Programmation en C/C++

Série 11
Les listes chaînées

 Important !  Difficile !

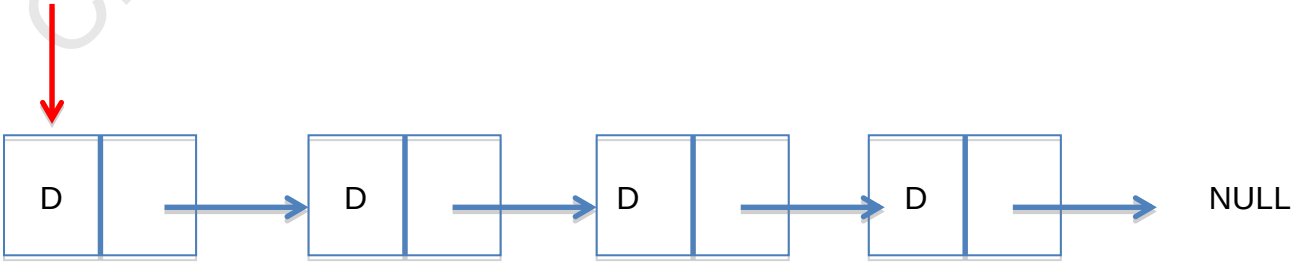
Objectifs
Créer une liste chaînée. Ajouter, supprimer des maillons en tête, milieu ou fin de liste. Utiliser une liste chaînée. Cette série est difficile.

Introduction

Nous avons déjà abordé précédemment les tableaux et les structures de données.
Les tableaux ont une limite : il est impossible d'agrandir ou rétrécir un tableau ou de supprimer certains de ses éléments en cours de route.
Imaginons une course à pieds avec des coureurs, leur numéro et leur âge.
On dimensionne un tableau de 100 coureurs au départ. Le programme tourne et on a besoin d'enregistrer un 101ème coureur : impossible !
Le 45ème coureur se désinscrit : il y aura une case vide et non utilisée dans le tableau… Imaginons enfin que 60 coureurs seulement s'inscrivent : il y a aura 40 cases allouées en mémoire, pour rien…
Les listes chaînées vont permettre une meilleure gestion de la mémoire. Les algorithmes pour gérer les listes chaînées pourront être itératifs mais gagneront à être récursifs.
Enfin, au delà de leur intérêt, les listes chaînées permettent d'élever son propre niveau de programmation. En effet, elles font appel aux fonctions, aux pointeurs, à la récursivité, aux structures et à des algorithmes demandant un minimum d'abstraction.

Définition d'une liste chaînée

Une liste chaînée est une structure récursive composée de maillons.
Chaque maillon est composé de donnée(s) et d'un pointeur vers… un autre maillon.



Exemple de liste avec 4 maillons

D symbolise des données, les flèches bleues symbolisent des pointeurs vers le maillon suivant. On parle de liste chaînée car tous les maillons sont chaînés les uns aux autres via des pointeurs. Le dernier pointeur est vide (fin de la liste). La flèche rouge représente un pointeur vers le premier maillon (tête de liste).

Structure d'une liste chaînée

Voici un exemple de liste chaînée où chaque maillon est constitué d'un entier (nommé "valeur" et d'un pointeur nommé "suivant").

```
typedef struct Maillon Maillon;
struct Maillon
{
int valeur;
Maillon *suivant;
}
```

Remarque

On voit que la structure ci-dessus est récursive car en définissant la structure "Maillon" on fait appel à un pointeur nommé "suivant" qui pointe sur un "Maillon".

Création de la liste

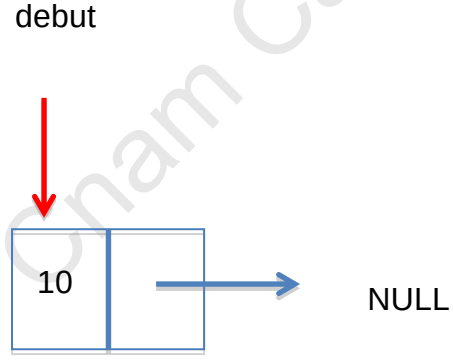
*Maillon *debut = malloc(sizeof(*P1));*

On crée un pointeur **debut* pointant sur un maillon ; on alloue de la mémoire pour la création de *debut*.

Exemple de cours 1

Création d'une première liste chaînée

On souhaite créer la liste chaînée suivante :



Et l'afficher.

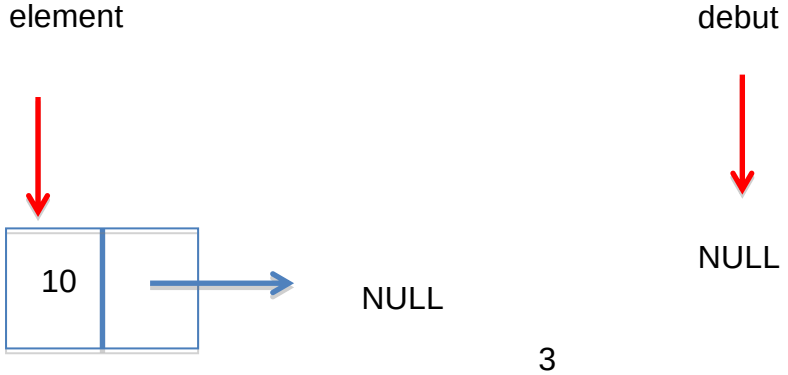
Corrigé

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

```
int main()  {
// Création de la structure de la liste
typedef struct Maillon Maillon;
        struct Maillon {
        int valeur;
        Maillon *suivant;
        };

// Création d'un maillon nommé element et d'un pointeur nommé debut pointant sur un maillon
Maillon *element = malloc(sizeof(*element));
Maillon *debut = malloc(sizeof(*debut));
debut =NULL;

// Affectation pour le premier maillon
element->valeur=10;
element->suivant=NULL;

//on fait pointer debut sur l'élément
debut = element;

// Affichage
printf("Liste : %d", debut->valeur);

return 0;
}
```

