

CORRIGES

Cahier de TD n° 4

THÈME 1 : DÉFINITION DE FONCTIONS, VRAI/FAUX

**QUELQUES DÉFINITIONS
VRAI/FAUX**

THÈME 2 : FONCTIONS SIMPLES, DÉFINITIONS ET APPELS

**EXEMPLES DE DÉFINITION ET D'APPELS DE FONCTION :
UN PROGRAMME MAL NOMME**

THÈME 3 : FONCTIONS ET TABLEAUX STATIQUES/DYNAMIQUES

**FONCTIONS UTILITAIRES
RETOURNER UN TABLEAU ?
CAS D'UN TABLEAU STATIQUE
CAS D'UN TABLEAU DYNAMIQUE**

THÈME 4 : LE PASSAGE DES PARAMÈTRES

LES PARAMÈTRES DE TYPE POINTEUR

Thème 5 : Gestion des valeurs de retour

LES PROCÉDURES

Thème 1 : Définition de fonctions, vrai/faux

Quelques définitions

Parmi les **entêtes/headers .h** (*déclarations prototypes de fonctions*) qui suivent, lesquelles sont incorrectes et pourquoi ?

```
3x(double x, double y, double z); INCORRECTE
void troix(double x, double y, double z); CORRECTE
appel conforme:
troix(5.9, 7, 99.88);
```

```
double racine_car(double); CORRECTE paramètre non présent, peu clair
```

```
double racine(double n); CORRECTE paramètre présent ==> sémantique
```

```
long, double foo(long a, b, c); INCORRECTE
long foo(long a, long b, long c); CORRECTE
```

```
char bar(); CORRECTE
```

```
void x3Y_ft2Aé(long Azrz4_); INCORRECTE mais BEURk pour les noms
void transformFourier(long x); CORRECTE (par exemple!)
```

```
double gauss(double x, long sigma); CORRECTE
```

VRAI/FAUX

déclaration/définition :	paramètre ou paramètre formel
appel:	argument ou paramètre effectif

- Un **argument** (ou un **paramètre effectif**) est une entrée pour la **définition** de la fonction; **FAUX**
- Un **argument** (ou un **paramètre effectif**) est une valeur donnée par un programme lors de l'**appel** d'une fonction; **VRAI**
- Une fonction a au moins une sortie; **FAUX 0 ok type de retour void**
- Une fonction a au plus une sortie; **VRAI si 0 sortie alors le type de retour est void**
- Il faut écrire autant d'exemplaires d'une fonction que de nombres de fois où on veut l'utiliser; **FAUX contraire à l'objectif assigné à une fonction!**
- Une fonction rend un programme plus lisible; **VRAI (si elle est bien écrite!)**
- Un programme avec des fonctions est plus dur à maintenir qu'un programme sans fonctions (maintenir = faire la chasse aux bugs); **FAUX (on circonscrit le problème, le bug est à recherché dans une ou plusieurs fonctions, pas dans la seule fonction main!)**

- On doit obligatoirement appeler une fonction que l'on définit; **FAUX**
- On doit obligatoirement définir une fonction que l'on appelle; **VRAI**
 aussi bien pour ses propres fonctions que pour celles des fichiers d'entête du C (stdio.h, stdlib.h, string.h, math.h ...) dont le code compilé en binaire se trouve dans la librairie standard **libc** ou dans la librairie mathématique **libm**
- `afficher` est une fonction; **VRAI s'il a été définie FAUX sinon**
- `while` est une fonction. **FAUX, c'est un mot-clé du langage C**

Thème 2 : fonctions simples, définitions et appels

exemples de définition et d'appels de fonction :

Ecrivez les programmes suivants avec des fonctions. Le programme principal devra appeler la ou les fonctions définies au moins 2 fois .

- Programme faisant la somme de deux entiers;

```
#include <iostream>

using namespace std;

// entête (déclaration prototype) de la fonction somme
long somme (long x, long y);

int main()
{
    // appel de la fonction somme
    long res = 0;
    res = somme(7 ,8);
    cout << "somme : " << res << endl;

    return 0;
}

// définition de la fonction somme
long somme (long x, long y)
{
    return (x + y);
}
```

- Programme calculant la moyenne de deux entiers

```
#include <iostream>
#include <iomanip>
using namespace std;

// entête (déclaration prototype) de la fonction moyenne
double moyenne (long x, long y);

int main()
{
    cout << fixed << setprecision(2); // précision au centième

    // appel de la fonction moyenne
    cout << "moyenne : " << moyenne(7, 8) << endl;

    return 0;
}

// définition de la fonction moyenne
double moyenne (long x, long y)
{
    return (x + y)/2.0;
}
```

