Module : Structures de données et programmation C++

Élément 1: Structures de données

Filière : Génie Informatique, Semestre 2

Année Universitaire 2021-2022

Pr. Rachid AIT DAOUD

Caractéristiques du cours

❑ Objectifs:

- Maitriser les structures de données de base et le principe de fonctionnement de chacune.
- Apprendre à concevoir des structures de données pour offrir une meilleure résolution d'un problèmes.
- Être capable d'implanter ces structures dans le langage C .

❑ Pré-requis

- Algorithmique et programmation C

❑ En TD: Les bases et les exemples fondamentaux

- Tout ce qui est vu en TD doit être connu.

❑ En TP: Implantation des structures vues en cours

- Tout les TPs sont à finir et à rendre.

❑ Présentation des exposés

Plan du cours

Chapitre 1: Rappels

- Les pointeurs et la mémoire
- Mémoire et la déclaration des variables locales
- Passage par valeurs et passage par adresses
- Le mémoire dynamique
- Les tableaux dynamiques
- Les structures

Chapitre 2: Les listes

- Les listes chainées
- Manipulation des listes simplement chainées
- Manipulation des listes doublement chainées.

Chapitre 3: Les piles et les files

- Les piles
- Les files

Chapitre 4: Les arbres binaires

Chapitre 1: Rappels

1. Accès à la mémoire via les identificateurs des variables

Les règles principales

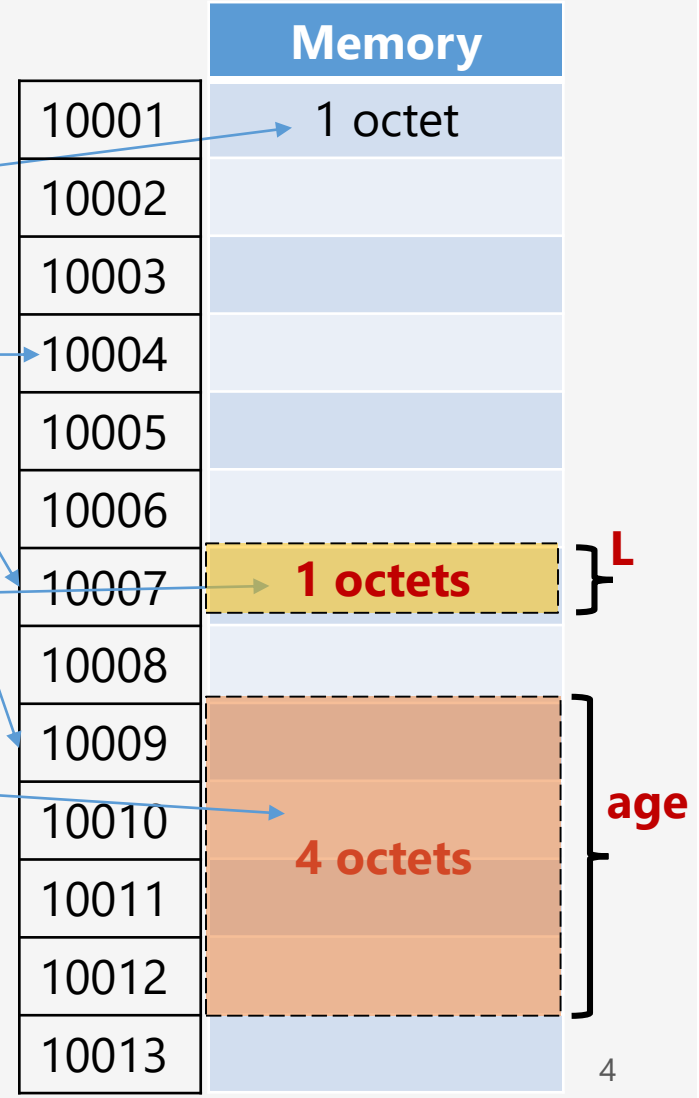
1. Chaque case mémoire est codée sur 1 octet (8bits)

2. Chaque case mémoire (1 octet) est identifiée par une adresse spécifique.

3. Le nombre d'octets réservé par le SE est dépend du type de la variable.
(**char**: 1 octet, **int**: 2 ou 4 octets, **long**: 4 octets, **pointeur: 4 octets**,
float: 4 octets, **double**: 8 octets)

Exemple: char L

int age



Chapitre 1: Rappels

1. Accès à la mémoire via les identificateurs des variables

1. Chaque case mémoire est codée sur 1 octet (8bits)

2. Chaque case mémoire (1 octet) est identifiée par une adresse spécifique.

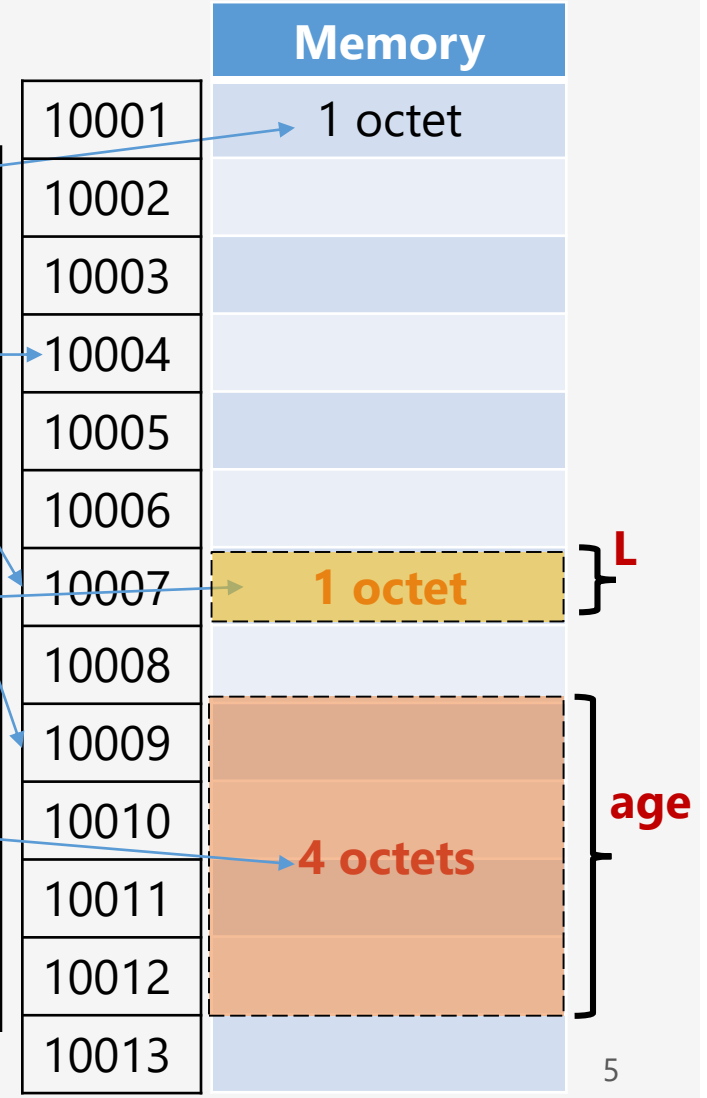
3. 3. Le nombre d'octets réservé par le SE est dépend du type de la variable. (**char**: 1 octet, **int**: 2 ou 4 octets, **long**: 4 octets, **pointeur: 4 octets**, **float**: 4 octets, **double**: 8 octets)

Exemple: char lettre

int age

4. L'adresse mémoire d'une variable est celle du dernièr octet

Exemple: L'adresse mémoire de la variable **age** est:



Chapitre 1: Rappels

1. Accès à la mémoire via les identificateurs des variables

1. Chaque case mémoire est codée sur 1 octet (8bits)

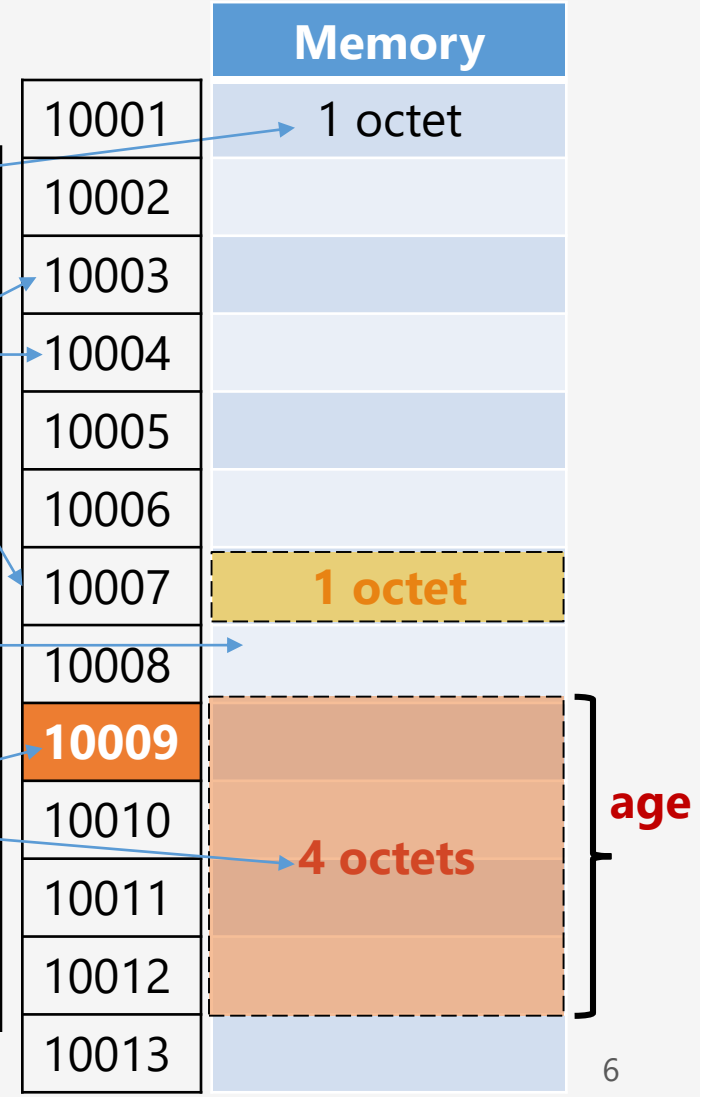
2. Chaque case mémoire (1 octet) est identifiée par une adresse spécifique.

3. 3. Le nombre d'octets réservé par le SE est dépend du type de la variable. (**char**: 1 octet, **int**: 2 ou 4 octets, **long**: 4 octets, **pointeur: 4 octets**, **float**: 4 octets, **double**: 8 octets)

Exemple: char lettre

int age

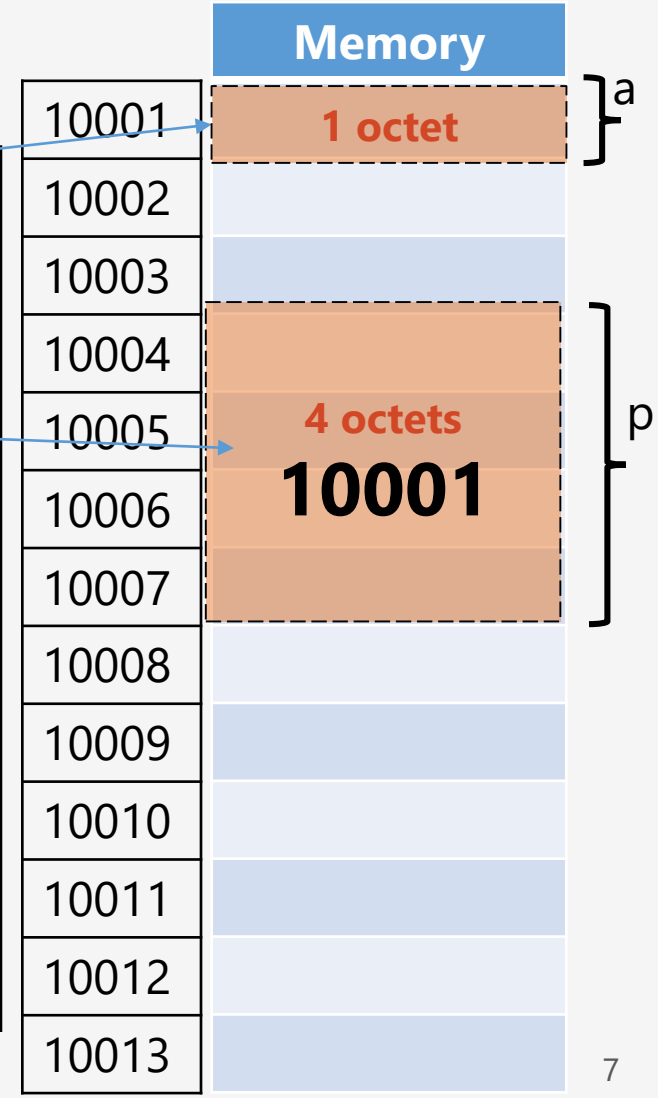
4. L'adresse mémoire d'une variable est celle du premier octet
Exemple: L'adresse mémoire de la variable **age** est **10009**



2. Les pointeurs

Les pointeurs: Variables spéciales qui contiennent l'adresse mémoire d'une autre variable.

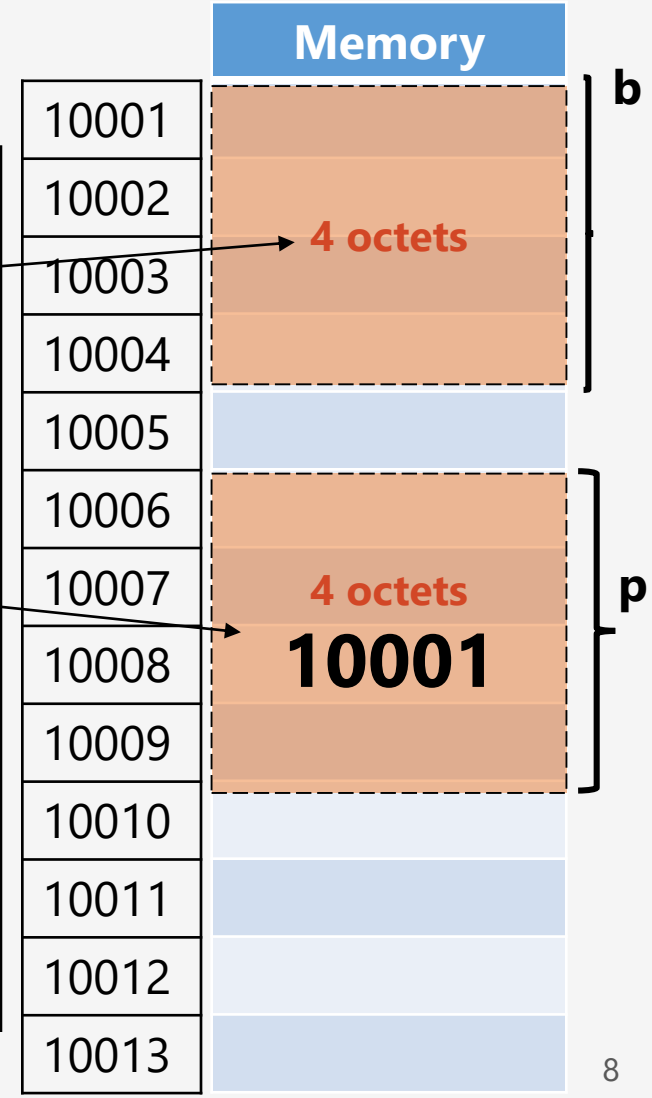
```
char a;
char *p = &a;
```



2. Les pointeurs

Considérons par exemple les instructions suivantes:

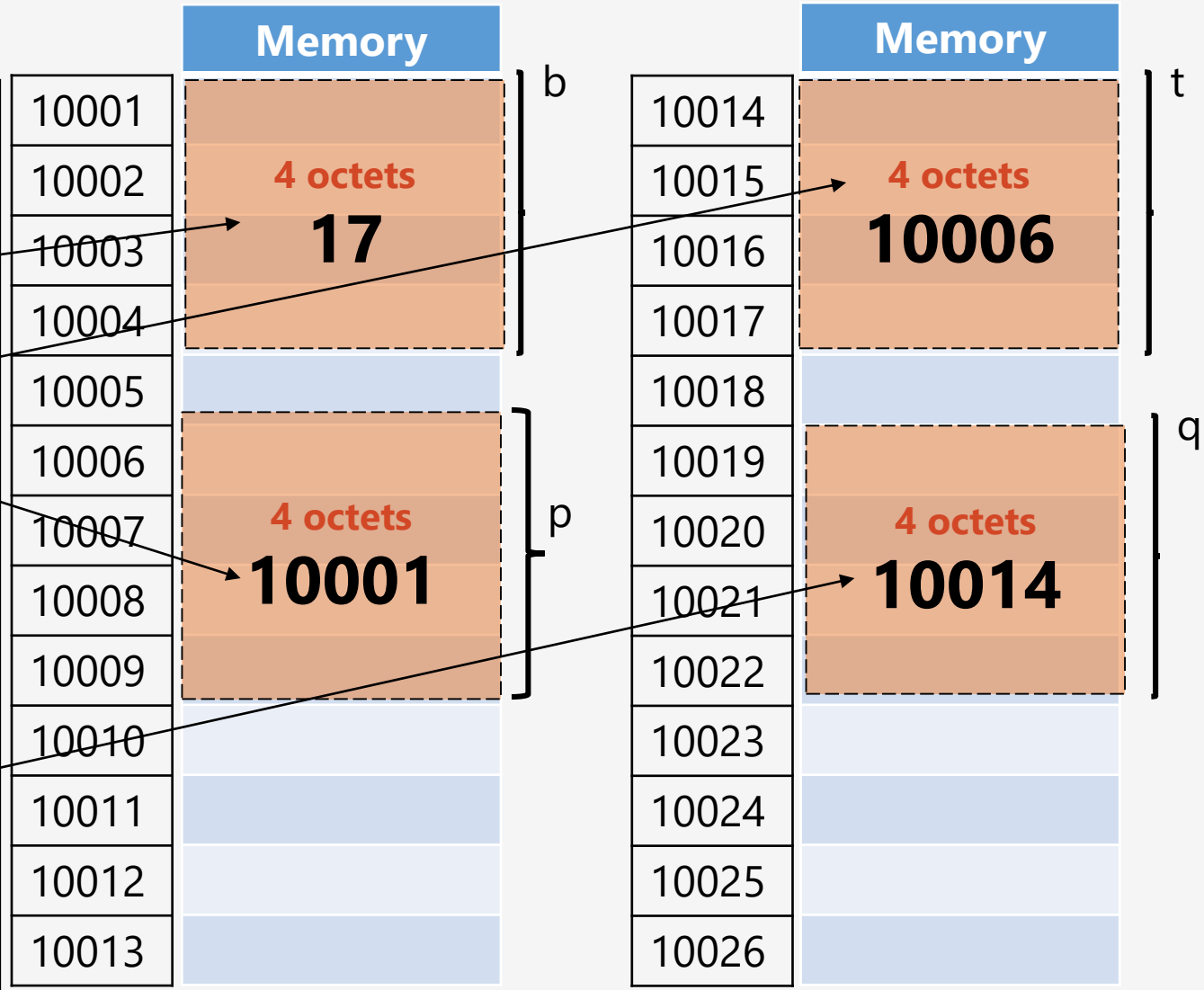
```
int b;
int *p = &b;
```



2. Les pointeurs

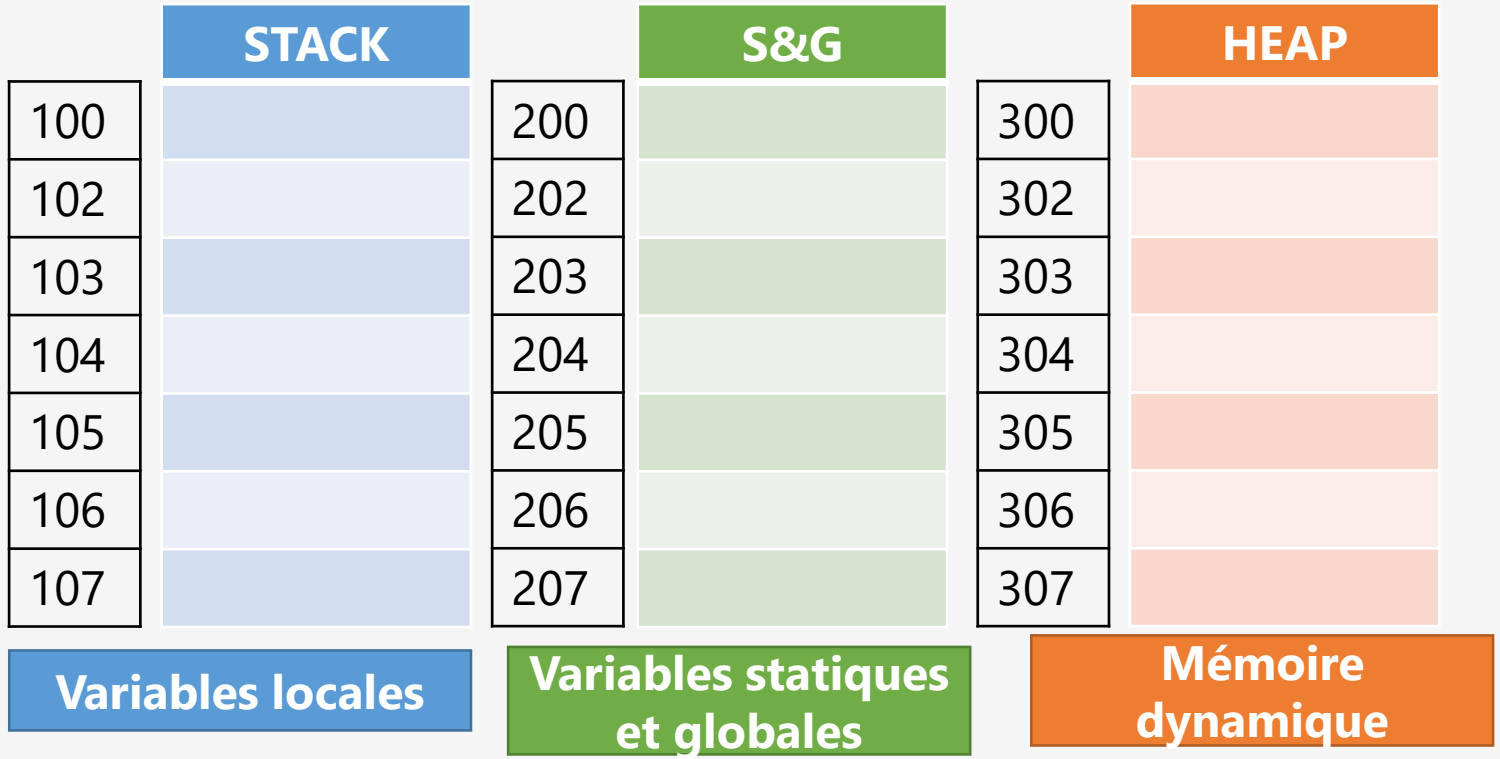
Considérons par exemple les instructions suivantes:

```
int b=17;
int *p = &b;
int ** t = &p;
printf("%d", b); 17
printf("%d", &b); 10001
printf("%d", p); 10001
printf("%d", *p); 17
printf("%d", t); 10006
printf("%d", *t); 10001
printf("%d", &t); 10014
int ***q=&t;
printf("%d", *q); 10006
printf("%d", **q); 10001
printf("%d", ***q); 17
```

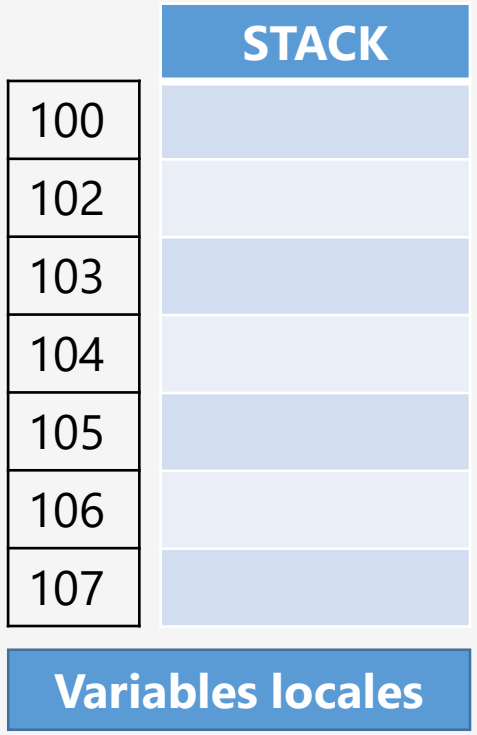


3. Les variables locales

Le mémoire Centrale (RAM)



3. Les variables locales

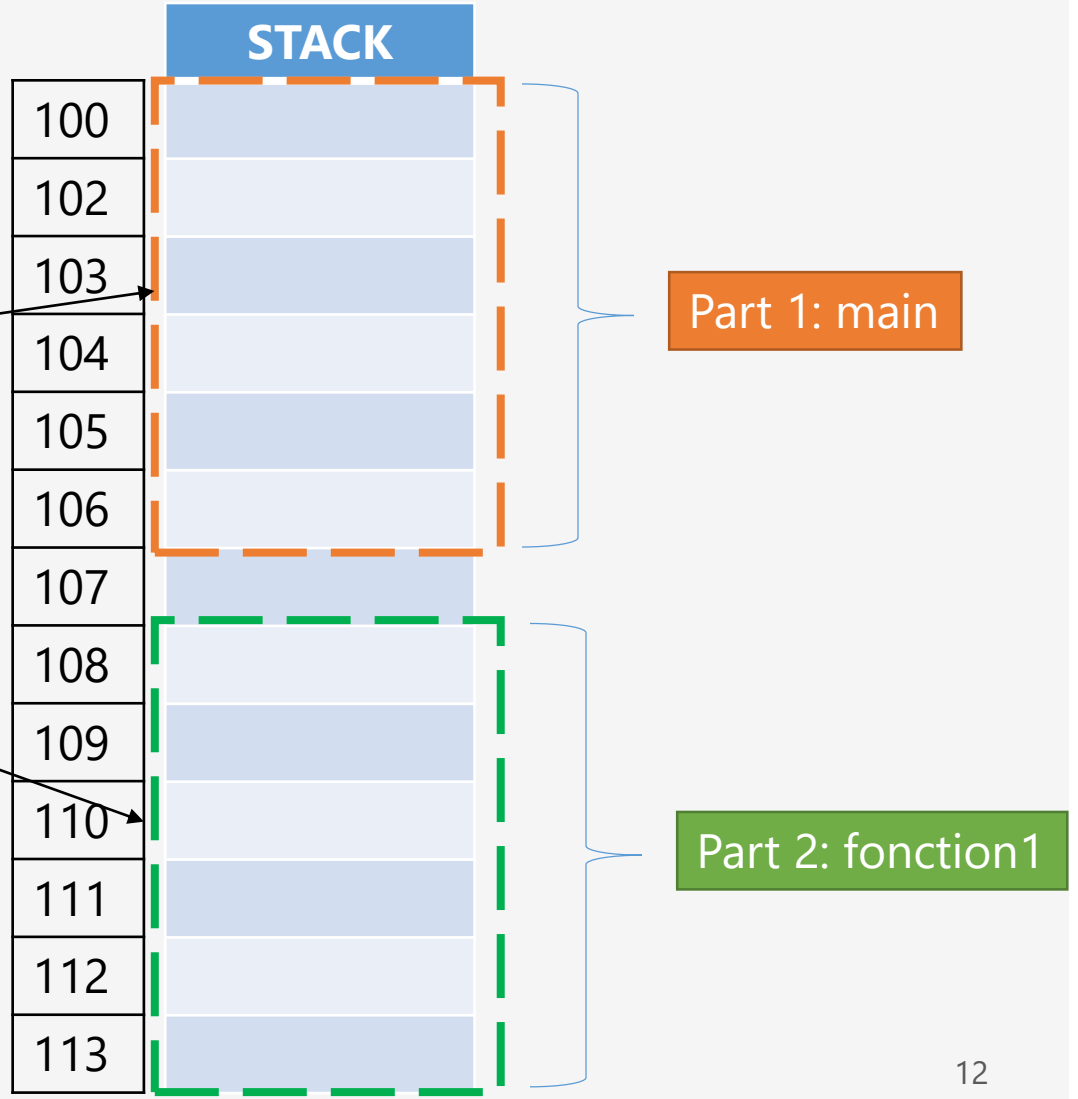


Caractéristiques:

- Taille limitée (Windows: 1Mo; Linux: 8Mo)
- Divisée en sous parties selon le nombre de fonction.
- Allocation statique

3. Les variables locales

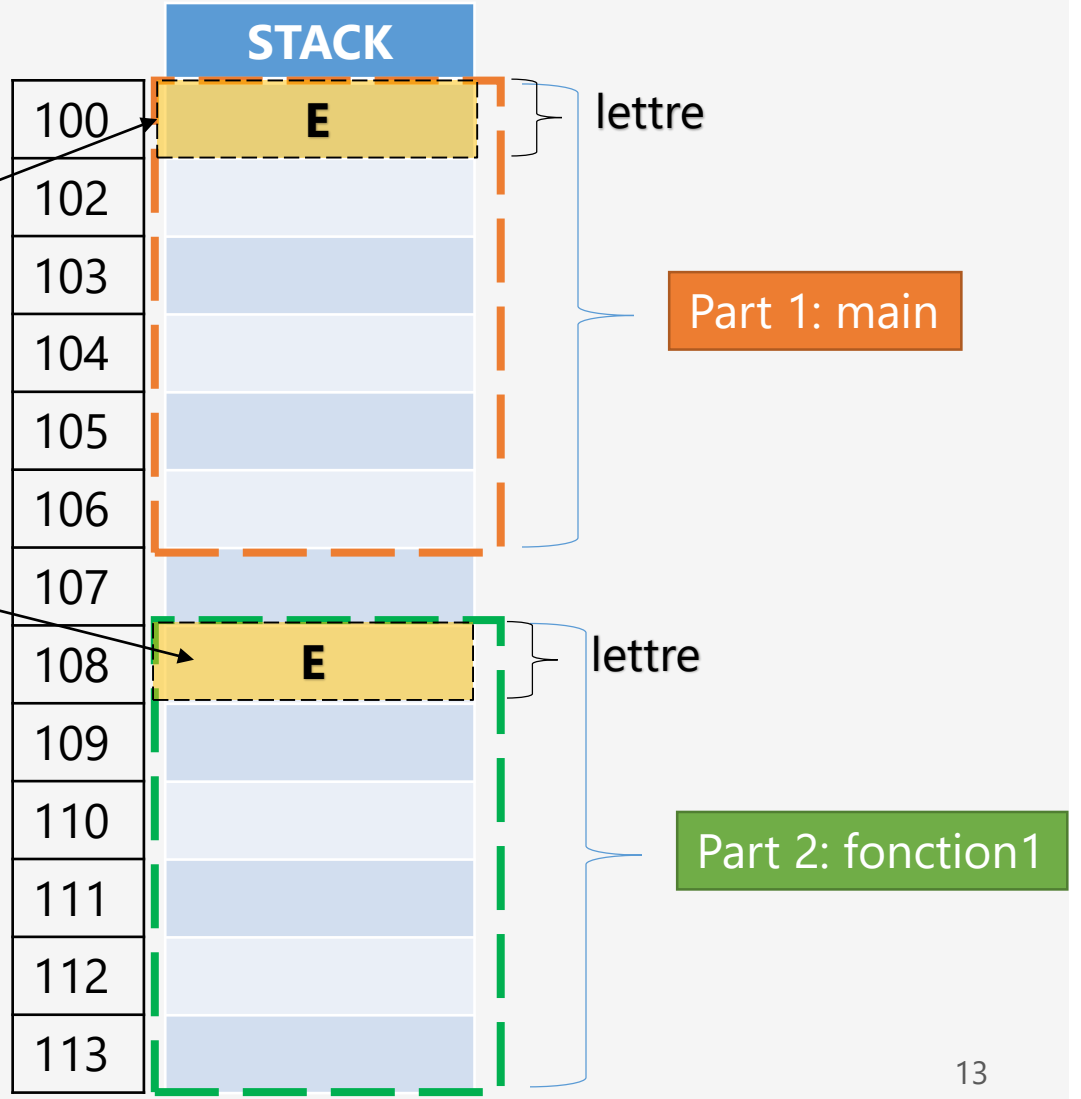
```c
#include <stdio.h>
#include <stdlib.h>
void fontion1 (char lettre)
{
    lettre='A';
}
int main()
{
    char lettre='E';

    fontion1(lettre);

    printf("La valeur de lettre apres l'appel de la fonction est:%c", lettre);

    return 0;
}
```



Les deux variables **lettre** ont aucune relation entre elles, chacune est déclarée dans une partie spécifique

3. Les variables locales

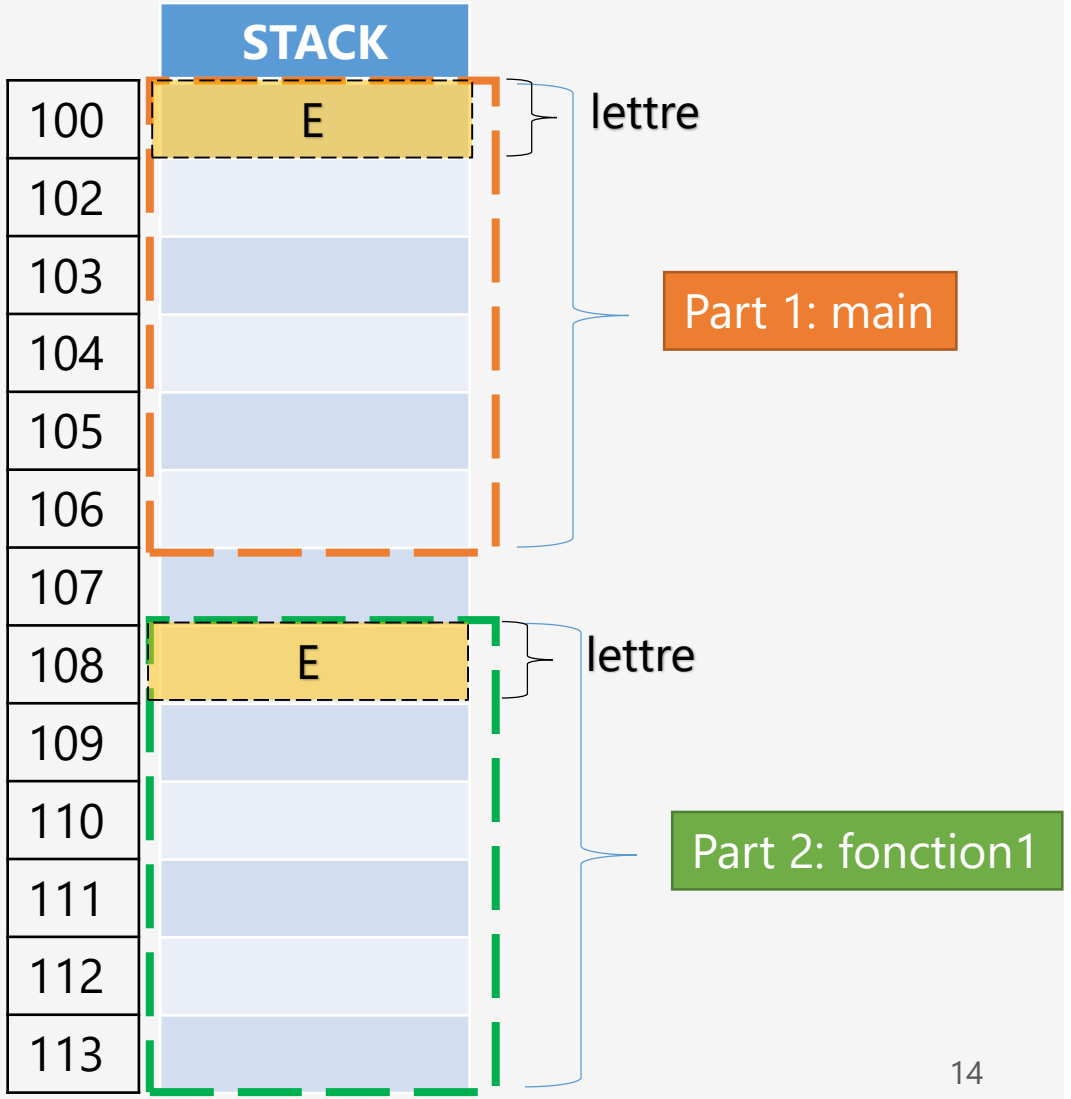
```c
#include <stdio.h>
#include <stdlib.h>
void fontion1 (char lettre)
{
    lettre='A';
}
int main()
{
    char lettre='E';

    fontion1(lettre);

    printf("La valeur de lettre apres l'appel de la fonction est:%c", lettre);

    return 0;
}
```



