

# Structures de données et algorithmes – TP6

Iulia-Cristina Stanica

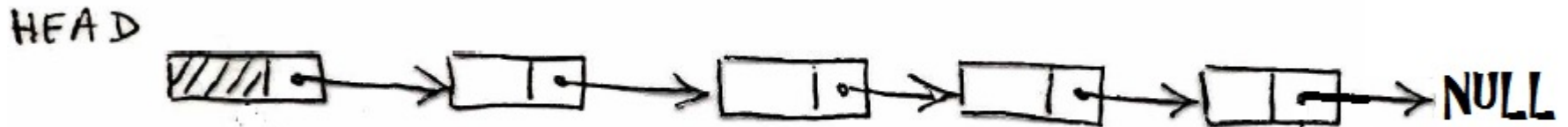
# Objectifs pour aujourd'hui

## Listes

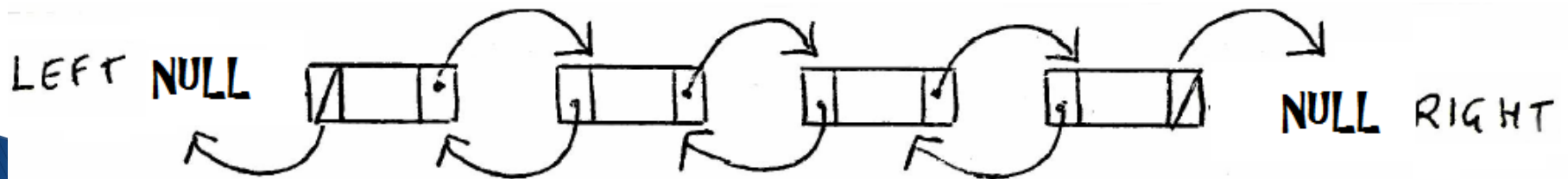
- ▶ L'implémentation des listes linéaires chaînées et leurs variations
- ▶ Exercices avec listes

# Types de listes (1)

- listes linéaires liées individuellement (singly-linked linear lists)

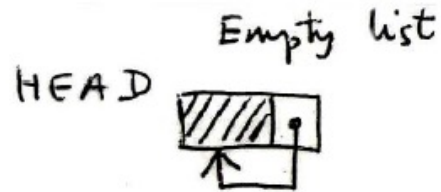
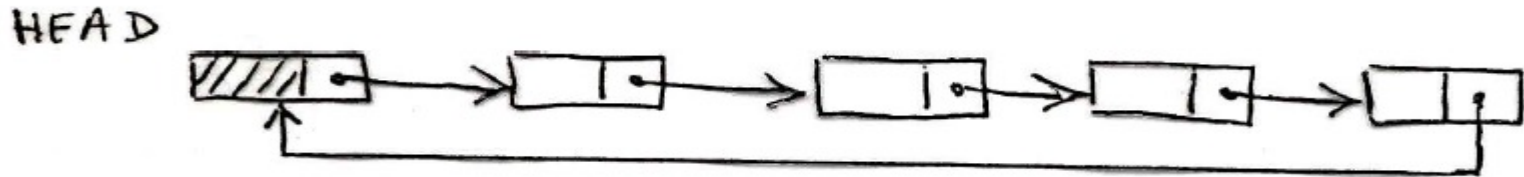


- listes linéaires doublement chaînées (doubly-linked linear lists)