- Programme proposant un affichage propre : on fournit un nombre entier, par exemple qui indique le nombre N d'€ que vous avez gagné dans un jeu. Selon la valeur de N, qui peut être nulle, égale à 1 ou supérieure, le programme affichera un message tenant compte de cette valeur de N, notamment pour gérer le cas singulier/pluriel : on veut éviter d'afficher, par exemple, les messages :

"Vous avez gagné 1 euros" (car on doit mettre euro au singulier ici).

```
#include <iostream>
using namespace std;
```

```
// entête (déclaration prototype) de la fonction afficher
void afficher (long n);
```

```
int main()
{
    // appels de la fonction afficher
    cout << "afficher 0 euro: "; afficher(0);
    cout << "afficher 1 euro: "; afficher(1);
    cout << "afficher 8 euros: "; afficher(8);
    return 0;
}
```

```
// définition de la fonction afficher
void afficher (long n)
{
    if(n < 0)
        return; // rien à faire, on sort de la fonction sans retourner de valeur

    switch(n) // if else plus simple....
    {
        case 0:
            cout << "Vous n'avez pas gagné à notre jeu" << endl; break;
        case 1:
            cout << "Vous avez gagné 1 euro" << endl; break;
        default: // n > 1
            cout << "Vous avez gagné " << n << " euros" << endl; break;
    }
}
```

Arguments / paramètres / variables locales

Dans le programme suivant, indiquez où sont les arguments, les paramètres, les appels et les définitions des fonctions :

```
#include <iostream>

using namespace std;

// entête (déclaration prototype) des fonctions
long gilb1(long g1, long g2);
double gilb2(double g1, char g3);

int main()
{
    long g1, g2, gcpt;
    double gx;

    cout << "entrez deux entiers : ";
    cin >> g1; // 3
    cin >> g2; // 5

    cout << gilb2(1.34543, 'X') << endl; // 1.34543/1
    cout << gilb2(6.34543, 'X') << endl; // 6.34543/6

    cout << "resultat : " << gilb1(g2 - g1, g1 - g2) << endl; // gilb1(2, -2) retourne 2 + (-2) = 0

    g1 = gilb1(g1, g2); // gilb1(3, 5) retourne 8
    cout << "resultat : " << g1 << endl;

    // gilb1(8, 5) retourne 8 + 5 = 13
    // gilb1(13, 5) retourne 13 + 5 = 18
    cout << "resultat : " << gilb1(gilb1(g1, g2), g2) << endl;

    cout << "entrez le nombre de points : ";
    cin >> g1; // 10

    gx = 0.0;

    for (gcpt = 0; gcpt <= g1; gcpt++)
    {
        // 5.0/10= 0.5
        gx = 1.0 + gcpt * (5.0/g1); // 1.0, 1.5, 2.0, 2.5, .... 6
        cout << gx << "\t" << gilb2(gx, '#') << endl; //
    }

    return 0;
}
```

```

// retourne g1 + g2, g1 et g2 ne sont pas modifiés
long gilb1(long g1, long g2)
{
    long gtemp;

    gtemp = g1 + g2;

    /* Pauvre Gilbert!!!!
    g2 = gtemp - g2;
    g1 = g1 - gtemp + g1;
    g2 = g1 - g2;
    */
    return (gtemp);
}

// retourne g1 / [g1]
double gilb2(double g1, char g3) // g3 ah bon!!!!
{
    long gtemp;

    gtemp = g1; // troncature , partie entière de g1, notée [g1]

    return(g1/gtemp);
}

```

Gilbert vous dit que toutes les variables sont locales. Etes-vous d'accord avec lui ?

Ben oui, variables locales aux 3 fonctions (main, gilb1, gilb2)

Que fait ce programme ?

Voir commentaires ci-dessus

Quelle serait l'allure de la courbe dessinée à partir des points calculés à la fin du programme ?

Difficile à dire! Nuage de points?

Conclusion: Notre très cher ami Gilbert a encore sévi: son programme est incompréhensible! Il estf encore dans les nuages.

