

# Chapitre 13

## Structures de données dynamiques : les listes chaînées

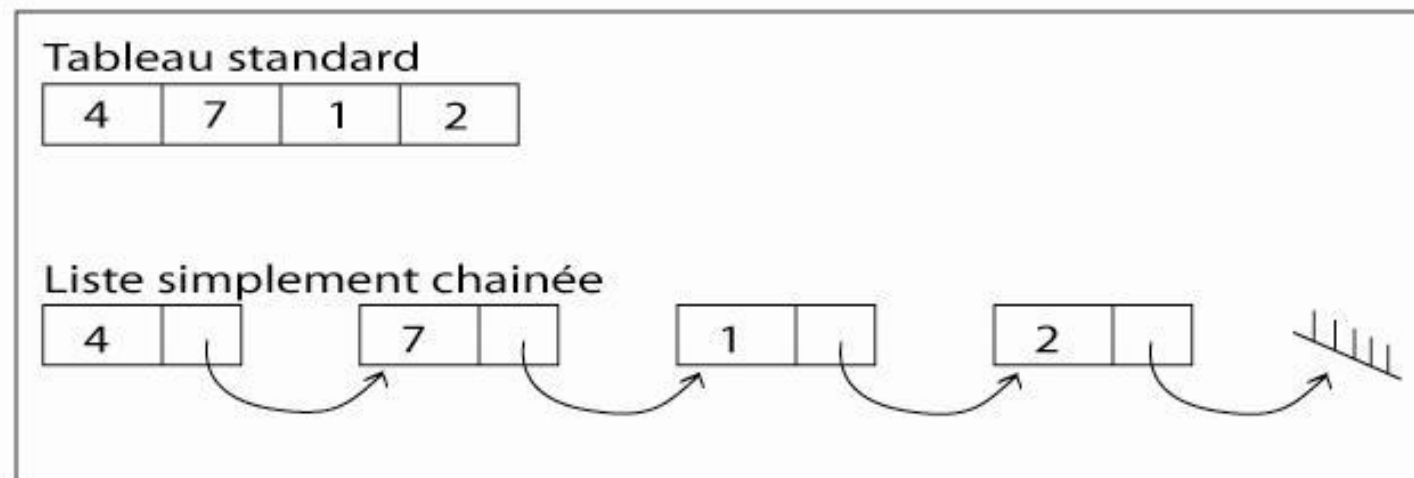
# Listes chaînées

## Problème

Données dont on ne connaît **pas la taille**.

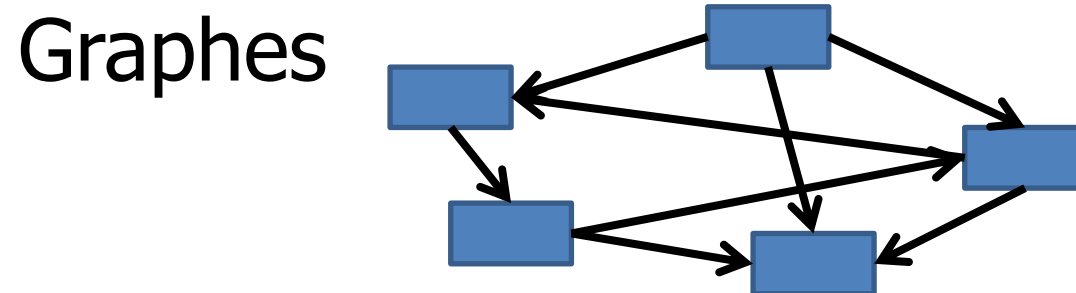
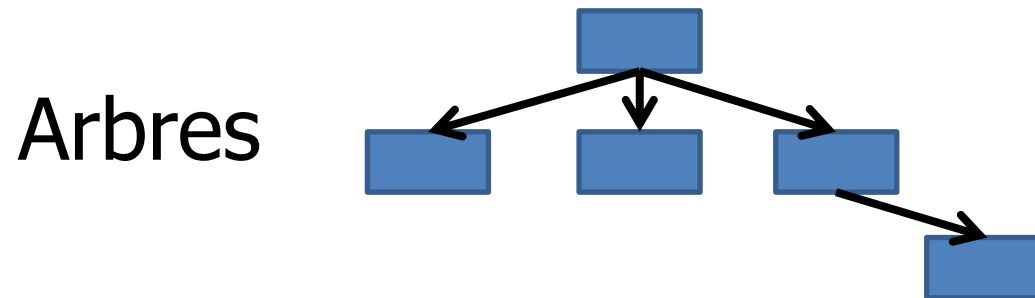
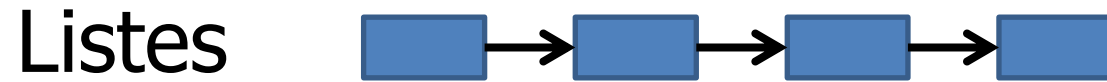
Données dont la **quantité et l'organisation varient** dans le temps.

**Solution : les structures dynamiques.**



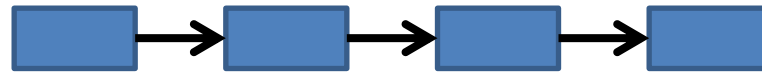
# Listes chaînées

## Structures dynamiques

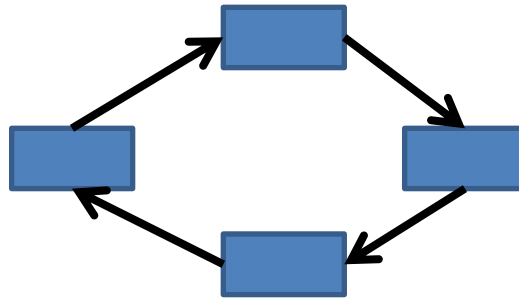


# Listes chaînées

Liste linéaire



Liste circulaire



Liste simplement chaînée

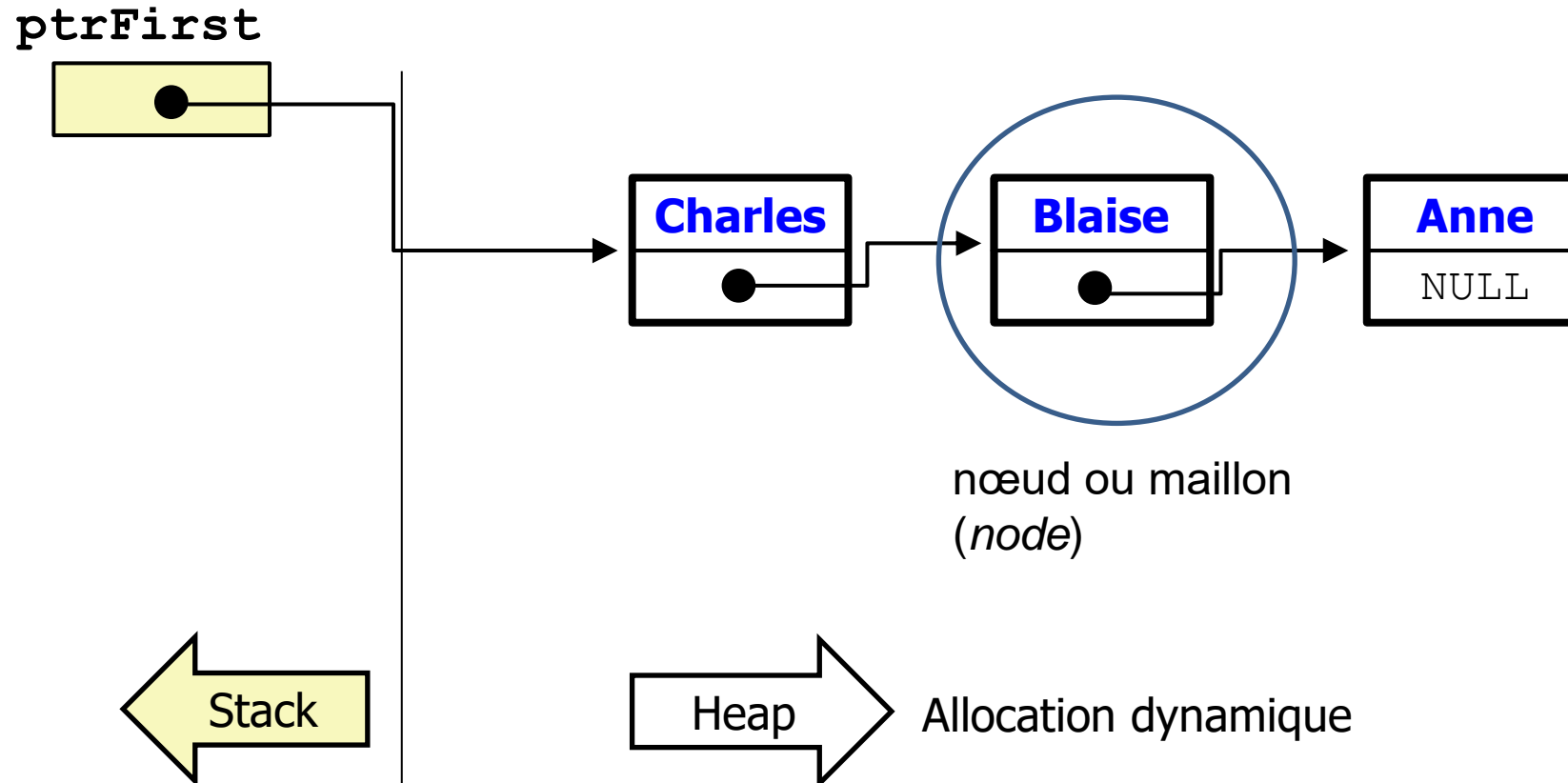


Liste doublement chaînée



# Listes chaînées : exemple

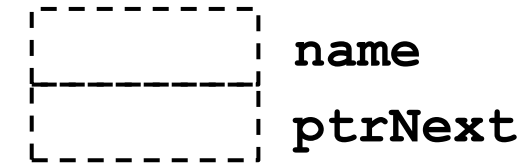
Liste simplement chaînée contenant des noms



