Structures et fonctions supplémentaires :

- *void viderBuffer()*
  - Fonction permettant de vider le buffer de l'entrée standard
  - L'intérêt est d'éviter des problèmes lors de la saisie de l'utilisateur
- *t_vaccin_elt* parcourirVaccins(t_vaccin_elt* vaccin[10], int vaccins_existants, char* nom_vaccin)*
  - Fonction permettant de rechercher un élément vaccin par son nom
  - Paramètres :
    - t_vaccin_elt* vaccin[10] : le tableau de vaccins à parcourir
    - Int vaccins_existants : le nombre de vaccins contenus dans le tableau
    - char* nom_vaccin : le nom du vaccin recherché
  - Retour : l'élément vaccin recherché
- *t_semaine_elt* supprimerSemaine(t_semaine_elt* liste_semaines, int semaine)*
  - Fonction permettant de supprimer une semaine en libérant son emplacement mémoire
  - Paramètres :
    - t_semaine_elt* liste_semaines : la liste de semaines à modifier
    - int semaine : le numéro de la semaine à supprimer
  - Retour : le premier élément de la liste de semaines modifiée
- *t_ville_elt* supprimerVille(t_ville_elt* liste_villes, char* ville)*
  - Fonction permettant de supprimer une ville en libérant son emplacement mémoire
  - Paramètres :
    - t_ville_elt* liste_villes : la liste de villes à modifier
    - char* ville : le nom de la ville à supprimer
  - Retour : le premier élément de la liste de villes modifiée
- *t_ville_elt* trierVilles(t_ville_elt* liste_villes, char* ville)*
  - Fonction permettant de trier la liste de villes en intégralité
  - Paramètres :
    - t_ville_elt* villes : la liste de villes à trier
  - Retour : le premier élément de la liste de villes triée
- *t_ville_elt* deplacerVille(t_ville_elt* liste_villes, t_ville_elt* ville)*
  - Fonction permettant de déplacer une ville de façon à ce qu'elle soit triée par rapport à son prédécesseur et son successeur dans la liste (cette fonction devra être appelée pour chaque ville de la liste pour effectuer un tri de la liste entière
  - Paramètres :
    - t_ville_elt* liste_villes : la liste de villes à modifier
    - t_ville_elt* ville : la ville à déplacer dans la liste

    - Retour : le premier élément de la liste de villes ainsi triée
- *int calculTotalVaccinsVille(t_semaine_elt* semaines_planifiees)*
    - Fonction calculant le nombre total de vaccins de la liste de semaines planifiées passée en paramètres. Notamment utilisée pour recalculer le total de vaccins d'une ville après déduction d'un nombre de doses (puisqu'il n'y a pas de moyen de savoir si l'insertion a été faite ou non, et donc si ce total a changé ou non)
    - Paramètres :
        - t_semaine_elt* semaines_planifiees : la liste de semaines dont on souhaite calculer le total de vaccins
    - Retour : le total de vaccins de la ville

Complexité des fonctions :

- *creerSemaine* : **O(1)**
- *creerVille* : **O(1)**
- *creerVaccin* : **O(1)**
- *ajouterVaccinS* :
  Cas défavorable : insertion en queue de liste
  Pour une liste de taille N, on parcourt alors toute la liste avant d'insérer
  => Complexité en **O(N)**
- *deduireVaccinS* :
  Cas défavorable : semaine recherchée en queue de liste
  Pour une liste de taille N, on parcourt alors toute la liste avant de déduire le nombre de vaccins
  => Complexité en **O(N)**
- *ajouterVaccinV* :
  Cas défavorable : ville recherchée en queue de liste
  Pour une liste de villes de taille M, on parcourt alors toute la liste (=> O(M))avant d'ajouter les stocks à la ville en appelant ajouterVaccinS, qui est elle-même de complexité O(N) avec N le nombre de semaines planifiées pour cette ville. On a alors une complexité totale **O(M+N)**.
  Un autre cas permettant d'arriver à cette complexité serait le suivant :
  On ajoute à une ville existante un nombre de vaccins tel que son nombre total de vaccins devient le plus grand de la liste de villes. On doit alors parcourir le reste de la liste afin d'y déplacer la ville modifiée, dans le but de garder la liste triée.
- *deduireVaccinV* :
  Cas défavorable : ville recherchée en bout de liste
  Pour une liste de villes de taille M, on parcourt alors l'ensemble de la liste pour trouver la ville cherchée et appeler deduireVaccinS, qui est elle-même de complexité O(N) avec

N le nombre de semaines planifiées pour cette ville. On a alors une complexité de **O(M+N)**. On doit cependant calculer le nouveau total de vaccins de la ville grâce à la fonction calculTotalVaccinsVille, ajoutant une complexité de **O(N)**. De plus, l'élément modifié doit être déplacé dans la liste dans le cas où son total de vaccins devient inférieur à celui de ses prédécesseurs. On utilise pour cela la fonction deplacerVille, qui a alors une complexité de **O(M)**. On obtient alors une complexité totale de **O(M+N)**.

