D Corrigés des exercices

D.1 Exercice sur les structures

1. Définition de données

```
/* il faut définir la structure permettant de représenter */
/* chaque étudiant. On définit le type type_etud, on      */
/* suppose que le type type_date défini plus haut existe  */

typedef struct un_etudiant {   char nom[20];
                               char prenom[20];
                               type_date ne_le;
                               int num_carte;
                               float resultats[6];
                             } type_etud ;

/* le type étant défini on peut ensuite définir une */
/* promotion comme un tableau de 25 étudiants       */

type_etud promotion[25];

/* une promotion contient 25 étudiants maximum. Le nombre */
/*  maximum peut ne pas être le nombre réel d'étudiants,  */
/* il faut donc comptabiliser le nombre d'étudiants.      */
/* Initialement ce nombre est à 0 puisque le tableau est  */
/* vide. Ce nombre ne pourra pas excéder 24 (indice du    */
/* dernier élément. Attention c'est au programmeur de     */
/* vérifier qu'il n'y a pas dépassement                   */

int nb_etud_promo = 0;

/* la valeur de cette variable devra être incrémentée     */
/* à chaque fois qu'un étudiant est ajouté à la promotion */
```

2. Détermination du major de la promotion.

```
/* il s'agit seulement de la partie du programme chargée    */
/* d'effectuer cette tâche. On suppose que le tableau a été */
/* rempli et que nb_etud_promo contient le nombre exact     */
/* d'étudiants dans la promotion à traiter. On considère    */
/* également que leur définition est globale c'est-à-dire   */
/* ici en dehors de la fonction main                        */

void main(void)
{ int i, major = 0; float moy, moy_max = 0.0;

  /* saisie des informations concernant les étudiants */
  /* de la promotion le code correspondant n'est pas  */
  /* donné ici                                        */

  for(i=0; i < nb_etud_promo; i++)
  { /* calcul de la moyenne generale d'un etudiant */
    moy = 0.0;
    for(j=0; j<6; j++)
       moy = moy + promotion[i].resultats[j];
    moy = moy / 6;
    /* comparaison avec la plus grande moyenne */
```

```
    /* obtenue jusqu'à present                */
    if(moy > moy_max)
    {  /* mémorisation de la plus grande moyenne et*/
       /* de l'indice de l'étudiant supposé major  */
       moy_max = moy;
       major = i;
    }
  }
  /* affichage du nom et prenom de l'étudiant major */
  /* le cas de major ex-aequo n'a pas été envisagé  */

  printf("\n%s %s est le major de la promotion\n",
        promotion[major].nom, promotion[major].prenom);
}
```

D.2 Exercices sur les pointeurs (bases)

Solution énoncé 1 :
Pas de modification du type.

Par contre, lors de la manipulation on écrira :

```
moy = moy + *(((promotion+i)->resultats)+j)
(promotion+major)->nom, (promotion+major)->prenom
```

Solution énoncé 2 :
Ecrire un programme qui permet de créer les 3 personnages suivants : TOTO, 10 ans, fils de EGLANTINE X née Y, 30 ans, et de ARTHUR X, 35 ans

```
struct membre_famille
 {          char nom[20];
            char prenom[20];
            int age;
            struct membre_famille *ptr_pere, *ptr_mere; } TOTO =
                                        {"X", "TOTO", 10, NULL, NULL} ;
struct membre_famille EGLANTINE = {"Y", "EGLANTINE", 30, NULL, NULL} ;
struct membre_famille  ARTHUR = {"X", "ARTHUR", 35, NULL, NULL} ;
TOTO.ptr_pere = &ARTHUR ;
TOTO.ptr_mere = &EGLANTINE ;
```

