Practical Work No. 7
The functions

I. Declaration

In C, sub-programs are called functions. A function always has a return type, which corresponds to the type of the result it can return and can be any of the types we previously studied; this return type can be void if we want the function to return nothing. It also has a name. And, finally, it has a list of zero, one, or more parameters.

Syntax:

```
ReturnType FNName (Type1 Parameter1, Type2 Parameter2){
    /*local variables*/

    /* function instructions */
   return value; /*optional if the function returns nothing (of type void)*/
}
```

II. Passing parameters:

Any parameter of a function can be of ordinary type (int, float,…) or of pointer type.

In the case of an ordinary parameter, passing it is called passing by value. In this case, the function makes a copy of the parameter's value. It only uses the copy, which can be modified. But at the end of the function's processing, the value of the variable passed as a parameter has not been modified, since only the copy has been modified.

In the case of a pointer type parameter, passing it is called passing by address. In this case, the function does not make a copy of the parameter value, as is done in the case of passing by value. Therefore, if the parameter changes value, then when the function exits, the variable passed as a parameter contains the last modification, i.e., the last value.

III. The prototypes

A function prototype is a description of a function that is defined later in the program. Therefore, the prototype is placed at the beginning of the program (before the main function main()). A function prototype is exactly the same as the function header, but not the body, which is replaced by a semicolon.

ReturnType FNName (Type1 Parameter1, Type2 Parameter2);

This description allows the compiler to "check" the validity of the function each time it encounters it in the program.

IV. Function Calls

A function is called as a program statement if it returns nothing, and it is called in an expression if an assignment, comparison, or other.

Examples:

```
#include <stdio.h>
#include<conio.h>
float square (float X) {
        X= X*X;
        return (X);
}

void main(){
        float A = 2, B;
        B= square (A);
        /*the value of A does not change after calling the square function */
        printf("the square of %f is=%f", A, B);
        getch();
}
```

```
#include <stdio.h>
#include<conio.h>
void permutation(int *, int *); /*prototype of the permute function */
void main(){
        int a = 2, b = 5;
        printf("start main program:\n a = %d \t b = %d\n",a,b);
        permutation(&a, &b);
        printf("end main program:\na = %d \tb = %d\n",a,b);
        getch();
}
void permutation (int *P1, int *P2){
        int temp;
        temp = *P1;
        *P1 = *P2;
        *P2 = temp;
}
```

V. Work requested

Exercise 1:
Write a C program that reads two non-zero natural numbers *m* and *n* and determines whether they are *Friend*. Two integers *n* and *m* are called *Friend* if the sum of the proper divisors of *n* equals *m*, and the sum of the proper divisors of *m* equals *n* (the number itself is not included among its divisors). Propose a modular solution.

Exercise 2:
Write a modular C program that includes a function to swap two integers passed as parameters. In the main program, read two integers from the user, call the swap function, and display the values before and after the swap.

Exercise 3:
1. Write a function that calculates x^y.
2. Write the main() function to enter two integers A and B on the keyboard, calculate and display A^B using the power function.

Exercise 4:
Create in C an algorithm for a function that searches for the first natural whole number whose square ends with n times the same digit.
Example: for n = 2, the result is 10 because 100 ends with the same digit twice.

Exercise 5:
1. Write a function that allows you to check whether an integer x belongs to an array T of n integers.
2. Test this function by writing a main program.

Exercise 6:
Write a C function that displays strictly increasing sub-sequences from an array of N integers.
Example for T = 5|7|9|2|3|1|20|25
The function displays:
5|7|9
2|3
1|20|25

Exercise 7: Sum, Product, and Average of the Elements
Write a C program that reads the size **N** ($5 < N \leq 20$) of an array **T** of type int. Fill this array with values entered from the keyboard and display the array. Then compute and display the **sum**, the **product**, and the **average** of the elements of the array.

Exercise 8: Occurrence of 0
Write a C program that reads the size **N** of an array **T** of type int (maximum size: 50 elements), fills the array with values entered from the keyboard, and displays the array. Then delete all occurrences of the value **0** from the array **T** and shift the remaining elements to remove the gaps. Display the resulting array.