SCIENCES ET MONTAGNE
UNIVERSITÉ SAVOIE MONT BLANC

UNIVERSITÉ SAVOIE MONT BLANC

INFO 505 - Programmation C II
L3 – 2024-25

CM3 : Structures de données et gestion mémoire dynamique, listes chainées

J.Y. RAMEL

Bureau 2D-204 - Polytech'Savoie - Bourget du Lac

Structure de données : struct

Rappel : tableau -> collection d'objets de même type

- Introduction d'un nouveau type de données permettant de s'affranchir des limites des tableaux : considérer des types de données différents, regroupés et accessibles de façon unifiée

Structure : définition

- Type de donnée, contenant un ou plusieurs objets pouvant être de type différent, regroupées au sein d'une même entité.

- Les objets contenus dans la structure sont désignés par champs (de la structure)

Illustration : Gestion d'un agenda → Un tableau dynamique de fiches Evenements

Structure de données : struct

Structure : définition

```
struct Nom_Structure {
        type_champ1 Id_Champ1;
        type_champ2 Id_Champ2;
        type_champ3 Id_Champ3;
        type_champ4 Id_Champ4;
        type_champ5 Id_Champ5;
};
```

```
struct coord3D
{
   double x ;
   double y ;
   double z ;
};
```

```
struct personne
{
   char nom[100];
   char prenom[100];
   char adresse[1000];

   int age;
   int sexe;
};
```

Nom_Structure et Nom_Champ1 :

- identifiants
- soumis aux mêmes règles syntaxiques que n'importe quel identifiant

Structure de données : struct

Structure : déclaration de variables

```
struct coord3D
{
    double x ;
    double y ;
    double z ;
};
struct Coord3D point1, point2 ;
```

```
struct personne
{
    char nom[100];
    char prenom[100];
    char adresse[1000];
    …
    int age;
    int sexe;
};
struct personne Paul;
```

Structure de données : struct

Structure : accès à un champ

- opérateur d'indirection

```
struct coord3D
{
   double x ;
   double y ;
   double z ;
};
struct Coord3D point1, point2 ;

point1.x = 0.0;
point2.x = point1.x;
```

```
struct personne
{
   char nom[100];
   char prenom[100];
   char adresse[1000];

   int age;
   int sexe;
};
struct personne Paul;

printf (''Age : %d\n'', Paul.age);
```

Structure de données : struct

Structure : cas des structures imbriquées

- Le champ d'une structure peut aussi être une structure

1

```
struct horloge
{
    int h ;
    int min ;
    int sec ;
};
```

2

```
struct date
{
    struct horloge heure;
    int jour ;
    char jour_semaine[10];
    int mois ;
    int annee ;
};
```

3

```
struct date date_evenement;
```

Attention à l'ordre de définition des structures

On se reveille…

- Structure ok ?
- Structures imbriquées ?
- Pointeurs de structures ?

```
struct date evenement;

struct date *pdate = (* date) malloc(…
```

```
struct horloge
{
    int heure ;
    int min ;
    int sec ;
};
```

```
struct date
{
    struct horloge temps;
    int jour ;
    char jour_semaine[10];
    int mois ;
    int annee ;
};
```

```
/* 15 septembre 2023, 10h20:30 */
evenement.temps.heure = 10;
evenement.temps.min = 20;
evenement.temps.sec = 30;

evenement.jour = 15;
evenement.jour_semaine[0] = 'J';
evenement.mois = 9;
evenement.annee = 2023;

pdate->année = 2024;
```

Structure de données : struct

Structure : cas des structures imbriquées

```
struct horloge
{
    int heure ;
    int min ;
    int sec ;
};
```

```
struct date
{
    struct horloge temps;
    int jour ;
    char jour_semaine[10];
    int mois ;
    int annee ;
};
```

Initialisation des champs d'une structure lors de la définition

```
/* 15 septembre 2023, 10h20:30 */
struct date date_evenement = {{10, 20, 30}, 15, "",9, 2023};
```

Structure de données : struct

Structure : tableau 1D et structure

Les éléments d'un tableau peuvent aussi être des structures

Exemple : tableau de coordonnées 3D de points

1

```
struct coord3D
{
    double x ;
    double y ;
    double z ;
};
```

2

```
struct   coord3D   t_coord3D[3] = {{0.0, 0.0, 0.0},
                                   {0.0, 0.0, 0.0},
                                   {0.0, 0.0, 0.0}
                                  };
```

3

```
t_coord3D[0].x = 1.0;
```

Accès au champ x de l'élément d'indice 0 du tableau t_coord3D

```
t_coord3D[1] = t_coord3D[0];
```

Copier l'élément d'indice 0 du tableau t_coord3D dans l'élément d'indice 1

Structure de données : struct

Structure : pointeur et structure

Structure : type de donnée à part entière
Possibilité de manipuler des variables de type « struct » par pointeur

Accès aux champs d'une structure en utilisant un pointeur :
Opérateur d'indirection : ->