On souhaite maintenant connaître les enfants d'un individu donné. Proposez une nouvelle structure et récrivez le programme créant TOTO et ses parents.

```
struct membre_famille
 {          char nom[20];
            char prenom[20];
            int age;
            struct membre_famille * ptr_pere, * ptr_mere;
            struct membre_famille * enfants[10]; } TOTO =
                                         {"X", "TOTO", 10, NULL, NULL, {NULL}} ;
struct membre_famille EGLANTINE = {"Y", "EGLANTINE", 30, NULL, NULL, {NULL}} ;
struct membre_famille ARTHUR = {"X", "ARTHUR", 35, NULL, NULL, {NULL}} ;
TOTO.ptr_pere = &ARTHUR ;
TOTO.ptr_mere = &EGLANTINE ;
ARTHUR.enfants[0] = &TOTO ;
EGLANTINE.enfants[0] = &TOTO ;
```

On apprend maintenant que TOTO a été adopté et que ses parents biologiques sont MARGUERITE Y, 25 ans, et ALPHONSE X, 25 ans. Sans recréer les personnages déjà existants, écrivez le programme mettant à jour la filiation de TOTO.

```
struct membre_famille MARGUERITE = {"Y", "MARGUERITE", 25, NULL, NULL, {NULL}} ;
struct membre_famille ALPHONSE = {"X", "ALPHONSE", 25, NULL, NULL, {NULL}} ;
TOTO.ptr_pere = &ALPHONSE ;
TOTO.ptr_mere = &MARGUERITE ;
ARTHUR.enfants[0] = NULL ;
EGLANTINE.enfants[0] = NULL ;
ALPHONSE.enfants[0] = &TOTO ;
MARGUERITE.enfants[0] = &TOTO ;
```

D.3 Exercices sur les passages de paramètres

D.3.1 Exercice 1

Question 1 Quand on déroule le programme à l'aide d'un tableau de situation, on obtient :

Variables	Valeurs en fc des étapes d'exécution				
i	0			1	
$a[0]$	10				
$a[1]$	20				
x		10		12	

Remarque : la case *a[0]* qui a servi de paramètre effectif n'est pas modifiée alors que le paramètre formel x est changé.

Question2 on obtient le tableau de situation correspondant.

Variables	Valeurs en fc des étapes d'exécution			
i	0			1
$a[0]$	10			12
$a[1]$	20			
x		adresse de $a[0]$		

Question3 Le tableau de situation correspondant serait le suivant :

Variables	Valeurs en fc des étapes d'exécution				
i	0		1		
$a[0]$	10			12	
$a[1]$	20				
x		10	12		

(plus le lien entre x et $a[0]$)

D.3.2 Exercice 2

Question 1 Quand on déroule le programme à l'aide d'un tableau de situation, on obtient :

Variables	Valeurs en fc des étapes d'exécution				
i	0		0	1	
$b[0]$	1		10		
$b[1]$	1				
x		1	3	5	

Remarquons que le paramètre effectif $b[0]$ est inchangé.

Question 2 Le tableau de situation correspondant est :

Variables	Valeurs en fc des étapes d'exécution				
i	0			0 1	
$b[0]$	1			3 10	12
$b[1]$	1				
x		adresse de $b[0]$			

Question 3 Le tableau de situation correspondant serait le suivant :

Variables	Valeurs en fc des étapes d'exécution
i	0 0 1
$b[0]$	1 10 5
$b[1]$	1
x	1 3 5

(plus le lien entre x et $b[0]$)

Remarquons qu'il y a ici un problème d'effet de bord (les passages par référence et par copie-recopie ne donnent pas le même résultat). Ce problème vient du fait que l'on modifie directement à l'intérieur du sous-programme la variable globale *b[0]* qui sert aussi de paramètre effectif.