# Listes chaînées : exemple

Déclaration d'une structure `node`

```
struct node
{
    char        name[20];
    struct node *ptrNext;
};
```



Déclaration d'un pointeur

```
struct node *ptrFirst = NULL;
```



Alternative avec `typedef`

```
typedef struct node NodeType;
NodeType *ptrFirst = NULL;
```

# Listes chaînées : opérations

## Opérations de base

**Initialisation** du pointeur sur le 1<sup>er</sup> nœud

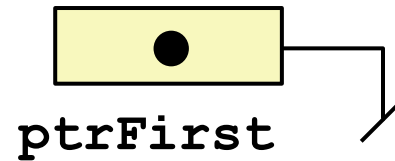
**Ajout** d'un nouveau nœud

- Allocation mémoire d'un nouveau nœud
- Initialisation du nouveau nœud
- Ajout du nouveau nœud au pointeur de liste
- Ajout d'un nouveau nœud en début de chaîne

**Affichage** de la liste

# Listes chaînées : initialisation

**Initialisation** du pointeur sur le premier noeud



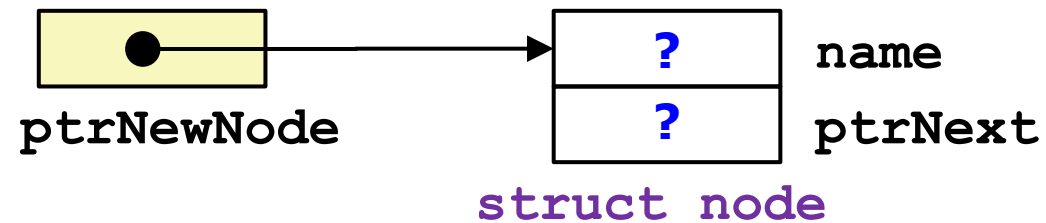
```
ptrFirst = NULL;
```



# Listes chaînées : ajout

## Ajout d'un nouveau nœud

1) Allocation en mémoire d'un **nouveau** nœud

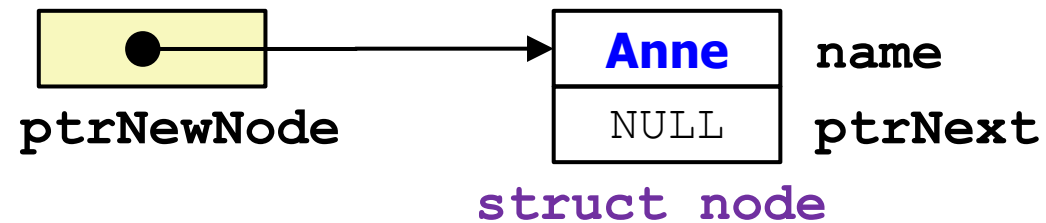


```
struct node* ptrNewNode = malloc(sizeof(struct node));
```

# Listes chaînées : ajout

**Ajout** d'un nouveau nœud (suite)

2) Initialisation en mémoire d'un **nouveau** nœud



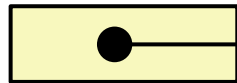
```
gets(ptrNewNode->name) ;  
ptrNewNode->ptrNext = NULL;
```

# Listes chaînées : ajout

**Ajout** d'un nouveau nœud (suite)

3) Ajout du nouveau nœud au **pointeur de liste**

ptrNewNode



Anne

name

NULL

ptrNext

NULL

ptrFirst

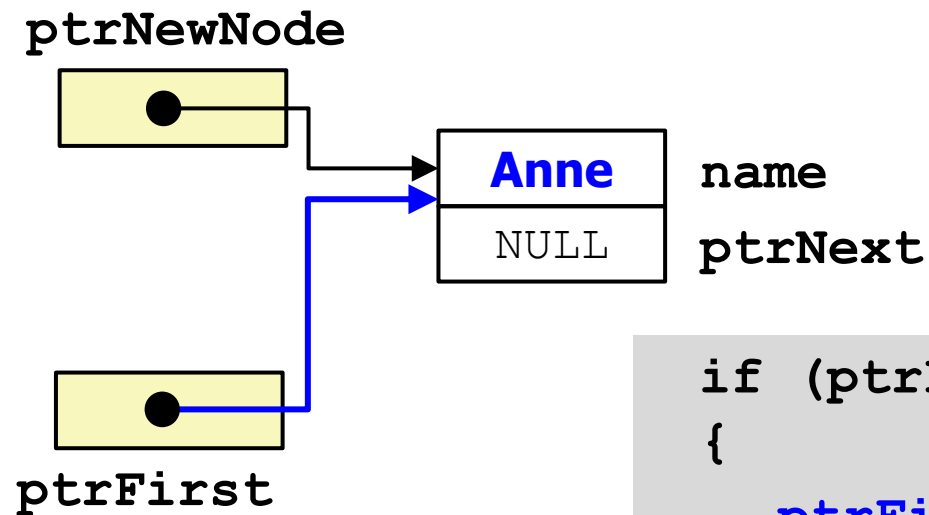
```
if (ptrFirst == NULL)
{

}
```

# Listes chaînées : ajout

**Ajout** d'un nouveau nœud (suite)

3) Ajout du nouveau nœud au **pointeur de liste**



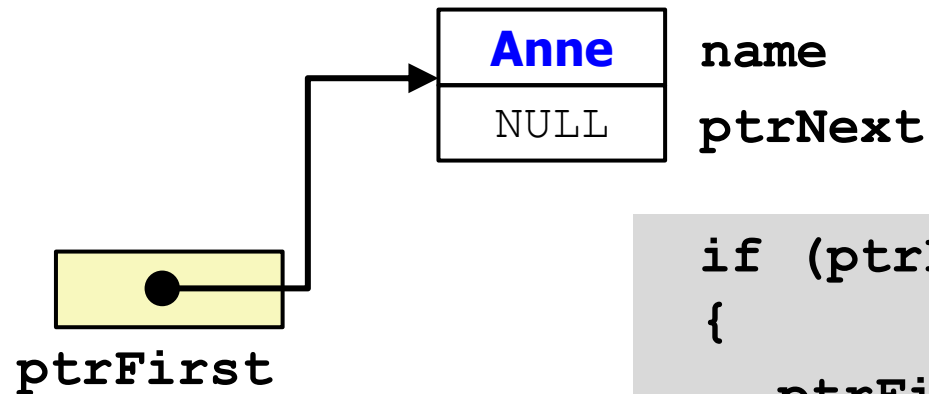
```
if (ptrFirst == NULL)
{
    ptrFirst=ptrNewNode;
}
```

# Listes chaînées : ajout

**Ajout** d'un nouveau nœud (suite)

3) Ajout du nouveau nœud au **pointeur de liste**

ptrNewNode



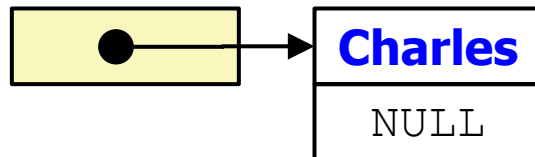
```
if (ptrFirst == NULL)
{
    ptrFirst=ptrNewNode;
}
```