Les deux variables **lettre** ont aucune relation entre elles, chacune est déclarée dans une partie spécifique

3. Les variables locales

```c
#include <stdio.h>
#include <stdlib.h>
void fontion1 (char lettre)
{
    lettre='A';
}
int main()
{
    char lettre='E';

    fontion1(lettre);

    printf("La valeur de lettre apres l'appel de la fonction est:%c", lettre);

    return 0;
}
```

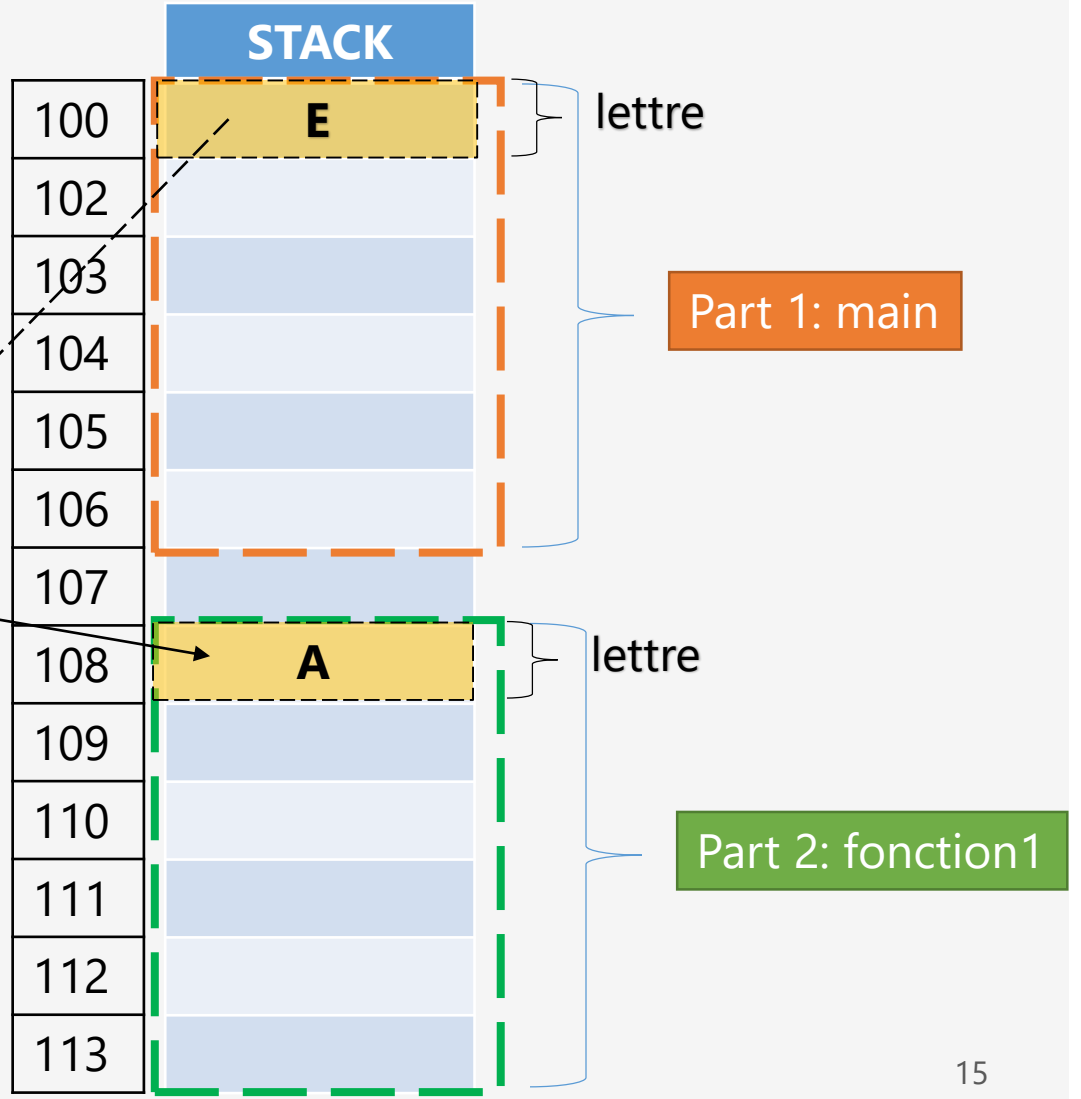


Les deux variables **lettre** ont aucune relation entre elles, chacune est déclarée dans une partie spécifique

3. Les variables locales

```c
#include <stdio.h>
#include <stdlib.h>
void fontion1 (char lettre)
{
    lettre='A';
}
int main()
{
    char lettre='E';

    fontion1(lettre);

    printf("La valeur de lettre apres l'appel de la fonction est:%c", lettre);

    return 0;
}
```

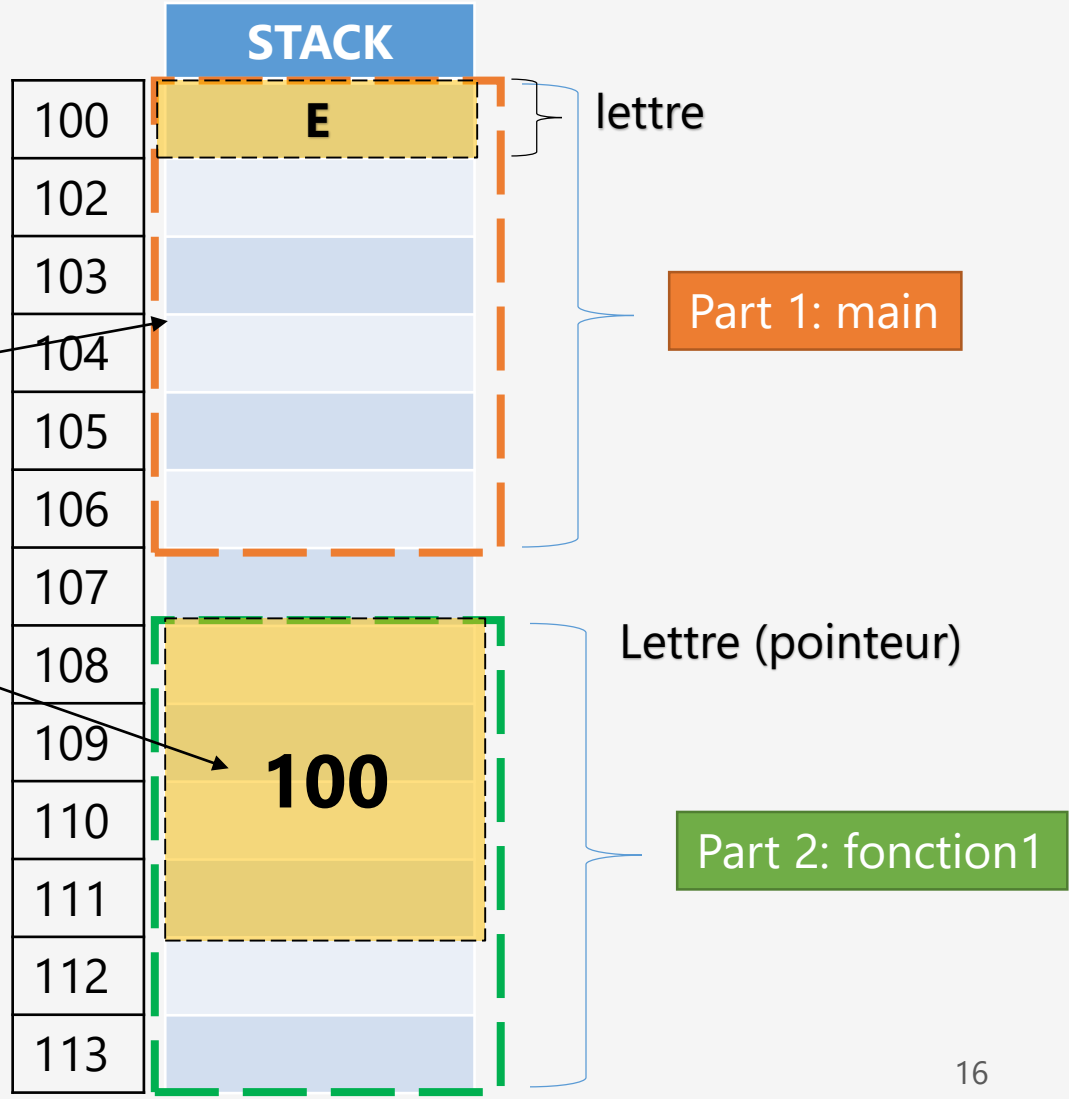
```
La valeur de R apres l'appel de la fonction est:E
Process returned 0 (0x0)   execution time : 0.004 s
Press any key to continue.
```



3. Les variables locales

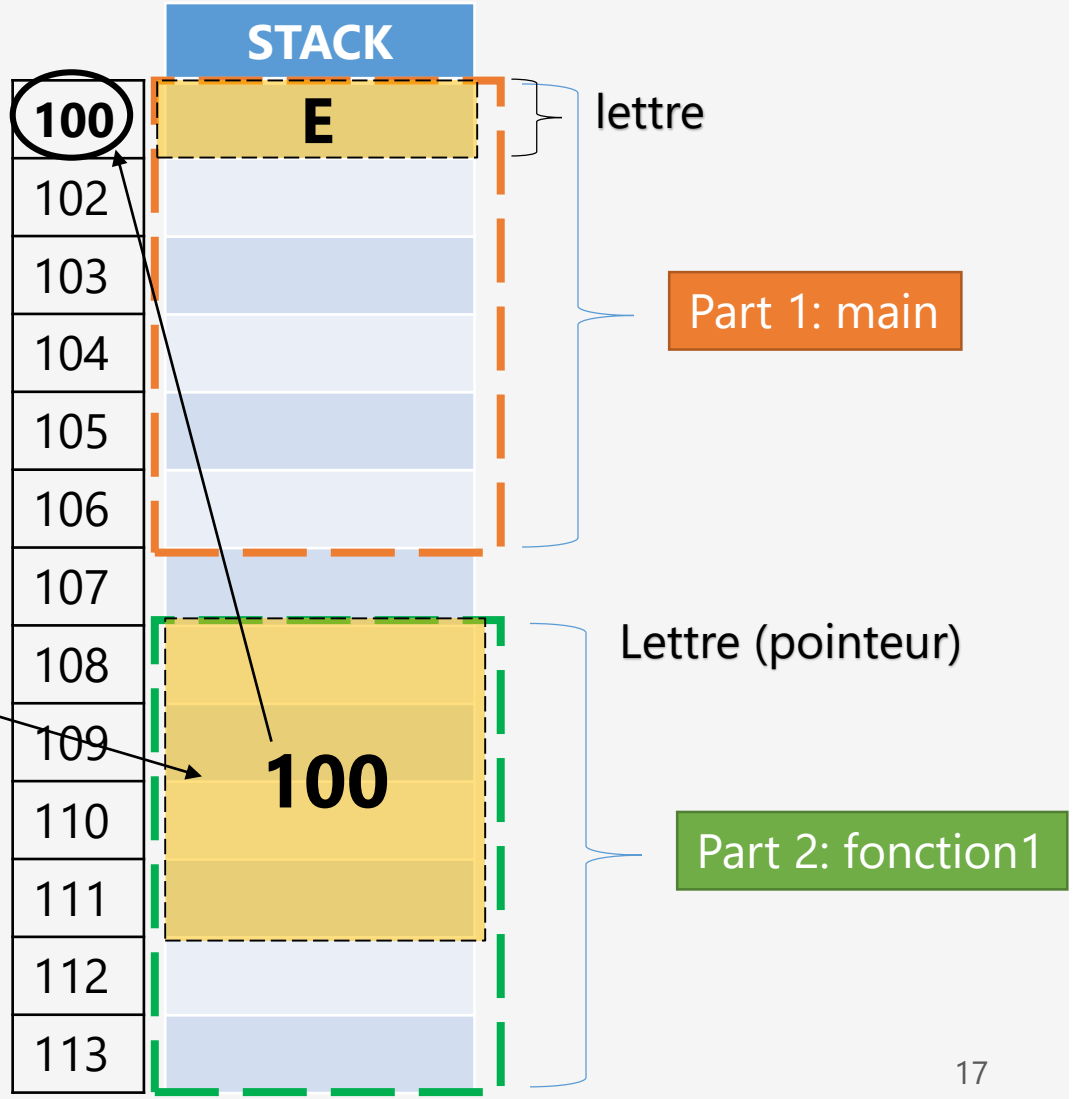
```c
#include <stdio.h>
#include <stdlib.h>
void fontion1 (char *lettre)
{
    *lettre='A';
}
int main()
{
    char lettre='E';

    fontion1(&lettre);

    printf("La valeur de lettre apres l'appel de la fonction est:%c", lettre);

    return 0;
}
```

pointeur



3. Les variables locales

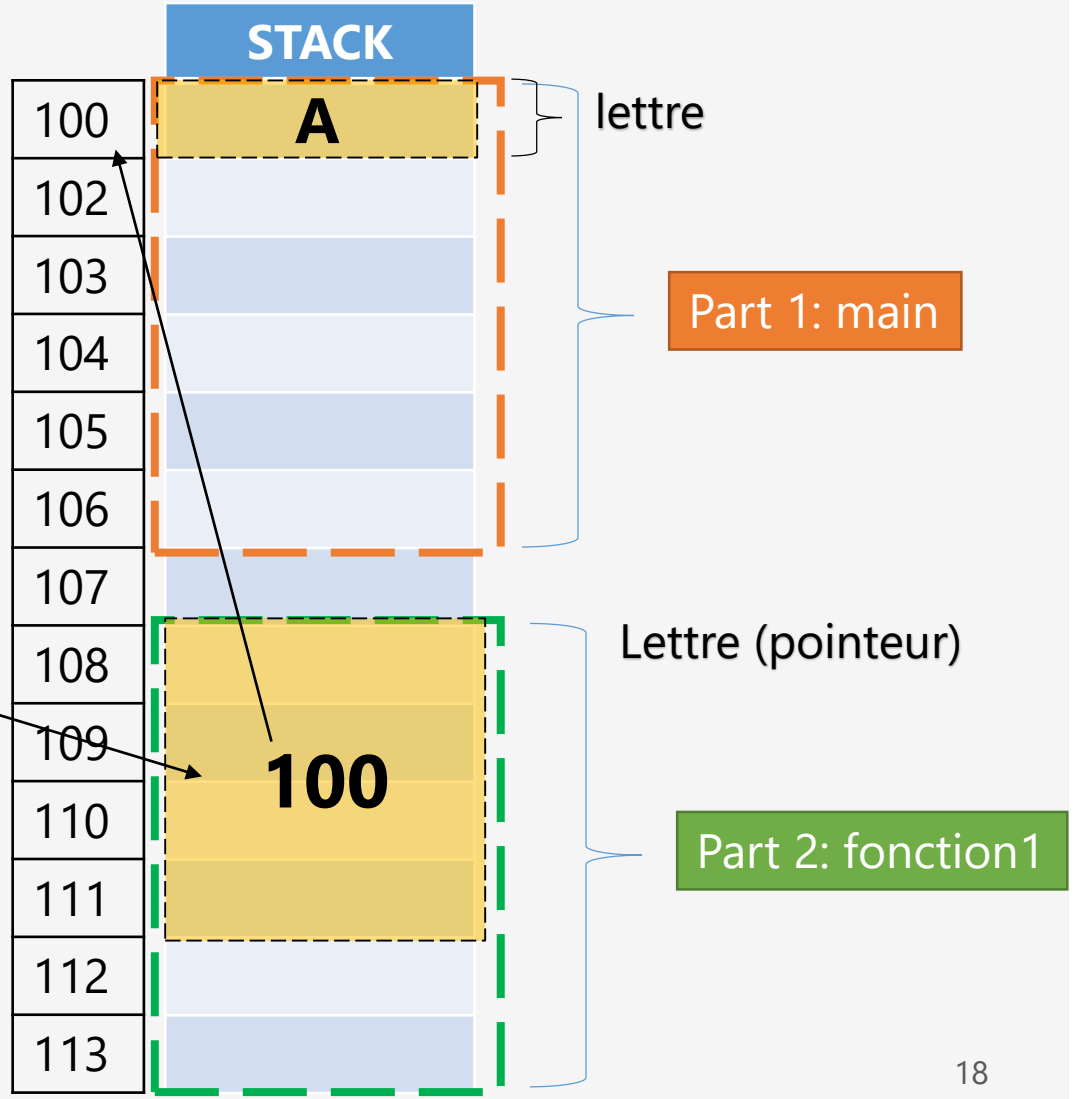
```c
#include <stdio.h>
#include <stdlib.h>
void fontion1 (char *lettre)
{
    *lettre='A';
}
int main()
{
    char lettre='E';

    fontion1(&lettre);

    printf("La valeur de lettre apres l'appel de la fonction est:%c", lettre);

    return 0;
}
```



