PW Sheet N° 4

Part 1: Repetitive instructions While and Do-While

Bank Loan (قرض بنكي)

A bank grants a loan to its customers if the sum of their interests exceeds 10,000 DA. The interest rate is 3.5% per year.

1

For example:

if a customer invests 200,000 DA, the interest earned in the first year would be (200,000 DA * 3.5 / 100 = 7,000 DA.

In the second year, the interest earned would be (207,000 DA * 3.5) / 100 = 7,245 DA.

In the third year, the interest earned would be (214 245 x 3.5) /100 = 7 498,575 DA.

…

The process would continue until the interest earned exceeded 10,000 DA

Following the example, write a C program that reads the initial amount of money placed, then calculates the number of years needed to qualify for a loan.

```
#include <stdlib.h>
#include<stdio.h>
int main() {
int annee;
float sum, interest;
interest=0. ; year=0;
printf("Initial Amount: \n");
scanf("%f",&sum);
printf("----------------------------\n");
printf("Initial Amount =%f\n",sum);
// printf("year\Interest\sum\n");
printf("----------------------------\n");
while(……)
        {
        ……
        printf("Year=%d\tinteret=%f\sum=%f\n",year, interest, sum );
        }
printf("----------------------------\n");
printf("It takes %d years to qualify for a loan\n",year );
printf("----------------------------\n");
return (0);
}
```

1. Create a new project.
2. Type this code, filling in what's missing, then compile and run.
3. Is it possible to replace ***the While loop*** with the ***Do-While loop***?

Concluding remarks

1. The looping statement in the C programming language is implemented using the ***while*** statement, which has the following syntax:

```
while (condition)
    {
        /* Instruction_Block */
    }
```

2. The instruction block will run as long as the condition is true.
3. The *Do-while* loop guarantees that the statement block will be executed at least once. In the C programming language, it is written as follows:

```
Do
    {
        /* Instruction_Block */
    }
while (condition) ;
```

4. These two instructions are used when the number of repetitions is unknown.

Part 2 : Repetitive instruction For

Multiplication Table (Algorithm done in the lecture session)

Write a program in C language that allows you to:

- Enter an integer number
- Test if it is a digit (an integer number between 1 and 9),
- Display its multiplication table, using repetitive structures.

Concluding remarks

1. When the number of repetitions is known, the C programming language has a control flow statement that allows you to repeat the execution of a block of code by automating the initialization and modification of the loop variable. This statement is called the ***for*** statement. The syntax of the ***for*** statement is as follow:

```
For (initialisation ; condition ; modification)
{
    /* Instruction_Block */
}
```

2. This statement is equivalent to

```
initialisation;
while (condition)
        {
         /* Instruction_Block */;
         modification;
         }
```

3. Initialization is performed only once, at the beginning of the loop.
4. The condition is evaluated at the beginning of each iteration.
5. The modification is executed at the end of each iteration.
6. Do not change the loop variable (the counter) in the instruction block.

Reminder: Comparison operators in c language

Relational Operators		Suppose X and Y are two variables
Operator	**Expression**	**Description**
<	X < Y	X is less than Y
<=	X <= Y	X is less than or equal to Y
>	X > Y	X is greater than Y
>=	X >= Y	X is greater than or equal to Y
==	X == Y	X is equal to Y
!=	X != Y	X is not equal to Y

Part 3 : Application Exercises

1. Write an algorithm to input ***N*** integers and calculate and display the minimum and maximum (Algorithm done in the lecture session).

2. Write an algorithm to input ***N*** integers (***N*** is a positive non-zero integer entered by the user), then calculate the average of the even integers (Algorithm done in the tutorial session).

3. Write a program that prompts the user for a positive integer and then outputs its mirror image (Algorithm done in the lecture session).

4. Write an algorithm that simulates the displays of a countdown timer from a given time (in minutes and seconds)

5. We want to test the Syracuse conjecture using an algorithm. We start with a positive integer different from 1.
   If it is odd, we multiply it by 3 and add 1;
   Otherwise, we divide it by 2.
   We repeat these steps with the new number obtained until it reaches the value 1. In this case, we observe an infinite cycle {• • •, 1, 4, 2, 1, • • •}. Regardless of the starting integer, we will always end up with the value 1. For example: 5, 16, 8, 4, 2, 1, 4, 2, 1, ...

   Write an algorithm that allows entering a positive integer, not equal to 1. The algorithm should count the number of iterations of the Syracuse sequence until it reaches the value 1.

6. Write a C program that prompts the user to enter N integer values, then finds and prints the largest multiple of 5.

   Example 1 : N = 6, the numbers entered by the user are: 3, -7, 10, 45, -20, 11. Your algorithm should print 45.

   Example 2 : N = 4, the numbers entered by the user are: 3, -7, -21, 11. Your algorithm should print "There is no multiple of 5."

7. Write a C program that prompts the user to enter a positive integer n and then calculates and prints the n^{th} term of the Fibonacci sequence U_n, which is defined by $U_0 = 1$, $U_1 = 1$, $U_{n+2} = U_{n+1} + U_n$.

8. We can approximate ***e*** using the following series: $\sum_{i=1}^{\infty} \frac{1}{i!}$

   Write an algorithm that allows entering a positive non-zero integer ***N*** and calculates an approximation of ***e*** up to the N^{th} term:

$$e = 1 + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{N!}$$

9. The Egyptians knew how to calculate the product of two positive integers by successive decompositions, without using multiplication except by 2, using the following decomposition:

$$a * b = \begin{cases} a + a * (b-1) & if\ b\ is\ odd \\ a * 2 * (b/2) & \text{otherwise} \end{cases}$$

   Until b = 1

Write a C program that allows you to enter two positive non-zero integers and calculate and display them using this method.

10. Implement a C program to calculate the greatest common divisor (GCD) of two positive non-zero integers using successive subtractions. The program should prompt the user to enter the two integers, then calculate and display the GCD.