1

```
struct coord3D
{
    double x ;
    double y ;
    double z ;
};
```

2

```
struct coord3D point1, *pt_coord3D;
```

point1 : structure
*pt_coord3D : pointeur

3

```
point1.x = 1.5;
pt_coord3D = &point1;
pt_coord3D -> y = 2.3; ou (* pt_coord3D).y = 2.3
```

Structure de données : struct

Structure : définition de type → typedef

typedef : permet de redéfinir un type déjà existant → **notion d'alias**

Structure : type de donnée à part entière
Objectifs :

« Faciliter » l'écriture du code source
Accroitre la lisibilité du code(? à discuter)

1

```
typedef struct
{
    double x ;
    double y ;
    double z ;
} coord3D;
```

2

```
coord3D   s1, s2, s3 = {0.0, 1.0, 2.0};
```

3

```
s1.x = 0.0;
```

Structure de données : struct

Structure : définition de type -> typedef

typedef : permet de redéfinir un type déjà existant pour en faire un type de donnée à part entière

Cas des structures auto référencées
le champ d'une structure est la structure elle-même.

1

```
typedef struct _individu
{
    char nom[20] ;
    char prenom[20] ;
    struct _individu pere ;
    struct _individu *mere;
} individu;
```

Typedef et _ pour + de clarté du code

2

```
individu personne_1 = {''Dupont'', ''Pascal''}
```

Utilisation aussi lors de la manipulation des listes chaînées par exemple.

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

// Déclaration d'une structure récursive pour une liste chaînée
struct Node {
    int data;
    struct Node* next; // Pointeur vers le prochain élément de type struct Node
};

int main() {
    // Allocation de mémoire pour le premier nœud
    struct Node* head = (struct Node*)malloc(sizeof(struct Node));
    head->data = 1;
    head->next = NULL;

    // Allocation de mémoire pour le deuxième nœud
    struct Node* second = (struct Node*)malloc(sizeof(struct Node));
    second->data = 2;
    second->next = NULL;

    // Relier le premier nœud au deuxième
    head->next = second;

    // Accéder aux données
    printf("First node data: %d\n", head->data);
    printf("Second node data: %d\n", head->next->data);

    // Libérer la mémoire
    free(head);
    free(second);
```

Structure de données : struct

Structure et fonction

```
struct coord3D
{
    double x ;
    double y ;
    double z ;
};
coord3D s1, s2, s3;
```

Possibilité de passer en paramètre d'une fonction :

- Une structure
- Le champ d'une structure

aussi bien par valeur que par adresse.

La valeur retournée par une fonction peut aussi être une structure.

Prototype de fonctions

```
void fct1 (struct coord3D p1, struct coord3D p2, struct coord3D p3);
void fct2 (struct coord3D *p1);
struct coord3D fct3 (void);
```

Appel fonction

```
fct1 (s1, s2, s3);
fct2 (&s1);
s3 = fct3 ();
```

Illustration : Gestion d'un agenda

Un tableau de structures « fiche »

Type_struct_fonction\main.c

```c
/**
 * @brief Illustration manipulation type struct et paramètre
 *   de fonction
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

struct agenda {
    uint8_t jour;
    uint8_t mois;
    uint16_t annee;
    uint8_t heure;
    uint8_t minute;
    char detail[60];
};

void affiche_contenu_agenda (struct agenda fiche);

void affiche_contenu_agenda (struct agenda fiche){
    printf ("Rdv le %u.%u.%u pour %uh%u -> %s\n", fiche.jour, fiche.mois, fiche.annee,
                                                  fiche.heure, fiche.minute, fiche.detail);
}

int main(void)
{
    struct agenda rdv1 = {1, 9, 2024, 10, 30, "Convocation"}, rdv2;
    rdv2.jour = 2; rdv2.mois = 10; rdv2.annee = 2025;
    rdv2.heure = 9; rdv2.minute = 20; strcpy(rdv2.detail, "Prochaine convocation");

    affiche_contenu_agenda (rdv1);
    affiche_contenu_agenda (rdv2);
    return EXIT_SUCCESS;
}
```

```
d:\Documents\ProgrammationLangageC\build-Debug\bin\Type_struct_fonction.exe
Rdv le 1.9.2024 pour 10h30 -> Convocation
Rdv le 2.10.2025 pour 9h20 -> Prochaine convocation

==== Program exited with exit code: 0 ====
Time elapsed: 000:00.047 (MM:SS.MS)
Press any key to continue...
```

Structure de données : struct

Occupation mémoire d'une structure

L'occupation mémoire d'une structure correspond à la somme des occupations mémoire de chaque champ.

Mais : la gestion de la mémoire peut introduire des mécanismes d'alignement mémoire.

Pour obtenir le nombre d'octets nécessaire au stockage d'une structure :

opérateur sizeof