3. Les variables locales

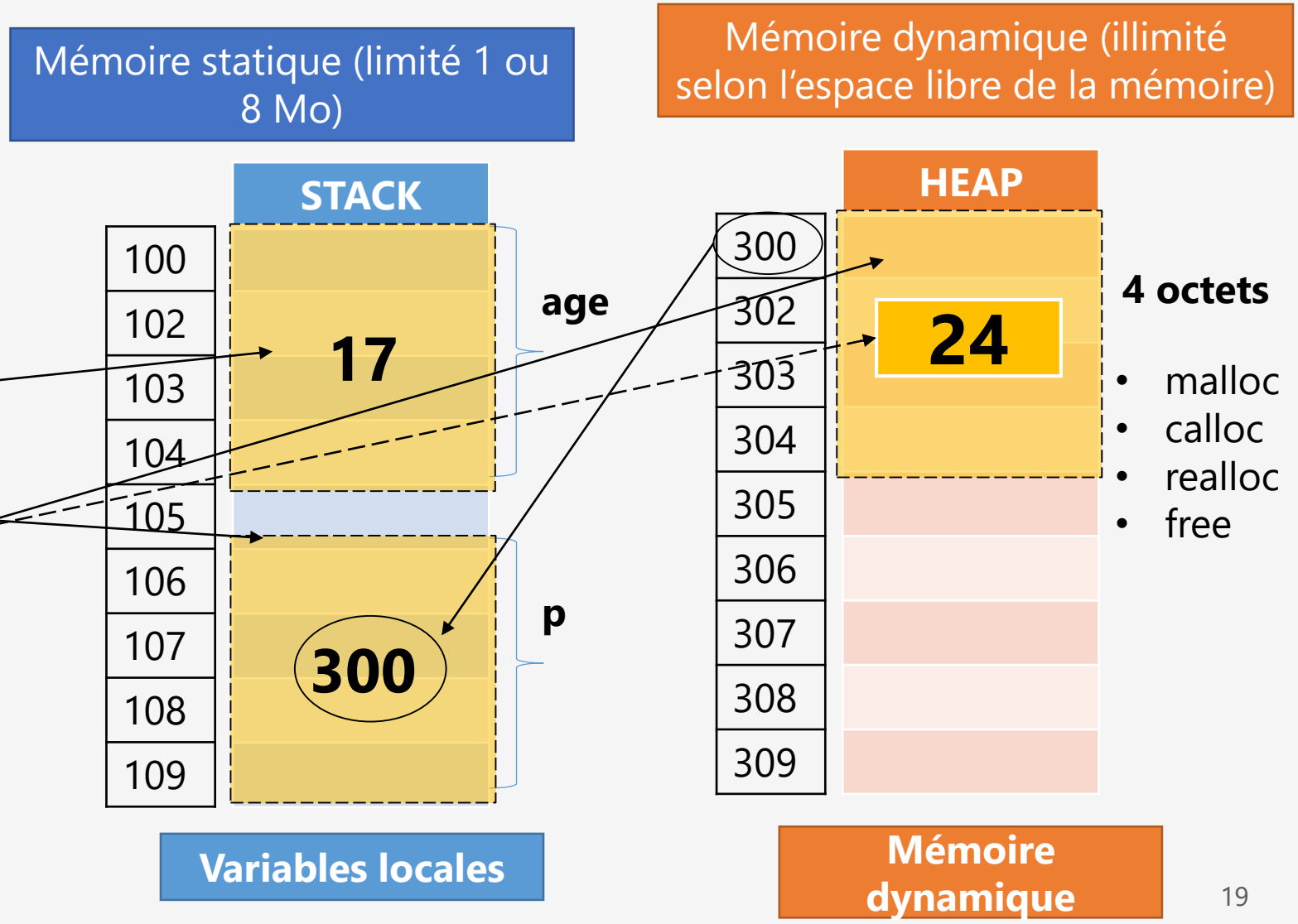
```c
#include <stdio.h>
#include <stdlib.h>
void fontion1 (char *lettre)
{
    *lettre='A';
}
int main()
{
    char lettre='E';

    fontion1(&lettre);

    printf("La valeur de R apres l'appel de la fonction est:%c", lettre);

    return 0;
}
```

```
La valeur de R apres l'appel de la fonction est:A
Process returned 0 (0x0)   execution time : 0.004 s
Press any key to continue.
```



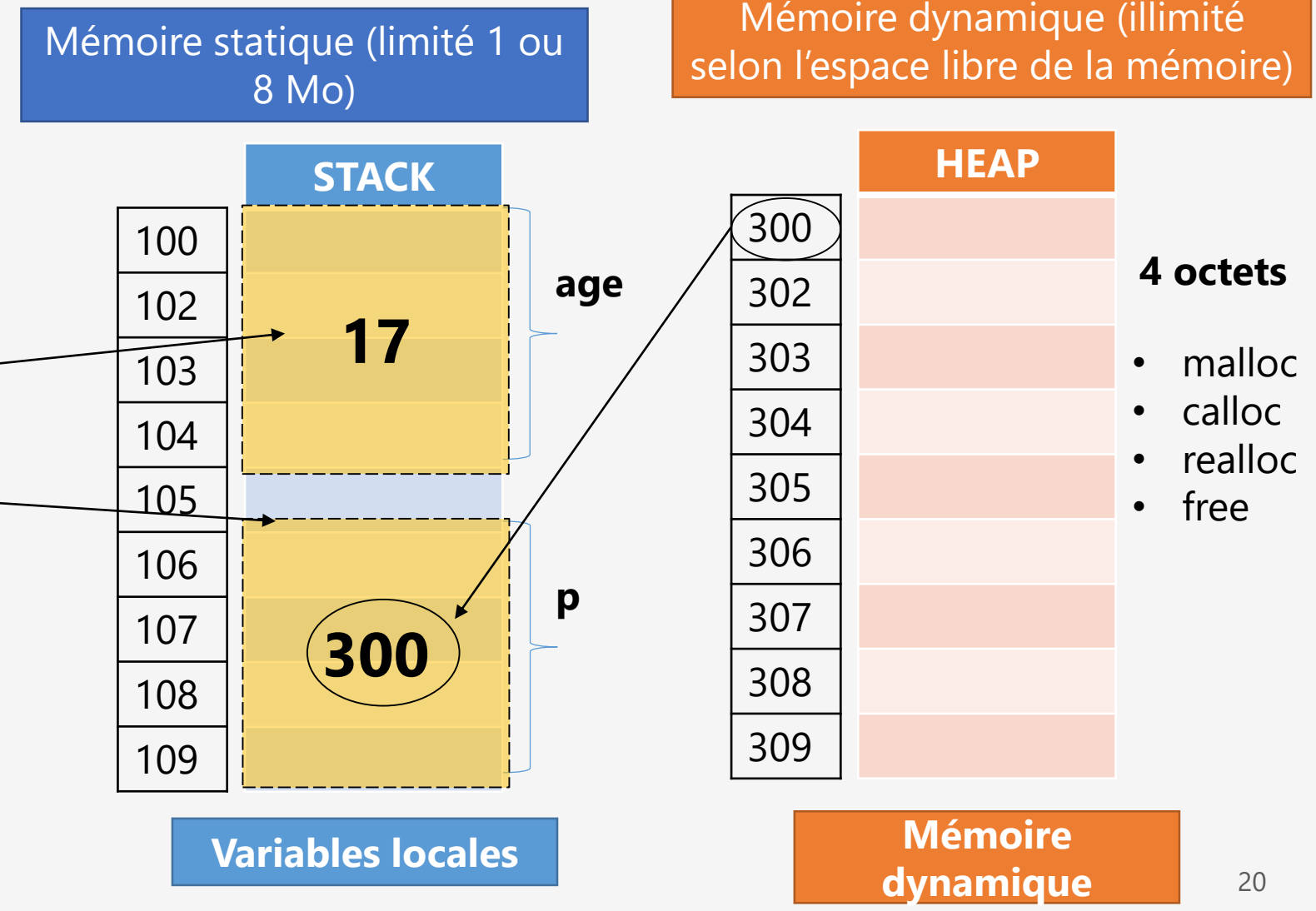
4. Mémoire dynamique

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

int main()
{
int age = 17;

int *p;

p = malloc (sizeof(int));
*p = 24;

free(p);
p = NULL;

    return 0;
}
```



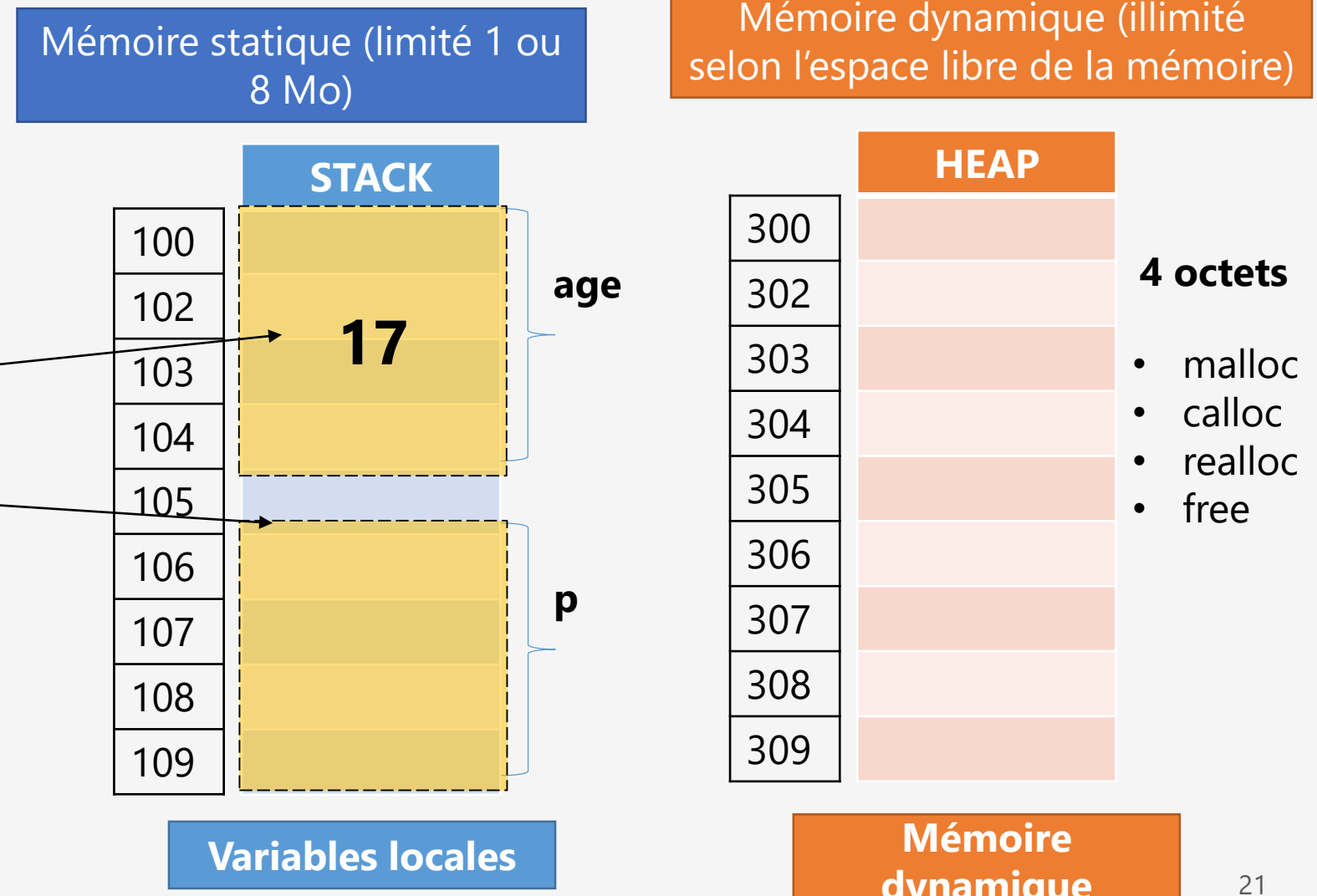
4. Mémoire dynamique

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

int main()
{
int age = 17;

int *p;

p = malloc (sizeof(int));
*p = 24;

free(p);
p = NULL;

    return 0;
}
```



4. Mémoire dynamique

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

int main()
{
int age = 17;

int *p;

p = malloc (sizeof(int));
*p = 24;

free(p);
p = NULL;

    return 0;
}
```



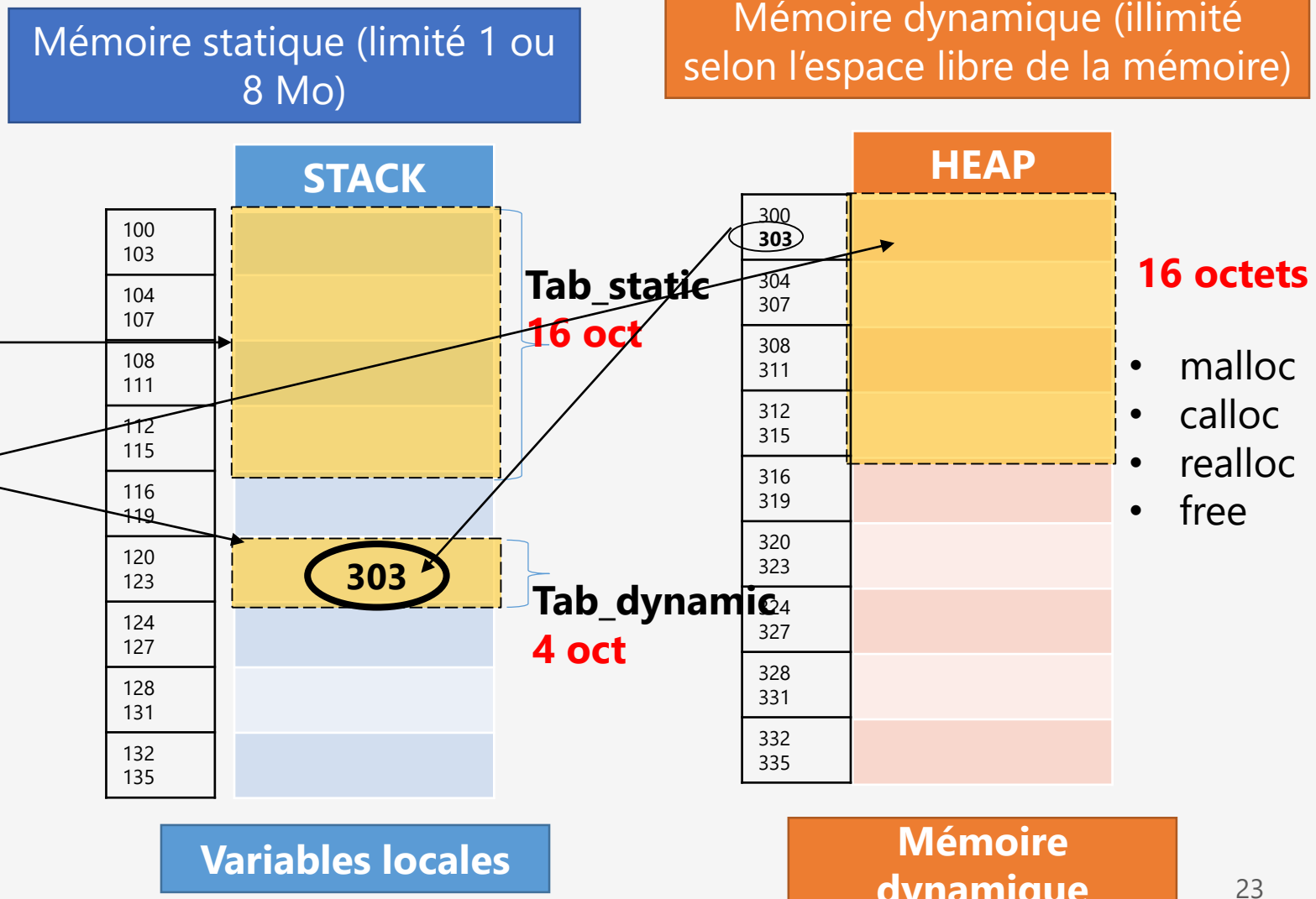
5. Les tableaux dynamiques

- **Il existe deux types de tableaux :**
  - **Les tableaux statiques: dont la taille est connue à la compilation.**
  - **Les tableaux dynamiques: dont la taille est connue à l'exécution.**

5. Les tableaux dynamiques

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

int main()
{
    int tab_static[4];

    int * tab_dynamic;

    tab_dynamic = malloc(4*sizeof(int));

    return 0;
}
```

4 oct → 1 oct | 1 oct | 1 oct | 1 oct



5. Les tableaux dynamiques

Exercice:

En utilisant un tableau dynamique, écrire un programme qui permet aux utilisateurs de saisir des notes au clavier (le nombre de notes n'est pas fixe). On suppose que le nombre de notes à saisir par défaut est égal à 3. (Au-delà de 3 notes le programme doit automatiquement augmenter la taille de la mémoire dynamique).