Remarque
Si l'instruction du type :
*Maillon *maListe = malloc(sizeof(*maListe));* ne fonctionne pas avec certains compilateurs, la remplacer par :
*Maillon *maListe = (Maillon *) malloc(sizeof(*maListe));*

Explications
element->valeur=10; affecte la valeur 10 au premier maillon,
element->suivant=NULL; le pointeur element->suivant est vide il ne pointe sur rien et correspond à la fin de liste (flèche bleue).
Maillon *debut = malloc(sizeof(*debut));
debut = NULL; On a créé un pointeur début qui ne pointe sur rien pour le moment.



debut = element; on fait pointer le pointeur *debut* sur l'élément créé.

printf("Liste : %d", debut->valeur); Il s'affichera donc à l'écran "Liste : 10".

Exemple de cours 2

Insertion en début de liste

On souhaite créer la liste suivante :
→3→2→1→0→NULL ; et l'afficher. On créera en premier le maillon 0 puis le 1 puis le 2 et enfin le 3. L'ajout se fait donc par le haut. On n'utilisera, dans cet exemple, aucune fonction. Voici l'état de la liste au fur et à mesure des étapes :

- *→0→NULL*
- *→1→0→NULL*
- *→2→1→0→NULL*
- *→3→2→1→0→NULL*

Corrigé

Pour un insérer un maillon par le haut, il faut réaliser la séquence suivante :

- Créer un nouveau maillon,
- Accrocher la liste existante à la suite du nouveau maillon
- Remonter le pointeur de tête de liste sur le nouveau maillon

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

typedef struct Maillon Maillon;

struct Maillon {
        int valeur;
        Maillon *suivant;
        };

int main() {
int i=0;
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *maListebis = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));


element->valeur = 0;
element->suivant=NULL;

maListe = element;
```

```
for (i=1; i<4;i++)  {
        Maillon *element = malloc(sizeof(*element));
        element->valeur = i;
        element->suivant = NULL;

        element->suivant = maListe;
        maListe = element;
        }

maListebis = maListe;

while (maListebis != NULL)     {
        printf("%d -> ", maListebis->valeur);
        maListebis = maListebis->suivant;
        }
printf(" NULL");
return 0;
}
```

Lors de l'affichage, le danger principal est de travailler sur la liste principale et d'utiliser des instructions du type *Liste=Liste->suivant* ; pour se décaler en fin de liste. En effet, on finira bien par avoir un *Liste->suivant* qui pointe sur *NULL* et l'on sera alors bien en fin de liste… Le problème est le suivant : *Liste* pointera sur le dernier maillon !
Vous avez donc perdu votre liste… car vous n'aurez aucun pointeur pointant sur le premier élément de la liste !! Il faut donc penser à dupliquer le pointeur pointant en tête de liste et travailler sur cette copie.

Exemple de cours 3

Insertion en fin de liste

On souhaite créer la liste suivante :
→3→2→1→0→NULL ; et l'afficher. On créera en premier le maillon 3 puis le 2 puis le 1 et enfin le 0. L'ajout se fait donc par le bas. On n'utilisera, dans cet exemple, aucune fonction. Cet exemple est un plus complexe dans la mesure où il faut se déplacer en fin de liste pour insérer le nouveau maillon.
Voici l'état de la liste au fur et à mesure des étapes :
→3→NULL
→3→2→NULL
→3→2→1→NULL
→3→2→1→0→NULL

Corrigé
Pour un insérer un maillon par le bas, il faut réaliser la séquence suivante :

- Créer un nouveau maillon,

- Se déplacer en fin de liste,
- Insérer le maillon.

 Encore une fois, le danger principal est de travailler sur la liste principale et d'utiliser des instructions du type *Liste=Liste->suivant* ; pour se décaler en fin de liste. En effet, on finira bien par avoir un *Liste->suivant* qui pointe sur *NULL* et l'on sera alors bien en fin de liste… Le problème est le suivant : *Liste* pointera sur le dernier maillon !
Vous aurez donc perdu votre liste car vous n'aurez aucun pointeur pointant sur le premier élément de la liste !! Il faut donc penser à dupliquer le pointeur pointant en tête de liste et travailler sur cette copie. Il faudra remonter cette copie en tête de liste après l'insertion. La séquence est alors un peu plus longue:

- Dupliquer la liste
- Créer un nouveau maillon,
- Se déplacer en fin de liste avec le pointeur de travail,
- Insérer le maillon,
- Remonter le pointeur de travail en début de liste

 Voyons le code source complet :

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

typedef struct Maillon Maillon;
struct Maillon {
int valeur;
Maillon *suivant; };

int main() {
int i=0;
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *maListebis = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));

//création du premier maillon
element->valeur = 3;
element->suivant=NULL;

maListe = element;

// duplication de la liste
maListebis=maListe;

for (i=2; i>=0;i--) {

        //Création d'un nouveau maillon
```

```
        Maillon *element = malloc(sizeof(*element));
        element->valeur = i;
        element->suivant = NULL;

        //parcours de la liste
        while (maListebis->suivant != NULL)
                {
                maListebis = maListebis->suivant;
                }

        //branchement du maillon en fin de liste
        maListebis->suivant = element;

        //Remontée de maListebis en tête de liste
        maListebis=maListe;
        }

//affichage de la liste
while (maListebis != NULL)    {
        printf("%d -> ", maListebis->valeur);
        maListebis = maListebis->suivant;
        }
printf(" NULL");

}
```

