Exercise “Programming in C”

1. Write a C program that keeps the actual values (temperature and pressure) of a sensor in a struct. It issues an alarm message (using printf) if the temperature is > 50° C or the pressure is < 3 bar.

2. Write a C program that displays a rectangle, built from the letters x, with width = 10 and height = 6:

```
xxxxxxxxxx
x        x
x        x
x        x
x        x
xxxxxxxxxx
```

3. Write a program that takes an integer value from the keyboard and returns the number with its digits reversed.
   For example, given the number 7631, the program should return 1367.

4. Convert the program from exercise 1 in a function that takes the width and height as parameters.
   Call that function from the main program.

5. Indicate in red the code lines that are syntactically incorrect. Indicate in green the expressions that are syntactically correct, but that are useless or might cause problems during execution.

   Finally, what will appear on the display? Do not use your computer.

```
int i,j;
double d;
int *ip = NULL, *jp = NULL;
double *dp = NULL;
i = 7;
ip = &7;
jp = &i;
*jp = j;
*ip = i;
ip = &j;
&i = ip;
j=4;
(*ip)++;
&d = dp;
```

```
*ip*= i;
*jp=*&j;
ip++;
i = ip-&i;
dp = &i;
dp = ip;
&dp = &&d;
*ip+=1;
*ip++;
cout <<endl<< i << j << d;
printf("i = %d, j = %d",i, j);
printf("i = %d, j = %d",*ip, *jp);
```

6. Use a two-dimensional array to solve the following problem:

   A company has 4 salespersons (1 to 4) and sells 5 different products (1 to 5).

   Each day, each salesperson submits a slip for each product sold.

   - This slip contains:
   - the salesperson's ID
   - the product ID
   - the total sales value of that product on that day.

   Each salesperson submits between 0 and 5 slips per day.

   Create a dummy-proof program to input the information from all these slips (over a period of, for example, one month), calculate the totals per person and per product.

   All totals must be stored in a two-dimensional array.

   Print the results as a table

   - where each column represents a salesperson each row represents a product
   - the product totals are in the rightmost column
   - the totals per salesperson are in the bottom row.

7. Write two functions:

   - tripleByValue, which receives an input number "by value" and multiplies it by 3.
   - tripleByReference, which receives an input number "by reference" and also multiplies it by 3.

   Place the prototypes in a header file and the function definitions in a .C file.

   Display the results of both functions, e.g.

```
Enter a decimal number: 123.65
tripleByValue: 3 x 123.65 = 370.95
tripleByReference: 3 x 123.65 = 370.95
```

8. Dynamically allocate memory for an array of integers whose length is requested from the user. Check if the allocation was successful. Experiment with testing to make the allocation fail. If the allocation was successful, fill the array with the elements of the Fibonacci sequence (a_n = a_{n-1} + a_{n-2}, with a_0 = 0 and a_1 = 1). Finally, display the contents of the array.

9. A function performs an operation on two doubles, which are passed as parameters. The function returns a double containing the result of the operation. In addition, it is also specified which operation should be performed: addition, subtraction, multiplication, etc. Implement this in a flexible way so that additional operations can be added later without having to modify the function.

10. Write a program that checks whether your system uses little-endian or big-endian.

11. Write a C program that performs several common bit manipulation operations on an 8-bit integer (uint8_t). The program should prompt the user for input and then perform the following actions:

    - **Set bit:**
        - Ask the user to enter a bit position (0-7) to be set (set to 1).
        - Use bit masking to set the specific bit position without changing the other bits.
    - **Clear bit:**
        - Ask the user to enter a bit position (0-7) to be cleared (set to 0).
        - Use bit masking to clear the specific bit position without changing the other bits.
    - **Read bit:**
        - Ask the user to enter a bit position (0-7) to be read.
        - Use bit masking to read the value of the specific bit and display it.
    - **Toggle bit:**
        - Ask the user to enter a bit position (0-7) to be toggled (inverted).
        - Use bit masking to toggle the specific bit position without changing the other bits.

Expected Output:

The program could work as follows, for example:

Enter a number (0-255): 172

```
Choose an operation:
1. Set bit
```

```
2. Clear bit
3. Read bit
4. Toggle bit
Choice: 4
Which bit position do you want to use? (0-7): 3
The entered number is: 172 (binary: 10101100)
The new number is: 164 (binary: 10100100)
```

This exercise helps you gain a deep understanding of bit manipulation in C, which is an essential skill for working with embedded systems and low-level programming.