CM – Listes chaînées :

1. Présenter la liste chaînée, simple, doublement chaînée, circulaire. Cas pratique : la liste chaînée d'entiers ;
2. Présenter le code source de l'archive `comparatif-1.0.tgz` comparant les performances en insertion entre une liste châinée et un tableau dynamique (vecteur) ;
3. Présenter le problème de Joséphus : `https://fr.wikipedia.org/wiki/Probl%C3%A8me_de_Jos%C3%A8phe`

TD/TP – Listes chaînées :

1. Résoudre le problème de Joséphus à l'aide d'une liste circulaire. Il sera nécessaire de pouvoir spécifier le nombre de soldats ainsi que le pas de déplacement permettant de sélectionner le suivant à éliminer : utiliser en les modifiant (conditions terminales lors du print et du free) les fonctions vues précédemment en cours, faire attention au nœud initial (il pointe sur lui-même) ;
2. Écrire une structure de données `node_t` utile au stockage d'un nombre indéfini de couples noms — prénoms (et faire un générateur de "mots" pour le test final) ;
3. Écrire la fonction `node_t * new_node(char * first_name, char * last_name)` qui crée un élément nom-prénom ;
4. Écrire la fonction `void add_node(node_t ** here, node_t * n)` qui ajoute l'élément `n` dans `here` ;
5. En ayant un pointeur vers le début d'une liste de noms et prénoms :
   (a) Comment ajouter un nouvel élément en début de liste ? L'écrire ;
   (b) Comment ajouter un nouvel élément en fin de liste ? L'écrire ;
   (c) Comment ajouter un nouvel élément en respectant un ordre pré-établi (liste déjà triée dans l'ordre croissant ou décroissant) ? L'écrire ;
   (d) Comment supprimer un élément de la liste ? L'écrire.
   (e) Comment libérer la liste ? L'écrire.