Exemple de cours 4

Suppression d'un maillon en début de liste

On souhaite supprimer le maillon en début d'une liste.
Au départ on a la liste : →3→2→1→0→NULL
A l'arrivée on aura la liste : 2→1→0→NULL
Pour simplifier on n'utilisera aucune fonction, dans cet exemple.

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

typedef struct Maillon Maillon;

struct Maillon {
        int valeur;
        Maillon *suivant;
        };

int main() {
int i=0;
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *maListebis = malloc(sizeof(*maListe));
```

```
Maillon *element = malloc(sizeof(*element));

//création du maillon 1
element->valeur = 0;
element->suivant=NULL;

maListe = element;

//création de la liste
for (i=1; i<4;i++)  {
        Maillon *element = malloc(sizeof(*element));
        element->valeur = i;
        element->suivant = NULL;

        element->suivant = maListe;
        maListe = element;
        }

maListebis = maListe;

//affichage de la liste
while (maListebis != NULL)     {
        printf("%d -> ", maListebis->valeur);
        maListebis = maListebis->suivant;
        }
printf(" NULL");

//remontée de malistebis en tete de liste
maListebis = maListe;

//suppression du maillon de tete
maListe=maListe->suivant;
free(maListebis);

//remontée de malistebis en tête de liste
maListebis = maListe;

//affichage de la liste finale
while (maListebis != NULL)     {
        printf("%d -> ", maListebis->valeur);
        maListebis = maListebis->suivant;
        }

printf(" NULL\n ");  }

}
```

Utilisation de fonctions… parfois récursives !

Afin de ne pas ajouter de difficultés supplémentaires nous avons travaillé sans utiliser de fonctions ; cela a permis d'éviter d'avoir à gérer les prototypes, les entêtes de fonctions et les passages de paramètres. Néanmoins, il est évident que les listes chaînées doivent être gérées avec des fonctions. L'utilisation de la récursivité dans ces fonctions ajoutera un niveau de difficulté supplémentaire mais les programmes seront plus lisibles. Nous allons voir dans le cadre de ce cours puis lors des exercices associés des exemples de fonctions itératives ou récursives travaillant sur les listes chaînées.
Voici, par exemple, une liste de fonctions élémentaires travaillant sur les listes ;

Exemple de fonctions élémentaires sur les listes

- Afficher/parcourir la liste
- Vérifier si la liste est vide
- Recopier la liste
- Supprimer la liste
- Insertion d'un maillon en début de liste
- Insertion d'un maillon en fin de liste
- Insertion d'un maillon à un endroit donné dans la liste
- Suppression d'un maillon en début de liste
- Suppression d'un maillon en fin de liste
- Suppression d'un maillon à un endroit donné dans la liste
- Afficher un maillon de la liste…

A partir de l'exemple suivant, il est vivement conseillé de chercher le code source de tous les exemples ou exercices proposés et corrigés. Si vous suivez vraiment ce conseil, implémenter tous les exemples et exercices corrigés vous occupera une trentaine d'heures.

Exemple de cours 5

Utilisation d'une fonction d'affichage non récursive

On cherchera ici à reprendre l'exemple de cours N°2 et à gérer l'affichage de la liste à l'aide d'une fonction non récursive, dans un premier temps.

Corrigé

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

typedef struct Maillon Maillon;

struct Maillon {
        int valeur;
        Maillon *suivant;
        };
```

```
// Prototype de la fonction affiche
void Affiche (Maillon *);


int main() {
int i;
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));


element->valeur = 0;
element->suivant=NULL;

maListe = element;

for (i=1; i<4;i++)  {
        Maillon *element = malloc(sizeof(*element));
        element->valeur = i;
        element->suivant = NULL;

        element->suivant = maListe;
        maListe = element;
        }


Affiche(maListe);
}

void Affiche(Maillon *liste) {
if (liste == NULL)
        {
        printf("Liste vide");
        exit(EXIT_FAILURE);
        }

while (liste != NULL)    {
        printf("%d -> ", liste->valeur);
        liste = liste->suivant;
        }
printf(" NULL");

}
```


Exemple de cours 6

Utilisation d'une fonction d'affichage récursive
On cherchera ici à reprendre l'exemple de cours précédent et à gérer l'affichage de la liste à l'aide d'une fonction récursive, cette fois.