Un programme mal nommé

Donnez les valeurs des variables au fur et à mesure du déroulement du programme suivant : que constatez-vous ?

```
#include <iostream>

using namespace std;

// entête (déclaration prototype) de la fonction échange
void échange(long a, long b);

int main()
{
    long a,b;

    a = 5; // original
    b = 2; // original
    cout << "Dans la fonction main avant les échanges : a = " << a << " b = " << b << endl;;
    échange(b, a);
    cout << "Dans la fonction main apres l'appel échange(b,a) : a = " << a << " b = " << b << endl;;
    échange(a,b);
    cout << "Dans la fonction main apres l'appel échange(a,b) : a = " << a << " b = " << b << endl;;
}

// définition de la fonction échange
// x et y sont des copies de a et b
void échange(long x, long y)
{
    cout << "parametres de la fonction échange : x = " << x << " y = " << y << endl;;
    long temp;
    temp = x;      // modifier une copie
    x = y;         // modifier une copie
    y = temp;

    cout << "Dans la fonction échange apres échange: x = " << x << " y = " << y << endl;;
}
```

CONCLUSION: les variables a et b locales à la fonction main, ne sont pas modifiées après les 2 appels à la fonction échange. Cette dernière modifie des copies des 2 variables et pas les variables.

Remède: les pointeurs en paramètres

Thème 3 : fonctions et tableaux statiques/dynamiques

Fonctions utilitaires

Note :vous pourrez réutiliser ces fonctions standard à de nombreuses occasions pour vos projets futurs !

DEJA FAIT cf fusion triée de 2 tableaux triés par ordre croissant

Ecrire une fonction qui affiche tous les éléments d'un tableau dont le type est connu (à vous de choisir le type). Précisez quelles sont les entrées et la sortie de cette fonction

Ecrire une fonction qui effectue la saisie d'un certain nombre d'éléments à ranger dans un tableau dont le type est connu. Précisez quelles sont les entrées et la sortie de cette fonction.

Rappelez pourquoi il n'est pas nécessaire de retourner un tableau dans une fonction qui modifie les éléments stockés dans le tableau.

PASSAGE DE PARAMETRE D'UN TABLEAU: l'adresse du début du tableau, donc un pointeur, on peut donc modifier le tableau dans la fonction.

Ecrire une fonction qui effectue le tri d'un tableau par ordre croissant ou décroissant, le choix de l'ordre de tri se fait par l'intermédiaire d'un paramètre `ordre_tri`. Précisez quelles sont les entrées et la sortie de cette fonction.

A RECHERCHER en utilisant un algorithme de tri vu en cours.

Retourner un tableau ?

Cas d'un tableau statique

Une fonction peut parfois avoir besoin de retourner un tableau, par exemple une fonction qui effectue une copie d'un tableau d'entiers, ou de réels. Que se passe-t-il exactement dans ce cas ? Nous allons tenter de le déterminer en regardant le programme suivant.

```
long * copie_tab(long tab[], long util)
{
    long tab_res[100]; // tableau statique MEMOIRE LA PILE
    long cpt;

    for(cpt=0;cpt<util;cpt++)
    {
        tab_res[cpt] = tab[cpt];
    }

    return tab_res; // adresse du début du tableau
}

void affich_tab(long tab[], long t_ut)
{
    long cpt;

    for(cpt=0;cpt<t_ut;cpt++)
    {
        cout << tab[cpt] << ' ';
    }
    cout << endl;
}

int main()
{
    long tablo[100] = {1,2,3,4,5,6,7,8};
    long util;
    long * resultat;

    util = 8;

    resultat = copie_tab(tablo, util);

    affich_tab(resultat, util);
}
```

COMPILATEUR:

warning: address of local variable 'tab_res' returned|

==> A CORRIGER!

question 1) Quel est le type de la sortie de la fonction `copie_tab` ?

`long *`

question 2) Pourquoi doit-on utiliser un tableau dynamique pour la variable `resultat` du programme principal et non une définition statique telle que : `long resultat[100]` ?