# Listes chaînées : ajout

**Ajout** d'un nouveau nœud (suite)

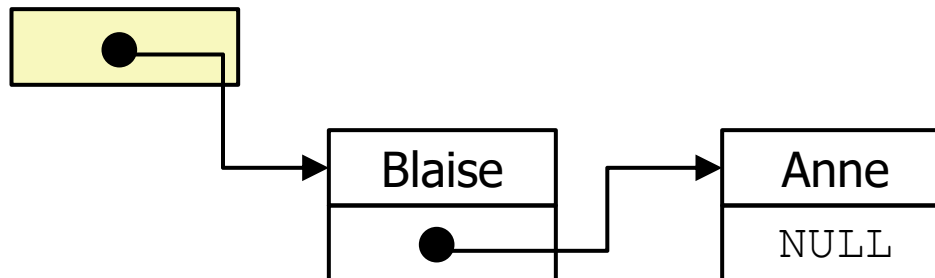
4) Ajout d'un nœud **au début** de chaîne

ptrNewNode



```
if (ptrFirst != NULL)
{
}
}
```

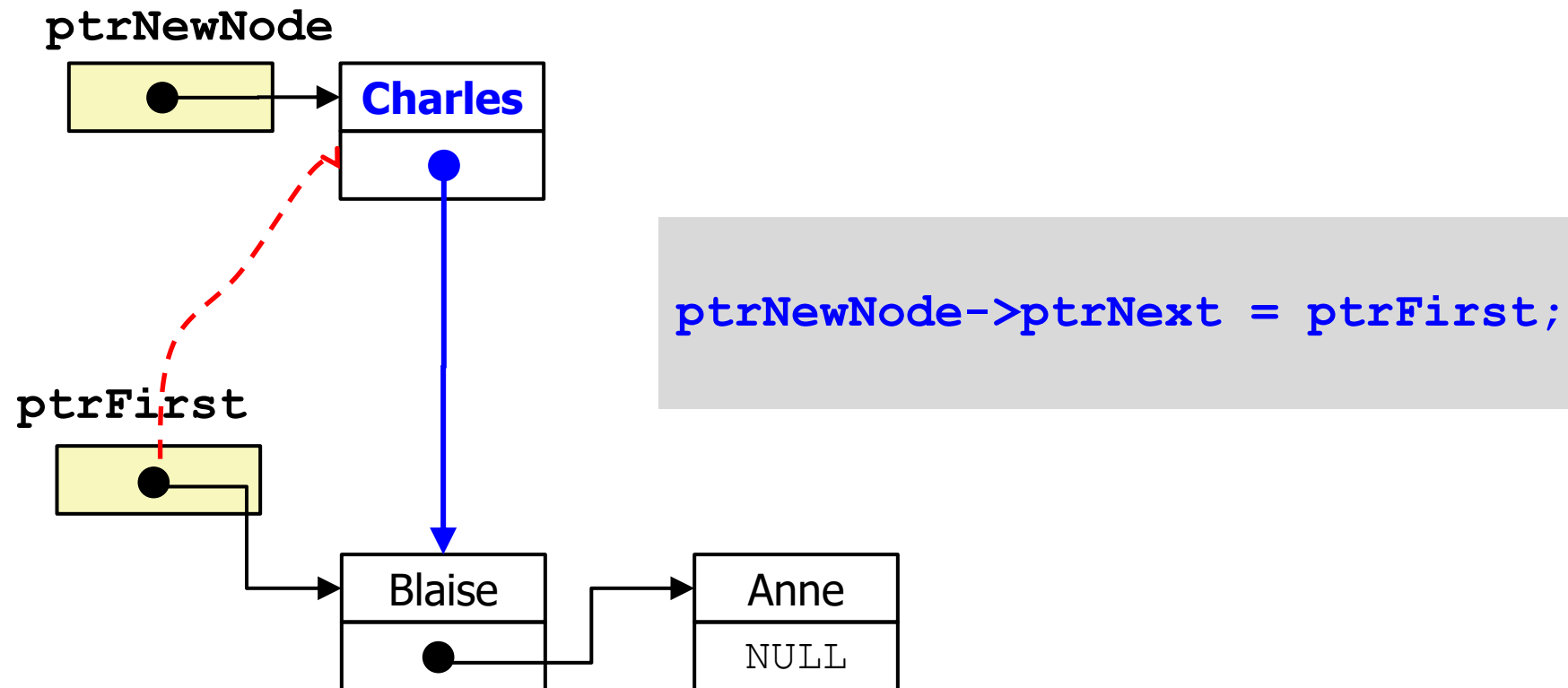
ptrFirst



# Listes chaînées : ajout

## Ajout d'un nouveau nœud (suite)

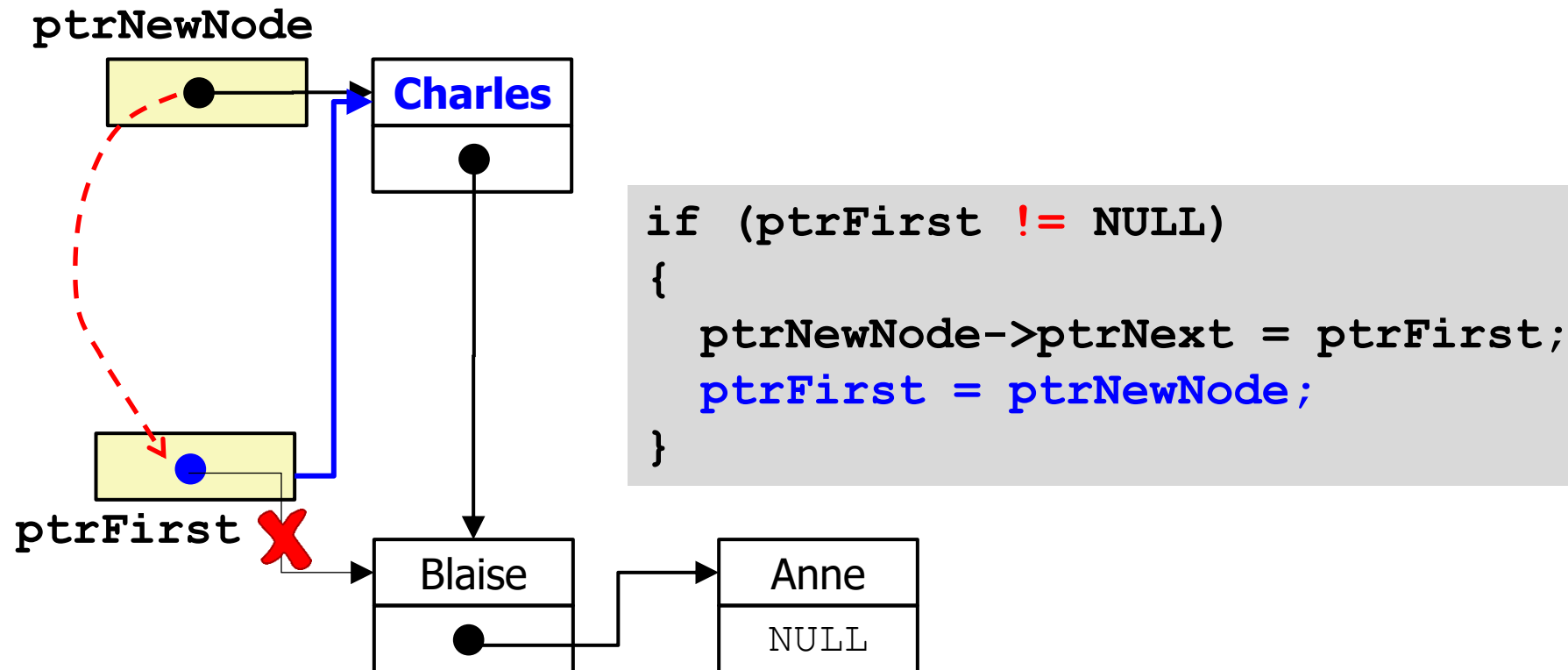
### 4) Ajout d'un nœud **au début** de chaîne



# Listes chaînées : ajout

## Ajout d'un nouveau nœud (suite)

### 4) Ajout d'un nœud **au début** de chaîne

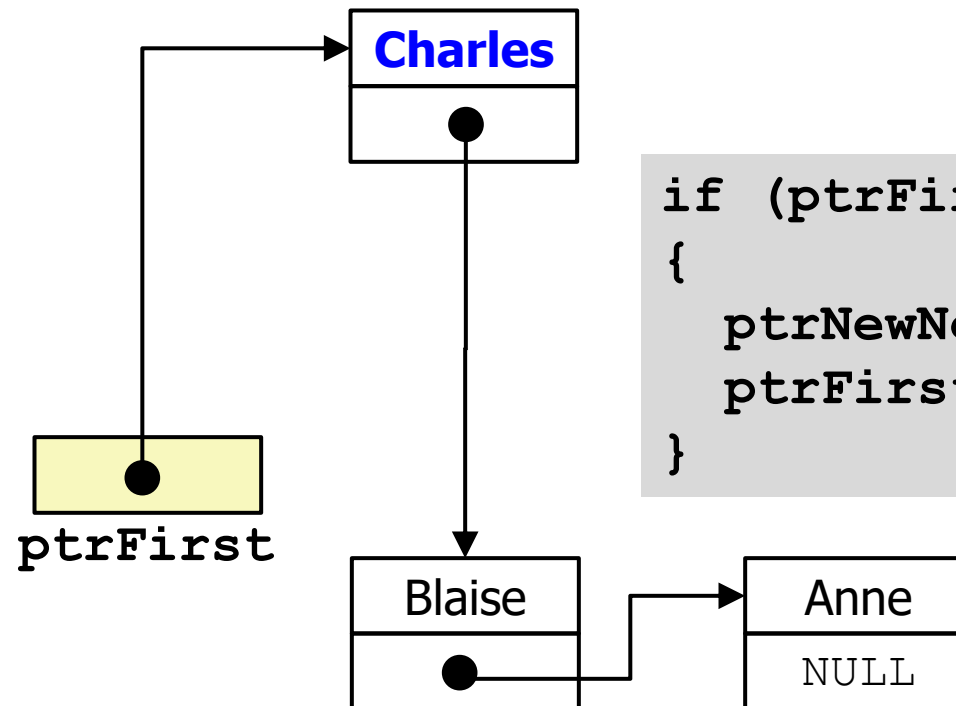




# Listes chaînées : ajout

**Ajout** d'un nouveau nœud (suite)

4) Ajout d'un nœud **au début** de chaîne



```
if (ptrFirst != NULL)
{
    ptrNewNode->ptrNext = ptrFirst;
    ptrFirst = ptrNewNode;
}
```