Corrigé

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

typedef struct Maillon Maillon;

struct Maillon {
        int valeur;
        Maillon *suivant;
        };

// Prototype de la fonction affiche
void Affiche (Maillon *);

int main() {
int i;
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));

element->valeur = 0;
element->suivant=NULL;

maListe = element;

for (i=1; i<4;i++)  {
        Maillon *element = malloc(sizeof(*element));
        element->valeur = i;
        element->suivant = NULL;

        element->suivant = maListe;
        maListe = element;
        }

Affiche(maListe);
}

void Affiche(Maillon *liste) {
if (liste == NULL)     {
        printf(" NULL");
else  {
        printf("%d -> ", liste->valeur);
        Affiche(liste->suivant);
       }
}
```


Exemple de cours 7

Insertion d'un maillon en début de liste avec une fonction

On souhaite avoir la liste suivante : 4→3→2→1→0→NULL.

Nous écrirons ici une fonction qui ajoute un maillon en début de liste. Au départ, on crée la liste 0→NULL puis on appellera quatre fois la fonction *Insere*.
Cette fonction va recevoir, à chaque fois, en argument, la liste initiale et la valeur du maillon à insérer.

Corrigé

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Maillon Maillon;
struct Maillon {
        int valeur;
        Maillon *suivant;
};

// Prototype des fonctions
Maillon * Insere (Maillon *, int);
void Affiche (Maillon *);

int main() {
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));
int i;

element->valeur = 0;
element->suivant=NULL;

maListe = element;

for (i=1;i<=4;i++)
        maListe = Insere (maListe,i);

Affiche(maListe);
}


Maillon * Insere (Maillon *liste, int Nb) {

Maillon *element = malloc(sizeof(*element));

if (liste == NULL || element == NULL)     {
        printf("Liste vide ou maillon vide");
        exit(EXIT_FAILURE);
}
element->valeur = Nb;
element->suivant = liste;
liste = element;

return liste;
}
```

```
void Affiche(Maillon *liste) {
if (liste == NULL)     {
        printf(" NULL");
}
else  {
        printf("%d -> ", liste->valeur);
        Affiche(liste->suivant);
        return;
}

}
```

Exemple de cours 8

Suppression du premier maillon d'une liste avec une fonction

On créera la liste suivante : 4→3→2→1→0→NULL.
Nous écrirons ensuite une fonction qui supprime le premier maillon de la liste : on aura donc à l'arrivée : 3→2→1→0→NULL. On libèrera la mémoire.

Corrigé

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Maillon Maillon;
struct Maillon {
        int valeur;
        Maillon *suivant;
};

// Prototype des fonctions
Maillon* Insere (Maillon *, int);
Maillon* SupprimeMaillon1(Maillon *);
void Affiche(Maillon *);


int main() {
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));
int i, nombre;
int val,valAinserer;

printf("Saisir un premier entier\n ");
scanf("%d", &nombre);

element->valeur = nombre;
element->suivant=NULL;
maListe = element;

for (i=1; i<=4; i++) {
```

```
        printf("Saisir un entier\n ");
        scanf("%d", &nombre);
        maListe = Insere (maListe,nombre);
}

printf("Avant suppression : \t");
Affiche(maListe);
maListe = SupprimeMaillon1(maListe);
printf("\nApres suppression : \t");
Affiche(maListe);
}

Maillon* Insere (Maillon *liste, int Nb) {
Maillon *element = malloc(sizeof(*element));
if (liste == NULL || element == NULL) {
        printf("Liste vide ou maillon vide\n ");
        exit(EXIT_FAILURE);
}
element->valeur = Nb;
element->suivant = liste;
liste = element;
return liste;
}

Maillon * SupprimeMaillon1 (Maillon *liste) {
Maillon *p = liste;

if (liste != NULL)  {
        p=liste->suivant;
        free(liste);
        return p;
}
else
        return NULL;
}


void Affiche(Maillon *liste) {
if (liste == NULL) {
    printf("NULL\n");
    return;
}

else  {
        printf("%d -> ", liste->valeur);
        Affiche(liste->suivant);
        return;
        }

}
```

Exemple de cours 9

Suppression du dernier maillon d'une liste avec une fonction

On créera la liste suivante : 4→3→2→1→0→NULL.

Nous écrirons ensuite une fonction qui supprime le dernier maillon de la liste : on aura donc à l'arrivée : 4→3→2→1→NULL.

Corrigé

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Maillon Maillon;
struct Maillon {
        int valeur;
        Maillon *suivant;
};

// Prototype des fonctions
Maillon* Insere (Maillon *, int);
Maillon* SupprimeMaillonLast(Maillon *);
void Affiche(Maillon *);


int main() {
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));
int i, nombre;
int val,valAinserer;

printf("Saisir un premier entier\n ");
scanf("%d", &nombre);

element->valeur = nombre;
element->suivant=NULL;
maListe = element;

for (i=1; i<=4; i++) {
        printf("Saisir un entier\n ");
        scanf("%d", &nombre);
        maListe = Insere (maListe,nombre);
}

printf("Avant suppression : \t");
Affiche(maListe);
maListe = SupprimeMaillonLast(maListe);
printf("\nApres suppression : \t");
Affiche(maListe);
}
```

```
Maillon* Insere (Maillon *liste, int Nb) {
Maillon *element = malloc(sizeof(*element));
if (liste == NULL || element == NULL) {
        printf("Liste vide ou maillon vide\n ");
        exit(EXIT_FAILURE);
}
element->valeur = Nb;
element->suivant = liste;
liste = element;
return liste;
}
```

```
Maillon * SupprimeMaillonLast (Maillon *liste) {
Maillon *p = liste;

if (p == NULL)
        return NULL;
if (p->suivant == NULL)
        return NULL;
else if (p->suivant->suivant == NULL) {
        p->suivant= NULL;
        free(p->suivant);
        return liste;
}
else
        SupprimeMaillonLast (p->suivant);
}

void Affiche(Maillon *liste) {
if (liste == NULL) {
        printf("NULL\n");
        return;
}

else  {
        printf("%d -> ", liste->valeur);
        Affiche(liste->suivant);
        return;
}

}
```