5. Les tableaux dynamiques

Exercice:
En utilisant un tableau dynamique, écrire un programme qui permet aux utilisateurs de saisir des notes au clavier (le nombre de notes n'est pas fixe). On suppose que le nombre de notes à saisir par défaut est égal à 3. (Au-delà de 3 notes le programme doit automatiquement augmenter la taille de la mémoire dynamique).

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

int main()
{
int i,n,nbr_note;
nbr_note=3;
int *p;

p = malloc (nbr_note * sizeof(int));
printf("Combien de notes voulez-vous inserer?\n");
scanf("%d",&n);

    for(i=0; i<n; i++)
    {
        if(i >= nbr_note)
        {
            nbr_note ++;
            p = realloc(p, nbr_note * sizeof(int));
        }
       printf("Entrer la note numero %d\n", i+1);
       scanf("%d",&p[i]);

    }
//Affichage des lettres insérées
printf("\n-----------------\n");
for (i=0;i<n;i++)
{
    printf("%d\n",p[i]);
   // p++;

}
free(p);
p = NULL;
    return 0;
}
```

6. Les structures

Une structure: Est un type de données défini par l'utilisateur en C / C++. Une structure crée un type de données qui peut être utilisé pour grouper des éléments de types éventuellement différents en un seul type, la première chose à réaliser est la description de celle-ci (techniquement, sa **définition**), c'est-à-dire préciser de quel(s) champ(s) cette dernière va se composer.

6. Les structures

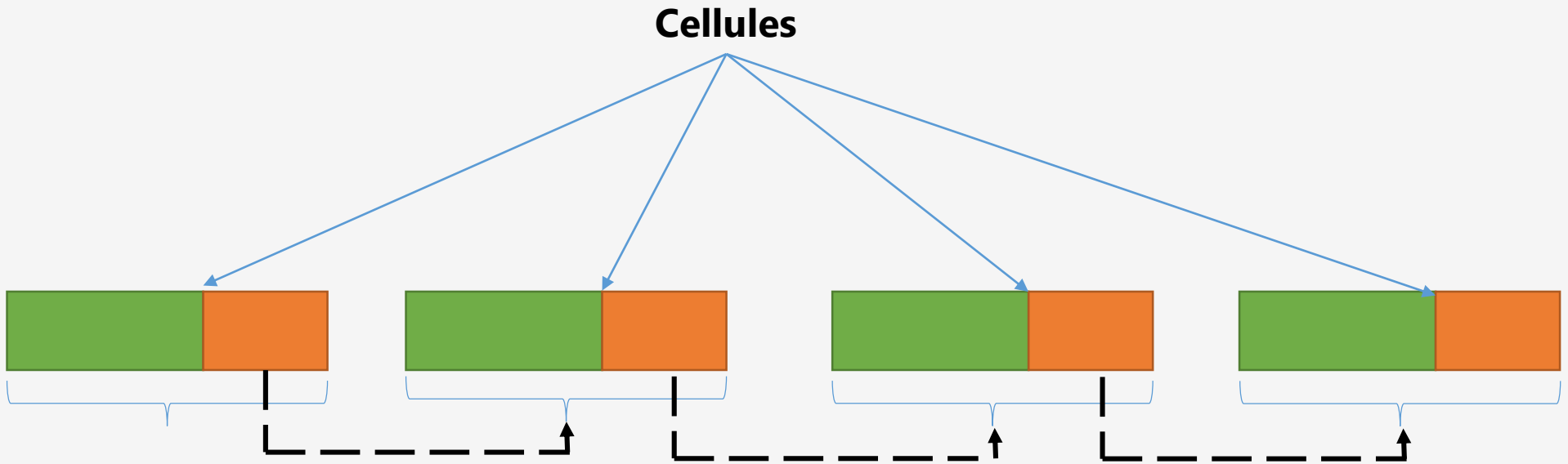
```c
#include <stdio.h>
#include <stdlib.h>
//définition de la structure
struct client
{
    char nom[20];
    long tel;
};
int main()
{
//déclaration d'une variable structurée
struct client client1;
//accès au champ d'une structure
client1.tel = 0669767699;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
//définition de la structure
typedef struct client
{
    char nom[20];
    long tel;
} clt;
int main()
{
//déclaration d'une variable structurée
clt client1;
//accès au champ d'une structure
client1.tel = 0669767699;
//déclaration d'un tableau de 10 structures client
clt tab[10];
//Pour modifier le nom du client qui a l'index i=1 dans tab on écrira
tab[1].nom = "TOTO"
//déclaration d'un pointeur qui peut pointer sur les types clt
clt *p;
//Réseravation dynamique
p = malloc(sizeof(clt));
return 0;
}
```

Chapitre 2: Les listes chainées

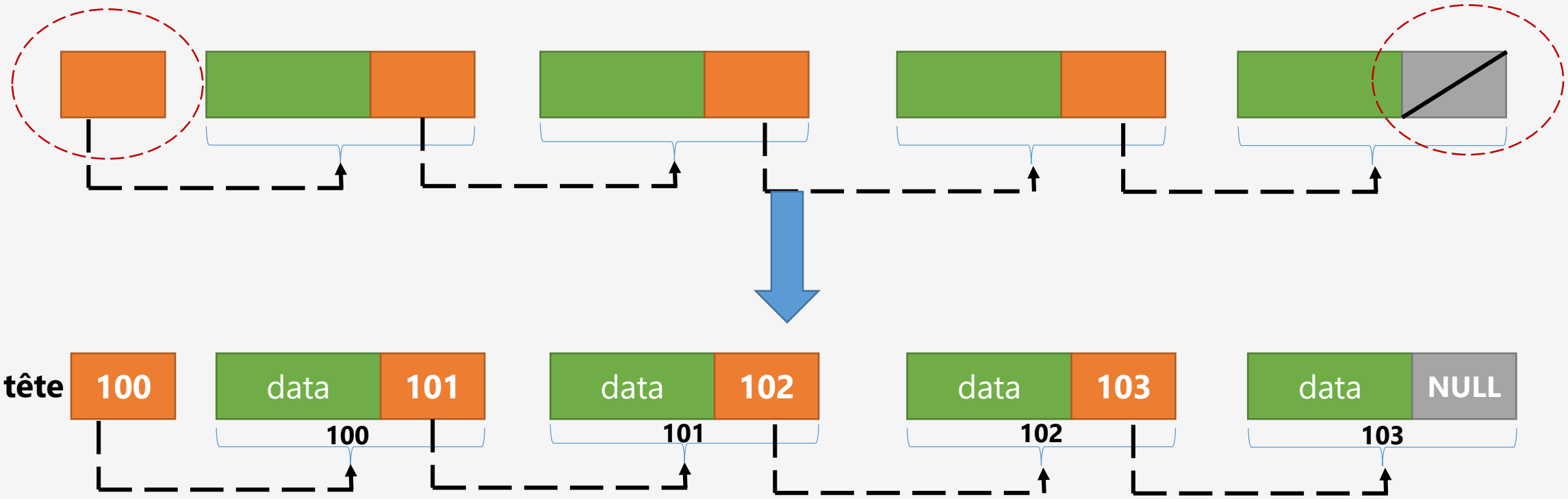
Chapitre 2: Les listes chainées

❑QU'EST-CE QU'UNE LISTE CHAÎNÉE ?



Chapitre 2: Les listes chainées

❑QU'EST-CE QU'UNE LISTE CHAÎNÉE ?

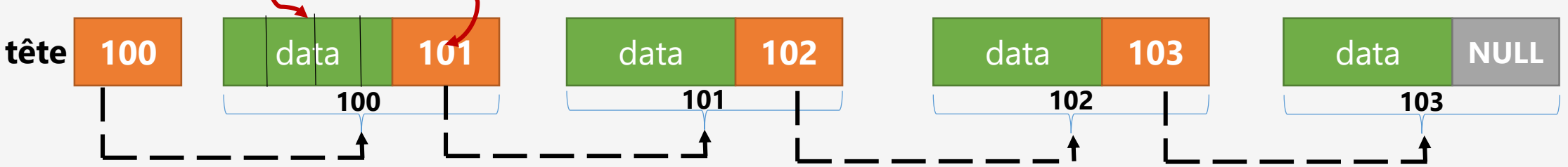


Chapitre 2: Les listes chainées

❑QU'EST-CE QU'UNE LISTE CHAÎNÉE ?

❑ Une liste chaînée est un **ensemble de cellules (éléments, nœuds)** liées entre elles **par des pointeurs**. Chaque cellule est une **structure** contenant les champs suivants :

- Une ou plusieurs **données** comme dans n'importe quelle structure ;
- Un **pointeur** "suivant" contient l'adresse mémoire de la cellule suivante.



Exemple de liste chaînée avec 4 cellules

❑ **L'entête** d'une liste chainée contient toujours l'adresse de la **première cellule**.

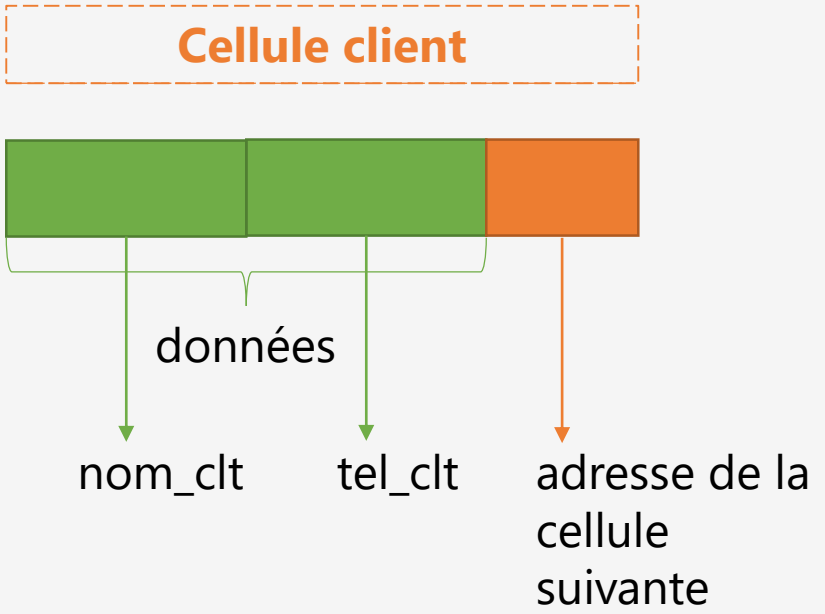
Chapitre 2: Les listes chainées

❑ **Exemple: cellule client**

On va créer une liste chainée qui regroupe un ensemble de cellules, et chaque cellule **"cellule client"** sera composée de:

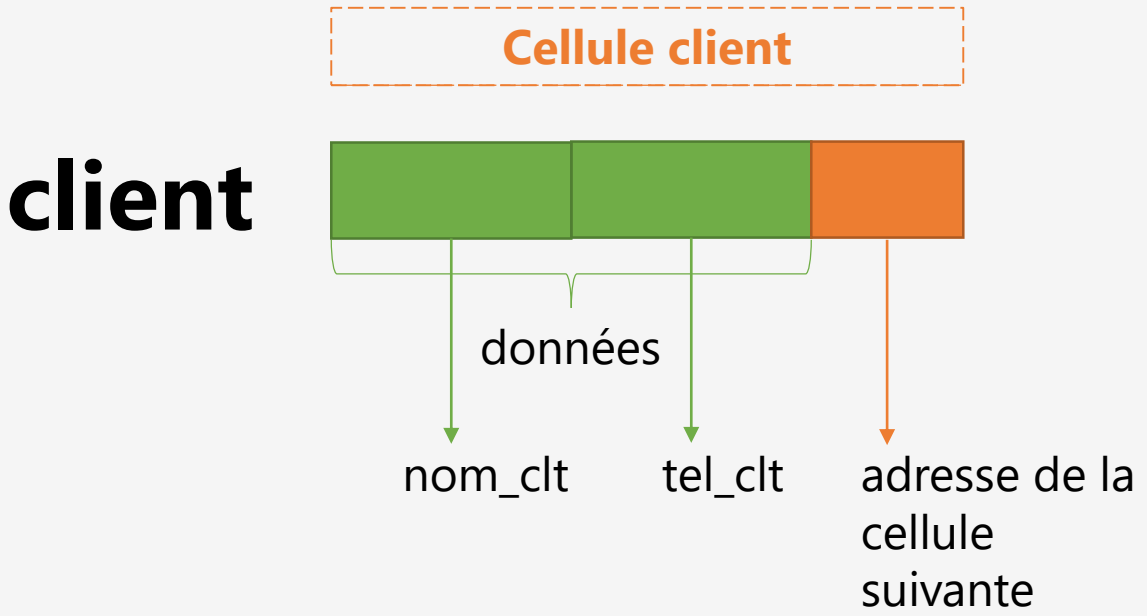
client

- **Partie données: nom et téléphone de client**
- **Pointeur: pointe sur la cellule suivante.**



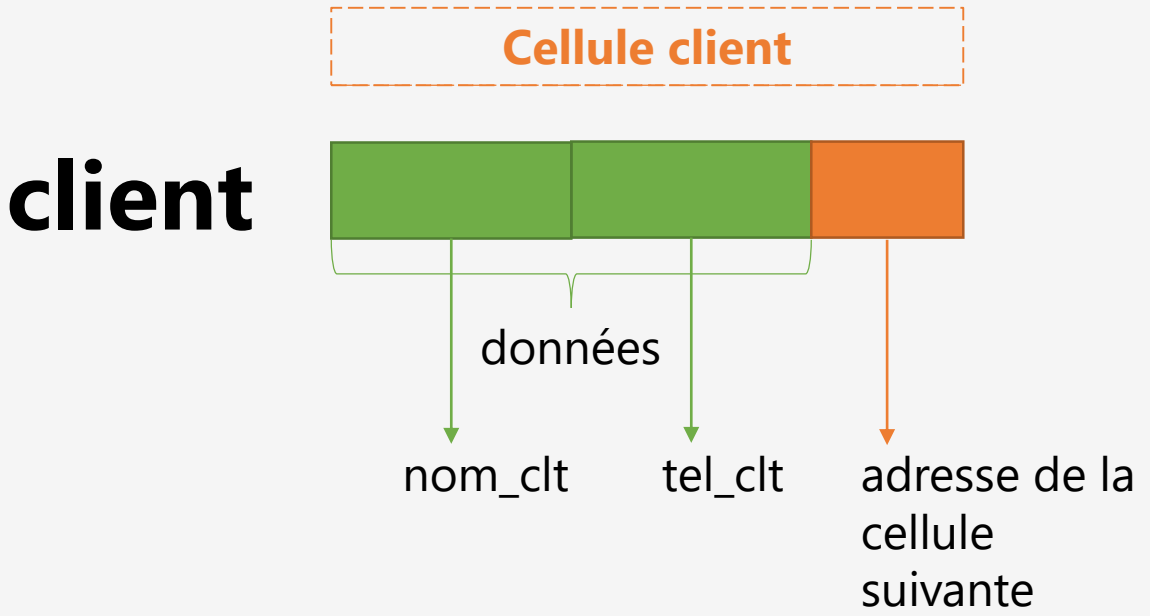
1. Définition de la cellule client