# Listes chaînées : ajout

## Ajout d'un nouveau nœud (suite)

On peut simplifier :

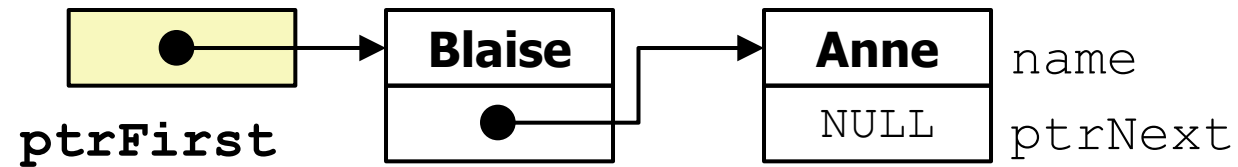
```
ptrNewNode->ptrNext = NULL;
if (ptrFirst == NULL)
    ptrFirst = ptrNewNode;
else
{
    ptrNewNode->ptrnext = ptrFirst;
    ptrFirst             = ptrNewNode;
}
```

Comme ceci :

```
ptrNewNode->ptrNext = ptrFirst;
ptrFirst            = ptrNewNode;
```

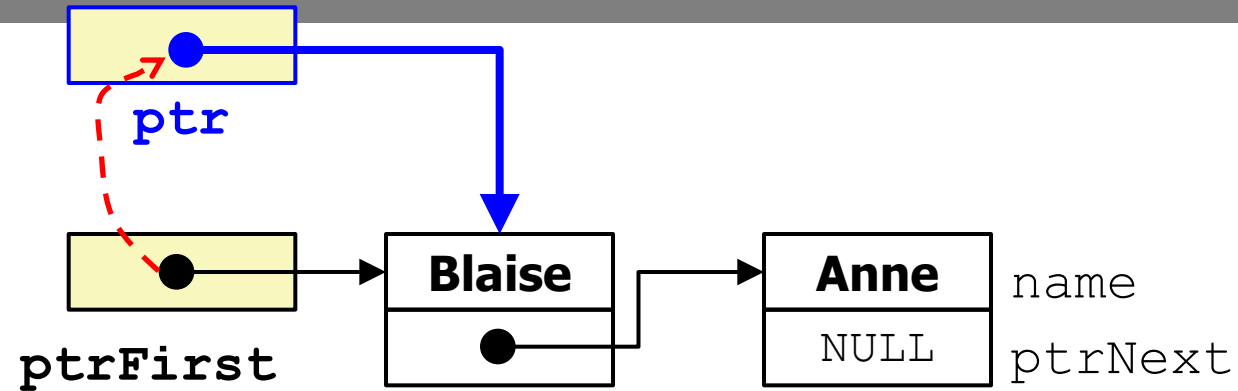
# Listes chaînées : affichage

## Affichage



# Listes chaînées : affichage

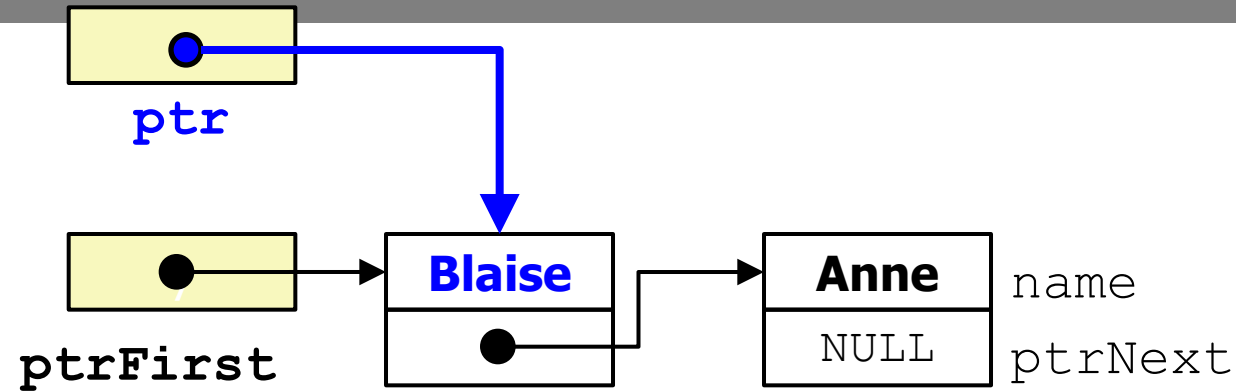
**Affichage**



```
struct node* ptr;  
ptr = ptrFirst;  
while (ptr != NULL)  
{  
    printf("%s\n", ptr->name);  
    ptr = ptr->ptrNext;  
}
```

# Listes chaînées : affichage

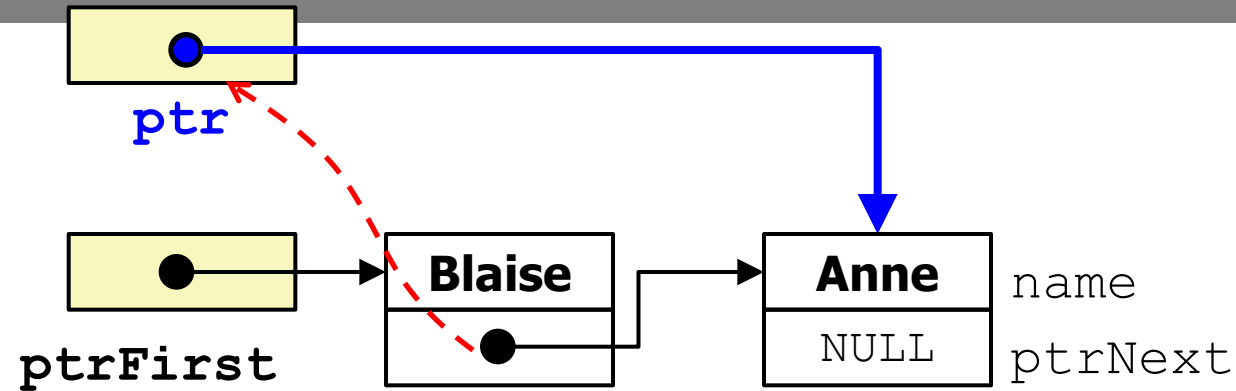
**Affichage**



```
struct node* ptr;  
ptr = ptrFirst;  
while (ptr != NULL) //(true)  
{  
    printf("%s\n", ptr->name);  
    ptr = ptr->ptrNext;  
}
```

# Listes chaînées : affichage

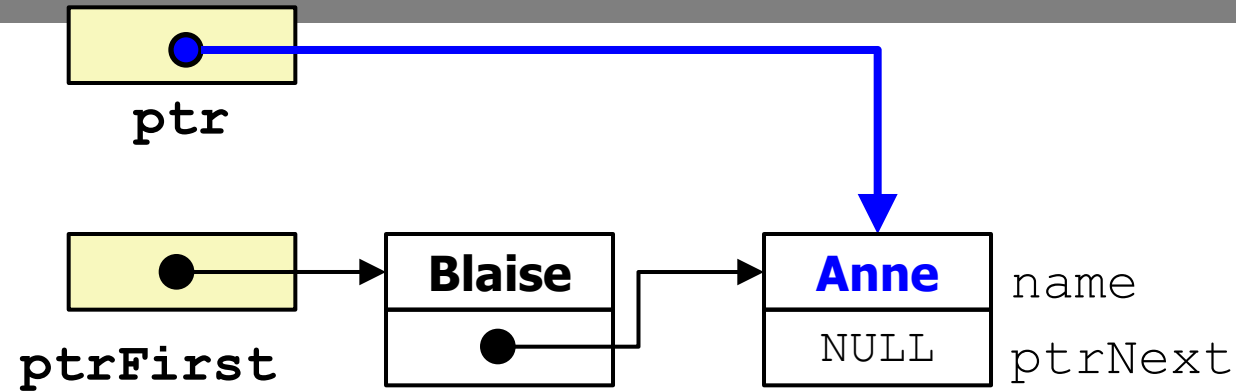
**Affichage**



```
struct node* ptr;  
ptr = ptrFirst;  
while (ptr != NULL)  
{  
    printf("%s\n", ptr->name);  
    ptr = ptr->ptrNext;  
}
```