EXERCICES

Exercice 1

Énoncé

Créer la liste chainée suivante avec cinq nombres entiers : 0→2→4→6→8→NULL

Le programme principal affiche ensuite la liste chaînée dans ce même ordre à l'aide d'une fonction récursive.

Corrigé

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

typedef struct Maillon Maillon;

struct Maillon {
        int valeur;
        Maillon *suivant;
        };

// Prototype de la fonction affiche
void Affiche (Maillon *);

int main() {
int i;
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));

element->valeur = 8;
element->suivant=NULL;

maListe = element;

for (i=6; i>=0;i--)  {
        if (i % 2 ==0) {
                Maillon *element = malloc(sizeof(*element));
                element->valeur = i;
                element->suivant = NULL;

                element->suivant = maListe;
                maListe = element;
        }
}

Affiche(maListe);
}

void Affiche(Maillon *liste) {
if (liste == NULL)     {
        printf("NULL");
```

```
        }
else  {
        printf("%d -> ", liste->valeur);
        Affiche(liste->suivant);
      }
}
```

Exercice 2

Énoncé

Créer une liste chaînée avec cinq nombres entiers saisis au clavier.

Le programme principal appellera une fonction récursive qui affichera la liste chaînée.

Corrigé

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

typedef struct Maillon Maillon;
struct Maillon {
        int valeur;
        Maillon *suivant;
};

// Prototype des fonctions
Maillon * Insere (Maillon *, int);
void Affiche (Maillon *);

int main() {
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));
int i, nombre;

printf("Saisir un premier entier");
scanf("%d", &nombre);
element->valeur = nombre;
element->suivant=NULL;

maListe = element;

for (i=1;i<=4;i++) {
        printf("Saisir un entier");
        scanf("%d", &nombre);
        maListe = Insere (maListe,nombre);
}

Affiche(maListe);
}
```

```
Maillon * Insere (Maillon *liste, int Nb) {

Maillon *element = malloc(sizeof(*element));

if (liste == NULL || element == NULL)     {
        printf("Liste vide ou maillon vide");
        exit(EXIT_FAILURE);
}
element->valeur = Nb;
element->suivant = liste;
liste = element;

return liste;
}


void Affiche(Maillon *liste) {
if (liste == NULL)
        {
        printf("NULL");
        }
else  {
        printf("%d -> ", liste->valeur);
        Affiche(liste->suivant);
        return;
        }

}
```

Exercice 3

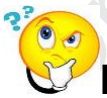
Énoncé

Créer une fonction récursive qui renvoie le nombre de maillons dans une liste.

Corrigé

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Maillon Maillon;
struct Maillon {
        int valeur;
        Maillon *suivant;
};

// Prototype des fonctions
Maillon * Insere (Maillon *, int);
int Compte (Maillon *);

int main() {
```

```
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));
int i;

element->valeur = 0;
element->suivant=NULL;

maListe = element;

for (i=1;i<=4;i++)
        maListe = Insere (maListe,i);

printf("Nombre de maillons : %d",Compte(maListe));
}


Maillon * Insere (Maillon *liste, int Nb) {

Maillon *element = malloc(sizeof(*element));

if (liste == NULL || element == NULL)     {
        printf("Liste vide ou maillon vide");
        exit(EXIT_FAILURE);
}
element->valeur = Nb;
element->suivant = liste;
liste = element;

return liste;
}

int Compte (Maillon *liste){
if (liste == NULL)
        return 0;
else
        return (1+Compte(liste->suivant));
}
```



Exercice 4

Énoncé

Créer une fonction récursive qui renvoie la plus grande valeur d'une liste.

Corrigé

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Maillon Maillon;
struct Maillon {
```

```
        int valeur;
        Maillon *suivant;
};

// Prototype des fonctions
Maillon * Insere (Maillon *, int);
int PlusGrand (Maillon *, int);

int main() {
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));
int i;

element->valeur = 0;
element->suivant=NULL;

maListe = element;

for (i=1;i<=4;i++)
        maListe = Insere (maListe,i);

printf("Plus grand maillon : %d",PlusGrand(maListe, maListe->valeur);
}


Maillon * Insere (Maillon *liste, int Nb) {

Maillon *element = malloc(sizeof(*element));

if (liste == NULL || element == NULL)     {
        printf("Liste vide ou maillon vide");
        exit(EXIT_FAILURE);
}
element->valeur = Nb;
element->suivant = liste;
liste = element;

return liste;
}

int PlusGrand (Maillon *liste, int max) {
if (liste == NULL) {
        printf("NULL");
        exit(EXIT_FAILURE);
        }
if (liste->suivant == NULL) {
        return max;
}

else if (liste->suivant->valeur > max)
```

```
        return (PlusGrand (liste->suivant , liste->suivant->valeur));
else
        return (PlusGrand (liste->suivant , max));
}
```