```
struct coord3D
{
   double x ;
   double y ;
   double z ;
};
coord3D s1, s2, s3;

sizeof (coord3D);
sizeof (s1);
```

Structure de données : struct

Occupation mémoire d'une structure

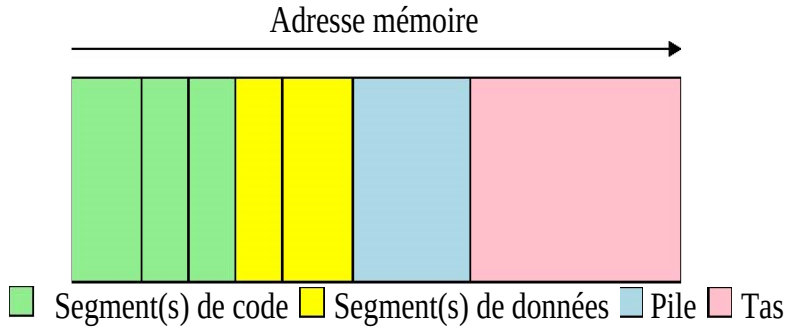
Espace_Mem_struct\main.c

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

struct agenda {
    uint8_t jour;
    uint8_t mois;
    uint16_t annee;
    uint8_t heure;
    uint8_t minute;
    char detail[35];
};

int main(void)
{
    struct agenda rdv = {6, 6, 2024, 17, 22, "DDay commeration"};
    size_t size;

    printf ("1. Nb octets : %llu\n", sizeof (rdv));
    printf ("2. Nb octets : %llu\n", sizeof (struct agenda));
    size = sizeof (rdv.jour) + sizeof (rdv.mois) + sizeof (rdv.annee)
         +sizeof (rdv.heure) + sizeof (rdv.minute) + sizeof (rdv.detail);
    printf ("3. Nb octets : %llu\n", size);

    return EXIT_SUCCESS;
}
```

```
d:\Documents\ProgrammationLangageC\build-Debug\bin\Espace_Mem_struct.exe
1. Nb octets : 42
2. Nb octets : 42
3. Nb octets : 41

==== Program exited with exit code: 0 ====
Time elapsed: 000:00.031 (MM:SS.MS)
Press any key to continue...
```

Allocation mémoire statique

- Segment de données = data segment
  - Zone contigüe de la mémoire
    - Zone mémoire réservée dans le code source
    - Contient les variables globales



- Pile = stack
  - Zone contigüe de la mémoire
    - Contient les variables locales
    - Emplacement de stockage temporaire des paramètres de fonctions lors de leur appel.

Allocation statique (data segment) **:**

- Associée à des variables dont la réservation s'effectue au moment de la compilation.
- La réservation mémoire est valide durant toute l'exécution du programme. Le volume de mémoire allouée ne peut être modifié.

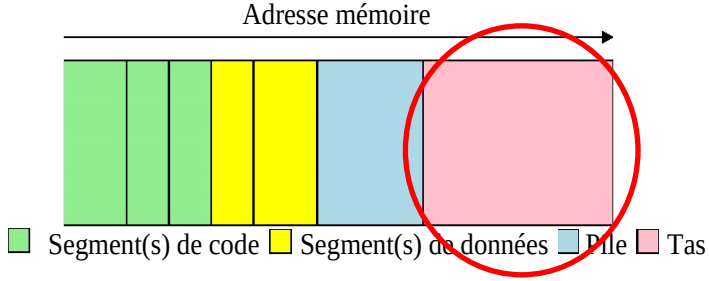
```
int tab[15]; // Tableau de 15 entiers de type int
```

Allocation automatique (pile / stack) :

- cas des variables locales et des paramètres de fonction. Durée de vie limitée à l'exécution d'un bloc d'instructions

Allocation mémoire dynamique

L'allocation dynamique est associée à des variables dont la création s'effectue en cours d'exécution d'un programme.



Utilisation du tas = heap

- Stockage de variables en mémoire dans le tas;
- Création des variables en cours d'exécution du programme
- Restitution / libération de zones mémoires précédemment allouées ⇒ perte de l'accès aux données
- Permettre de gérer des structures de données dont le nombre d'éléments n'est pas connu a priori (arbres, listes,…)

Fonctions de base de gestion de l'allocation dynamique :
- *malloc* : pour allouer dans le tas (heap) une zone mémoire composée de n octets contigus
- *free* : pour libérer une zone mémoire précédemment allouée avec malloc

Autres fonctions :
- calloc
- realloc

Nécessite d'inclure le fichier *stdlib.h*

Allocation mémoire dynamique

malloc : pour allouer dans le tas (heap) une zone mémoire composée de n octets contigus

void *malloc(size_t taille);

taille : nombre d'octets réservés
Valeur retournée :
un pointeur sur le premier octet de la zone mémoire réservée
si échec, le pointeur NULL

Cast facultatif en C obligatoire en C++

Exemple : réserver un zone mémoire pour stocker 15 réels de type float

```
float *p_f = NULL;
p_f = (float *) malloc (15 * sizeof (float));
if (p_f == NULL) exit (-1);    // Pour s'assurer que l'allocation mémoire s'est effectuée avec succès
```

Allocation mémoire dynamique

free : pour libérer une zone mémoire précédemment allouée avec malloc

```
void free(void *ptr);
```

ptr : pointeur sur une zone mémoire crée avec malloc
Valeur retournée : aucune
Une fois la zone mémoire libérée mettre : ptr = NULL;

