🧩 PARTIE 1 — Exercices : LISTES CHAÎNÉES SIMPLES

📝 Exercice 1 — Gestion de notes

On souhaite gérer une liste chaînée simple contenant des notes (entiers).

Travail demandé :

1. Définir les structures nécessaires.
2. Insérer une note au début et à la fin.
3. Supprimer toutes les notes inférieures à 10.
4. Calculer la moyenne des notes.
5. Trier la liste par ordre croissant.
6. Afficher la liste.

📝 Exercice 2 — Liste de chiffres d'un nombre

Un nombre est saisi sous forme de chaîne.

Travail demandé :

1. Créer une liste chaînée où chaque nœud contient un chiffre.
2. L'ordre dans la liste doit être l'inverse du nombre saisi.
3. Supprimer les chiffres pairs.
4. Inverser la liste.
5. Trier les chiffres.

📝 Exercice 3 — Fusion de deux listes triées

Deux listes chaînées simples triées sont données.

Travail demandé :

1. Vérifier si chaque liste est triée.
2. Fusionner les deux listes en conservant le tri.
3. Supprimer les doublons dans la liste résultante.

🔁 PARTIE 2 — Exercices : LISTES DOUBLEMENT CHAÎNÉES

📝 Exercice 4 — Historique de navigation

Une liste doublement chaînée représente un historique.

Travail demandé :

1. Insérer une page à la fin.
2. Se déplacer en avant et en arrière.
3. Supprimer une page donnée.
4. Afficher l'historique dans les deux sens.
5. Inverser la liste.

📝 Exercice 5 — Playlist musicale

Chaque nœud contient le nom d'une chanson.

Travail demandé :

1. Ajouter une chanson.
2. Supprimer une chanson.
3. Avancer et reculer dans la playlist.
4. Trier les chansons par ordre alphabétique.

🔄 PARTIE 3 — Exercices : LISTES CHAÎNÉES CIRCULAIRES

📝 Exercice 6 — Jeu de chaises musicales

Une liste circulaire représente des joueurs.

Travail demandé :

1. Créer la liste circulaire.
2. Parcourir la liste sans boucle infinie.
3. Supprimer un joueur donné.
4. Supprimer chaque 3^{e} joueur jusqu'à un seul gagnant.

📝 Exercice 7 — Tour de garde

Une liste circulaire représente des employés.

Travail demandé :

1. Affecter le tour de garde.
2. Passer au suivant.
3. Supprimer un employé absent.
4. Réaffecter le tour correctement.

🔁 PARTIE 4 — Exercices : LISTES DOUBLEMENT CHAÎNÉES CIRCULAIRES

📝 Exercice 8 — Gestion d'un parking

Les voitures sont stockées dans une liste doublement circulaire.

Travail demandé :

1. Ajouter une voiture.
2. Retirer une voiture.
3. Parcourir le parking dans les deux sens.
4. Trouver la voiture la plus ancienne.
5. Trier les voitures par numéro.

🧠 PARTIE 5 — Codes des TRIS (LISTE CHAÎNÉE SIMPLE)

⚠️ **Langage : C++ classique (Dev-C++)**
⚠️ Utilisation de NULL (pas nullptr)

🔹 Structure utilisée

```
struct node {
   int val;
   node* next;
};


struct linkedlist {
   node* head;
```

```
};
```

🔸 1️⃣ Tri par sélection (Selection Sort)

```
void triSelection(linkedlist &lst){
   for(node* i = lst.head; i != NULL; i = i->next){
      node* min = i;
      for(node* j = i->next; j != NULL; j = j->next){
         if(j->val < min->val)
            min = j;
      }
      int tmp = i->val;
      i->val = min->val;
      min->val = tmp;
   }
}
```

🔸 2️⃣ Tri à bulles (Bubble Sort)

```
void triBulles(linkedlist &lst){
   if(lst.head == NULL) return;

   bool permute;
   do{
      permute = false;
      node* p = lst.head;
      while(p->next != NULL){
         if(p->val > p->next->val){
            int tmp = p->val;
            p->val = p->next->val;
```

```
            p->next->val = tmp;
            permute = true;
        }
        p = p->next;
    }
  } while(permute);
}
```

🔸 3️⃣ **Tri par fusion (Merge Sort) — ⭐ IMPORTANT**

Division

```
void diviser(node* source, node* &a, node* &b){
  node* lent = source;
  node* rapide = source->next;

  while(rapide != NULL){
    rapide = rapide->next;
    if(rapide != NULL){
      lent = lent->next;
      rapide = rapide->next;
    }
  }

  a = source;
  b = lent->next;
  lent->next = NULL;
}
```

Fusion

```
node* fusion(node* a, node* b){
```

```
    if(a == NULL) return b;
    if(b == NULL) return a;

    if(a->val <= b->val){
        a->next = fusion(a->next, b);
        return a;
    } else {
        b->next = fusion(a, b->next);
        return b;
    }
}
```

Tri fusion

```
void triFusion(node* &head){
    if(head == NULL || head->next == NULL)
        return;

    node* a;
    node* b;

    diviser(head, a, b);
    triFusion(a);
    triFusion(b);

    head = fusion(a, b);
}
```