Exercice 5

Énoncé

Comment s'arrête cette fonction récursive ?

```
void affiche_liste (Maillon *liste)
{
   if (liste!=NULL)
   {
      printf("%d",liste->valeur);
      affiche_liste (liste->next);
   }
}
```

Corrigé

Il faut regarder l'appel récursif : si la liste n'est pas nulle on affiche la valeur du maillon puis on appelle la liste avec le maillon suivant. Quand on se trouve sur le dernier maillon on affiche sa valeur et on appelle *affiche_liste* avec le maillon suivant qui n'existe pas donc on envoie NULL à *affiche_liste*. Comme la liste n'est pas différente de *NULL* on ne "passe pas" dans le *if* et on s'arrête (plus d'appel récursif).

Exercice 6

Créer une fonction qui affiche une liste de 5 entiers saisis au clavier. Le programme principal va demander ensuite de saisir une valeur puis appelle une autre fonction récursive qui renvoie si la valeur est présente dans la liste.

Corrigé

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

typedef struct Maillon Maillon;
struct Maillon {
        int valeur;
        Maillon *suivant;
};

// Prototype des fonctions
Maillon* Insere (Maillon *, int);
Maillon* EstPresent (Maillon *, int );

int main() {
```

```
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));
int i, nombre, NbAchercher;

printf("Saisir un premier entier\n ");
scanf("%d", &nombre);
element->valeur = nombre;
element->suivant=NULL;
maListe = element;

for (i=1; i<=4; i++) {
printf("Saisir un entier\n ");
scanf("%d", &nombre);
maListe = Insere (maListe,nombre);
}

printf ("Quelle est la valeur a chercher ? \n ");
scanf ("%d", &NbAchercher);

//opérateur ternaire
(EstPresent (maListe, NbAchercher) != NULL ) ? printf("Valeur presente\n") :
printf("Valeur absente\n");
}


Maillon* Insere (Maillon *liste, int Nb) {
Maillon *element = malloc(sizeof(*element));

if (liste == NULL || element == NULL)     {
        printf("Liste vide ou maillon vide\n ");
        exit(EXIT_FAILURE);
        }
element->valeur = Nb;
element->suivant = liste;
liste = element;
return liste;
}


Maillon* EstPresent (Maillon *liste, int NbAchercher) {
if (liste==NULL)
        return NULL;
if (liste->valeur==NbAchercher)
        return liste;
else
        return EstPresent(liste->suivant, NbAchercher);
}
```

Exercice 7

Énoncé

Créer une fonction qui affiche une liste de 5 entiers saisis au clavier puis appelle une autre fonction qui va détruire la liste (la liste devient vide) tout en libérant la mémoire qu'elle occupait.

Corrigé

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Maillon Maillon;
struct Maillon {
int valeur;
Maillon *suivant;
};

// Prototype des fonctions
Maillon* Insere (Maillon *, int);
void Libere (Maillon *);
void Affiche(Maillon *);

int main() {
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));
int i, nombre, NbAchercher;

printf("Saisir un premier entier\n ");
scanf("%d", &nombre);

element->valeur = nombre;
element->suivant=NULL;
maListe = element;

for (i=1; i<=4; i++) {
        printf("Saisir un entier\n ");
        scanf("%d", &nombre);
        maListe = Insere (maListe,nombre);
}

Affiche(maListe);
Libere (maListe);
}


Maillon* Insere (Maillon *liste, int Nb) {
Maillon *element = malloc(sizeof(*element));
if (liste == NULL || element == NULL) {
        printf("Liste vide ou maillon vide\n ");
        exit(EXIT_FAILURE);
```

```
}
element->valeur = Nb;
element->suivant = liste;
liste = element;
return liste;
}

void Libere (Maillon *liste) {
Maillon *p = liste;
if (liste != NULL)     {

        if (liste->suivant != NULL)
        {
                liste = liste -> suivant;
                free(p);
                Libere (liste);
        }
        else
                free(p);
}

}

void Affiche(Maillon *liste) {
if (liste == NULL)  {
        printf("NULL");
        return;
}
else  {
        printf("%d -> ", liste->valeur);
        Affiche(liste->suivant);
        return;
}
}
```

Exercice 8

Énoncé

On souhaite gérer une production d'articles de sport. Chaque article comporte un identifiant (code alphanumérique à 4 caractères, un libellé à 10 caractères, un taux de TVA (réel), un prix hors taxe (réel)).

Créer une liste chaînée avec *n* articles (on demande combien d'articles sont souhaités). Les *n* articles seront stockés par une fonction récursive. Les caractéristiques de ces *n* articles seront affichées par une fonction récursive appelée par le programme principal.