# Listes chaînées : affichage

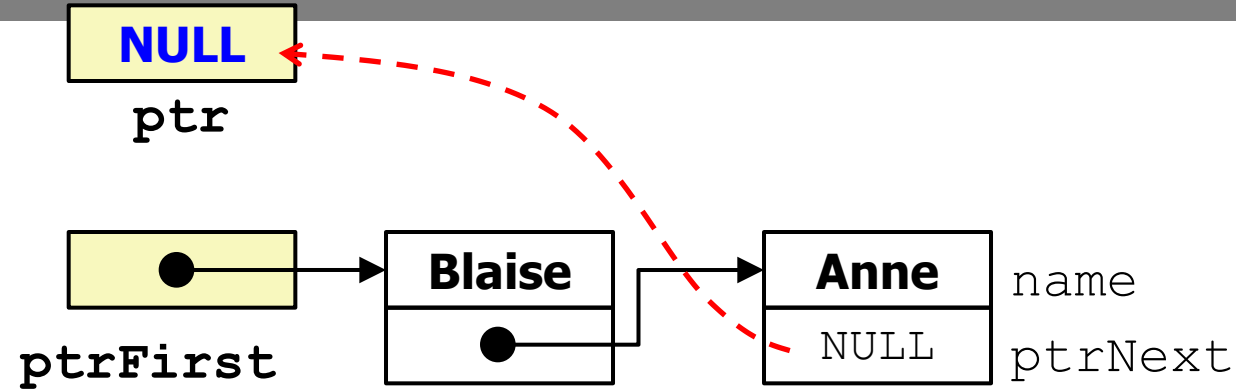
**Affichage**



```
struct node* ptr;  
ptr = ptrFirst;  
while (ptr != NULL) //(true)  
{  
    printf("%s\n", ptr->name);  
    ptr = ptr->ptrNext;  
}
```

# Listes chaînées : affichage

**Affichage**



```
struct node* ptr;  
ptr = ptrFirst;  
while (ptr != NULL)  
{  
    printf("%s\n", ptr->name);  
    ptr = ptr->ptrNext;  
}
```

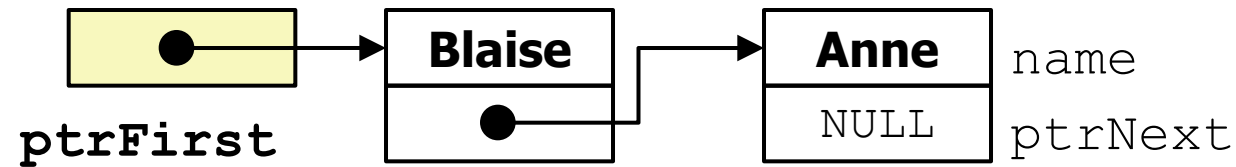


# Listes chaînées : affichage

Affichage

NULL

ptr



```
struct node* ptr;  
ptr = ptrFirst;  
while (ptr != NULL) → false  
{  
    printf("%s\n", ptr->name);  
    ptr = ptr->ptrNext;  
}
```

# Listes chaînées : programme (1)

```
#include <stdio.h>

struct node
{
    char name[20];
    struct node* ptrNext;
};

struct node *ptrFirst=NULL;

void addNode(void) ;
void displayList (void) ;
```

# Listes chaînées : programme (2)

```
int main(void)
{
    char choice;
    printf("Insert a new name (Y/N)? ");
    scanf(" %c", &choice);
    while (toupper(choice) != 'N')
    {
        addNode();
        printf("Insert a new name (Y/N)? ");
        scanf(" %c", &choice);
    }
    displayList();
    return 0;
}
```

# Listes chaînées : programme (3)

```
void addNode(struct node *ptrFirst)
{
    struct node* ptrNewNode;
    ptrNewNode = malloc(sizeof(struct node));

    printf("Enter a name: ");
    fflush(stdin);
    gets(ptrNewNode->name);

    ptrNewNode->ptrNext = ptrFirst;
    ptrFirst             = ptrNewNode;
}
```

# Listes chaînées : programme (4)

```
void displayList(struct node *ptrFirst)
{
    struct node * ptr;
    ptr = ptrFirst;
    if (ptr == NULL)
        printf("La liste est vide\n");
    else
    {
        printf("\nContenu de la liste: \n");
        while (ptr != NULL)
        {
            printf("%s\n", ptr->name);
            ptr = ptr->ptrNext;
        }
    }
}
```

# Listes chaînées : programme (5)

```
void emptyList (struct node *ptrFirst)
{
    struct node *ptr, *nextPtr;
    ptr = ptrFirst;
    while (ptr != NULL) {
        nextPtr = ptr->ptrNext;           // (1)
        free(ptr);                       // (2)
        ptr = nextPtr;                   // (3)
    }
}

// Remarque : le pointeur sur le nœud suivant
// est pris avant (1) l'appel de la fonction
// free(). Il est possible que l'espace libéré
// puisse être réutilisé avant l'exécution de
// la ligne suivante (3) et que ptrSuivant soit invalide
```

# Exercices



## Exercices du chapitre 13

# ANNEXE

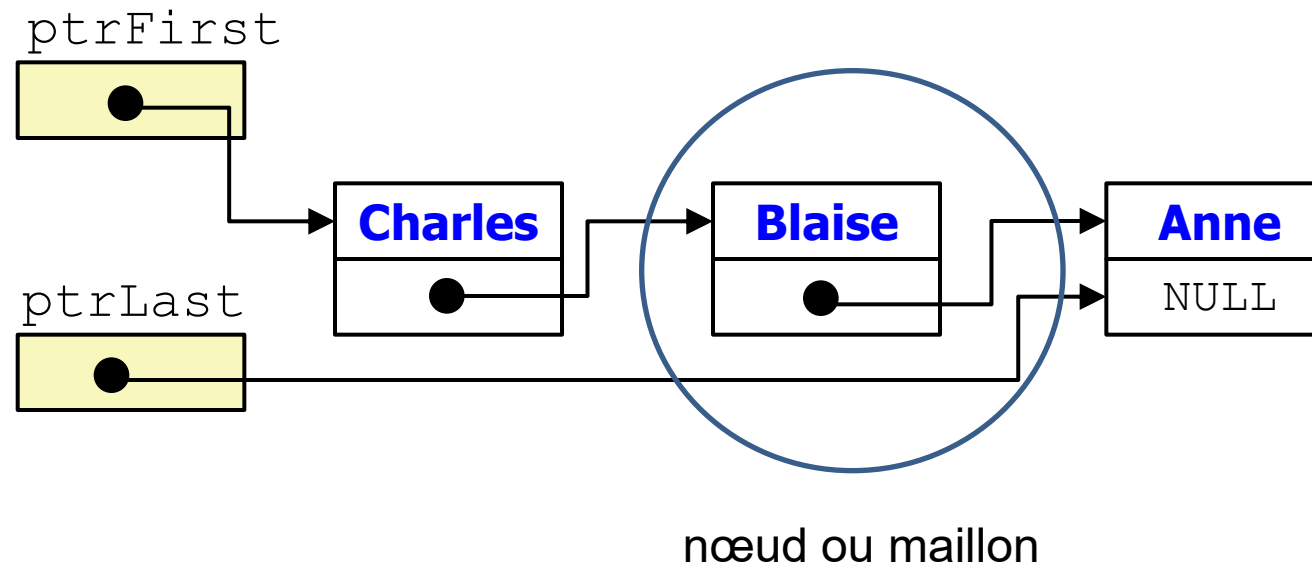
## Liste chaînées avec deux pointeurs

ptrFirst : pointe sur le premier nœud

ptrLast : pointe sur le dernier nœud

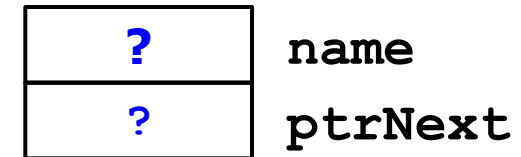


# Exemple Liste simplement chaînée contenant des noms



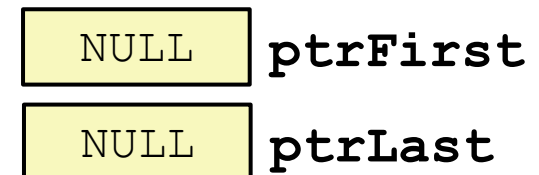
## Déclaration du type noeud:

```
struct node
{
    char        name[20] ;
    struct node *ptrNext;
};
```



## Déclaration des deux pointeurs :

```
struct node *ptrFirst = NULL,
            *ptrLast = NULL;
```



## Alternative avec typedef

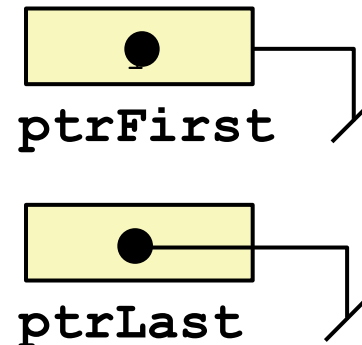
```
typedef struct node NodeTy;
NodeTy *ptrFirst, *ptrLast;
```

## Opérations de base:

- a) Initialisation des pointeurs
- b) Ajouter un nœud
  - 1) Allocation mémoire d'un nouveau nœud
  - 2) Initialisation du nœud
  - 3) Ajout du nœud à la liste
    - Au début de la liste
    - A la fin de la liste
- c) Afficher la liste

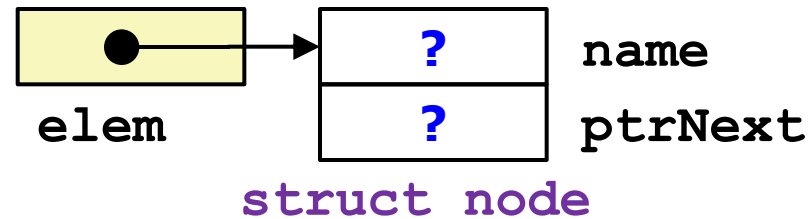
### a) Initialisation des pointeurs

```
ptrFirst = NULL;  
ptrLast  = NULL;
```



