Licence 2ème Année
UFR de Mathématiques et Informatique
Université Paris Descartes
F. Cloppet

Correction TD-TP n° 5 Programmation Impérative

I- Partie TD Fonctions

Exercice 1

Soit la situation suivante :
La fonction *main* appelle la fonction *fonctionA* qui prend 2 entiers en paramètres et renvoie un entier
La fonction *fonctionA* appelle la fonction *fonctionB* qui prend un entier et un réel en paramètres et ne renvoie rien

a) Ecrire les déclarations des fonctions dans le cas où celles-ci ne sont pas prototypées
b) Ecrire les déclarations des fonctions dans le cas où celles-ci sont prototypées

a) sans prototypage

```
int fonctionB(int p1, float p2){
        ……..
}
int fonctionA(int x, int y){
    float z=5.4 ;
    …..
    fonctionB(z,y) ;
    return(x+y) ;
}
int main(){
    int a=3,b=4 ;

    fonctionA(a,b) ;
    return 1;
}
```

b) avec prototypage

```
int fonctionA(int , int ) ;
int fonctionB(int , float ) ;

int fonctionA(int x, int y){
    float z=5.4 ;
    …..
    fonctioB(z,y) ;
    return(x+y) ;
}
int fonctionB(int p1, float p2){
        ……..
}
int main(){ /* par convention on mettra toujours la fonction main à la fin mais elle
               pourrait être définie à n'importe quel endroit */
    int a=3,b=4, x ;

    x=fonctionA(a,b) ;
    return 1;
}
```

Exercice 2

1. Ecrire une fonction Somme qui effectue la somme de 2 entiers passés en paramètres et qui renvoie le résultat de la somme de ces 2 entiers.

```
int Somme(int p1,int p2){
   return(p1 +p2 );
}
int  main(){
   int x=3 , y=12, s ;

   s = Somme(x,x) ;
   printf("%d + %d = %d\n", x,y,s) ;

  /* ou printf("%d + %d = %d\n", x,y,Somme(x,y)) ; */
  return EXIT_SUCCESS;
}
```

2. Modifier la fonction écrite précédemment de manière à ce que le type de retour de la fonction Somme soit void, tout en "récupérant" le résultat de cette sommation à l'extérieur de la fonction.

```
void Somme(int p1,int p2, int *res){
   *res = p1 +p2 ;
}
int  main(){
   int x=3 , y=12, s ;

   Somme(x,y, &s) ;
   printf("%d + %d = %d\n", x,y,s) ;
   return EXIT_SUCCESS;
}
```

Licence 2ème Année
UFR de Mathématiques et Informatique
Université Paris Descartes
F. Cloppet

Exercice 3

La fonction factorielle est définie par :

0 ! = 1

n ! = n*(n-1) !

a) Ecrire une version itérative pour calculer f(i)

```
/*version itérative*/
double factorielleIter(int nb){ /* résultat de factorielle pouvant être grand, on met le type
                                   le + grand pour le type de retour*/
        int i;
        double res=1;

        for(i=nb;i>1;i--)
                res*=i;
        return res;
}
```

b) Ecrire une version récursive

```
/*version récursive*/
double factorielleRec(int nb){ /*résultat de factorielle pouvant être grand, on met le type
                                  le +  grand pour le type de retour*/
   if((nb==0) || (nb==1))
      return(1);
   return (nb*factorielleRec(nb-1));
}
```

Exercice 4

Ecrire une fonction récursive Hanoi, proposant une solution au problème dit des tours de Hanoï:

On dispose de 3 piquets numérotés 1, 2, 3 et de n disques de tailles différentes. Au départ ces disques sont empilés par taille décroissante sur le piquet n°1. Le but du jeu est de déplacer ces n disques du piquet n° 1 sur le piquet n° 3 en respectant les contraintes suivantes:

1) On ne déplace qu'un seul disque à la fois d'un piquet à 1 autre
2) Un disque ne doit jamais être placé au dessus d'un disque plus petit que lui

On peut montrer que cela revient à déplacer n disques du piquet n° 1 vers le piquet n° 3 en utilisant le piquet 2 comme piquet intermédiaire ce qui revient à faire:

- Déplacement des n-1 disques supérieurs du piquet 1 vers le piquet 2 en utilisant le piquet 3 comme piquet intermédiaire
- Déplacer le disque restant sur le piquet 1 vers le piquet 3 (*)
- Déplacer les n-1 disques du piquet 2 vers le piquet 3 en utilisant le piquet 1 comme piquet intermédiaire

(*) cette opération sera matérialisée par "printf("déplacer disque de %d vers %d\n", …); "

```
void hanoi(int nb, int deb, int fin, int inter){
    if(nb>0){
                hanoi(nb-1, deb, inter, fin);
                printf("déplacement de %d vers %d\n", deb, fin);
                hanoi(nb-1, inter, fin, deb);
    }
}
```

II-Partie TP Machine : Fonctions

Exercice 1

Ecrire 2 fonctions **Mult_2** et **Mult_3** à 1 argument entier et 1 valeur de retour entière permettant de préciser si l'argument reçu est multiple de 2 ou 3. Utiliser ces fonctions dans un programme qui lit un nombre entier et précise s'il est pair, ou multiple de 3, et/ou multiple de 6.

```
/*************************************/
/*      exo1 TP5                     */
/*************************************/
#include <stdio.h>
#include <stdlib.h>

int Mult_2(int n){
 if(n%2)
   return (0);
 return(1);
}

int Mult_3(int n){
  if(n%3)
    return(0);
 return(1);
}

int  main(void){
 int nb;

 printf("Donner un entier\n");
 scanf("%d", &nb);
 if(Mult_2(nb))
          printf("%d  est pair\n");
 if(Mult_3(nb))
          printf("%d est multiple de 3\n");
 if(Mult_2(nb)&&Mult_3(nb))
          printf("%d est donc multiple de 6\n");
 return EXIT_SUCCESS;
}
```

Licence 2ème Année
UFR de Mathématiques et Informatique
Université Paris Descartes
F. Cloppet

Exercice 2

1- Ecrire une fonction Swap qui prend 2 paramètres de type entier x et y et qui ne renvoie pas de valeur. Cette fonction réalise un échange de valeurs entre les paramètres formels x et y.

2- Ecrire un programme qui permette de saisir 2 paramètres effectifs param1 et param2, qui seront utilisés lors de l'appel de Swap. Afficher leurs valeurs avant d'appeler la fonction Swap. Afficher les valeurs des 2 paramètres formels avant de sortir de la fonction Swap. Afficher les valeurs des 2 paramètres effectifs après l'appel de la fonction Swap.

3- Modifier le programme de manière à ce que les valeurs des paramètres effectifs soient inversées par la fonction Swap.

```
4-  /**************************************/
5-  /*      exo2 TP5                      */
6-  /**************************************/
7-  #include<stdio.h>
8-
9-  void Swap(int x, int y){
10-   int aux;
11-
12-   aux = x;
13-   x = y;
14-   y = aux;
15-   printf("Pendant Appel de Swap   : %d\t %d\n", x,y);
16-  }
17-  void  Swap1(int *x, int *y){
18-   int aux;
19-
20-   aux = *x;
21-   *x= *y;
22-   *y = aux;
23-   printf("Pendant Appel de Swap1  : %d\t %d\n", *x,*y);
24-  }
25-  int  main (void){
26-   int a,b;
27-
28-   printf("Rentrer 2 valeurs entières séparées par un espace\n");
29-   scanf("%d %d", &a,&b);
30-   printf("Avant appel de Swap      : %d\t %d\n", a, b);
31-   Swap(a,b);
32-   printf("Après appel de Swap      : %d\t %d\n", a, b);
33-   printf("Avant appel de Swap1     : %d\t %d\n", a, b);
34-   Swap1(&a,&b);
35-   printf("Après appel de Swap1     : %d\t %d\n", a, b);
36-   return 1;
37-  }
```