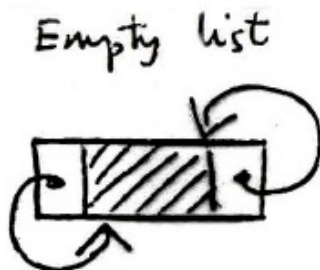
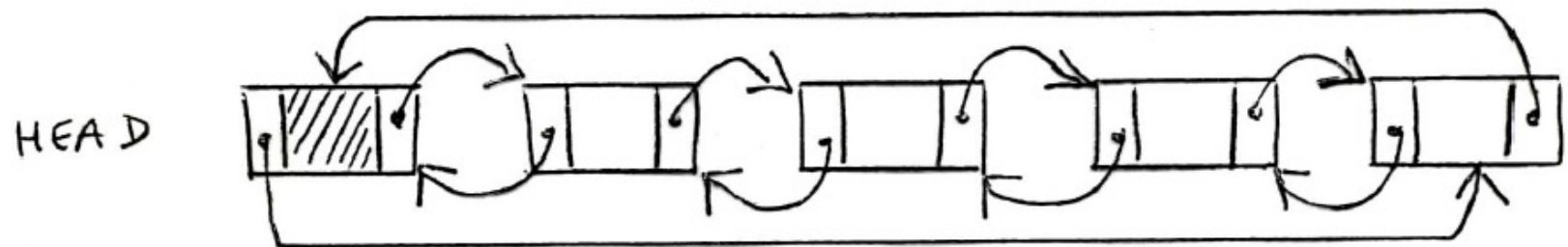


# Types de listes (2)

- listes circulaires liées individuellement (singly-linked circular lists)



- listes circulaires doublement chaînées (doubly-linked circular lists)

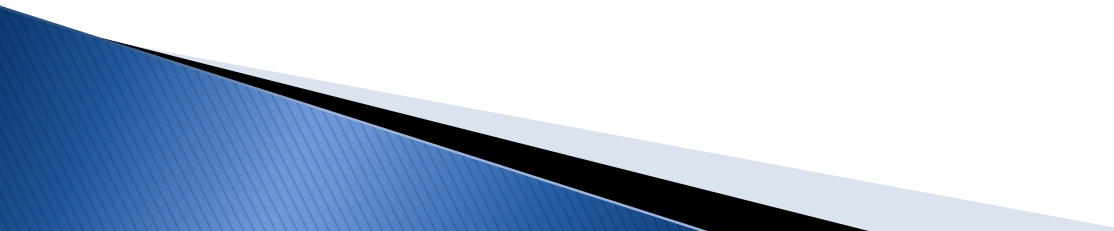


# Exercices

- Ex1) Testez les méthodes d'une liste linéaire doublement chaînée (liste1.h – sur Moodle) dans un fichier de type .cpp. Ajoutez dans le header une fonction **printListInverse()** qui affiche le contenu de chaque nœud de la liste, dans l'ordre inverse.
- Ex2) Modifiez l'ex1) pour créer deux nouvelles listes à partir de la liste initiale: une avec les nombres paires, l'autre avec les nombres impaires. Vous devez ajouter deux méthodes dans la classe **LinkedList** du header pour créer ces deux nouvelles listes. Affichez les trois listes pour voir l'effet.
- Ex3) Utilisez l'implémentation de l'Exo 1 pour créer une file d'attente (queue) avec les listes.
- Ex4) Modifiez le header liste1.h pour obtenir une liste circulaire doublement chaînée (doubly-linked circular list). Concentrez-vous sur les méthodes **add** et **remove**. Est-ce que l'affichage fonctionne toujours?

5) Étant donné une liste, divisez-la en deux sous-listes – une pour la moitié avant et une pour la moitié arrière. Si le nombre d'éléments est impair, l'élément supplémentaire doit aller dans la liste avant. Donc `FrontBackSplit ()` sur la liste {2, 3, 5, 7, 11} devrait produire les deux listes {2, 3, 5} et {7, 11}.

Vous devez vérifier votre solution par rapport à quelques cas (longueur = 2, longueur = 3, longueur = 4) pour vous assurer que la liste est divisée correctement près des conditions limites de la liste courte. Si cela fonctionne correctement pour longueur = 4, cela fonctionne probablement correctement pour longueur = 1000. Vous aurez probablement besoin d'un code de cas spécial pour traiter les cas (longueur < 2)