```
struct client {
        char nom_clt[20];
        long tel_clt;
                *next_clt;
};
```



1. Définition de la cellule client

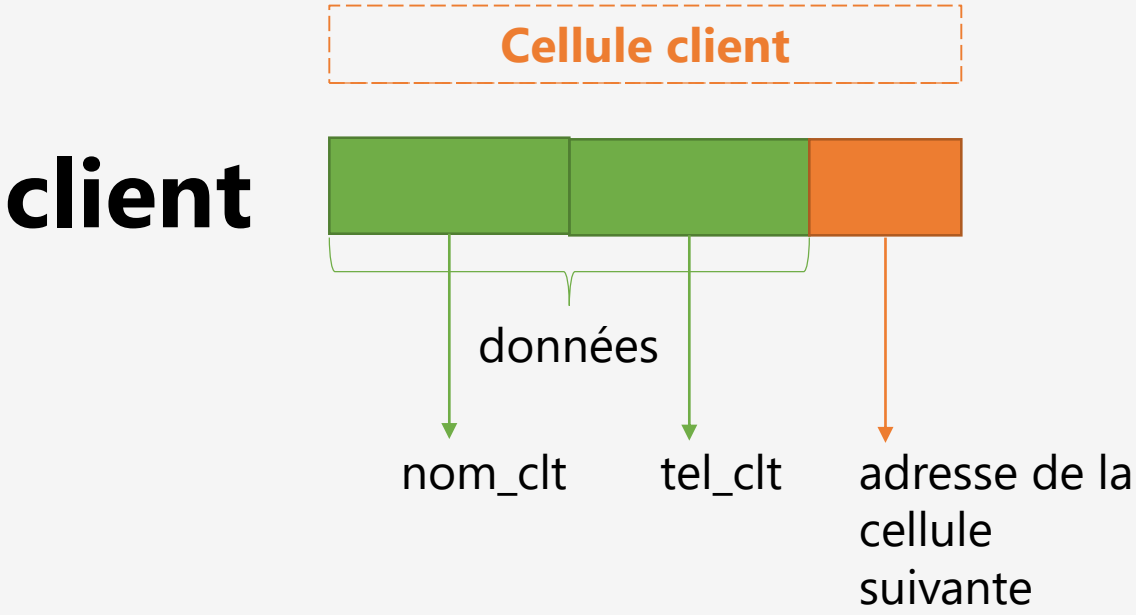
```
struct client {
        char nom_clt[20];
        long tel_clt;
        struct client   *next_clt;
};
```



Chapitre 2: Les listes chainées – Opérations sur les listes chainées

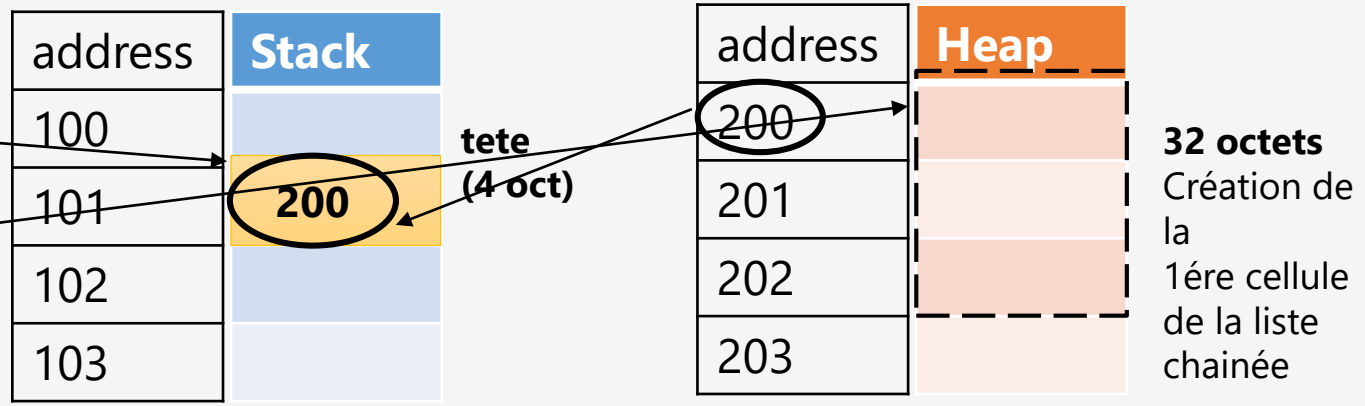
1. Définition de la cellule client

```
struct client {
        char nom_clt[20];
        long tel_clt;
        struct client  *next_clt;
};
```



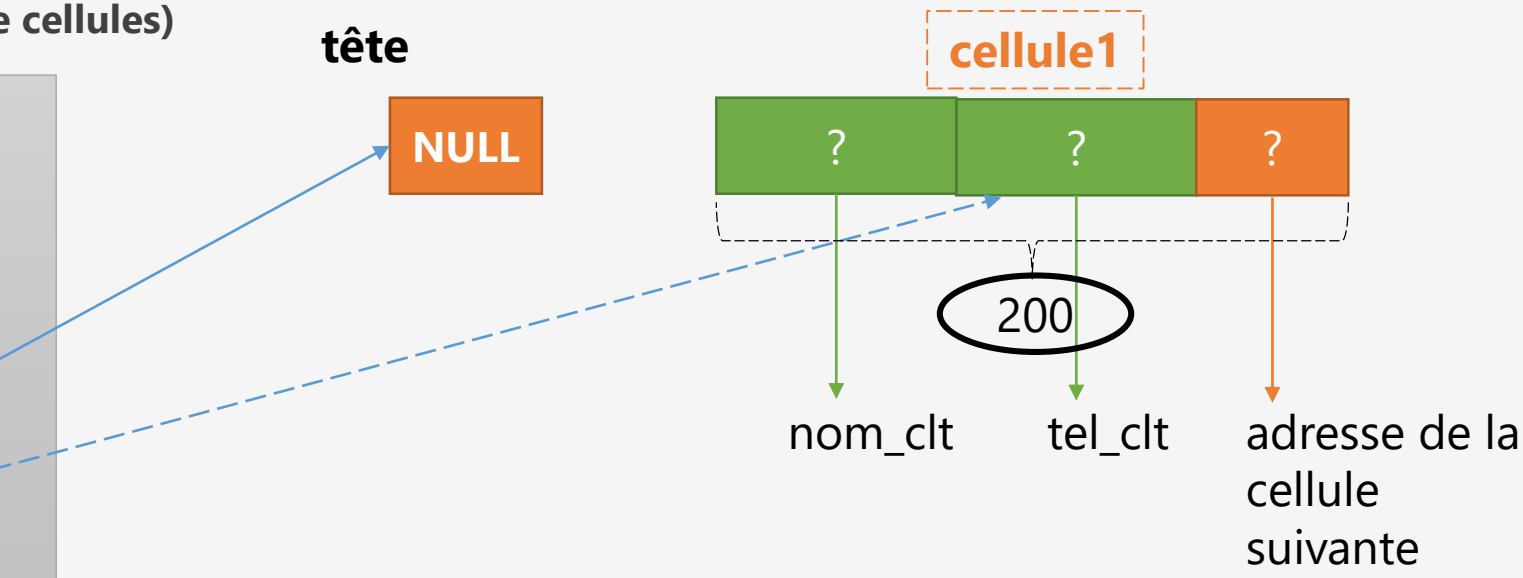
2. Déclaration de l'entête de la liste chainée.

```
struct client    *tete = NULL;

tete = malloc(sizeof(struct client));
```



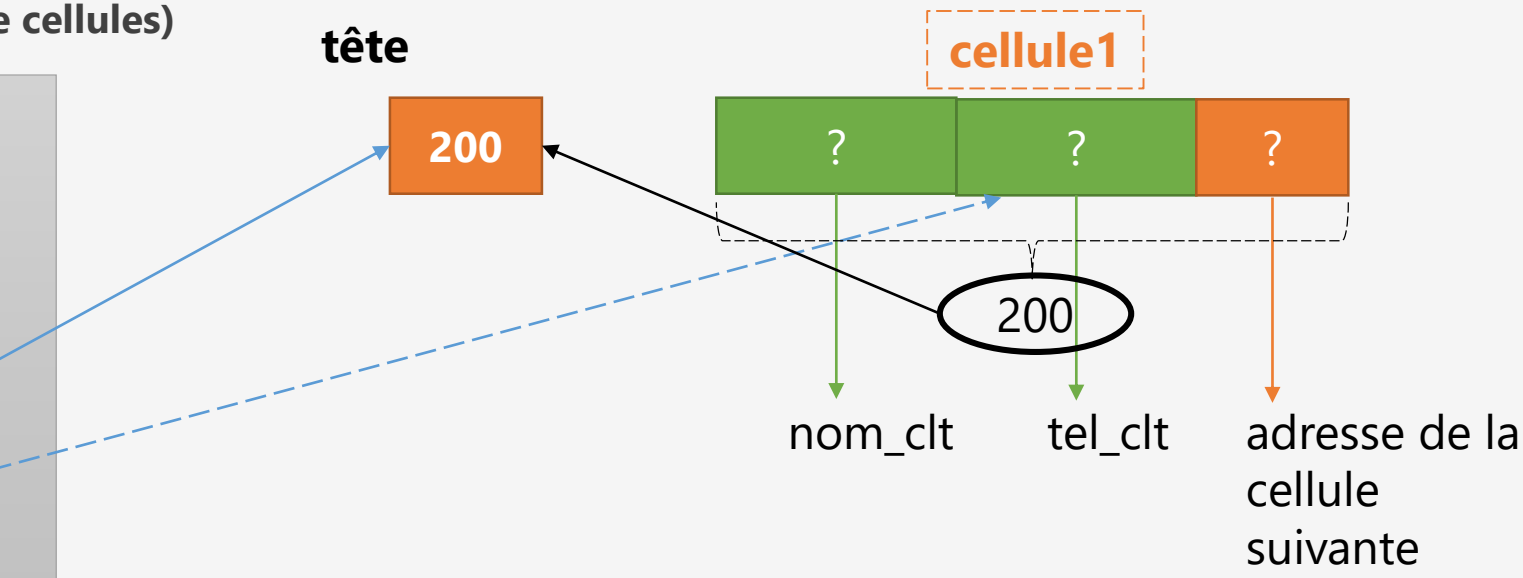
3. Création de la liste chainée (création de cellules)

```
struct client {
            char nom_clt[20];
            long tel_clt;
            struct client  *next_clt;
};
struct client    *tete = NULL;
tete = malloc(sizeof(struct client);
```



3. Création de la liste chainée (création de cellules)

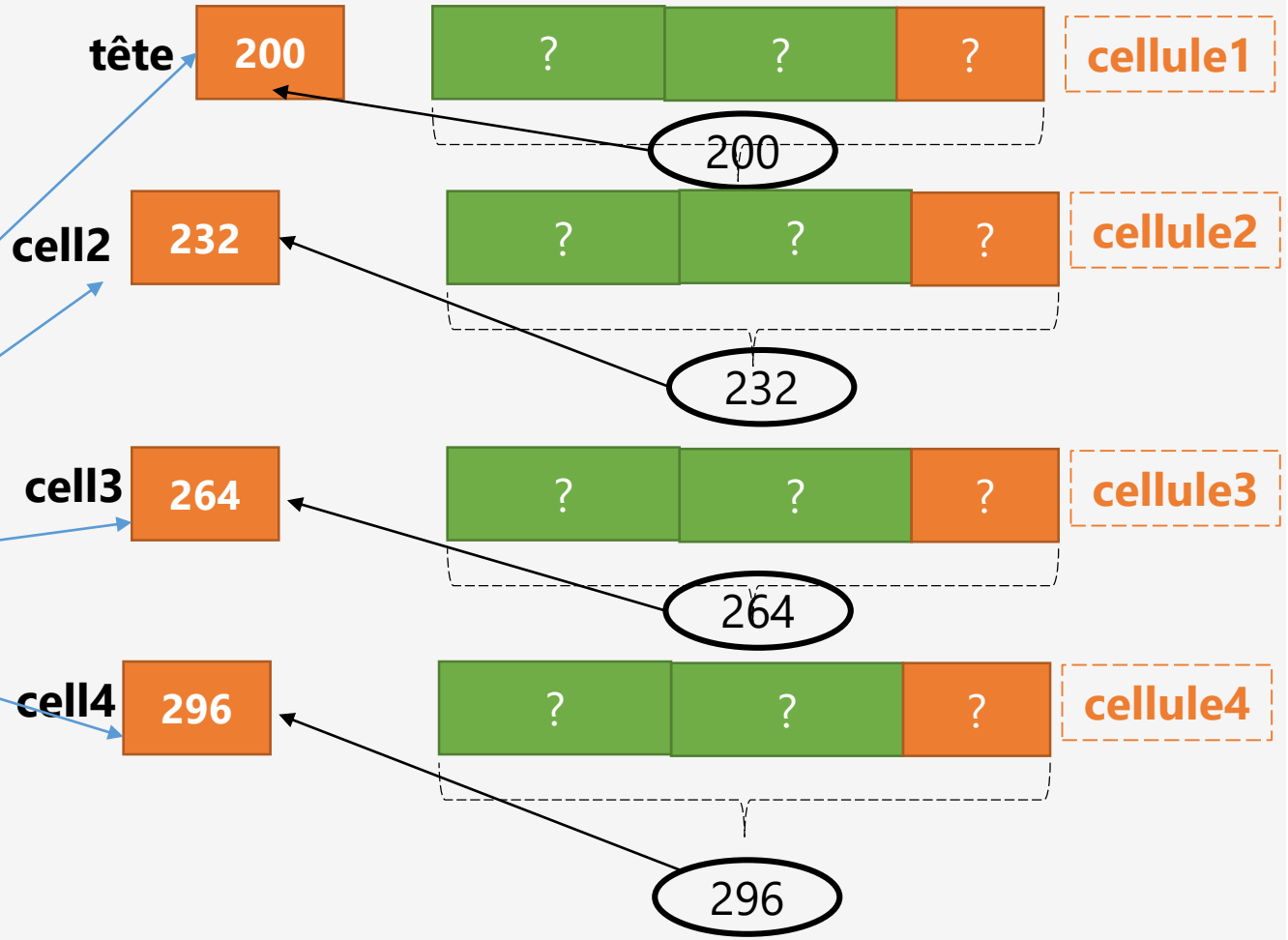
```
struct client {
            char nom_clt[20];
            long tel_clt;
            struct client  *next_clt;
};
struct client    *tete = NULL;
tete = malloc(sizeof(struct client);
```



3. Création de la liste chainée (création de cellules)

```
struct client {
        char nom_clt[20];
        long tel_clt;
        struct client *next_clt;
};
struct client   *tete = NULL; // déclaration de l'entête
tete = malloc(sizeof(struct client); // creation de la 1ère cellule client
struct client *cell2 = malloc(sizeof(struct client);
struct client * cell3 = malloc(sizeof(struct client);
struct client * cell4 = malloc(sizeof(struct client);
```

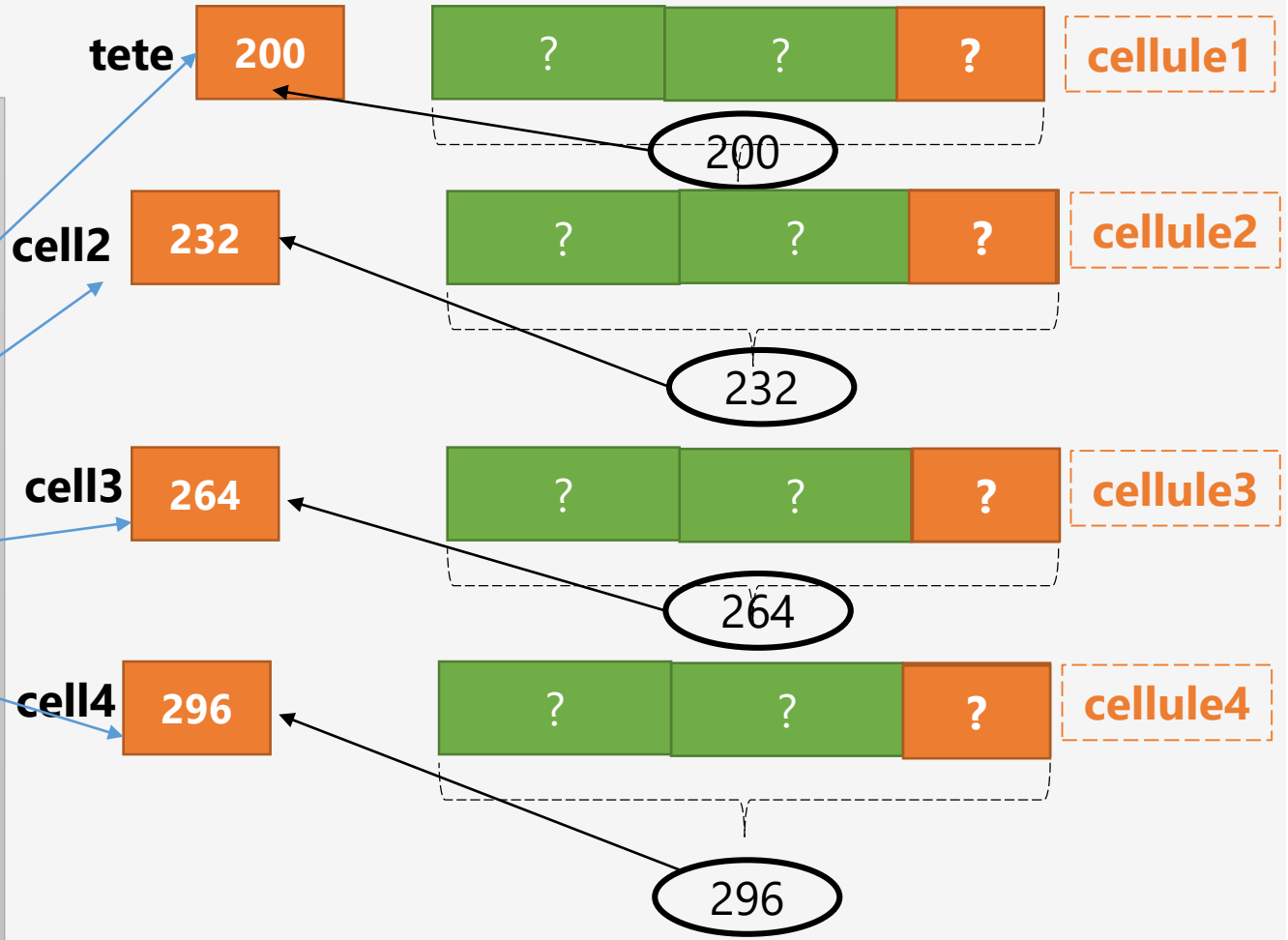
//Définition de la cellule



3. Création de la liste chainée (création de cellules)

```
struct client {
        char nom_clt[20];
        long tel_clt;
        struct client  *next_clt;
};
struct client    *tete = NULL;
tete = malloc(sizeof(struct client);
struct client * cell2 = malloc(sizeof(struct client);
struct client * cell3 = malloc(sizeof(struct client);
struct client * cell4 = malloc(sizeof(struct client);
tete ->next_clt = cell2;
cell2 ->next_clt = cell3;
cell3 ->next_clt = cell4;
cell4 ->next_clt = NULL;
```

Pour faire lier les cellules entre elles



3. Remplir les cellules

```
struct client {
            char nom_clt[20];
            long tel_clt;
            struct client  *next_clt;
};
struct client    *tete = NULL;
tete = malloc(sizeof(struct client);
struct client * cell2 = malloc(sizeof(struct client);
struct client * cell3 = malloc(sizeof(struct client);
struct client * cell4 = malloc(sizeof(struct client);
tete ->next_clt = cell2;
cell2 ->next_clt = cell3;
cell3 ->next_clt = cell4;
cell4 ->next_clt = NULL;
struct client *ptr =tete;
while (ptr != NULL)
{
   printf("Nom :\t");
   scanf("%s",&ptr->nom_clt);
   printf("Tel :\t");
   scanf("%d",&ptr->tel_clt);
   ptr = ptr->nextclt;
}
```

??

4. Parcourir une liste chainée (Affichage)

Exercice:
Ecrire une fonction qui permet d'afficher tous les clients (Nom + N° de téléphone) de la liste chainée créée précédemment.