Exemple : zone mémoire pour stocker 15 réels de type float

```
float *p_f;
p_f = (double *) malloc (15 * sizeof (double));
if (p_f == NULL) exit (-1); // Pour s'assurer que l'allocation mémoire s'est effectuée avec succès
free (p_f);
p_f = NULL;
```

Allocation mémoire dynamique

calloc : pour allouer dans le tas (heap) une zone mémoire composée de *number* éléments dont la taille est de *size* octets

```
void * calloc( size_t number, size_t size );
```

Valeur retournée :

un pointeur sur le premier octet de la zone mémoire réservée
si échec, le pointeur NULL
Tous les octets de la zone allouée sont initialisés à 0

Exemple : zone mémoire pour stocker 15 réels de type float

```
float * p_f;
p_f = (float *) calloc (15, sizeof (float));
if (p_f == NULL) exit (-1);   // Pour s'assurer que l'allocation mémoire s'est effectuée avec succès
```

Allocation mémoire dynamique

realloc : pour changer la taille d'une zone déjà allouée

Expliquer ce code

```
for (int i = 0; i < nb_evenements; i++) {
    if (strcmp(agenda[i]->titre, titre) == 0) {
        free(agenda[i]);                                    // Libérer la mémoire de l'événement
        for (int j = i; j < nb_evenements -1 ;j++)
            agenda[j] = agenda[j + 1];                      // Décaler les événements

        nb_evenements -= 1;

        agenda = (Evenement**)realloc(agenda, nb_evenements * sizeof(Evenement*));

        printf("Événement supprimé avec succès.\n");
        return agenda;
    }
}
printf("Événement non trouvé.\n");
return agenda;
```

Illustration allocation dynamique : tableau 1D

Ecrire un programme en C permettant de :

- De créer un tableau pour stocker des valeurs réelles.
  → Dont le nombre de valeurs à stocker sera saisi au clavier

- Chaque élément du tableau sera initialisé avec une valeur aléatoire réelle dans [0.0; 1.0]

- De calculer la valeur moyenne des éléments du tableau et l'afficher

- De libérer la mémoire allouée en fin de traitement

Rappels
tab_double[0] = 1; ⇔ *tab_double = 1;
et
tab_double[i] ⇔ *(tab_double + i) = …

Illustration allocation dynamique : tableau 1D

Malloc_Tab1D\main.c

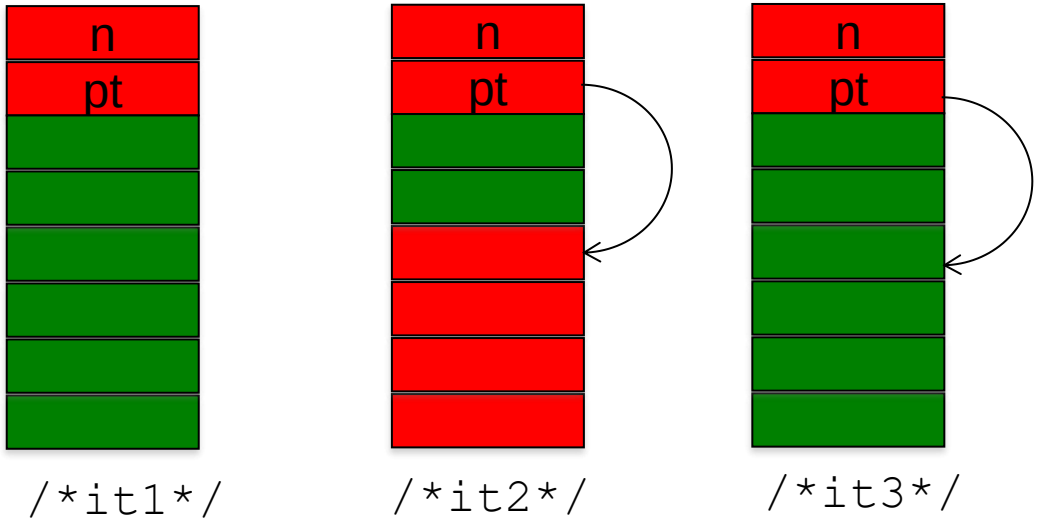
```c
/**
 * @brief Allocation dynamique tableau 1D
 * Utilisation du générateur de nombre pseudo aléatoire
 * Nombre pseudo aléatoire dans [0.0 - 1.0]
*/

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

int main(void)
{
    long int n = 0;
    double moyenne = 0.0;
    double * tab_double = NULL;

    printf("Nombre d'elements du tableau : "); scanf("%ld", &n);
    if (n> 0) {
        tab_double = (double *) malloc (n * sizeof(double));
        if (tab_double == NULL){
            printf ("Echec allocation dynamique\n");
            exit (-1);
        }
        srand(time(NULL)); // Initialisation de la séquence de nombres
        for (int i = 0; i < n; i++){
            tab_double[i] = rand() / (double)RAND_MAX; // <=> *(tab_double+i) = valeur dans [0.0 - 1.0]
        }
        for (int i = 0; i < n; i++){
            moyenne += tab_double[i];
        }
        moyenne /= n;
        printf ("Moyenne = %.3lf\n", moyenne);

        free(tab_double);

    }// Fin if

    return EXIT_SUCCESS;
} //Fin main()
```

```
d:\Documents\ProgrammationLangageC\build-Debug\bin\Malloc_Tab1D.exe
Nombre d'elements du tableau : 125
Moyenne = 0.502

==== Program exited with exit code: 0 ====
Time elapsed: 000:07.265 (MM:SS.MS)
Press any key to continue...
```

On se reveille…

Que manque t il ?