D.3.3 Exercice 3

Quand on déroule le programme à l'aide d'un tableau de situation, on obtient 2 cas possibles :

— soit $f(i)$ est évaluée avant y (donc y sera modifiée AVANT d'avoir été évaluée en vue de l'addition) :

Variables	Valeurs en fc des étapes d'exécution
y	10 11
i	1
z	24 (13 + 11)
x	1 2
Valeur retournée par f	13

— soit $f(i)$ est évaluée après y (donc y sera modifiée APRÈS avoir été évaluée en vue de l'addition) :

Variables	Valeurs en fc des étapes d'exécution
y	10 (valeur mémorisée pour calculer z) 11
i	1
z	23 (13 + 10)
x	1 2
Valeur retournée par f	13

La différence de résultat obtenue pose problème car on ne peut jamais être sûr du sens de l'évaluation ! Cela dépend du compilateur et de la machine.

Remarque : Cet exercice est aussi un exemple d'effet de bord (mise à jour de la variable globale y lors de l'exécution de la fonction f).

D.3.4 Exercice 4 : Cas d'appels multiples

Quand on déroule le programme à l'aide d'un tableau de situation, on obtient :

Variables	Valeurs en fc des étapes d'exécution			
i	10 15	on oublie	20	
j	5	le contexte		
y	10	d'appel précédent	15	
x	5	et on charge un	5	
z	5 6	nouveau contexte	5 6	
Valeur retournée par p	15	d'appel	20	

D.3.5 Exercice 5 : Cas des tableaux

Quand on déroule le programme à l'aide d'un tableau de situation, on obtient :

Variables	Valeurs en fc des étapes d'exécution
TE	@1
TF	@1
à l'adresse @1	?/? 0/? 0/0
i	0 1

Ce que l'on a donc passé en paramètre était l'adresse de la première case du tableau, ce qui permet alors de modifier le contenu du tableau sans difficulté. En aucun cas, il n'y a copie de la TOTALITÉ du tableau !

D.3.6 Exercice 6 : Ecriture de fonctions

Solution énoncé 3 :

```
float moy (int t[N])
{
int i;
float s = 0.;
for (i=0; i<N; i++)
        s+=t[i];
return (s/N);
}
```

Solution énoncé 4 :

```
void moyminmax (int t[N], float* moy, int* max, int* min)
{
int i;
float s;
s = s[0];
*max = t[0];
*min = t[0];
for (i=1; i<N; i++)
        {
        s+=t[i];
        if (t[i] > *max) *max = t[i];
        else if (t[i] < *min) *min = t[i];
        }
*moy = s/N;
}
```

Solution énoncé 5 :

```
void decalage (int t[N])
{
int i, aux;
aux = t[0];
for (i=0; i<N-1; i++)
        t[i] = t[i+1];
t[N-1] = aux;
}
```

Solution énoncé 6 :

```
void somtab (int t1[N], int t2[N], int t3[N])
{
int i;
for (i=0; i<N; i++)
        t3[i] = t1[i] + t2[i];
}
```

Solution énoncé 7 :

```
void recutab(int mat[M][N], int recu[M], int * nbrecu)
{
int i;
float moyen;
*nbrecu = 0;
for (i=0; i<M; i++)
        {
        moyen = moy(mat[i]);
        if (moyen >= 10)
                {
                recu[*nbrecu] = i;
                (*nbrecu) ++;
                }
        }
}
```

Solution énoncé 8 :

```
int encode(char c)
{
return c;
}
```

Solution énoncé 9 :

```
void encodechaine (char * s, int t[N], int * taille)
{
int i;
*taille = strlen(s);
for (i=0; i<*taille; i++)
        t[i] = encode(s[i]);
}
```

Solution énoncé 10 :

```
int appartient(int t[N], int X)
{
int i, res =0;
for (i=0; ((i<N)&&(!res)); i++)
        if (t[i]==X) res = 1;
return res;
}
```

ou bien :

```
int appartient(int t[N], int X)
{
int i;
for (i=0; i<N; i++)
        if (t[i]==X) return 1;
return 0;
}
```