`long resultat[100];`

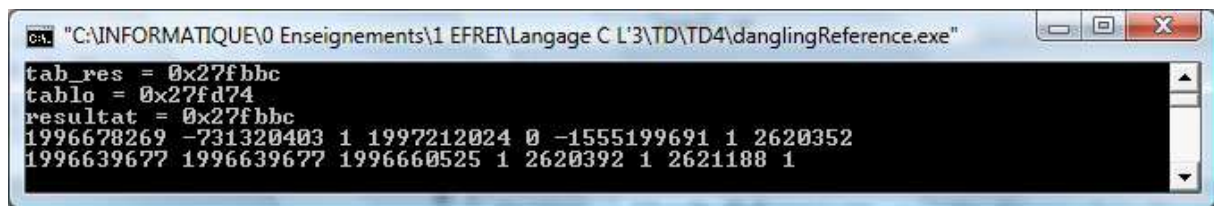
`resultat = copie_tab(tablo, util);`

Message du compilateur:

error: incompatible types in assignment of '`long int*`' to '`long int [100]`'

`resultat` est un pointeur constant

question 3) Que vaut, dans la fonction `copie_tab`, la variable `tab_res` ? (vous pouvez faire une hypothèse sur la valeur numérique de cette variable).



```
tab_res = 0x27fbbc
tablo = 0x27fd74
resultat = 0x27fbbc
1996678269 -731320403 1 1997212024 0 -1555199691 1 2620352
1996639677 1996639677 1996660525 1 2620392 1 2621188 1
```

question 4) Que vaut, dans le programme principal, la variable `tab_res` ?

Elle n'existe pas!

question 5) Que vaut la variable `resultat` du programme principal après l'appel à la fonction `copie_tab` ?

8 valeurs interdites sur la pile

question 6) Que trouve-t-on en mémoire à l'adresse stockée dans la variable `resultat` du programme principal ?

Même adresse que `tab_res` de `copie_tab`

question 7) Le programme affichera-t-il les valeurs du tableau ou affichera-t-il un message d'erreur ?

BUG : valeurs affichées ne sont pas des copies du tableau tablo.

cas d'un tableau dynamique

voici une nouvelle version de la fonction copie_tab, fonctionnant avec un tableau dynamique. Répondez de nouveau aux questions 4 à 7 de l'exercice précédent, et indiquez les instructions à ajouter éventuellement dans et/ou à la fin du programme principal pour adapter le programme à la nouvelle version de cette fonction.

```
long * copie_tab(long tab[], long util)
{
    long * tab_res;
    long cpt;

    tab_res = new long [util]; // tableau dynamique MEMOIRE LE TAS

    for(cpt=0; cpt<util; cpt++)
    {
        tab_res[cpt] = tab[cpt];
    }

    return tab_res;
}
```

CONCLUSION

Ne jamais retourner l'adresse d'une variable locale! (tableau, double, long, structure....)

COMPILATEUR:

warning: address of local variable 'tab_res' returned|

==> A CORRIGER!

Problème de la référence pendante (dangling reference)

Thème 4 : le passage des paramètres

les paramètres de type pointeur

rappelez l'utilité des paramètres qui sont des pointeurs

MODIFICATION DES VALEURS DES VARIABLES VIA UN APPEL DE FONCTION

Ecrire un programme avec une fonction qui réalise l'échange de 3 valeurs entières : la première reçoit la valeur de la deuxième, la deuxième celle de la troisième et enfin la troisième celle de la première. Cet échange doit être répercuté dans le programme principal.

```
#include <iostream>
using namespace std;

// entête (déclaration prototype) de la fonction d'échange
void echange3entiers(long * a, long * b, long * c);

int main()
{
    long a, b,c;

    a = 5; // original
    b = 2; // original
    c = 7;
    cout << "adresses : a = " << &a << " b = " << &b << " c = " << &c << endl;
    cout << "Dans la fonction main avant les echanges" << endl;
    cout << " a = " << a << " b = " << b << " c = " << c << endl; // 5 2 7
    echange3entiers(&a, &b, &c);
    cout << "Dans la fonction main apres l'appel echange(&a, &b, &c))";
    cout << "a = " << a << " b = " << b << " c = " << c << endl; // 2 7 5
}
```

```

// définition de la fonction d' échange
// les pointeurs x, y et z sont des copies de &a, &b et &c
// x pointe sur a, y pointe sur b, z pointe sur c
void echange3entiers(long * x, long * y, long * z)
{
    cout << "parametres de la fonction echange : x = " << x << " y = " << y << " z = " << z << endl;
    long temp;
    temp = *x;
    *x = *y;      // modifier l'original
    *y = *z;      // modifier l'original
    *z = temp;    // modifier l'original

    cout << "Dans la fonction echange apres echange: *x = " << *x << " *y = " << *y << " *z = " << *z
<< endl;
}

```

```

C:\INFORMATIQUE\0 Enseignements\1 EFREI\Langage C L'3\TD\TD4\echange3entiers.exe
adresses : a = 0x27ff0c b = 0x27ff08 c = 0x27ff04
Dans la fonction main avant les echanges
a = 5 b = 2 c = 7
parametres de la fonction echange : x = 0x27ff0c y = 0x27ff08 z = 0x27ff04
Dans la fonction echange apres echange: *x = 2 *y = 7 *z = 5
Dans la fonction main apres l'appel echange(&a, &b, &c))a = 2 b = 7 c = 5

```

Ecrire une fonction à qui l'on **fournit un tableau statique** et qui **retourne un tableau dynamique contenant des copies** des **éléments utilisés** du tableau statique passé en paramètre.

