Correction of the Lab Sheet N°1

2) Example 1: calculation of the value of $f(x)$

1. Create a new project and type the above code.
2. Modify the program to calculate $f(x)$ for multiple values of x entered by the user. The number of these values should also be entered by the user.

Answer:

```
#include <stdio.h>
int calculate_f(int a){
 int f;
 f = a * a - 2 * a + 1;
 return f;
}
int main() {
 int x,y,n,i;
 printf("Enter the number of values: ");
 scanf("%d",&n);
 for(i=1;i<=n;i++){
        printf("Enter the %d-th value of x: ",i);
        scanf("%d",&x);
        y=calculate_f(x);
        printf("f(%d)=%d\n",x,y);
 }
 return 0;
}
```

3. Make the necessary modifications so that the calculation of $f(x)$ is performed in a procedure instead of a function.

Answer:

```
#include <stdio.h>
void calculate_f(int a,int* f){
 *f = a * a - 2 * a + 1;
}
int main() {
 int x,y,n,i;
 printf("Enter the number of values: ");
 scanf("%d",&n);
 for(i=1;i<=n;i++){
        printf("Enter the %d-th value of x: ",i);
        scanf("%d",&x);
        calculate_f(x,&y);
        printf("f(%d)=%d\n",x,y);
 }
 return 0;
}
```

3) Application Exercises

1. Given A and B, two natural numbers, we want to establish a program that obtains a number C equal to the concatenation of the two numbers A and B.

   For example, if $A = 276$ and $B = 15$, the result of concatenation would be the number $C = 27615$.

   However, the number C can be obtained using the following formula:

   $$C = A \times 10^{nb} + B$$

   Where nb is the number of digits in B.

   To solve this problem, write the following functions:

   a) A function **`countDigits(n)`** that counts and returns the number of digits in a number **`n`** passed as a parameter.

   b) A function **`power(x, y)`** that takes two numbers **`x`** and **`y`** as parameters and returns **`x`** to the power of **`y`**. The use of the predefined **`pow`** function is prohibited.

   c) A function **`concatenate(A, B)`** that calculates and returns the number resulting from the concatenation of two numbers **`A`** and **`B`** passed as parameters. Concatenation is done using the formula described above.

   d) The **`main`** function that allows the user to enter two natural numbers, concatenate them, and display the result.

   Answer:

```c
#include <stdio.h>
int countDigits(int n){
 int nb;
 nb=0;
 while(n!=0){
        nb++;
        n=n/10;
 }
 return nb;
}
int power(int x,int y){
    int i,p;
    p=1;
    for(i=1;i<=y;i++)p = p * x;
    return p;
}
int concatenate(int A,int B){
    int nb,C;
    nb = countDigits (B);
    C = A * power(10,nb)+B;
    return C;
}
int main() {
 int A,B,C;
 printf("Enter the two numbers: ");
 scanf("%d%d",&A,&B);
 C=concatenate(A,B);
 printf("The concatenation result is: %d",C);
 return 0;
}
```

2. We want to calculate the number of combinations of P elements taken from N elements, knowing that this number is determined by the following formula:

$$C_N^P = \frac{N!}{P! \times (N-P)!}$$

a) Write the function **factorial** that returns the factorial of an integer passed as a parameter.

b) Write the function **combination** that takes two numbers N and P as input and calculates and returns the combination of P elements among N.

c) Write the **main** function that prompts the user to enter two numbers N and P, calculates, and displays the number of combinations of P elements taken from N elements by calling the previous **combination** function.

Answer:

```
#include <stdio.h>
int factorial(int n){
    int i,f;
    f=1;
    for(i=1;i<=n;i++)
        f=f*i;
    return f;
}
float combination(int n, int p){
    return (float)factorial(n)/(factorial(p)*factorial(n-p));
}
main(){
  int n,p;float comb;
  printf("Enter N and P: ");
  scanf("%d%d",&n,&p);
  if(n<=0 || p<=0 || p>n)printf("Input error");
  else{
    comb = combination(n,p);
    printf("The combination of %d among %d is %.2f",p,n,comb);
  }
}
```

3. A perfect number is a number that has the special property of being equal to the sum of all its divisors, excluding itself. For example, 28 is a perfect number because the proper divisors of 28 are 1, 2, 4, 7, and 14, and in addition, 28 = 1 + 2 + 4 + 7 + 14.

a) Write the function **isDivisor** that takes two numbers **a** and **b** as input and returns **True** if **b** is a divisor of **a** and returns **False** otherwise.

b) Write the procedure **sumDivisors** that calculates and returns the sum of divisors of an integer passed as a parameter.

c) Write the function **isPerfect** which, given an integer **n**, returns whether it is a perfect number or not.

d) Write the procedure **displayAllPerfect** which takes an integer **n** as input and displays all perfect numbers between 1 and **n**.

e) Write the **main** function that displays all perfect numbers between 1 and 1000 by calling the **displayAllPerfect** procedure.

