Practical Work No. 4

Iterative structures

Goals :

- Recall the iterative structures
- Present the syntaxes offered by the C language allowing the manipulation of iterative structures in C.

I. The structure for …

Syntax:

```
for (initialization; true continuity condition; modification) {
            …………..; /*instruction block*/
            ………….;
}
```

Noticed :

- The {} are not needed when the block has only one statement.
- The 3 for instructions do not necessarily relate to the same variable.
- An instruction can be omitted, but not the ;

Example :

```
for (i = 0 ; i < 10 ; i++) {
            printf(“%d”, i);

}
```

⇔

```
i=0 ;
for ( ; i < 10 ; ) {
            printf(“%d”, i);
            i++;
}
```

```
for (i = 0 , j = 10 ; i < j ; i++ , j--) {
            printf(“%d %d”, i, j);
}
```

⇔

```
i=0 ;
for (  j=10; i < j ; j-- ) {
   printf(“%d %d”, i, j);
   i++ ;
}
```

II. The while Structure

Syntax:

```
 while (expression) {
            …………..; /*instruction block*/
            ………….;
 }
```

The test is done first, the instruction block is not necessarily executed.

Remarks :

- The {} are not needed when the block has only one statement.
- The processing is executed 0 or n times! (depending on the condition)

III. The do … while Structure

<u>Syntax:</u>

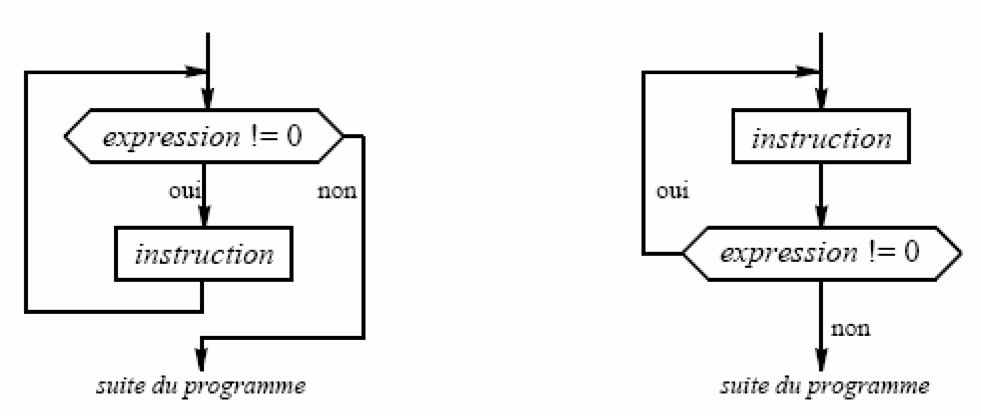
```
do{
        ………… ;
        …………; /*instruction block*/
         …………;
} while (expression) ;
```

Since the test is done after, the block is executed at least once

<u>Remarks :</u>

- the {} are not needed when the block has only one statement.
- The processing is executed 1 or n times! (depending on the condition)

The operation of while and do...while is described by the following flowcharts:



<u>Example:</u>

```
int sum=0, i=1;
while (i<=5) {
        sum += i ;
        i++ ;
}
```

```
int sum=0, i=1;
do {
        sum += i ;
        i++ ;
} while (i<=5) ;
```

IV. Work requested

Exercise 1:

Given a non-zero real number **X** and an integer **N**, write a C program that calculates $\mathbf{X}^{\mathbf{N}}$.

Consider all possible cases (when **N** is positive or negative).

Exercise 2:

Write a C program that calculates the sum of the digits of a given positive integer.
Example:
For **N = 25418**, we have **2 + 5 + 4 + 1 + 8 = 20**.

Exercise 3:

A natural number of three digits is said to be cubic if it is equal to the sum of the cubes of its three digits.

Example:

153 is cubic because $153 = 1^3 + 5^3 + 3^3$

Implement an algorithm in C that searches for and displays all three-digit cubic integers.

Exercise 4:

Write a C program that reads a positive integer and determines all its prime factors.

Examples:

30=2 * 3 * 5
36=2 * 2 * 3 * 3
99 = 3 * 3 * 11

Exercise 5:

Write a program that determines the minimum and maximum of n numbers entered on the keyboard.

Exercise 6:

Two integers are coprime if they have no common divisors other than 1.

- 7 and 13 have only 1 as a common factor, so 7 and 13 are coprime.
- 12 and 32 have several common divisors: 1; 2 and 4 so 12 and 32 are not coprime.

Write a C program that takes two integers N1 and N2, checks and displays whether they are relatively prime or not.

Exercise 7:

A number is said to be palindrome if it is written the same way from left to right or from right to left.

Examples: 101; 22; 3663; 10801, etc.

Write a C program to determine and display all palindromic numbers in the interval [100..9999].

Exercise 8:

Write a C program that displays all perfect numbers less than or equal to a given positive integer **n**.

A number is called **perfect** if it is equal to the sum of its divisors other than itself.

Example:
28 = 1 + 2 + 4 + 7 + 14

Perfect numbers less than 10000: 6, 28, 496, 8128.

Exercise 9:

Write a C program that reads two non-zero natural numbers **m** and **n**, and determines whether they are **amicable numbers**.

Two integers **n** and **m** are said to be **amicable** if the sum of the divisors of **n** is equal to **m**, and the sum of the divisors of **m** is equal to **n** (the number itself is not counted among its divisors).

Exercise 10:

Two integers **N1** and **N2** are said to be **"brothers"** if each digit of **N1** appears at least once in **N2**, and vice versa.

Examples:

- If **N1 = 1164** and **N2 = 614**, the program should display:

  ```
  N1 and N2 are brothers
  ```

- If **N1 = 405** and **N2 = 554**, the program should display:

  ```
  N1 and N2 are not brothers
  ```

Write a C program that reads two integers **N1** and **N2**, checks whether they are brothers, and displays the result.