# Support théorique

# L'allocation dynamique

- ▶ L'allocation dynamique: allocation de la mémoire au moment de l'exécution du programme.
- ▶ Opérateurs utilisés: new et delete
- ▶ Utilisation: quand on ne connaît pas en avance la mémoire dont on a besoin
- ▶ Syntaxe: new **data-type**;
- ▶ Libérer la mémoire: delete pointeur;
- ▶ Plus d'infos:  
[http://www.tutorialspoint.com/cplusplus/cpp\\_dynamic\\_memory.htm](http://www.tutorialspoint.com/cplusplus/cpp_dynamic_memory.htm)



# Allocation dynamique – exemple

```
#include <iostream>
using namespace std;
int main ()
{
    int* pvalue = NULL; // Pointeur initialise avec NULL
    pvalue = new int;    // Demander de la mémoire pour la
variable
    *pvalue = 29;        // Sauvegarder la valeur a l'adresse
allouee
    cout << "Value of pvalue : " << *pvalue << endl;
    delete pvalue;      // Libérer la mémoire
    return 0;
}
```

# Allocation dynamique – exemple objets

```
#include <iostream>
using namespace std;
int main ()
{
    complex* number = new complex (2,3);
    number->display(); //on utilise l'opérateur « -> » et
    pas le « . » car number est un pointeur, pas l'objet réel
    //equivalent avec:
    (*number).display(); // *number – l'objet réel
}
```

# Allocation dynamique – tableaux

```
#include <iostream>
using namespace std;
int main ()
{
    int n; cin>>n;
    int* a = new int [n]; // Demander de la mémoire pour
    le tableau; a est un pointeur
    for (int i=0; i<n; i++) {
        a[i] = 0; // Initialiser les éléments du tableau
        //a[0] ou *a contiennent la valeur du premier élément
    }

    delete [] a; // Libérer la mémoire
    return 0;
}
```

# Allocation dynamique – tableau d'objets

```
#include <iostream>  
using namespace std;
```

```
int main()  
{  
    ClassName* myArray = new ClassName[4];  
    delete [] myArray ; // Delete array  
  
    return 0;  
}
```

# Les listes

- ▶ Instance d'un type de données abstraites (ADT)
- ▶ En C++ : `list` / `slist`.
- ▶ Une collection ordonnée d'entités.
- ▶ Le contenu de chaque élément de la liste:
  - informations utiles
  - pointeur(s) vers un ou plusieurs éléments voisins de la liste

Info sur les listes standard en C++:

<http://www.cplusplus.com/reference/list/list/>

# Opérations de base

- **Add (ajouter)** – ajoute un élément à la liste (au début, à la fin, à une position arbitraire)
- **Remove (supprimer)** – supprime un élément qui se trouve au début, à la fin ou tenant compte de l'indice/du contenu
- **Get** – obtenir un élément qui a un certain indice
- **Update** – mettre à jour l'information/le contenu d'un certain élément
- ▶ **OBS: la liste a les propriétés suivantes:**
  - La longueur (size) – nombre d'éléments de la liste (dans une fonction `getSize()`)
  - Le type – le type des éléments de la liste

# Mise en oeuvre

- Chaque nœud contient:
  - une information (le contenu)
  - le lien (pointeur):
    - vers ses voisins (listes doublement chaînées)
    - vers l'élément suivant de la liste (listes liées individuellement)
    - le dernier élément est lié au premier élément (listes circulaires)
- Les nœuds sont **alloués dynamiquement**, donc nous pouvons obtenir des listes d'une taille limitée seulement par la mémoire du programme

# Quiz – pour exercer 😊

- ▶ Pointeurs:

<https://www.cprogramming.com/tutorial/quiz/quiz6.html>

- ▶ Allocation dynamique:

<https://www.careerride.com/mcq/memory-management-c-mcq-questions-and-answers-120.aspx>



# Plus d'infos: Différences new et malloc()

NEW	MALLOC
Calls constructor	Does not call constructors
It is an operator	It is a function
Returns exact data type	Returns void *
On failure, Throws	On failure, returns NULL
Memory allocated from free store	Memory allocated from heap
Can be overridden	Cannot be overridden
Size is calculated by compiler	Size is calculated manually