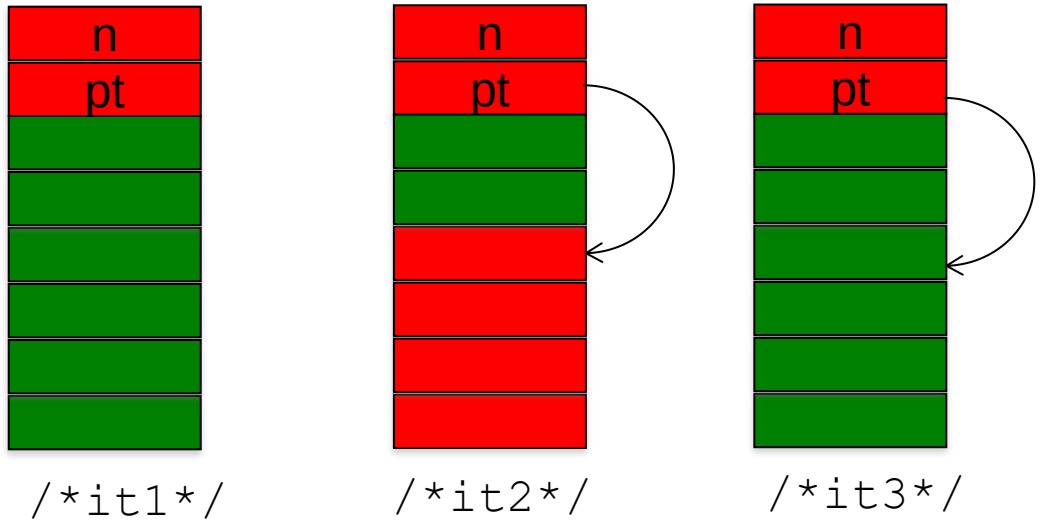
```
int n ;
double * pt;
scanf(" %d " , &n)              /*it1*/
pt = (double*) malloc(n*sizeof(double));
/*it2*/
…
free (pt);                      /*it3*/
…
```



Allocation mémoire dynamique

Illustration

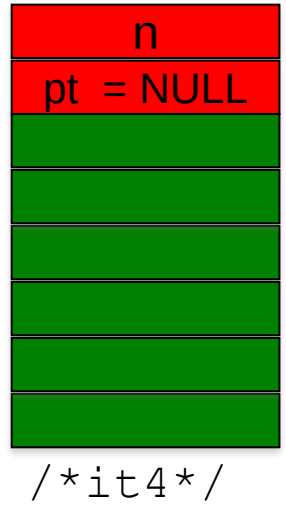
```
int n ;
double * pt;
scanf(" %d " , &n)                          /*it1*/
pt = (double*) malloc(n*sizeof(double));
/*it2*/
…
free (pt);                                  /*it3*/
…
```



Après free, pt ne vaut pas NULL et il indique pourtant une adresse inaccessible.

Faire toujours suivre un free par une mise à NULL du pointeur

```
free (pt); /*t4*/
pt =NULL;
```



On se reveille…

```
double *pt;
pt = (double*) malloc(5*sizeof(double));
pt = pt+1
free (pt);
```

On se reveille…

double *pt;

pt = (double*) malloc(5*sizeof(double));

pt = pt+1

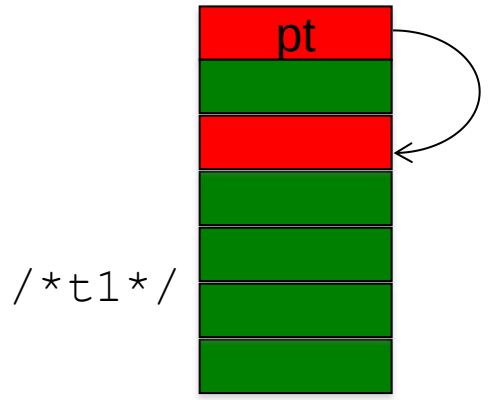
free (pt);

l'adresse donnée comme paramètre à la fonction free doit correspondre à une adresse renvoyée par un malloc précédent

```
double *pt;
pt = (double*) malloc(5*sizeof(double));
pt = pt+1
free (pt);    => ERREUR
```

On se reveille

```
double * pt;
pt = (double*) malloc(sizeof(double)); /*t1*/
pt = (double*) malloc(sizeof(double)); /*t2*/
pt = (double*) malloc(sizeof(double)); /*t3*/
```