Ecrire un programme appelant cette fonction.

```
long * staticToDynamicTab(long * tab, long tailleUtile);
```

```
int main()
{
    long tab[] = {1, 3, 5, 2, 33, 22}; // tableau statique
    long tailleUtile = 6;

    long * p; // tableau dynamique

    p = staticToDynamicTab(tab, tailleUtile); // appel

    // affichage du tableau
}
```

```
// VERSION INDICE
```

```
long * staticToDynamicTab(long * tab, long tailleUtile)
```

```
{
    long * dyntab;

    dyntab = new long [tailleUtile];

    for (long i = 0; i < tailleUtile; i++)
    {
        dyntab[i] = tab[i];
    }

    return dyntab;
}
```

```
// VERSION N°1 POINTEUR LA MEILLEURE POUR LES TABLEAUX
```

```
// pas de cases dans les listes chaînées
```

```
long * staticToDynamicTab(long * tab, long tailleUtile)
{
    long * dyntab;

    dyntab = new long [tailleUtile];

    for (long * ptr = tab; ptr < tab + tailleUtile; ptr++, dyntab++)
    {
        *dyntab = *ptr;
    }

    // dyntab est à la fin; on doit reculer de tailleUtile cases
    // ARITHMETIQUE DES POINTEURS
    return dyntab - tailleUtile;
}
```



```

// VERSION N°2 POINTEUR OK tableaux
// idée à reprendre pour les listes chaînées
/*
long * staticToDynamicTab(long * tab, long tailleUtile)
{
    long * dyntab;

    dyntab = new long [tailleUtile];

    long * ptr = NULL;
    long * pdyn = NULL;
    for (ptr = tab, pdyn = dyntab; ptr < tab + tailleUtile; ptr++, pdyn++)
    {
        *pdyn = *ptr;
    }

    return dyntab;
}
*/

```

Thème 5 : Gestion des valeurs de retour

Les procédures

On appelle procédure une fonction dont la valeur de retour est un entier qui n'est pas le résultat d'un calcul. Cette valeur de retour est utilisée comme indicateur pour que le programme puisse savoir s'il peut se fier aux traitements que la fonction a effectué. Cela suppose également que la sortie de la fonction étant utilisée pour communiquer un entier, toute autre valeur que la fonction doit communiquer doit être gérée par un paramètre de type pointeur.

Status ou code de retour d'une fonction

Ecrivez une fonction `racine_carrée` qui calcule la racine carrée d'un nombre positif. La valeur de retour de cette fonction n'est pas le résultat du calcul, mais indique si le calcul a pu être effectué ou non.

```
int racineCarre(double x, double * y);
```

appel:

```
#include <limits.h>
```

```
double a = 76.35;
```

```
double racine = DOUBLE_MAX;
```

```
int status;
```

```
status = racineCarre(a, &racine);
```

<pre> #include <iostream> #include <stdio.h> #include <limits.h> #include <math.h> using namespace std; int racineCarre(double x, double * y); int main() { double a; double racine = DOUBLE_MAX; int status; cout << "nombre : "; cin >> a; status = racineCarre(a, &racine); if (status == 1) { cout << "nombre négatif" << endl; } else // status vaut 0 cout << a << " a pour racine " << racine << endl; } </pre>	<pre> /* retour 1 si x est négatif retour 0 si x >= 0 */ int racineCarre(double x, double * y) { int status = 0; if (x < 0) { status = 1; } else { *y = sqrt(x); } return status; } </pre>
---	---

Ecrivez une **fonction bruit** qui reçoit un tableau de réels et calcule la moyenne, la médiane et l'écart-type de ces valeurs. Si l'écart-type des valeurs est inférieur à un seuil S fourni en paramètre, alors on considère que les calculs effectués ne sont pas fiables, sinon on considère que les résultats sont corrects.

```
int bruit (double * t, double * moyenne, double * mediane,  
          double * ecartType, double seuil);
```

Ecrivez une fonction sensée recevoir un tableau t de nombres tous positifs et qui effectue un calcul de moyenne pondérée suivant : $m = (t[0] + 2*t[1] + 2*t[2] + t[3])/6$.