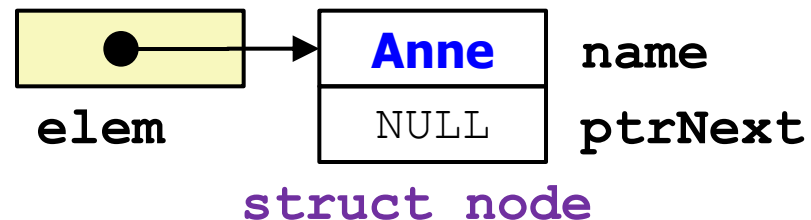
## b) Ajouter un nœud

### 1) Allocation en mémoire d'un nouveau nœud



```
struct node* elem;  
elem = malloc(sizeof(struct node));
```

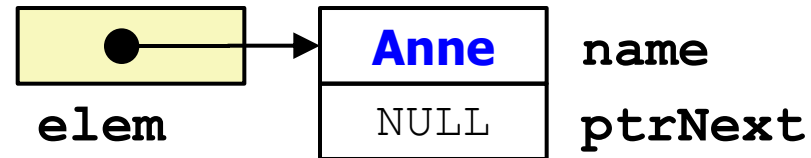
### 2) Initialisation d'un nœud



```
gets(elem->name);  
elem->ptrNext = NULL;
```

## b) Ajouter un nœud

### 3) Ajout d'un nœud à une liste vide



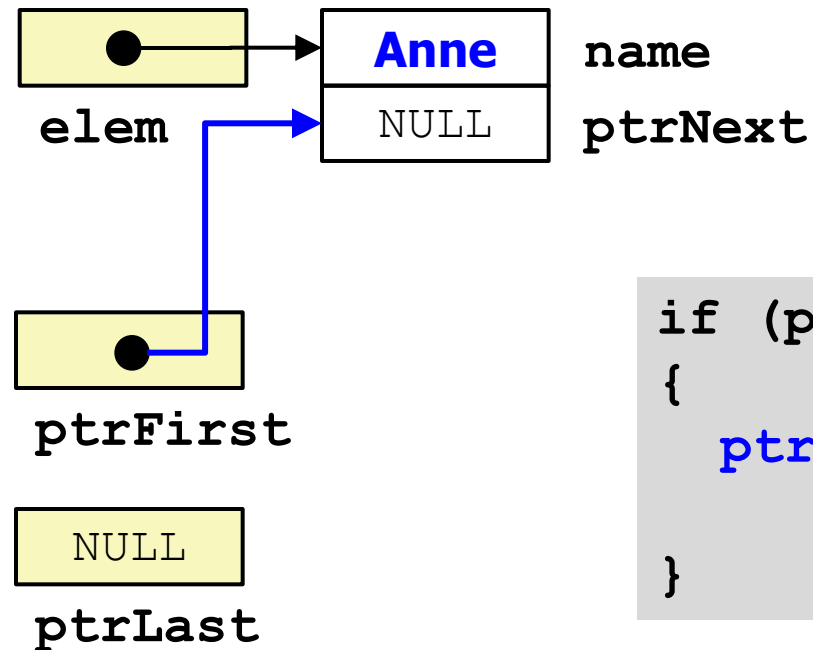
NULL  
ptrFirst

NULL  
ptrLast

```
if (ptrLast == NULL)
{
}
}
```

## b) Ajouter un nœud

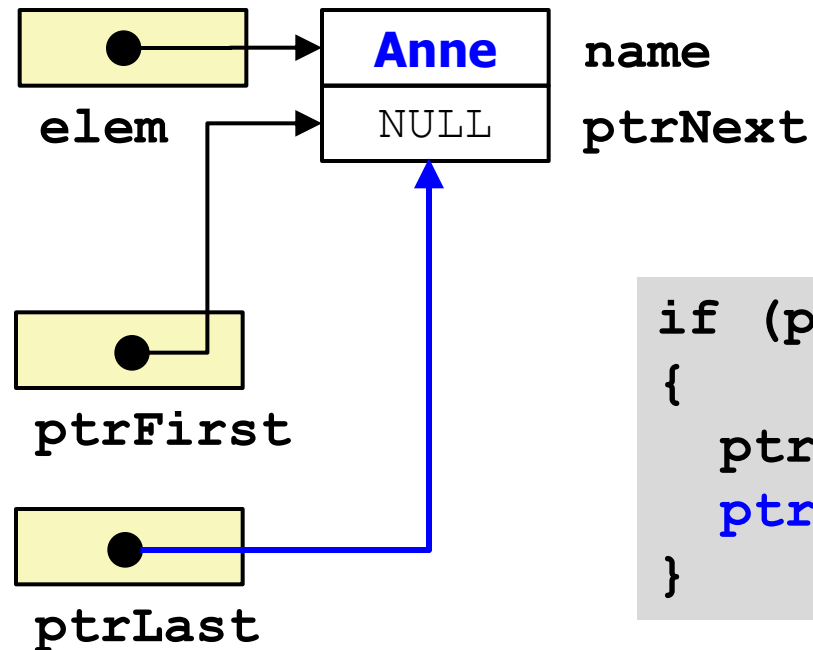
### 3) Ajout d'un nœud à une liste vide



```
if (ptrLast == NULL)
{
    ptrFirst = elem;
}
```

## b) Ajouter un nœud

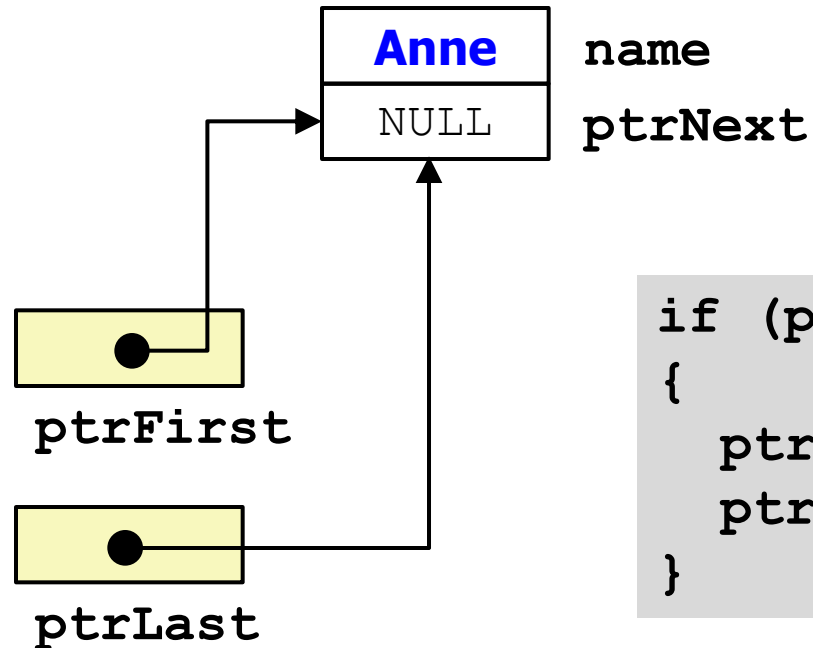
### 3) Ajout d'un nœud à une liste vide



```
if (ptrLast == NULL)
{
    ptrFirst = elem;
    ptrLast = elem;
}
```

## b) Ajouter un nœud

### 3) Ajout d'un nœud à une liste vide

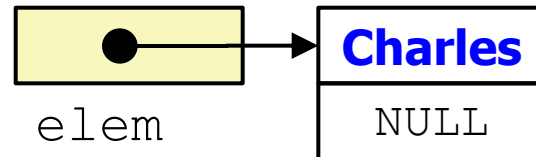


```
if (ptrLast == NULL)
{
    ptrFirst = elem;
    ptrLast = elem;
}
```

