Série 11.1 : Fonction II

Exercice 1 : Passage de paramètres [facile]

Ecrire une fonction `void echange(...)` qui permette l'échange du contenu de 2 variables déclarées dans le programme principal. Faire un programme pour la tester.

```
void Echange (int* val1, int* val2)
{
    int temp = *val1;
    *val1 = *val2;
    *val2 = temp;
}

int main()
{
    int var1 = 3, var2 =5;
    printf("var1:%d    var2:%d\n", var1, var2);
    Echange(&var1, &var2); printf("Echange(&var1, &var2)\n");
    printf("var1:%d    var2:%d\n", var1, var2);
    return 0;
}
```

Exercice 2 : Passage de paramètres [moyen]

Ecrire une fonction `void raz(...)` qui permette de mettre à zéro une valeur entière passée en argument. Ecrire une fonction `void testETraz(...)` qui appelle `raz()`si la valeur reçue est différente de 0.

```
void raz(int *px)
{
    *px = 0;
}

void testETraz(int* px)
{
    if (*px!=0)
    {
        raz(px);              //Le pointeur est simplement passé à raz
    }
}

int main()
{
    int x=19;
    testETraz(&x);
    printf("x: %d\n", x);
    return 0;
}
```

Exercice 3 : Passage de paramètres [moyen+]

Ecrire une fonction `void resetPointer (...)` qui permette de mettre à `NULL` un pointeur.

```
void resetPointer (void** pptr)  //il faut l'adresse du pointeur … donc **
{
    *pptr = NULL;
}

int main()
{
    int x=1;
    int *ptr = &x;
    printf("ptr:%p\n", ptr);
    resetPointer((void**)&ptr); //cast explicite pour éviter le warning
    printf("ptr:%p\n", ptr);
}
```

Exercice 4 : Passage de paramétres [moyen]

Reprendre l'exercice 2 de la série 10, et « factoriser » le programme écrit en encapsulant le code dans les fonctions `lireTaille`, `allocationTableau`, `remplirTableau`, `affichageTableau`, et `LiberationTableau`.
Le programme principal (main) prendra alors une forme concise et modulaire :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int lireTaille()
{
    int N;
    printf("Entrez la taille du tableau: ");
    scanf("%d", &N);
    return N;
}
int* allocationTableau(int n)
{
    int* adr = (int*) malloc(n*sizeof(int));
    return adr;
}
void remplirTableau(int* t, int n)
{
    int i;
    srand(time(NULL));
    for(i=0;i<n;i++)
        *t++ = 32+rand()%(127-32);
}
void afficherTableau(int* t, int n)
{
    int i;
    for(i=0;i<n;i++)
        printf("%d ",*t++);
```

```
    printf("\n");
}
void liberationTableau(int** t)
{
    free(*t);
    *t=NULL;
}
int main(){
    int* ptrTab=NULL;
    int N;
    N=lireTaille();
    ptrTab=allocationTableau(N);
    remplirTableau(ptrTab, N);
    afficherTableau(ptrTab, N);
    printf("%p %d\n",ptrTab, sizeof(ptrTab));
    liberationTableau(&ptrTab);
    printf("%p %d\n",ptrTab, sizeof(ptrTab));
}
```

Exercice 5 : Passage d'arguments à un programme **[facile]**

a) Ecrire un programme `Salutations` qui accepte un argument de type char à son lancement. Celui-ci déterminera la langue de l'utilisateur et affichera un message de salutations approprié.
Exemples :

```
C:\Salutations
Hello World !
Exécution du programme terminée normalement
```

```
C:\Salutations.exe I
Ciao Mondo !
```

```
C:\Salutations.exe F
Salut Tout le Monde !
```

...

b) Si la langue n'est pas supportée définir un code d'erreur et le renvoyer au système d'exploitation. Afficher un message informatif:

```
C:\Test.bat
Langue non supportee, contactez-nous pour souscrire
a l'abonnement PREMIUM.
```

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
```

```
    if (argc==1)
        printf("Hello world!\n");
    else
        switch (argv[1][0]) // on prend le premier caractère du premier argument
    {
        case 'E':
        case 'e': printf("Hello World!\n"); break;
        case 'F':
        case 'f': printf("Salut tout le monde!\n"); break;
        case 'D':
        case 'd': printf("Salu Tzame!\n");break;
        case 'I':
        case 'i': printf("Ciao Mondo!\n");break;
        default:  printf("Langue non supportee, Achetez la version PREMIUM!");
                  return -1;
    }
    return 0;
}
```

Exercice 6 : Callback **(moyen)**

Ecrire une fonction *CalculIntegrale(ptrf,a,b)* qui permette de calculer l'intégrale I d'une fonction f(x) quelconque définie entre deux bornes x=a et x=b :

$$I = \int_a^b f(x)\,dx$$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>       // pour des fonctions "toute pretes"