Quelles sont les cas de figures dans lesquels la fonction ne pourra pas effectuer correctement ses calculs ? Vous en déduirez les codes d'erreur possibles pour cette fonction.

Ecrivez une deuxième fonction qui affiche un message d'erreur compréhensible en fonction de la valeur de retour de la fonction précédente.

RETOUR SUR LA FUSION TRIEE DE DEUX TABLEAUX TRIES PAR ORDRE CROISSANT

```
#include <iostream>
#define TAILLE_MAXIMALE1 5
#define TAILLE_MAXIMALE2 7
using namespace std;

void saisirCroissant(long * tab, long tailleMaximale, long * tailleUtile);
void afficher(long * tab, long tailleUtile);
void fusionTrice(long * tab1, long tailleUtile1, long * tab2, long tailleUtile2,
                 long * tab3, long * tailleUtile3);

int main()
{
    long tab1[TAILLE_MAXIMALE1];
    long tab2[TAILLE_MAXIMALE2];
    long tab3[TAILLE_MAXIMALE1 + TAILLE_MAXIMALE2];
    long tailleUtile1 = 0;
    long tailleUtile2 = 0;
    long tailleUtile3 = 0;

    cout << "Saisie des " << TAILLE_MAXIMALE1 << " valeurs du premier tableau par ordre croissant : " <<
endl;
    saisirCroissant(tab1, TAILLE_MAXIMALE1, &tailleUtile1);
    cout << "Saisie des " << TAILLE_MAXIMALE2 << " valeurs du second tableau par ordre croissant : " <<
endl;
    saisirCroissant(tab2, TAILLE_MAXIMALE2, &tailleUtile2);

    cout << "Tableau1 : ";
    afficher(tab1, tailleUtile1);
    cout << "Tableau2 : ";
    afficher(tab2, tailleUtile2);

    fusionTrice(tab1, tailleUtile1, tab2, tailleUtile2, tab3, &tailleUtile3);

    cout << "Tableau3 : ";
    afficher(tab3, tailleUtile3);
    return 0;
}

void saisirCroissant(long * tab, long tailleMaximale, long * tailleUtile)
{
    long * ptr = NULL;
    for (ptr = tab; ptr < tab + tailleMaximale; ptr++)
    {
        cout << "entier : ";
        cin >> *ptr;
        (*tailleUtile)++;
    }
}
```

```
void afficher(long * tab, long tailleUtile)
```

```
{  
  
    long * ptr = NULL;  
    for (ptr = tab; ptr < tab + tailleUtile; ptr++)  
        cout << *ptr << ' ';  
    cout << endl;  
}
```

```
void fusionTrice(long * tab1, long tailleUtile1, long * tab2, long tailleUtile2, long * tab3, long * tailleUtile3)
```

```
{  
    long * ptr1 = tab1;  
    long * ptr2 = tab2;  
    long * ptr3 = tab3;  
    // tant qu'il y a des éléments dans les 2 tableaux  
    while ((ptr1 < tab1 + tailleUtile1) && (ptr2 < tab2 + tailleUtile2))  
    {  
        if (*ptr1 <= *ptr2)  
        {  
            *ptr3 = *ptr1;  
            ptr1++;  
            ptr3++;  
        }  
        else // *ptr1 > *ptr2  
        {  
            *ptr3 = *ptr2;  
            ptr2++;  
            ptr3++;  
        }  
        // (*tailleUtile3)++;  
    }  
    // tant qu'il y a encore des éléments dans le premier tableau  
    while (ptr1 < tab1 + tailleUtile1)  
    {  
        *ptr3 = *ptr1;  
        ptr1++;  
        ptr3++;  
        // (*tailleUtile3)++;  
    }  
    // tant qu'il y a encore des éléments dans le second tableau  
    while (ptr2 < tab2 + tailleUtile2)  
    {  
        *ptr3 = *ptr2;  
        ptr2++;  
        ptr3++;  
        //(*tailleUtile3)++;  
    }  
    }  
    *tailleUtile3 = ptr3 - tab3; // arithmétique des pointeurs (nombre de cases sans se soucier de la taille en  
    octets des cases  
}
```