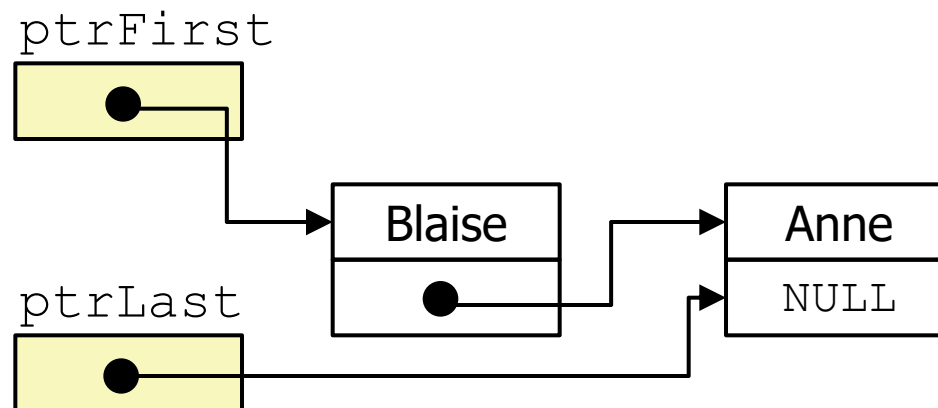


## b) Ajouter un nœud

### 3) Ajout d'un nœud **au début** de liste

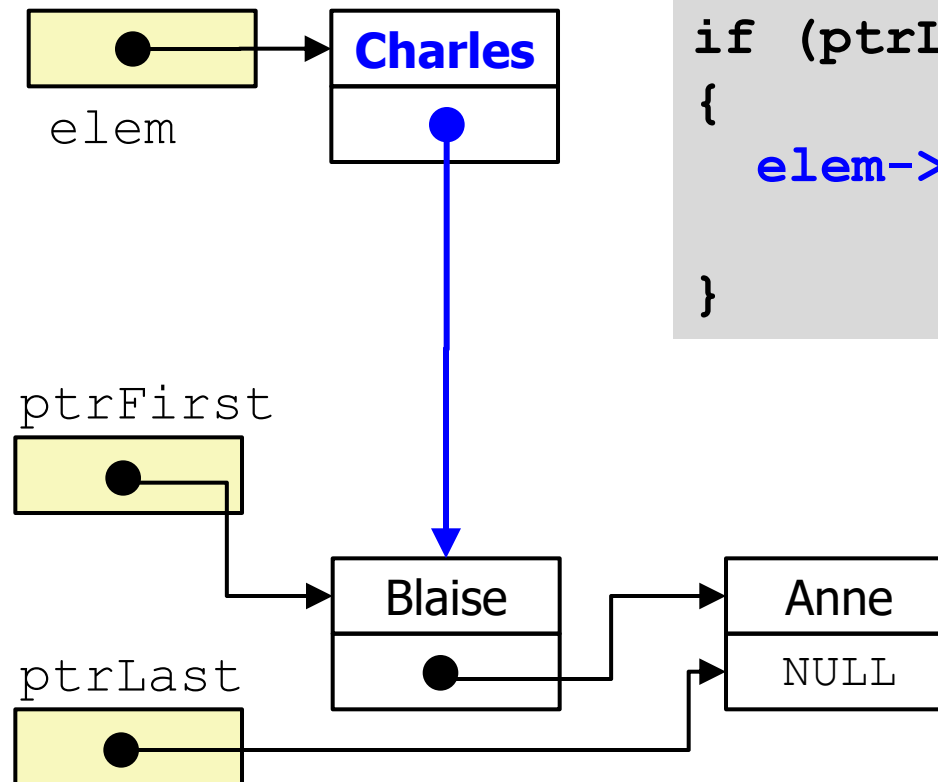


```
if (ptrLast != NULL)
{
}
}
```



## b) Ajouter un nœud

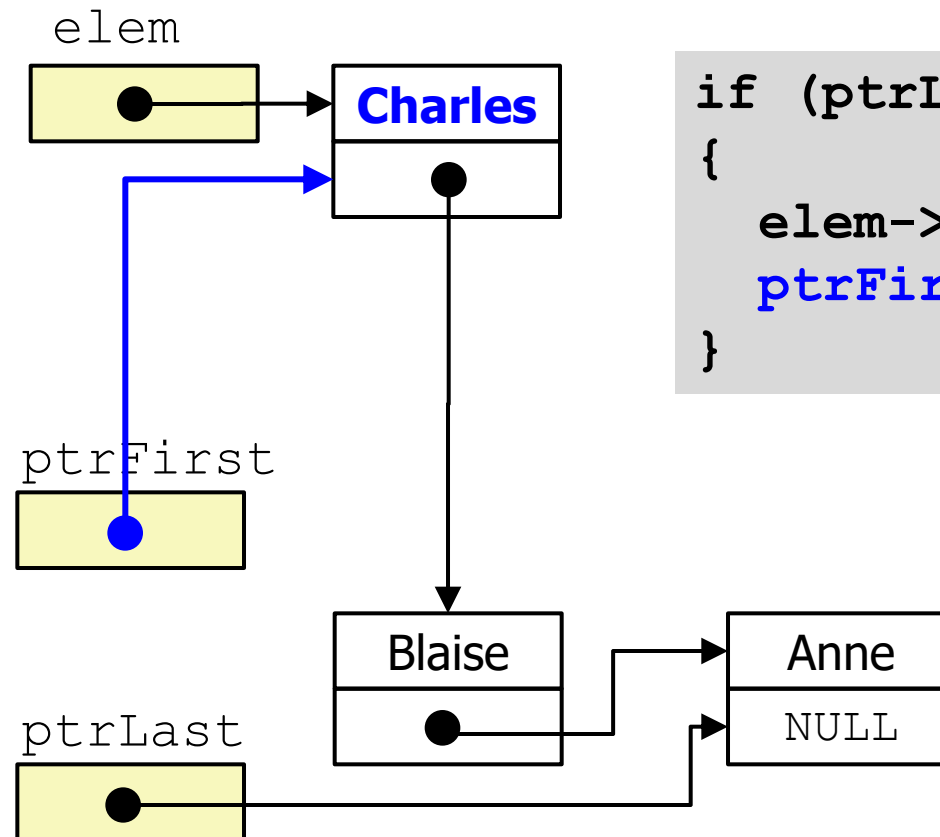
### 3) Ajout d'un nœud **au début** de liste



```
if (ptrLast != NULL)
{
    elem->ptrNext = ptrFirst;
}
```

## b) Ajouter un nœud

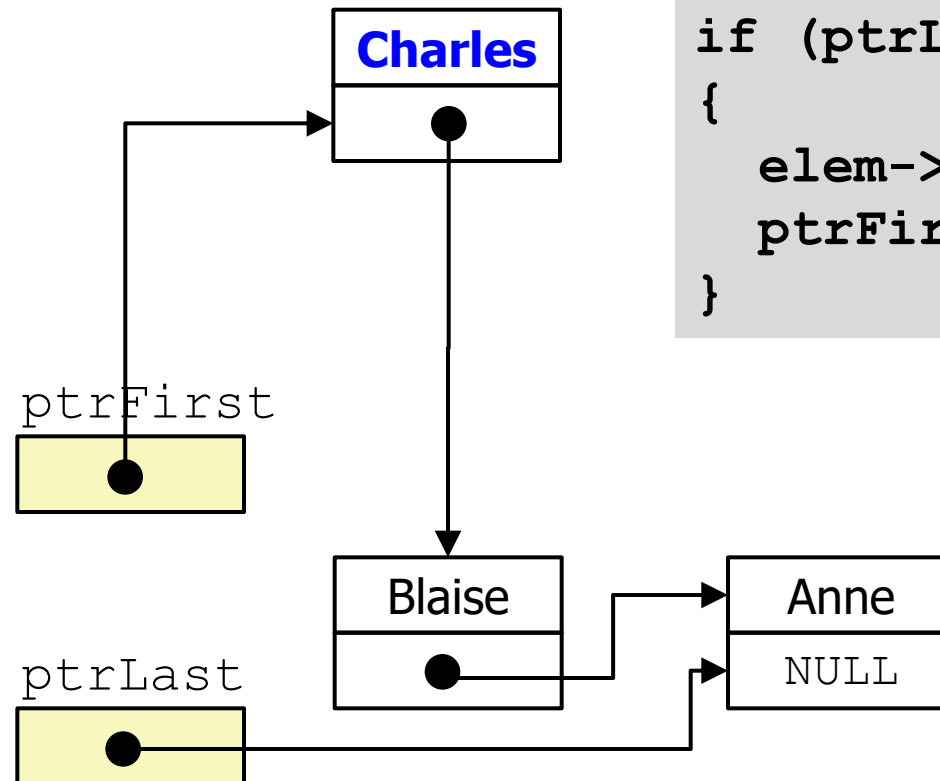
### 3) Ajout d'un nœud **au début** de liste



```
if (ptrLast != NULL)
{
    elem->ptrNext = ptrFirst;
    ptrFirst = elem;
}
```

## b) Ajouter un nœud

### 3) Ajout d'un nœud **au début** de liste

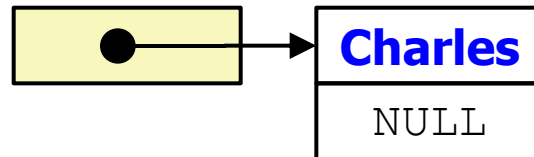


```
if (ptrLast != NULL)
{
    elem->ptrNext = ptrFirst;
    ptrFirst = elem;
}
```