Exercice 3

Ecrire 1 fonction Min_Max qui permet de "renvoyer" le minimum et le maximum de 3 valeurs passées en paramètres. Tester cette fonction dans un programme.

```
/**************************************/
/*                exo 3 TP5           */
/**************************************/
#include <stdio.h>

void Min_Max(int x, int y, int z, int *min, int *max){
 *min=x;
 *min=(*min<y?*min:y);
 *min=(*min<z?*min:z);

 *max=x;
 *max=(*max > y ? *max:y);
 *max=(*max> z ? *max:z);
}
int main(){
 int a,b,c,minimum, maximum;

 printf("Donnez 3 nombres séparés par des espaces \n");
 scanf("%d %d %d", &a,&b,&c);
 Min_Max(a,b,c, &minimum, &maximum);
 printf("Maximum: %d\t Minimum: %d\n", maximum, minimum);
 return 1;
}
```

Exercice 4

Les coefficients C_n^p du binôme de Newton ($(x+y)^n = \sum_{p=0}^{n} C_n^p x^{n-p} y^p$) peuvent être calculés récursivement grâce à la formule du Triangle de Pascal :

$$\forall n, n \in N, C_n^O = C_n^n = 1$$

$$\forall n, n \in N, n >= 2$$

$$\forall k, k \in N, 0 < k < n \quad C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$$

a) Ecrire une fonction CoeffBinome récursive qui calculera les coefficients du binôme de Newton.

```
double binomeRec(int n, int k){
 if((k==0)||(k==n))
   return 1;
 return(binomeRec(n-1, k-1)+ binomeRec(n-1, k));
}
```

b) Ecrire une version itérative de CoeffBinome sachant que C_n^p = n! / (p !*(n-p) !)

```
double factorielle(int nb){ /*résultat de factorielle pouvant être grand, on met le type le +
                              grand pour le type de retour */
        int i;
        double res=1;

        for(i=nb;i>1;i--)
                res*=i;
        return (res);
}

double binomeIter(int n, int k){
     return(factorielle(n)/(factorielle(k)*factorielle(n-k)));
}
```

c) Le programme devra finalement afficher le résultat du calcul du binôme de Newton pour les valeurs x, y, et n rentrées par l'utilisateur.

Pour le calcul de la puissance, réutilisez la fonction écrite pour l'exercice n°4 du TP4.

Ex : x=2, y=3, n=4 => résultat = 625

```
#include <stdio.h>
#include <stdlib.h>
double puissance(int a, int b){
  int i;
  double    res=1;

  for(i=0; i<b; i++)
     res*=a;
   return(res);
}

int  main(){
  int x,y,n,p;
  double res_rec=0., res_iter=0.;

  printf("Rentrer x:\n");
  scanf("%d", &x);
  printf("Rentrer y:\n");
  scanf("%d", &y);
  printf("Rentrer n: \n");
  scanf("%d", &n);

  /* version récursive */
  for(p=0; p<=n; p++){
      printf("BinomeRec(%d %d ): %.0lf\n", n,p,binomeRec(n,p));
      res_rec+= binomeRec(n,p)* puissance(x, n-p)*puissance(y,p);
  }
  printf("Le résultat de V. récursive est : %.0f\n", res_rec);

  /* version itérative */
  for(p=0; p<=n; p++){
    printf("BinomeIter(%d %d ): %.0lf\n", n,p,binomeIter(n,p));
     res_iter+= binomeIter(n,p)* puissance(x, n-p)*puissance(y,p);
  }
  printf("Le résultat de V. itérative est : %.0f\n", res_iter);
  return EXIT_SUCCESS;
}
```