Allocation mémoire dynamique

Il ne faut jamais "oublier" l'adresse renvoyée par un malloc, seul moyen d'atteindre les cases réservées => risque de saturation de la mémoire

```
double * pt;
pt = (double*) malloc(sizeof(double)); /*t1*/
pt = (double*) malloc(sizeof(double)); /*t2*/
pt = (double*) malloc(sizeof(double)); /*t3*/
```

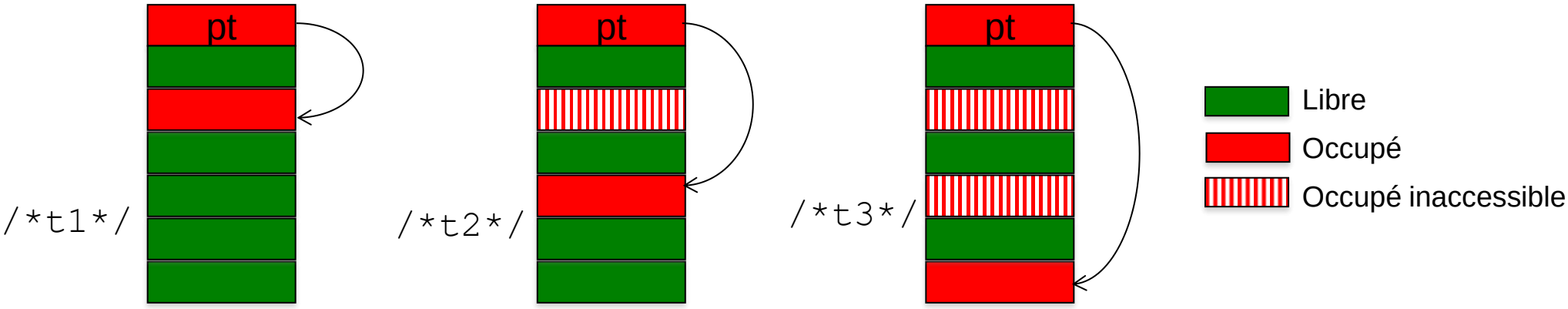
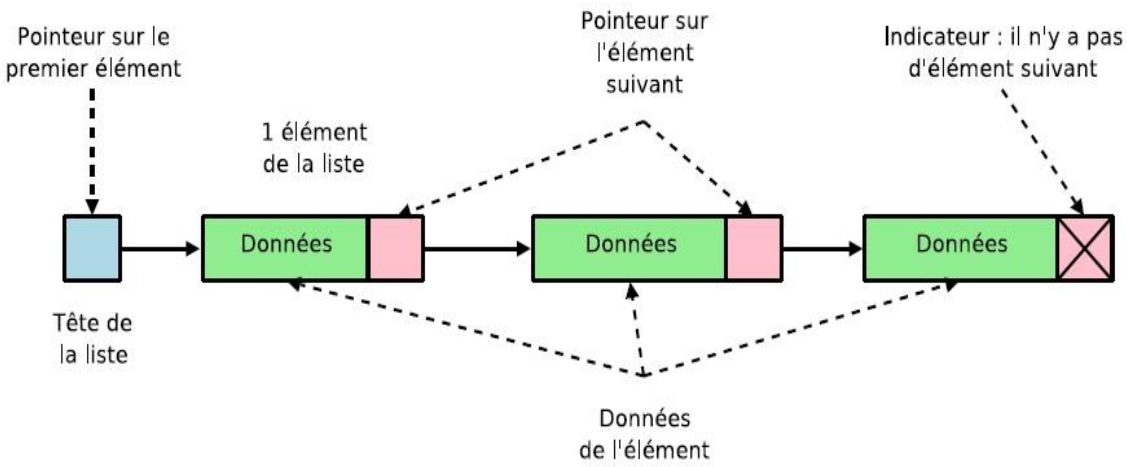


Illustration allocation dynamique : liste simplement chainée



- **Faite la structure elmt_liste**

- **Expliquez :**
  - **&tete ???**
  - **tete->next ???**
  - **&tete.next ???**

Rappels:

Liste vide : le pointeur de tête a la valeur NULL ;

Un élément de la liste est composé :

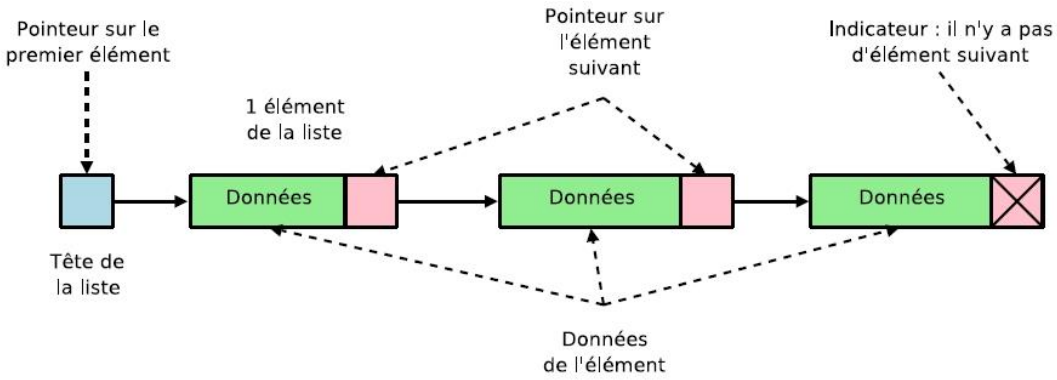
Un champ de données ;

Un pointeur vers l'élément suivant.