## b) Ajouter un nœud

### 3) Ajout d'un nœud **à la fin** de liste

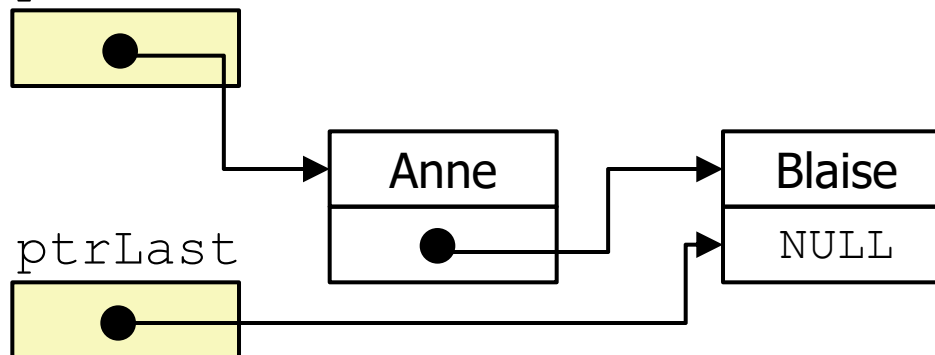
elem



```
if (ptrLast != NULL)
{
}

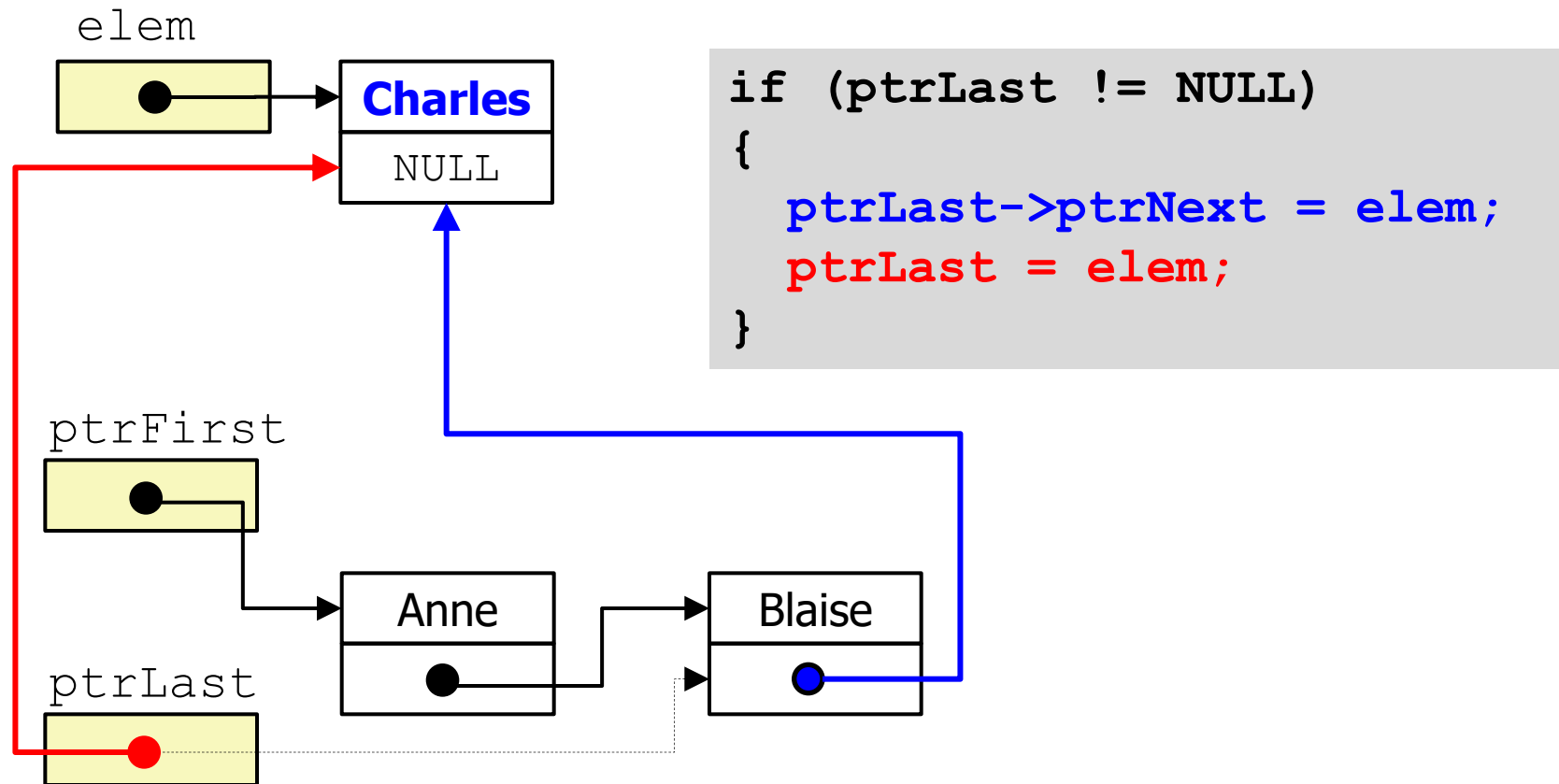
}
```

ptrFirst



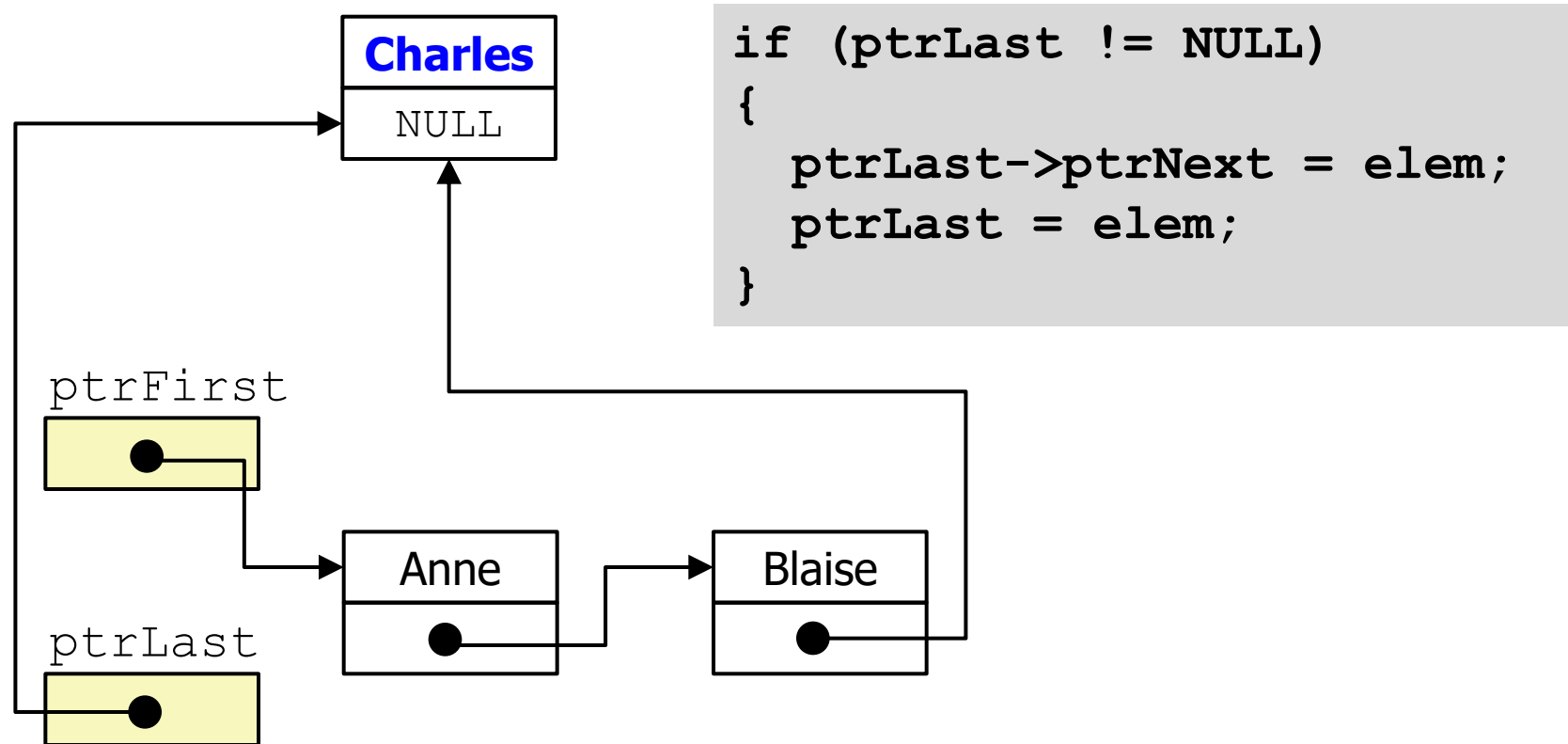
## b) Ajouter un nœud

### 3) Ajout d'un nœud **à la fin** de liste

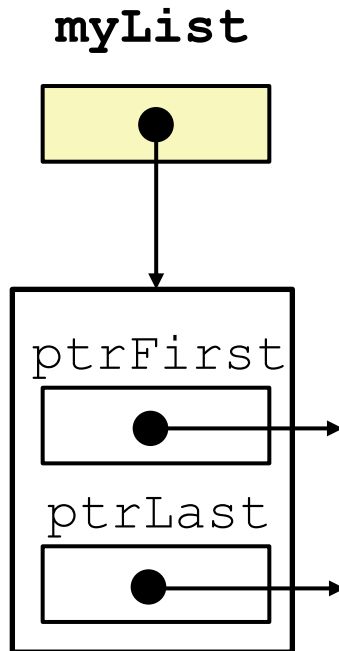


## b) Ajouter un nœud

### 3) Ajout d'un nœud **à la fin** de liste



## Bon usage : définir une structure pour la liste !



```
struct list
{
    struct node *ptrFirst;
    struct node *ptrLast;
};
```

```
struct list *myList;
myList = malloc(sizeof(struct list));
```

```
AddNode(myList);
```

```
//plutôt que AddNode(&ptrFirst,&ptrLast);
```