) 123456^7@
(d) 1^458@
(e) ^^^4567^^8912@