Champ de données : exemple
Nom : chaine de caractères
Prénom : chaine de caractères
Age : entier

Illustration allocation dynamique : liste simplement chainée

Définir un élément de la liste



Champ de données : exemple
Nom : chaine de caractères
Prénom : : chaine de caractères
Age : entier

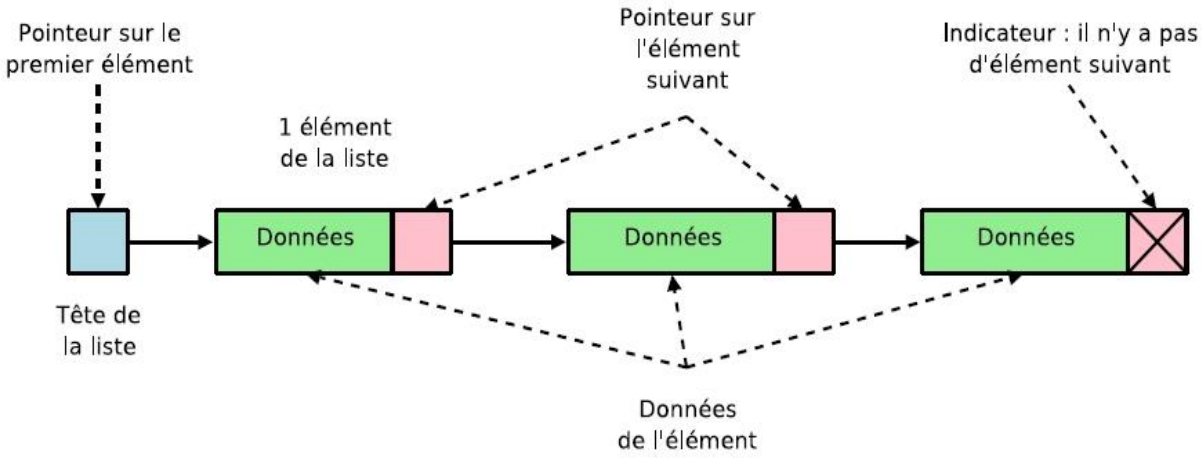
```
typedef struct _elmt_liste
{
    char nom[30];
    char prenom [30];
    int age;
    struct _elmt_liste * next;
} e_liste;
```

Pointeur vers l'élément suivant

Illustration allocation dynamique : liste simplement chainée

Initialisation de la liste

e_liste * tete = NULL;




```
typedef struct _elmt_liste
{
        char nom[30];
        char prenom [30];
        int age;
        struct _elmt_liste *next;
} e_liste;
```

Illustration allocation dynamique : liste simplement chainée

Opérations sur liste simplement chainée :

- Insertion d'un élément (en tête de liste)
- Suppression d'un élément (en tête de liste)
- Compter le nombre d'éléments de la liste
- Afficher les données d'un élément,…

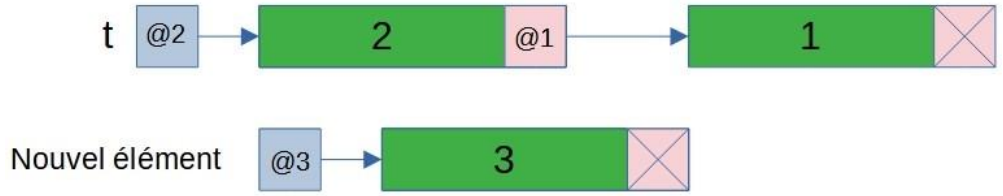
Illustration allocation dynamique : liste simplement chainée