#define NB_INTERVALLES 1000

double calculIntegral (double (*ptrFonc)(double x), double a, double b   );
double fxEgalUn(double x) { return 1;}  // fonction constante unitaire f(x)=1
double fxEgalX(double x) { return x;}  // fonction linéaire f(x)=x

int main()
{
   double resultat;

   resultat = calculIntegral(sin, 0, 0.5);
   printf("\nL'integrale de sin(x) entre 0 et 0.5 vaut:%f\n", resultat);

   resultat = calculIntegral(exp, 0, 3);
   printf("\nL'integrale de exp(x) entre 0 et 3 vaut:%f\n", resultat);

   resultat = calculIntegral(log10, 1, 10);
   printf("\nL'integrale de log10(x) entre 1 et 10 vaut:%f\n", resultat);

   resultat = calculIntegral(fxEgalUn, 2, 5);
   printf("\nL'integrale de f(x)=1 entre 2 et 5 vaut:%f\n", resultat);

   resultat = calculIntegral(fxEgalX, 0, 4);
   printf("\nL'integrale de f(x)=x entre 0 et 4 vaut:%f\n", resultat);

   system("Pause");
   return 0;
}


double calculIntegral (double (*ptrFonc)(double x), double a, double b   )
{
    double x, surfaceRectangle, sommeRectangles=0.0;

    for (int i=0; i<NB_INTERVALLES; i++)
    {
        x = a + (b-a)* ((double)i/NB_INTERVALLES);
        surfaceRectangle= (*ptrFonc)(x) *((b-a)/NB_INTERVALLES);
        sommeRectangles += surfaceRectangle;
    }
    return sommeRectangles;
}
```

Exercice 7 : Pointeur fonction **(Facultatif)**

Ecrire une fonction **`modifierTableau`** qui reçoit en paramètre un tableau, sa taille ainsi qu'une fonction qui va modifier les éléments du tableau.
Deux fonctions seront utilisées, **`fois2`** et **`fois4`** (voir plus bas). Ces fonctions seront appliquées à tous les éléments du tableau. Elles sont passées en arguments à la fonction `modifierTableau` de 3 manières :

- La fonction elle-même
- Un pointeur de fonction
- Un élément d'un tableau de pointeur de fonction

Résultat :

```
               Tableau initial :     1     2     3     4
                   Fonction x2 :     2     4     6     8
                   Fonction x4 :     8    16    24    32
        Pointeur de fonction x2 :    16    32    48    64
        Pointeur de fonction x4 :    64   128   192   256
Tableau de pointeur de fonction :   128   256   384   512
Tableau de pointeur de fonction :   512  1024  1536  2048

Process returned 0 (0x0)   execution time : 0.650 s
Press any key to continue.
```

```c
#include <stdio.h>
#include <stdlib.h>

int fois2( int x );
int fois4( int x );
void modifierTableau( int *t, int n, int (*f) ( int ) );
void afficherTableau( char texte[], int t[], int n);

int main(void)
{
    int tab[ ] = { 1 , 2 , 3 , 4 } ;
    int size=sizeof(tab)/sizeof(int);
    afficherTableau( "Tableau initial :", tab, size );

    /* a) Fonction */
    /*-------------*/
    modifierTableau( tab, size, fois2 ) ;
    afficherTableau( "Fonction x2 :", tab, size );

    modifierTableau( tab, size, fois4 ) ;
    afficherTableau( "Fonction x4 :",tab, size );

    /* b) Pointeurs de fonctions */
    /*---------------------------*/
    int (*ptr_fois) (int) = fois2 ;

    modifierTableau( tab, size, ptr_fois ) ;
    afficherTableau( "Pointeur de fonction x2 :",tab, size );

    ptr_fois = fois4;
    modifierTableau( tab, size, ptr_fois ) ;
    afficherTableau( "Pointeur de fonction x4 :",tab, size );

    /* c) Tableau de fonction */
```

```
    /*------------------------*/
    int i;
    int (*tab_ptr_fois[2]) (int) = {fois2,fois4} ;
    for (i=0;i<2;i++)
    {
        modifierTableau( tab, size, tab_ptr_fois[i] ) ;
        afficherTableau( "Tableau de pointeur de fonction :",tab, size );
    }
    return EXIT_SUCCESS;
}

int fois2( int x )
{
    return 2*x;
}

int fois4( int x )
{
    return 4*x;
}

void modifierTableau( int *t, int n, int(*fonction)(int) )
{
  int i ;
  for ( i=0 ; i<n ; i++)
  {
    t[i] = (*fonction)(t[i]);
  }
}