Corrigé

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

```
typedef struct Article Article;
struct Article {
        char Code[5];
        char Libelle[11];
        float TauxTVA;
        float PrixHT;
};

typedef struct Maillon Maillon;
struct Maillon {
        Article valeur;
        Maillon *suivant;
};

// Prototype des fonctions
Maillon* Insere (Maillon *, int);
void Affiche(Maillon *);


int main() {
Maillon *maListe = malloc(sizeof(*maListe));
maListe=NULL;
int NbVal;

printf("Combien d'articles ? \n");
scanf("%d",&NbVal);

maListe = Insere (maListe,NbVal);

printf("Affichage des articles de la liste : \n");
Affiche(maListe);
}

Maillon* Insere (Maillon *liste, int Nb) {
Maillon *element = malloc(sizeof(*element));
element->suivant = NULL;

if (Nb == 0) {
        return liste;
}
if (liste == NULL) {
        printf("Quel est le code ? \n");
        scanf("%s",&element->valeur.Code);
        printf("Quel est le libelle ? \n");
        scanf("%s",&element->valeur.Libelle);
        printf("Quel est le taux de TVA ? \n");
        scanf("%f",&element->valeur.TauxTVA);
        printf("Quel est le prix HT ? \n");
        scanf("%f",&element->valeur.PrixHT);
```

```
        element->suivant = NULL;
        liste = element;
        liste = Insere (liste,--Nb);
        return liste;
}
else
{
        printf("Quel est le code ? \n");
        scanf("%s",&element->valeur.Code);
        printf("Quel est le libelle ? \n");
        scanf("%s",&element->valeur.Libelle);
        printf("Quel est le taux de TVA ? \n");
        scanf("%f",&element->valeur.TauxTVA);
        printf("Quel est le prix HT ? \n");
        scanf("%f",&element->valeur.PrixHT);
        element->suivant = liste;
        liste = element;
        liste = Insere (liste,--Nb); return liste ;
}
}


void Affiche(Maillon *liste) {
if (liste == NULL) {
        printf("Fin de liste\n");  }
else   {
        printf("\n\nARTICLE : \n");
        printf("%s  \n", liste->valeur.Code);
        printf("%s  \n", liste->valeur.Libelle);
        printf("%.2f \n", liste->valeur.TauxTVA);
        printf("%.2f  \n", liste->valeur.PrixHT);
        Affiche(liste->suivant);
        return;
        }
}
```

Exercice 9

Énoncé

En prenant l'exemple de dessus, créer deux listes chaînées ayant *n* articles puis écrire une fonction qui ajoute la deuxième liste à la fin de la première. Afficher la nouvelle liste récursivement.

Corrigé

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

typedef struct Article Article;
struct Article {
```

```
        char Code[5];
        char Libelle[11];
        float TauxTVA;
        float PrixHT;
};

typedef struct Maillon Maillon;
struct Maillon {
        Article valeur;
        Maillon *suivant;
};

// Prototype des fonctions
Maillon* Insere (Maillon *, int);
void Affiche(Maillon *);


int main() {
Maillon *maListe1 = malloc(sizeof(*maListe1));
maListe1=NULL;
Maillon *maListe2 = malloc(sizeof(*maListe2));
maListe2=NULL;
Maillon *Pcopie = malloc(sizeof(*Pcopie));
Pcopie = NULL;
int NbVal;

//Saisie de la liste 1
printf("Combien d'articles dans la liste 1 ? \n");
scanf("%d",&NbVal);

maListe1 = Insere (maListe1,NbVal);

printf("\n Affichage des articles de la liste 1: \n");
Affiche(maListe1);

//Saisie de la liste 2
printf("Combien d'articles dans la liste 2 ? \n");
scanf("%d",&NbVal);

maListe2 = Insere (maListe2,NbVal);

printf("\n Affichage des articles de la liste 2: \n");
Affiche(maListe2);

//Concaténation des listes
Pcopie = maListe1;
while (Pcopie->suivant !=NULL)
        {
        Pcopie = Pcopie->suivant;
        }
```

```
Pcopie->suivant = maListe2;

//Affichage des listes concaténées
printf("\n\n Affichage des listes concatenees \n");

Affiche(maListe1);
}



Maillon* Insere (Maillon *liste, int Nb) {
Maillon *element = malloc(sizeof(*element));
element->suivant = NULL;

if (Nb == 0) {
        return liste;
}
if (liste == NULL) {
        printf("Quel est le code ? \n");
        scanf("%s",&element->valeur.Code);
        printf("Quel est le libelle ? \n");
        scanf("%s",&element->valeur.Libelle);
        printf("Quel est le taux de TVA ? \n");
        scanf("%f",&element->valeur.TauxTVA);
        printf("Quel est le prix HT ? \n");
        scanf("%f",&element->valeur.PrixHT);
        element->suivant = NULL;
        liste = element;
        liste = Insere (liste,--Nb);
        return liste;
}
else
{
        printf("Quel est le code ? \n");
        scanf("%s",&element->valeur.Code);
        printf("Quel est le libelle ? \n");
        scanf("%s",&element->valeur.Libelle);
        printf("Quel est le taux de TVA ? \n");
        scanf("%f",&element->valeur.TauxTVA);
        printf("Quel est le prix HT ? \n");
        scanf("%f",&element->valeur.PrixHT);
        element->suivant = liste;
        liste = element;
        liste = Insere (liste,--Nb); return liste ;
}
}