Solution énoncé 11 :

```
int appartientpos(int t[N], int X, int * pos)
{
int i, res = 0 ;
*pos = -1 ;
for (i=0 ; ((i<N)&&( !res)) ; i++)
        if (t[i]==X)
                {
                res = 1 ;
                *pos = i ;
                }
return res ;
}
```

ou bien :

```
int appartientpos(int t[N], int X, int * pos)
{
int i ;
*pos = -1 ;
for (i=0 ; i<N ; i++)
        if (t[i]==X)
                {
                *pos = i ;
                return 1 ;
                }
return 0 ;
}
```

Solution énoncé 12 :

```
void remplacepremier(int t[N], int X, int Y)
{
int i ;
for (i=0 ; (t[i] !=X) ; i++) ;
t[i] = Y ;
}
```

Solution énoncé 13 :

```
void remplacedernier(int t[N], int X, int Y)
{
int i ;
for (i=N-1 ; (t[i] !=X) ; i--) ;
t[i] = Y ;
}
```

Solution énoncé 14 :

```
int remplacepremierbis(int t[N], int X, int Y)
{
int i = 0, res = 0 ;
while ((i < N) && ( !res))
        if (t[i] == X)
                {
                t[i] = Y ;
                res = 1 ;
                }
        else i++ ;
return res ;
}
```

Solution énoncé 15 :

```
int remplacetous(int t[N], int X, int Y)
{
int i = 0, nb = 0;
while (i < N)
        {
        if (t[i] == X)
                {
                t[i] = Y;
                nb++;
                }
        i++;
        }
return nb;
}
```

Solution énoncé 16 :

```
int remplacetousbis(int t[N], int X, int Y, int remplaces[N])
{
int i = 0, nb = 0;
while (i < N)
        {
        if (t[i] == X)
                {
                t[i] = Y;
                remplaces[nb] = i;
                nb++;
                }
        i++;
        }
return nb;
}
```

D.4 Exercices sur les pointeurs

D.4.1 Exercice 1

```
pile * p ;
x = (*(*p)).info ;   /* OK et x est un entier */
x = (*p).info ;      /* NOK car p est un ptr sur un ptr donc *p est un ptr */
                     /* et l'opérateur . est interdit sur les ptr          */
x = p->info ;        /* NOK : idem second cas puisque       */
                     /* "p->info" est equiv à "(*p).info"   */
x = (*p)->info ;     /* OK et x est un entier */
```

D.4.2 Exercice 2

```
typedef  struct cel {
   int info;
   struct cel *  suiv;
   } cel ;      /* cel = structure à 2 champs (entier, pointeur sur cel) */

typedef struct  cel *  pile ;              /* pile = pointeur sur une cel */
                                  /* cel et struct cel sont des synonymes */
                        /* cel *, struct cel * et pile sont des synonymes */
pile creer_pile ( )
    {                                  /* création d'une pile à vide */
    return NULL ;
```

```
    }
int pile_vide (pile p)
    {              /* vérifier si une pile est vide (renvoie 0 si non */
                   /*    et une valeur diff. de 0 si oui)  */
    return (p == NULL);
    }
int sommet_pile (pile p, int * pcoderr)
    {                  /* récupérer la valeur en sommet d'une pile */
    *pcoderr = 0 ;    /* pcoderr = adresse du code d'erreur */
                      /*    (code  = 1 si Nok, 0 sinon)     */
    if (pile_vide(p))  {
        printf("erreur") ;
       *pcoderr = 1;
       }
    else return( p->info) ;
    }
pile depiler (pile p,  int * x, int * pcoderr)
    {                       /* supprime et renvoie le sommet de pile */
    pile  paux;
    *pcoderr = 1 ;                    /* pcoderr = idem sommet_pile */
    if (pile_vide(p))  printf("erreur") ;
    else  {
       paux = p ;
       *x = p->info ;
       p = p->suiv ;
       free(paux);
       *pcoderr = 0 ;
       }
    return p ;
    }
pile empiler (pile p, int x, int * pcoderr)
    {                                 /* rajoute une valeur à la pile */
    pile paux ;
    *pcoderr = 1 ;                     /* pcoderr = idem sommet_pile */
    paux = (pile)malloc(sizeof(struct cel));
    if (paux == NULL)  printf("erreur") ;
    else {
       paux->info = x ;
       paux->suiv = p ;
       p = paux ;
       *pcoderr = 0 ;
       }
    return p ;
    }
```