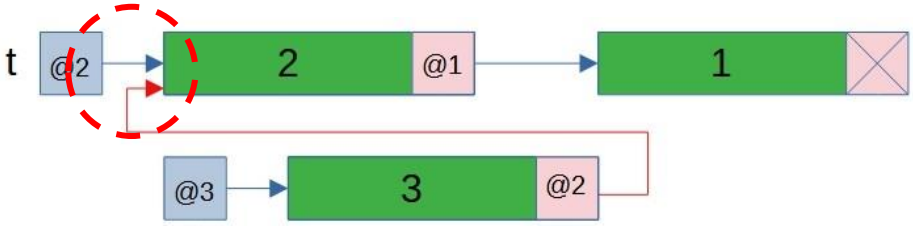
Insérer un élément en tête de liste

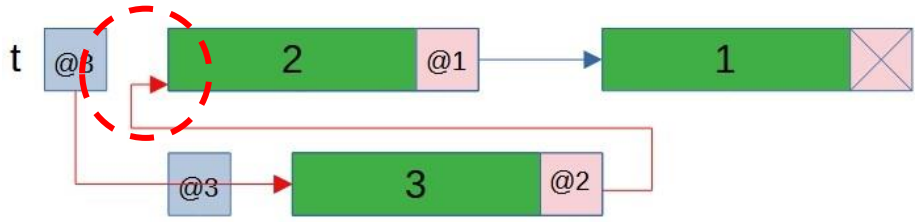
Créer le nouvel élément

Mettre à jour le chainage des pointeurs: 2 étapes

Etat initial →







Etat final →

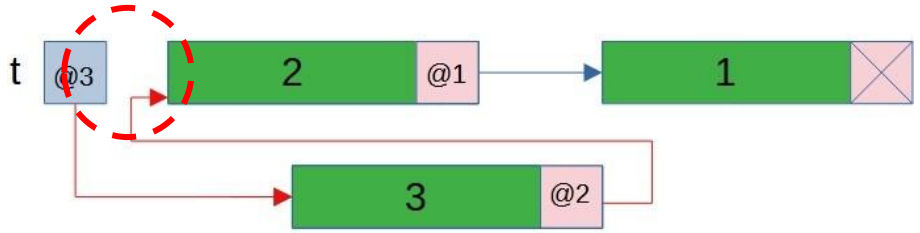


Illustration allocation dynamique : liste simplement chainée

Insérer un élément en tête de liste

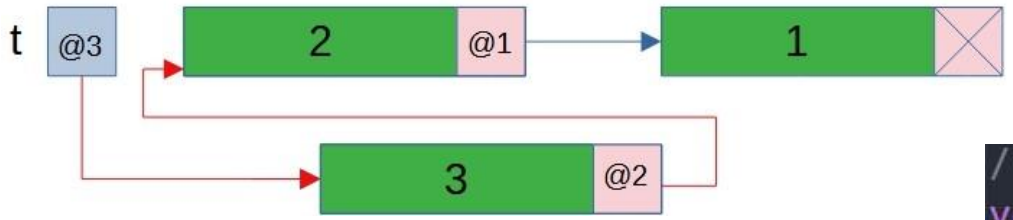
```c
//-------------------------------------------------------------
void insere_tete_liste (e_liste ** tete) {
    e_liste *tmp;

    // Créer le nouvel élément
    tmp = (e_liste *) malloc (sizeof (e_liste));
    if (tmp == NULL) {
        printf ("Echec allocation\n");
        return;
    } else {
        scanf("%s %s %d", tmp->nom, tmp-> prenom, &(tmp->age));
        // Insertion en tête de liste
        tmp->next = *tete;
        *tete = tmp;
    }
}
```

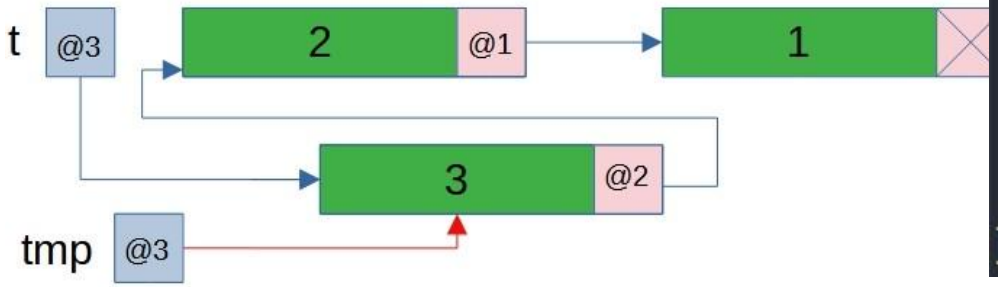
```c
/**
 * @brief Définition de la structure d
 * @struct _elemt_liste
 */
typedef struct _elemt_liste {
    char nom[30];
    char prenom[30];
    int age;
    struct _elemt_liste * next;
} e_liste;
```

```c
int main(void)
{
    e_liste * tete = NULL;

    insere_tete_liste (& tete); // Insérer un premier élément
    insere_tete_liste (& tete); // Insérer un second élément
    insere_tete_liste (& tete); // Insérer un troisième élément
    printf ("Nombre d'elements de la liste : %d\n", parcours_liste (tete));
    supprime_tete_liste (&tete); // Supprimer le premier élément
    printf ("Nombre d'elements de la liste : %d\n", parcours_liste (tete));
    Affiche_element_liste(tete);
    supprime_tete_liste (&tete); // Supprimer le premier élément
    supprime_tete_liste (&tete); // Supprimer le premier élément
    printf ("Nombre d'elements de la liste : %d\n", parcours_liste (tete));

    return EXIT_SUCCESS;
}
```

Illustration allocation dynamique : liste simplement chainée

Supprimer un élément en tête de liste






```c
//-----------------------------------------------
void supprime_tete_liste (e_liste ** tete){
    e_liste *tmp = NULL;

    if (*tete != NULL){
        tmp = *tete;
        *tete = (*tete)->next;
        free(tmp);
    }
}
```