```
void affichageList(struct client *p)
{
while (p!= NULL) {
printf("\nNom du client: %s", p->nom_clt);
printf("\nTel du client: %d", p->tel_clt);
printf("\n------------\n");
p=p->nextclt; }
}
int main()
 { ....
ptr=tete;
affichageList(ptr);   }
```

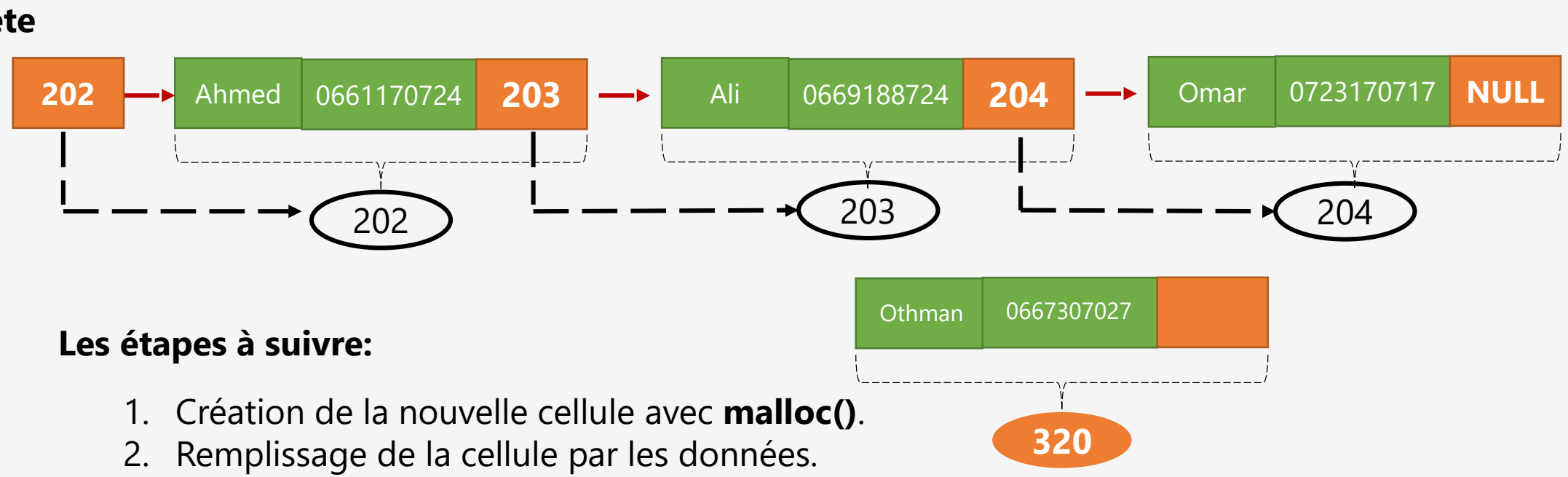
Appel de la fonction

Programme complet (création, liaison, remplissage, affichage)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct client {
    char nom_clt[10];
    int tel_clt;
    struct client *nextclt;
};
int main(){
   struct client *tete = NULL;
  tete = malloc(sizeof(struct client));
  struct client *ptr = tete;
  char reponse;
  do
  {
      printf("\n-Nom du client:\t");
      scanf("%s",&ptr->nom_clt);
      printf("Tel du client:\t");
      scanf("%d",&ptr->tel_clt);
      printf("Voulez-vous ajouter un nouveau client (O/N)?:");
      reponse = getche();
      if(reponse == 'O' || reponse == 'o')
      {
          ptr->nextclt=malloc(sizeof(struct client));
          ptr = ptr->nextclt;
      }
      else {ptr->nextclt = NULL;}
  } while ( reponse == 'O' || reponse == 'o');

  ptr = tete;
  while (ptr != NULL)
  {
      printf("\n Nom du client : %s\n", ptr->nom_clt);
    printf("Tel du client : %d\n", ptr->tel_clt);
    ptr = ptr->nextclt; // Passing to the next node
  }

    return 0;
}
```

Travail à faire "Faire une description schématique"

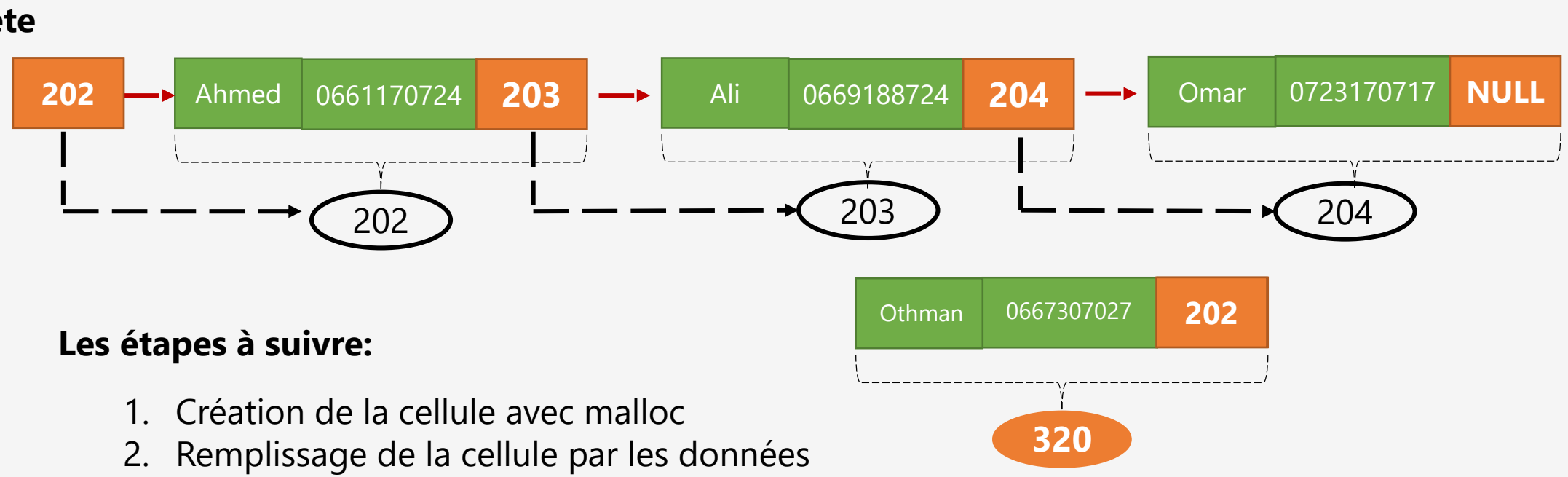
5. L'ajout d'une cellule (client) au début de la liste chainée.



Les étapes à suivre:

1. Création de la nouvelle cellule avec **malloc()**.
2. Remplissage de la cellule par les données.
3. Faire la liaison entre le nouvelle cellule et la première cellule de la liste chainée.
4. Modification de la tête de la liste pour qu'elle pointe sur la nouvelle cellule.

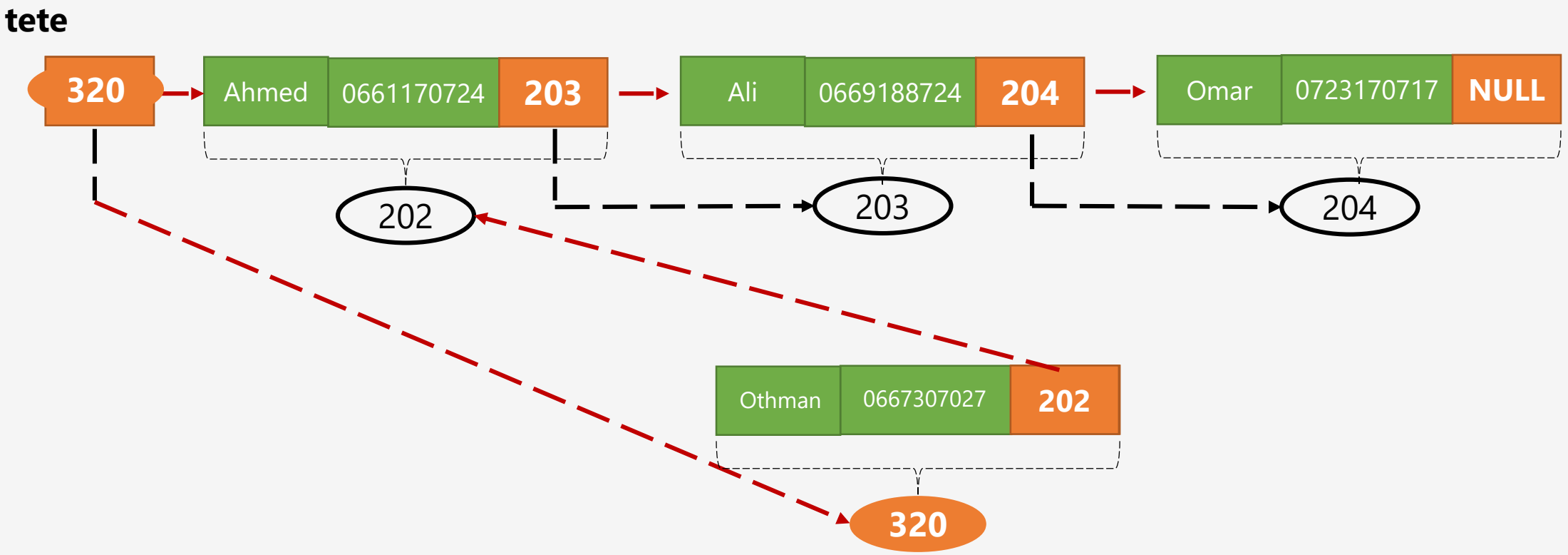
5. L'ajout d'une cellule (client) au début de la liste chainée.



Les étapes à suivre:

1. Création de la cellule avec malloc
2. Remplissage de la cellule par les données
3. Faire la liaison entre le nouvelle cellule et la première cellule de la liste chainée.
4. Modification de la tête de la liste pour qu'elle pointe sur la nouvelle cellule.

5. L'ajout d'une cellule (client) au début de la liste chainée.



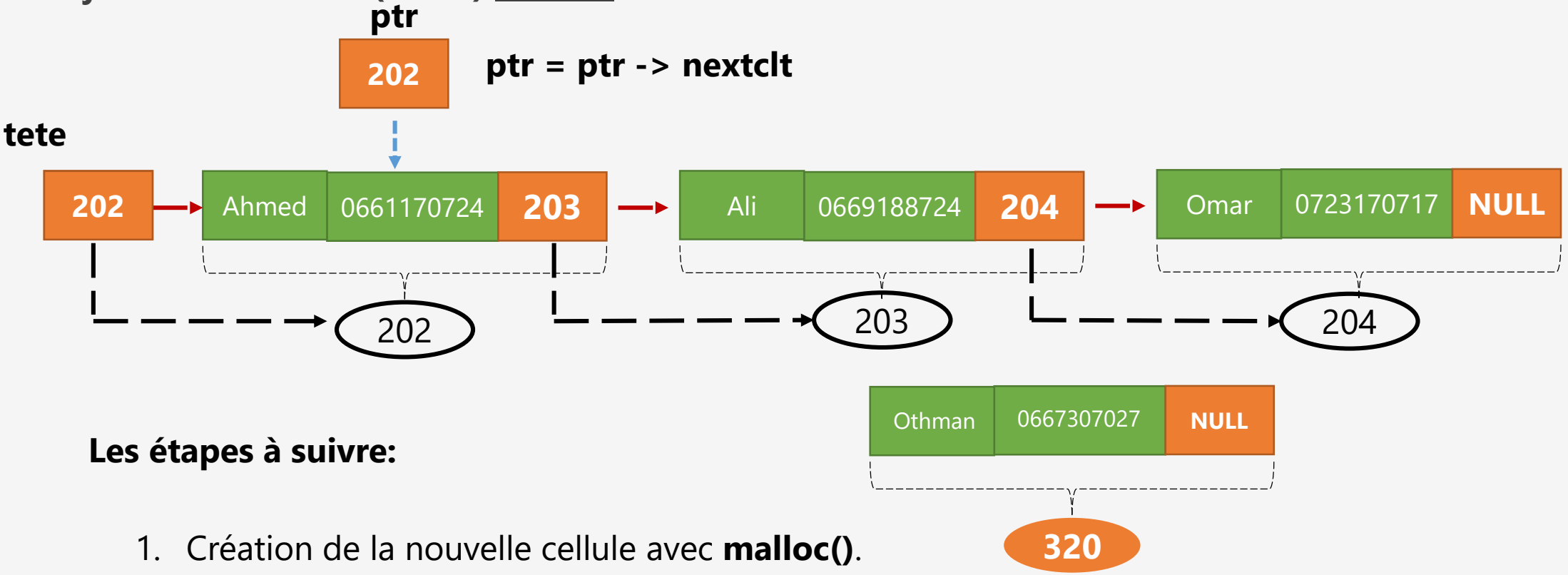
5. L'ajout d'une cellule (client) au début de la liste chainée.

Exercice:

En utilisant une fonction ajoutDebut(), écrire un programme qui permet d'ajouter des nouveaux clients au début de la liste chainée.
(le nombre de client à ajouter est fixé par l'utilisateur).

```c
//---Défintion de la fonction qui ajoute des nouveaux clients au début de la liste
struct client* ajoutDebut(struct client *t)
{
    struct client *NewClient;
    NewClient = malloc(sizeof(struct client));
    printf("\n Nom du client SVP:\t");
    scanf("%s", &NewClient->nom_clt);
    printf("\n Tel du client SVP:\t");
    scanf("%d", &NewClient->tel_clt);
    NewClient->next_clt = t;
    return NewClient;
}
int main()
{
    int i,nbr_clt_ajt;
    struct client *tete = NULL;
    tete = initialiseList(tete); //appel de la fonction qui initialise la liste
    afficheList(tete);// appel de la fonction qui affiche les clients de la liste
    printf("Donner le nombre de clients a ajouter\n");
    scanf("%d",&nbr_clt_ajt);
    for (int i=1; i<=nbr_clt_ajt; i++)
    {tete=ajoutDebut(tete);}
    afficheList(tete);
    return 0;
}
```

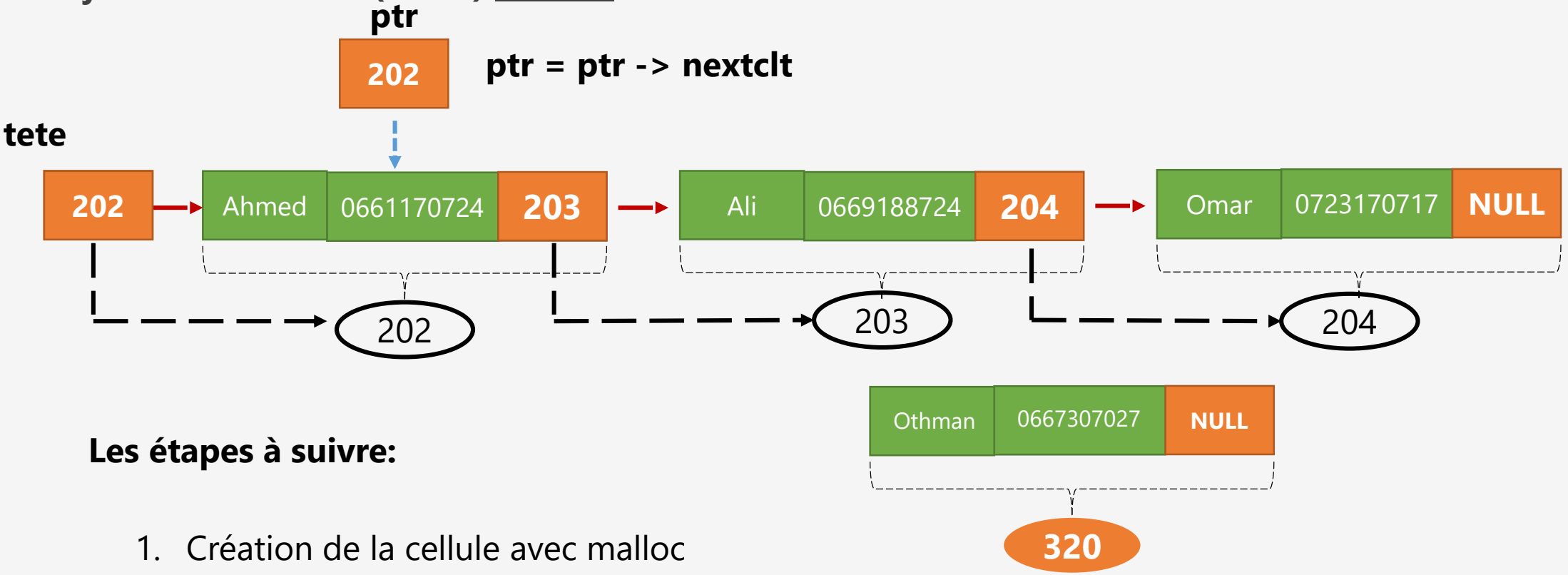
6. L'ajout d'une cellule (client) à la fin de la liste chainée.