D.4.3 Exercice 3

```
/** CEL = struct (element, ptr sur la suivante) **/
typedef struct cel {
    element  info;
    struct   cel *    suiv;
} cel ;

/** MAFILE = struct (ptr sur 1ere cellule,
                     ptr sur derniere cellule) **/
typedef struct  cel2 {
  struct cel *     tete ;
```

```
  struct cel *     queue ; } mafile ;


/*** INIT_FILE = initialisation file **/
mafile init_file ( )
{
  mafile f ;
  f.tete = NULL ;
  f.queue = NULL ;
  return f ;
}

/*** FILE_EST_VIDE = verifie si file est vide **
 **  parametre : la file a verifier           **
 **  renvoie : 1 si vide, 0 sinon             **/
int    file_est_vide (mafile f)
{
  return (f.tete == NULL);
}


/** ENFILERENQUEUE = ajoute 1 elt en queue de file **
 ** parametres :                                   **
 **  - la file a maj,                              **
 **  - l'element a rajouter                        **
 **  - code d'erreur (1 si ok, 0 sinon)            **/
void enfilerEnQueue (mafile * f, element x, int * res)
{ struct cel * faux ;
  *res = 0 ;
  faux = (struct cel *)malloc(sizeof(struct cel));
  if (faux == NULL)
    printf("Erreur : file pleine => pas de rajout ");
  else {
    faux->info = x ;
    faux->suiv = NULL ;
    if (file_est_vide(*f)) {
        f->queue = faux ;
        f->tete = faux ;}
    else  {
        f->queue->suiv = faux ;
        f->queue = faux ;}
    *res = 1 ;}}

/** DEFILERENTETE = supprime 1 elt en tete d'une file **
 ** parametres :                                      **
 **  - la file a maj,                                 **
 **  - l'element que l'on a supprime                  **
 **  - code d'erreur (1 si ok, 0 sinon)               **/
void defilerEnTete (mafile * f,  element * x, int * res)
{ struct cel *  faux;
  *res = 0 ;
  if (file_est_vide(*f))
    printf("Erreur : file vide => pas de suppression ") ;
  else  {
    faux = f->tete ;
    *x = faux->info ;
```

```
    f->tete = faux->suiv ;
    free(faux);
    if (file_est_vide(*f)) f->queue = f->tete ;
    *res = 1 ;}}

/** ENFILERENTETE = ajoute 1 elt en tete de file **
 ** parametres :                                 **
 **  - la file a maj,                            **
 **  - l'element a rajouter                      **
 **  - code d'erreur (1 si ok, 0 sinon)          **/
void enfilerEnTete (mafile * f, element x, int * res)
{ struct cel * faux ;
  *res = 0 ;
  faux = (struct cel *)malloc(sizeof(struct cel));
  if (faux == NULL)
    printf("Erreur : file pleine => pas de rajout ");
  else {
    faux->info = x ;
    faux->suiv = f->tete ;
    if (file_est_vide(*f)) f->queue = faux ;
    f->tete = faux ;
    *res = 1 ; }}

/** DEFILERENQUEUE = supprime 1 elt en queue d'une file **
 ** parametres :                                        **
 **  - la file a maj,                                   **
 **  - l'element que l'on a supprime                    **
 **  - code d'erreur (1 si ok, 0 sinon)                 **/
void defilerEnQueue (mafile * f,  element * x, int * res)
{ struct cel * faux;
  struct cel * fprec;
  *res = 0 ;
  if (file_est_vide(*f))
    printf("Erreur : file vide => pas de suppression ") ;
  else  {
    faux = f->tete ;
    fprec = NULL ;
    while (faux != f->queue) {
        frec = faux ;
        faux = faux->suiv ; }
    /* faux pointe sur la derniere cellule de la file                      */
    /* fprec pointe sur l'avant-derniere cellule de la file si elle existe */
    *x = faux->info ;
    if (fprec != NULL) frec->suiv = NULL ;
    else f->tete = NULL ;
    free(faux);
    if (file_est_vide(*f)) f->queue = f->tete ;
    *res = 1 ;}}
```