void afficherTableau( char texte[], int t[], int n)
{
  int i ;
  printf( "%40s ", texte );
  for ( i=0; i<n; i++)
  {
    printf( "%5d ", t[i] );
  }
  printf( "\n" );
}
```

Exercice 8 : Callback **(facultatif, mais important)**

Compléter l'exercice 8, de manière à pouvoir trier le tableau une fois dans l'ordre croissant et une fois dans l'ordre décroissant avec la fonction **`qsort()`** de la libraire **`<stdlib.h>.`**

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <math.h>

/** Prototype de qsort
*** void qsort (void* base, size_t num, size_t size,
***                                 int (*compar)(const void*,const void*));
**/

/* Fonction call back pour qsort */
int  comparaisonCroissant(const void * a, const void * b);
int  comparaisonDecroissant(const void * a, const void * b);

/* Affichage du tableau*/
void printTab(char * text,float* t, short n);

int main()
{
    static float tab[]={4.6, 4.8, 4.79, -6.1, 0, -6.11, 4.79, -6.09, 4.81 , 0};
    short n=sizeof tab/sizeof(*tab);

    printTab("tableau initial: ",tab,n);

    qsort(tab,n,sizeof(*tab),comparaisonCroissant);
    printTab("Tableau croissant: ",  tab,n);

    qsort(tab,n,sizeof(*tab),comparaisonDecroissant);
    printTab("\n\ntableau decroissant: ",  tab,n);

    return 0;
}


void printTab(char * texte, float* t, short n)
{
   int i ;
   printf("%s\n", texte);
   for (i=0;i<n;i++)
        printf("%+4.2f | ", t[i]);
    printf("\n");
}
```

```
int comparaisonCroissant(const void * pa, const void * pb)
{
    float a = *(float*)pa;
    float b = *(float*)pb;

    if (a > b )
        return 1;
    else if (a < b )
        return -1;
    else
        return 0;

   // En une seule ligne pour rendre le programme moins comprehensible…
   //return (int) ceilf( a - b );
}

int comparaisonDecroissant(const void * pa, const void * pb)
{
    if (*(float *)pa < *(float *)pb )
        return 1;
    else if (*(float *)pa > *(float *)pb )
        return -1;
    else
        return 0;

    // En une seule ligne pour rendre le programme moins comprehensible…
    //return (int) ceilf( *(float *)pa - *(float *)pb );
}
```

Version 2 : Ajout d'une fonction permettant de vérifier si le tableau est correctement trié

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>

/** Prototype de qsort
*** void qsort (void* base, size_t num, size_t size,
***                                 int (*compar)(const void*,const void*));
**/

/* Fonction call back pour qsort */
int  comparaisonCroissant(const void * a, const void * b);
int  comparaisonDecroissant(const void * a, const void * b);

/* permet de tester si le tableau est correctement trie */
bool testTriTab(float* t, short n, bool (*comp)(float,float));
bool  testCompCroissant(float a, float b);
bool  testCompDecroissant(float a, float b);

/* Affichage du tableau*/
void printTab(char * text,float* t, short n);

int main()
{
    static float tab[]={4.6, 4.8, 4.79, -6.1, 0, -6.11, 4.79, -6.09, 4.81 , 0};
    short n=sizeof tab/sizeof(*tab);

    printTab("tableau initial: ",tab,n);

    qsort(tab,n,sizeof(*tab),comparaisonCroissant);
    printTab("Tableau croissant: ",  tab,n);
    if (testTriTab(tab,n,testCompCroissant))
        printf("OK: Trie croissant\n");
    else
        printf("KO: Pas trie croissant\n");

    qsort(tab,n,sizeof(*tab),comparaisonDecroissant);
    printTab("\n\ntableau decroissant: ",  tab,n);
    if (testTriTab(tab,n,testCompDecroissant))
        printf("OK: Trie decroissant\n");
    else
        printf("KO: Pas trie decroissant\n");
    return 0;
}

int comparaisonCroissant(const void * pa, const void * pb)
{
    float a = *(float*)pa;
    float b = *(float*)pb;

    if (a > b )
        return 1;
    else if (a < b )
        return -1;
    else
        return 0;

    //return (int) ceilf( a - b );
}
```

```
int comparaisonDecroissant(const void * a, const void * b)
{
    if (*(float *)a < *(float *)b )
        return 1;
    else if (*(float *)a > *(float *)b )
        return -1;
    else
        return 0;

    // En une seule ligne pour rendre le programme moins comprehensible…
    //return (int) ceilf( *(float *)a - *(float *)b );
}


 /* TESTE si le tabéeau est correctement trié */
bool testTriTab(float* t, short n, bool (*comp)(float,float))
{
    int i=0;
    while (i<n-1 && comp(t[i],t[i+1]))
    {
       i++;
    }
    return i==n-1;
}

bool  testCompCroissant(float a, float b)
{
    return a<=b;
}

bool testCompDecroissant(float a, float b)
{
    return a>=b;
}

void printTab(char * texte, float* t, short n)
{
   int i ;
   printf("%s\n", texte);
   for (i=0;i<n;i++)
        printf("%+4.2f | ", t[i]);
    printf("\n");
}
```