Les étapes à suivre:

1. Création de la nouvelle cellule avec **malloc()**.
2. Remplissage de la cellule par les données + mettre la valeur **NULL** dans le champ **next_clt**
3. Parcourir la liste jusqu'à l'arrivé à l'adresse de la dernière cellule de la liste.

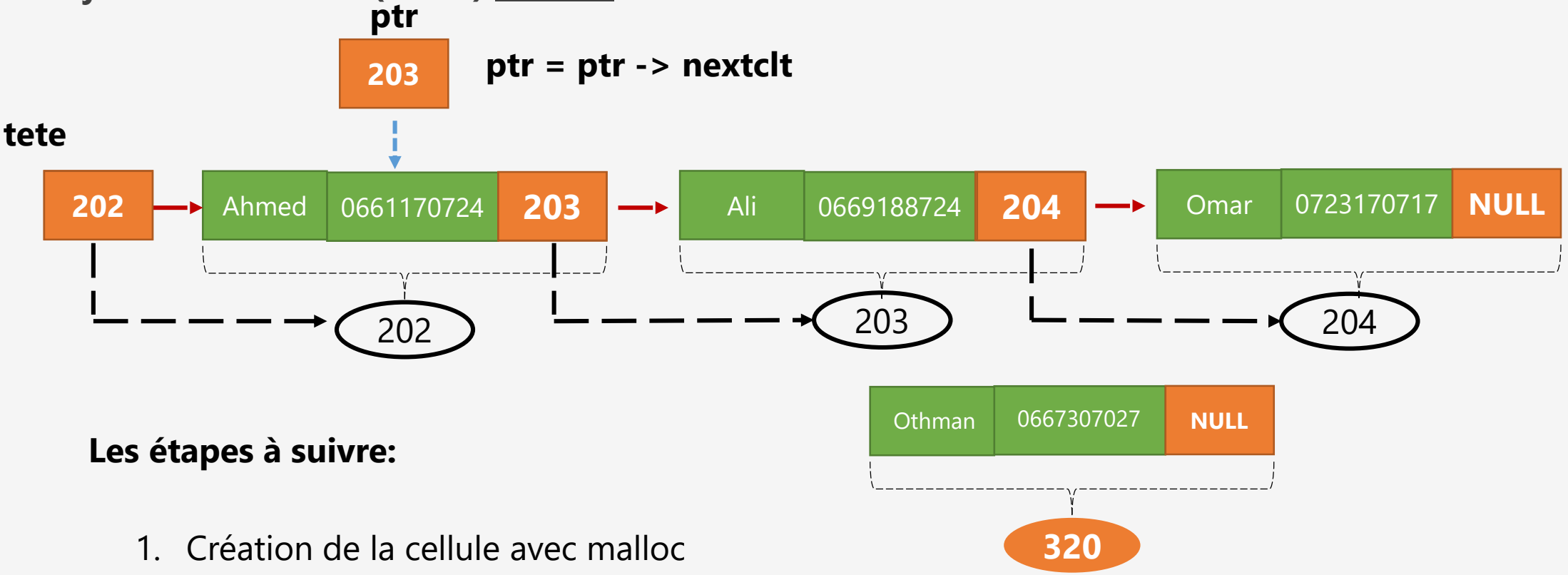
6. L'ajout d'une cellule (client) à la fin de la liste chainée.



Les étapes à suivre:

1. Création de la cellule avec malloc
2. Remplissage de la cellule par les données + mettre la valeur **NULL** dans le champ **next_clt**
3. Parcourir la liste jusqu'à l'arrivé à l'adresse de la dernière cellule de la liste.

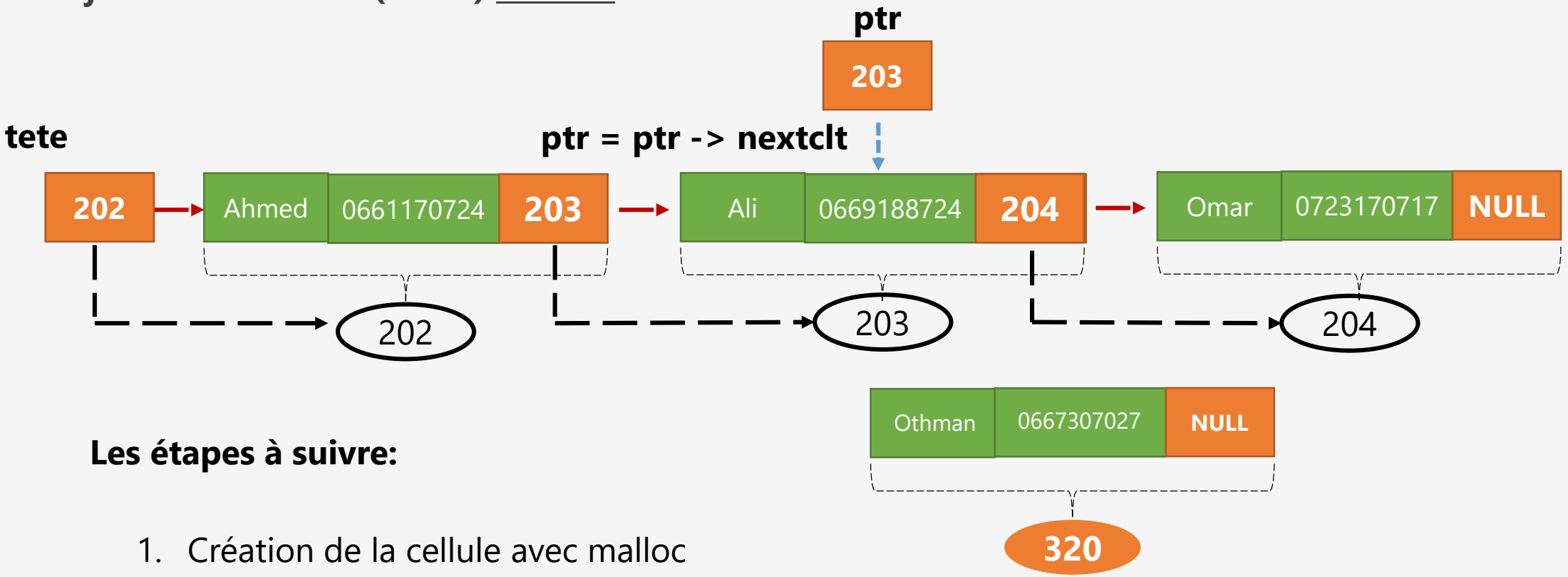
6. L'ajout d'une cellule (client) à la fin de la liste chainée.



Les étapes à suivre:

1. Création de la cellule avec malloc
2. Remplissage de la cellule par les données + mettre la valeur **NULL** dans le champ **next_clt**
3. Parcourir la liste jusqu'à l'arrivé à l'adresse de la dernière cellule de la liste.

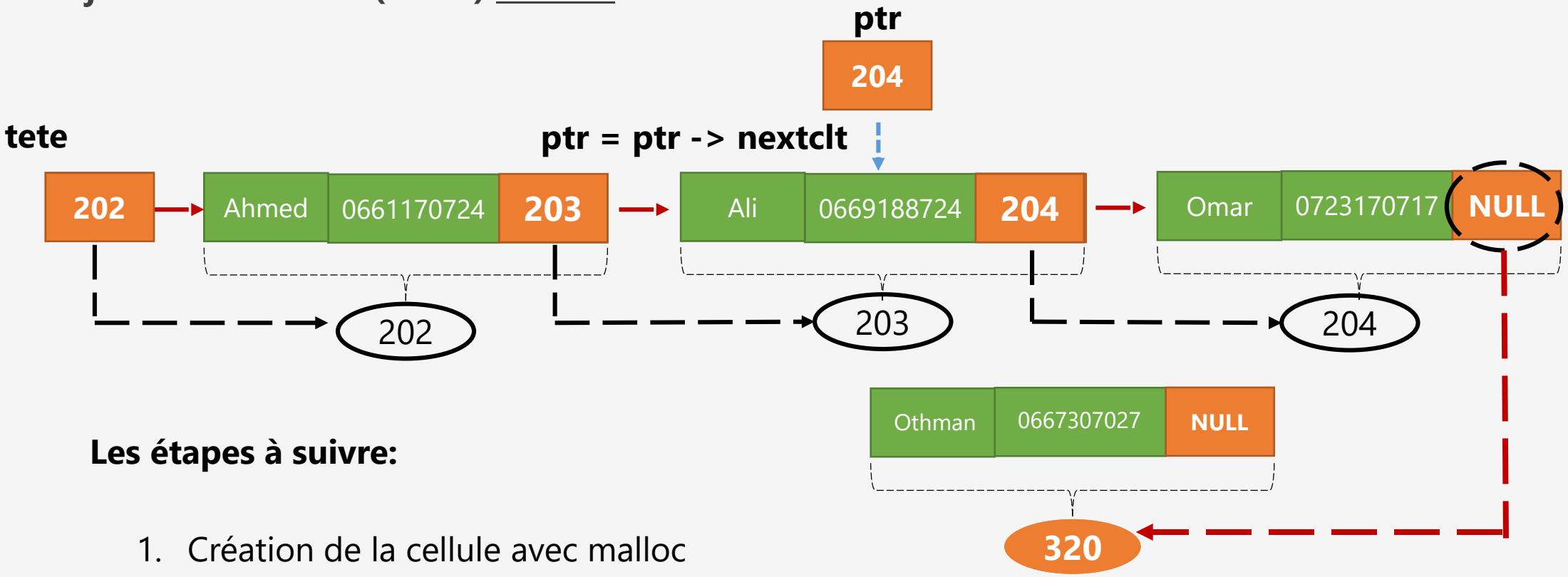
6. L'ajout d'une cellule (client) à la fin de la liste chainée.



Les étapes à suivre:

1. Création de la cellule avec malloc
2. Remplissage de la cellule par les données + mettre la valeur **NULL** dans le champ **next_clt**
3. Parcourir la liste jusqu'à l'arrivé à l'adresse de la dernière cellule de la liste.

6. L'ajout d'une cellule (client) à la fin de la liste chainée.



Les étapes à suivre:

1. Création de la cellule avec malloc
2. Remplissage de la cellule par les données + mettre la valeur **NULL** dans le champ **next_clt**
3. Parcourir la liste jusqu'à l'arrivé à l'adresse de la dernière cellule de la liste.
4. Faire lier l'ancienne dernière cellule avec la nouvelle.

6. L'ajout d'une cellule (client) à la fin de la liste chainée.

Exercice:

En utilisant une fonction ajoutFin(), écrire un programme qui permet d'ajouter des nouveaux clients à la fin de la liste chainée.

@ de la dernière cellule

```c
//Définition de la fonction qui ajoute des clients à la fin de la liste.
struct client* ajoutFin(struct client * t)
{
   struct client *NewClient = malloc(sizeof(struct client));
   printf("Saisir le nom du client:\t");
   scanf("%s",&NewClient->nom_clt);
   printf("Saisir Tel du client:\t");
   scanf("%d",&NewClient->tel_clt);
   NewClient->next_clt=NULL; // le dernier client
   struct client *ptr = t;
            while(ptr->next_clt != NULL)
            {
                ptr = ptr->next_clt;
            }
            ptr->next_clt = NewClient;
    return t;
}
int main()
{
    int i,nbr_clt_ajt;
    struct client *tete = NULL;
    tete = initialiseList(tete); //appel de la fonction qui initialise la liste
    afficheList(tete);// appel de la fonction qui affiche les clients de la liste
    printf("Donner le nombre de clients a ajouter\n");
    scanf("%d",&nbr_clt_ajt);
    for (int i=1; i<=nbr_clt_ajt; i++)
    {tete=ajoutDebut(tete);}
     printf("---------Affichage apres un ajout au debut---------\n");
    afficheList(tete);
    tete = ajoutFin(tete);
    printf("---------Affichage apres un ajout a la fin---------\n");
    afficheList(tete);
    return 0;
}
```

La taille et la recherche dans une liste chainée

Exercice:

Ecrire trois fonctions **tailleList () et rechercheList()** qui permettent de:

- **tailleList () retourne** sa taille (le nombre de cellules)
- **rechercheList() recherche** si un client existe dans la liste (Exemple liste client).

Chapitre 2: Les listes chainées

Taille d'une liste chainée.

```
//Défintion de la fonction qui permet de calculer la taille d'une liste.
int tailleList(struct client *t)
{
    int nbr_clt = 0;
    struct client *ptr = t;
    while (ptr != NULL)
    {
        nbr_clt++;
        ptr = ptr->next_clt;
    } return nbr_clt;
}
```

Chapitre 2: Les listes chainées

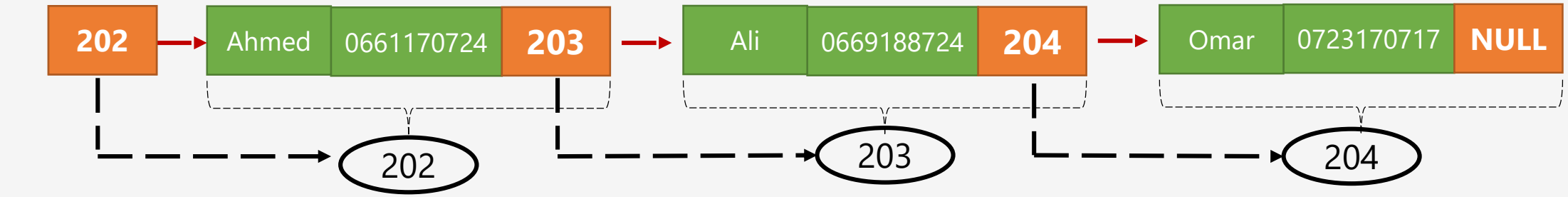
Recherche un élément dans une liste chainée.

```
//Défintion de la fonction qui permet de rechercher un élément dans une liste
bool rechercheList(struct client *t, char Xclient[20])
{
    struct client *ptr = t;
    while (ptr != NULL)
    {
        if(strcmp(ptr->nom_clt,Xclient) == 0)
           {
               return true; //return arrete la fonction
               //break;
           } else {ptr= ptr->next_clt;}
    } return false;
}
```

Appel de la fonction

```
printf("\n -----------Recherche d'un client-----------\n");
if(rechercheList(tete, "Ali") == true) printf ("Le client existe\n");
else printf ("Le client n'existe pas\n");
```

7. L'ajout d'une cellule (client) au milieu de la liste chainée.



Objectif: Ajouter un nouveau client après le client **Ali**

Les étapes à suivre:

1. Vérifier que la liste n'est pas vide (**tete != NULL**).
2. Parcourir la liste jusqu'à l'arrivé à la cellule ou le client concerné **clientX(=Ali)**.
3. Si le client existe, on va créer une nouvelle cellule avec **malloc()**.
4. Remplir la nouvelle cellule par des données (Nom et Tél du client).
5. Faire lier le nouveau client avec le client qui suive le client recherché **clientX(=Ali)**.
6. Faire lier le client recherché **clientX(=Ali)** avec le nouveau client.

N.B: l'ajout au milieu d'une liste chainée ne modifié par l'adresse de la tête de la liste

7. L'ajout d'une cellule (client) au milieu de la liste chainée.

Exercice:

En utilisant une procédure ajoutMilieu(), écrire un programme qui permet d'ajouter un nouveau client <u>après le client "Ali"</u> .

```c
//Défintion de la procédure qui ajoute un client au milieu d'une liste
void ajoutMilieu(struct client *t, char clientX[20]){
    if(t == NULL) printf("Desole la liste est vide, vous ne pouvez pas ajouter des clients au milieu\n");
    else {
            struct client *ptr= t;
            bool exist = false;
            while(ptr != NULL && exist== false)
            {
                if(strcmp(ptr->nom_clt, clientX)==0) {
                        exist = true;
                   struct client *NewClient = malloc(sizeof(struct client));
                   printf("Saisir le nom du client:\t");
                   scanf("%s",&NewClient->nom_clt);
                   printf("Saisir Tel du client:\t");
                   scanf("%d",&NewClient->tel_clt);
                   NewClient->next_clt= ptr->next_clt;
                   ptr->next_clt = NewClient;
                  // ptr = NULL;
                } else ptr = ptr->next_clt;
            } if(exist == false) {
                                    printf("Le client %s n'existe pas dans la liste\
                                    }
        }

}
```

```c
int main()
{
    int i,nbr_clt_ajt;
    struct client *tete = NULL;
    tete = initialiseList(tete); //appel de la fonction qui initialise la liste
    afficheList(tete);// appel de la fonction qui affiche les clients de la liste
    printf("\n -----------Ajout au debut-----------\n");
    printf("Donner le nombre de clients a ajouter\n");
    scanf("%d",&nbr_clt_ajt);
    for (int i=1; i<=nbr_clt_ajt; i++)
    {tete=ajoutDebut(tete);}
     printf("---------Affichage apres un ajout au debut---------\n");
    afficheList(tete);
    printf("\n -----------Ajout a la fin-----------\n");
    tete = ajoutFin(tete);
    printf("---------Affichage apres un ajout a la fin---------\n");
    afficheList(tete);
    printf("\n -----------Ajout au milieu-----------\n");
    ajoutMilieu(tete, "Ali");
    printf("---------Affichage apres un ajout au milieur---------\n");
    afficheList(tete);
    return 0;
}
```