On en déduit que le `defilerqueue` n'est pas du tout efficace puisqu'il faut tout parcourir pour pouvoir enlever une cellule. Il vaut donc mieux enfiler par la queue et defiler par la tête.

D.5 Corrigé pour le secrétariat médical

Solution énoncé 17 :

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

```
#include <string.h>

typedef struct Date  {
  int jour;
  char mois[10];
  int annee;
}     date;

typedef struct Personne  {
  char nom[20];
  char prenom[20];
  date ne_le;
}     personne;

typedef struct maladie {
  char nom[20];
  date diagnostic_le;
  char etat[20];
}     maladie;

typedef struct cel_dossier_medical {
  maladie une_maladie ;
  struct cel_dossier_medical * maladie_suiv ;
}     cel_dossier_medical ;

typedef cel_dossier_medical * dossier_medical ;

typedef struct client {
  personne ident_client ;
  dossier_medical dossier ;
}     client;

typedef struct cel_liste_clients {
  client un_client;
  struct cel_liste_clients * client_suiv ;
}     cel_liste_clients;

typedef cel_liste_clients * liste_clients ;

typedef struct medecin {
  personne ident_medecin ;
  char specialite[20];
  liste_clients liste;
}      medecin ;

typedef struct cel_secretariat {
  medecin un_medecin ;
  struct cel_secretariat * medecin_suiv ;
}      cel_secretariat ;

typedef cel_secretariat * secretariat ;


secretariat init_secretariat()  {
  return NULL ;
}
```

```
dossier_medical init_dossier_medical()  {
  return NULL ;
}

int dossier_medical_est_vide(dossier_medical d)  {
  return d == NULL ;
}

client init_client (personne p)  {
  client c ;
  c.ident_client = p ;
  c.dossier = init_dossier_medical() ;
  return c ;
}

int date_identique(date d1, date d2)  {
  return ((d1.jour == d2.jour) &&
  (!strcmp(d1.mois, d2.mois)) &&
  (d1.annee == d2.annee));
}

int personne_identique(personne p1, personne p2)  {
  return ((!strcmp(p1.nom, p2.nom)) &&
  (!strcmp(p1.prenom, p2.prenom)) &&
  (date_identique (p1.ne_le, p2.ne_le)));
}

int client_identique(client c1, client c2)  {
  return (personne_identique(c1.ident_client, c2.ident_client));
}

secretariat ajout_medecin_secretariat(secretariat s, medecin m)  {
  secretariat saux ;
  saux = (secretariat)malloc(sizeof(cel_secretariat));
  saux->un_medecin = m;
  saux->medecin_suiv = s ;
  return saux ;
}

dossier_medical ajout_maladie_dossier_medical(dossier_medical d, maladie m)  {
  dossier_medical daux ;
  daux = (dossier_medical)malloc(sizeof(cel_dossier_medical));
  daux->une_maladie = m;
  daux->maladie_suiv = d ;
  return daux ;
}

void modif_maladie_dossier_medical(dossier_medical d, maladie m)  {
  int trouve = 0 ;
  while ((!dossier_medical_est_vide(d)) && (! trouve))
      if (d->une_maladie.nom == m.nom)
{
  d->une_maladie = m ;
  trouve = 1 ;
}
```

```
      else
d = d->maladie_suiv;
}

liste_clients supprimer_client_de_liste_clients(liste_clients l, client c)  {
  liste_clients laux ;
  liste_clients lprec = NULL;
  int trouve = 0;
  laux = l ;
  while ((laux != NULL) && (!trouve))
    if (client_identique(l->un_client, c))
      trouve = 1 ;
    else
      {
lprec = laux ;
laux = laux->client_suiv;
      }
  if (trouve)
    {
      if (lprec == NULL)
l = laux->client_suiv ;
      else
lprec->client_suiv = laux->client_suiv ;
      free (laux);
    }
  return l ;
}


main()  {
}
```