void Affiche(Maillon *liste) {
```

```
if (liste == NULL) {
        printf("Fin de liste\n");  }
else  {
        printf("\n\nARTICLE : \n");
        printf("%s  \n", liste->valeur.Code);
        printf("%s  \n", liste->valeur.Libelle);
        printf("%.2f \n", liste->valeur.TauxTVA);
        printf("%.2f  \n", liste->valeur.PrixHT);
        Affiche(liste->suivant);
        return;
}
}
```

Exercice 10

Énoncé

Créer une liste chaînée circulaire du type :

34.5 → 567.78 → 890.46 → 3423.11

Le programme principal appelle ensuite une fonction récursive *Affiche*.
Conséquence ?

Corrigé

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

typedef struct Maillon Maillon;
struct Maillon {
        float valeur;
        Maillon *suivant;
};

// Prototype des fonctions
Maillon * Insere (Maillon *, float);
void Affiche (Maillon *);

int main() {
Maillon *maListe = malloc(sizeof(*maListe));
Maillon *element = malloc(sizeof(*element));
Maillon *Pcopie = malloc(sizeof(*Pcopie));
Pcopie = NULL;

element->valeur = 3423.11;
element->suivant=NULL;

maListe = element;
```

```
maListe = Insere (maListe, 890.46);
maListe = Insere (maListe, 567.78);

maListe = Insere (maListe, 34.5);

//Liste devient circulaire
Pcopie = maListe;
while (Pcopie->suivant !=NULL)
        Pcopie= Pcopie->suivant;

Pcopie->suivant=maListe;

Affiche(maListe);
}



Maillon * Insere (Maillon *liste, float Nb) {

Maillon *element = malloc(sizeof(*element));

if (liste == NULL || element == NULL)     {
        printf("Liste vide ou maillon vide");
        exit(EXIT_FAILURE);
}
element->valeur = Nb;
element->suivant = liste;
liste = element;

return liste;
}


void Affiche(Maillon *liste) {
if (liste == NULL)
        {
        printf("NULL");
        }
else  {
        printf("%.2f -> ", liste->valeur);
        Affiche(liste->suivant);
        return;
        }

}
```

Le programme va multiplier les appels récursifs à la fonction *Affiche* puisque le "dernier élément" est chaîné au premier pour s'arrêter (plantage) quand la capacité mémoire sera dépassée. Vous pouvez modifier le programme pour afficher le nombre d'appels avant plantage.

(Après un test, 174 700 appels ont été réalisés, 261 000 sur les machines de l'établissement en seulement…5,4 secondes !).
Il suffit de modifier *Affiche* comme suit :

```
void Affiche(Maillon *liste, int i) {
if (liste == NULL)  {
        printf("NULL");
}
else  {
        printf("%.2f -> ", liste->valeur);
        printf("\n%d\n",i);
        Affiche(liste->suivant,++i);
        return;
}
```

Pensez à modifier également le prototype de la fonction *Affiche* qui reçoit, désormais, 2 paramètres : la liste et *i* qui comptera le nombre d'appels.

Exercice 11

Énoncé

Créer une fonction récursive qui renvoie la moyenne des valeurs entière d'une liste.

Corrigé

#

Exercice 12

Énoncé

Créer une fonction qui ajoute un maillon en milieu de liste (ni au début, ni à la fin).
On précisera par saisie au clavier après quelle valeur présente dans un maillon on souhaite ajouter le nouveau maillon (on suppose que la valeur existe).
Le programme principal utilisera la fonction qui ajoute un maillon puis la fonction qui affiche la liste.

Corrigé

#

Exercice 13

Énoncé

Créer une fonction qui supprime un maillon au milieu d'une liste (ni au début, ni à la fin).
On précisera par saisie au clavier quel numéro de maillon doit être supprimé.
Le programme principal utilisera une fonction qui supprime un maillon puis une fonction qui affiche la nouvelle liste.

Corrigé

#

Exercice 14

Énoncé

Écrire un programme qui demande combien d'entiers on souhaite stocker et qui crée un tableau avec autant de nombres entiers, générés aléatoirement.
Écrire ensuite la fonction qui recopie ces nombres, dans le même ordre, dans une liste chaînée ainsi que la fonction qui permettra d'afficher cette liste.

Corrigé

#

Exercice 15

Énoncé

Écrire un programme qui demande combien d'entiers on souhaite stocker et qui crée une liste chaînée avec autant de nombres entiers générés aléatoirement.
Écrire ensuite la fonction qui recopie ces nombres, dans le même ordre, dans un tableau dynamique. Le contenu du tableau est ensuite affiché par une fonction récursive.

Corrigé

#

Exercice 16

Énoncé

Écrire un programme qui demande combien d'entiers on souhaite stocker et qui crée un fichier texte avec autant de nombres entiers générés aléatoirement.
Écrire ensuite la fonction qui remonte ces entiers, à partir du fichier texte, dans une liste chaînée ainsi que la fonction récursive qui permettra d'afficher cette liste.

Corrigé

#

Exercice 17

Énoncé

Écrire un programme qui demande combien d'entiers on souhaite stocker et qui crée une liste chaînée avec autant de nombres entiers générés aléatoirement.
Écrire ensuite la fonction qui enregistre les nombres présents dans cette liste chaînée, dans un fichier texte.

Corrigé

#

Exercice 18 Hors programme 2016

Énoncé

Créer une liste doublement chaînée avec 6 nombres réels.
Écrire la fonction récursive Affiche1 qui affiche les nombres du premier au dernier.
Écrire la fonction récursive Affiche2 qui affiche les nombres du dernier au premier.

Corrigé

#