Answer:

```
#include <stdio.h>
int isDivisor(int a,int b){
    if(a % b == 0)return 1;
    else return 0;
}
void sumDivisors(int n,int* s){
    int i=1;
    *s=0;
    for(i=1;i<n;i++)
       if(isDivisor(n,i)==1)*s=*s+i;
}
int isPerfect(int n){
    int s;
    sumDivisors(n,&s);
    if(n==s)return 1;
    else return 0;
}
void displayAllPerfect(int n){
    int i;
    for(i=1;i<=n;i++)
        if(isPerfect(i)==1)
            printf("%d is a perfect number\n",i);
}
main(){
    displayAllPerfect(1000);
}
```

4. Write a program to input an array of 10 real elements, reduce it, and display the result. The reduction involves subtracting the standard deviation of the array from each element. The standard deviation (σ) is given by:

$$\sigma = \sqrt{\frac{1}{n}\left(\sum_{i=0}^{n-1}\left(T[i]-m\right)^2\right)}$$

Where m is the mean of the elements in the array.

The program should be decomposed into the following subprograms:

- Procedure **`fillArray`** to input values from the keyboard into an array passed as a parameter.
- Procedure **`displayArray`** to display the elements of an array passed as a parameter.
- Function **`calculateMean`** that calculates and returns the mean of the elements in an array.
- Function **`calculateStd`** which takes an array as a parameter and calculates and returns its standard deviation.
- Procedure **`reduceArray`** to reduce an array passed as a parameter.
- The **`main`** function which only calls the aforementioned subprograms.

Anwser:

```
#include <stdio.h>
#include <math.h>
#define n 6
```

```
typedef float Tab[n];
void fillArray(Tab T){
    int i;
    for(i=0;i<n;i++){
        printf("Element %d: ",i);
        scanf("%f",&T[i]);
    }
}
void displayArray(Tab T){
    int i;
    printf("The elements of the array\n");
    for(i=0;i<n;i++)
        printf("%.2f\t",T[i]);
}
float calculateMean(Tab T){
    int i;float s,mean;
    s=0;
    for(i=0;i<n;i++)
        s=s+T[i];
    mean=s/n;
    return mean;
}
float calculateStd(Tab T){
    int i;float m,s,std;
    m = calculateMean(T);
    s = 0;
    for(i=0;i<n;i++)
        s=s+(T[i]-m)*(T[i]-m);
    std=sqrt(s/n);
    return std;
}
void reduceArray(Tab T,float std){
    int i;
    for(i=0;i<n;i++)
        T[i] = T[i] - std;
}
int main() {
    Tab T;float std;
    fillArray(T);
    std = calculateStd(T);
    reduceArray(T,std);
    displayArray(T);
    return 0;
}
```

5. Consider the sequence U_n defined by: $U_0 = 0$, $U_1 = 1$, $U_n = 3U_{n-1} - U_{n-2} + 5$
   a) Write a recursive function that takes an integer `n` as input and returns the n^{th} term of the sequence U_n.
   b) Test this function in a main program.

Answer:

```
#include <stdio.h>
int U(int n){
    if(n==0)return 0;
    else if(n==1)return 1;
    else return 3*U(n-1)-U(n-2)+5;
}
int main() {
    int n;
    printf("Give n: ");
    scanf("%d",&n);
    if(n<=0)printf("Input error");
    else printf("U(%d) = %d",n,U(n));
    return 0;
}
```

6. Let **`T`** be an array of 6 integer elements:

   a) Write a recursive procedure **`sum_Max(T, i, s, max)`** that calculates and returns the sum and the maximum of the elements in an array **`T`**.

   b) Write the **`main`** function that fills the array **`T`** and calculates, then displays the sum and the maximum of its elements. The calculation of the sum and the maximum should be done using the previous **`sum_Max`** procedure.

Answer:

```
#include <stdio.h>
#define n 6
typedef int Tab[n];
void sum_Max(Tab T,int i,int* s,int* max){
    if(i==n-1){
        *s=T[i];
        *max=T[i];
    }
    else{
        sum_Max(T,i+1,s,max);
        *s = *s + T[i];
        if(T[i] > *max)
            *max = T[i];
    }
}
int main() {
   Tab T;int i,s,max;
   for(i=0;i<n;i++){
        printf("Element %d: ",i);
        scanf("%d",&T[i]);
   }
   sum_Max(T,0,&s,&max);
   printf("The sum is %d\n",s);
   printf("The max is %d\n",max);
   return 0;
}
```