Avec les modifications demandées, on obtient :

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

typedef struct Date  {
  int jour;
  char mois[10];
  int annee;
}     date;

typedef struct Personne  {
  char nom[20];
  char prenom[20];
  date ne_le;
}     personne;

typedef struct maladie {
  char nom[20];
  date diagnostic_le;
  char etat[20];
}     maladie;

typedef struct cel_dossier_medical {
  maladie une_maladie ;
```

```
  struct cel_dossier_medical * maladie_suiv ;
}     cel_dossier_medical ;

typedef cel_dossier_medical * dossier_medical ;

typedef struct cel_client {
  personne ident_client ;
  dossier_medical dossier ;
}      cel_client;

typedef cel_client * client; /* c'est desormais un pointeur et non une structure */

typedef struct cel_liste_clients {
  client un_client;
  struct cel_liste_clients * client_suiv ;
}     cel_liste_clients;

typedef cel_liste_clients * liste_clients ;

typedef struct medecin {
  personne ident_medecin ;
  char specialite[20];
  liste_clients liste;
}      medecin ;

typedef struct cel_secretariat {
  medecin un_medecin ;
  struct cel_secretariat * medecin_suiv ;
}      cel_secretariat ;

typedef cel_secretariat * secretariat ;


secretariat init_secretariat()  {
  return NULL ;
}

dossier_medical init_dossier_medical()  {
  return NULL ;
}

int dossier_medical_est_vide(dossier_medical d)  {
  return d == NULL ;
}

client init_client (personne p)  {
  client c ;
  c = (client)malloc (sizeof (cel_client));
  c->ident_client = p ;
  c->dossier = init_dossier_medical() ;
  return c ;
}

int client_identique(client c1, client c2)  {
  return (c1 == c2);
}
```

```
int appartient_client_a_une_liste_clients(liste_clients l, client c)  {
  int trouve = 0 ;
  while ((l != NULL) && (! trouve))
    if (client_identique(l->un_client, c))
      trouve = 1 ;
    else
      l = l->client_suiv ;
  return trouve ;
}

int nb_occurrences_client_a_un_secretariat(secretariat s, client c)  {
  int nb = 0 ;
  while (s != NULL)
    {
      if (appartient_client_a_une_liste_clients(s->un_medecin.liste, c))
nb++ ;
      s = s->medecin_suiv ;
    }
  return nb ;
}

liste_clients supprimer_client_de_liste_clients(liste_clients l, client c, secretariat s)  {
  liste_clients laux ;
  liste_clients lprec = NULL;
  int trouve = 0, nb ;
  laux = l ;
  nb =  nb_occurrences_client_a_un_secretariat(s, c);
  while ((laux != NULL) && (!trouve))
    if (client_identique(l->un_client, c))
      trouve = 1 ;
    else
      {
lprec = laux ;
laux = laux->client_suiv;
      }
  if (trouve)
    {
      if (lprec == NULL)
l = laux->client_suiv ;
      else
lprec->client_suiv = laux->client_suiv ;
      if (nb == 1) free (laux);
    }
  return l ;
}


main()  {
}
```