- *afficherStock* :
  Dans tous les cas, parcours de toutes les villes O(M) de la liste et pour chaque ville, parcours de toutes les semaines planifiées O(N). On peut donc estimer la complexité de cette fonction à **O(M*N)**.
- *afficherPlanification* :
  Cas défavorable : le numéro de semaine à afficher est supérieur ou égal à celui de toutes les semaines planifiées de chaque ville. Dans ce cas, toutes les semaines de chaque ville seraient parcourues, avec donc une complexité de **O(M*N)**.
- *fusionnerStocks* :
  Cette fonction commence par ajouter à la liste des villes planifiées toutes les villes de la liste du premier vaccin, pour une complexité de O(Ma) avec Ma le nombre de villes dans la première liste. Toutes les semaines de ces villes sont parcourues pour être recopiées, pour une complexité de O(Na) avec Na le nombre de semaines dans la ville parcourue. On obtient alors pour ce parcours de la première liste de villes une complexité de : O(Na + Ma*Na) = O(Na*(Ma+1)) = **O(Ma*Na)**.
  On parcourt ensuite la liste de villes du second vaccin, avec une complexité de O(Mb). Dans ce parcours, pour chaque ville (complexité de O(Mb)), on parcourt la liste de villes précédemment créées pour vérifier si cette ville existe déjà. Si la ville existe déjà, on fusionne les deux villes existantes et leurs semaines ; on déplace alors la ville dans la liste de façon à garder celle-ci triée par le nombre total de vaccins (complexité de O(Ma+Mb)). De plus, on cherchera à fusionner ou à défaut insérer de la même façon chaque semaine de la ville parcourue, pour une complexité de O(Na+Nb).
  Ainsi, on peut estimer la complexité de la fonction à O(Ma*Na + Mb*(Ma+Mb + Na+Nb)), pouvant être développée en **O(Mb² + Ma*Na + Mb*Ma+ Mb*Na + Mb*Nb)**.
- *viderBuffer* : **O(1)**
- *parcourirVaccins* :
  Cette fonction parcourt l'ensemble des vaccins du tableau pour retourner le vaccin dont le nom correspond à celui recherché. On a donc une complexité de **O(N)** où N est le nombre de vaccins contenus dans le tableau.
- *supprimerSemaine* :
  Cette fonction parcourt l'ensemble des semaines (complexité O(N) avec N le nombre de semaines planifiées dans la liste) pour trouver l'emplacement de la semaine à supprimer,

tout en conservant un pointeur sur la semaine précédente pour la raccorder sur la semaine suivante celle que l'on veut supprimer. On obtient donc une complexité de **O(N)**.

- *supprimerVille* :
  Cette fonction parcourt l'ensemble des villes (complexité O(M) avec M le nombre de villes planifiées dans la liste) pour trouver l'emplacement de la ville à supprimer, tout en conservant un pointeur sur la ville précédente pour la raccorder sur la ville suivante celle que l'on veut supprimer.
  À la suite de ce parcours s'ajoute la suppression de toutes les semaines de la ville afin d'éviter toute fuite de mémoire, en appelant la fonction supprimerSemaine pour une complexité de **O(N)** avec N le nombre de semaines planifiées pour la ville. On obtient donc une complexité de **O(M+N)**.
  Note : Le parcours des semaines de la ville n'est fait qu'une seule fois, **après** le parcours des villes : il n'est en effet nécessaire que de supprimer les semaines de la ville à supprimer, ajoutant ainsi N à la complexité une seule fois. Donc on obtient une complexité de **O(M+N)**.
- *deplacerVille* :
  Cette fonction parcourt la liste des villes pour y insérer la ville déplacée, pour une complexité de O(M) avec M le nombre de villes dans la liste : en effet, on parcourt dans le cas défavorable toute la liste avant de pouvoir insérer la ville. On obtient donc une complexité de **O(M)**.
- *trierVilles* :
  Cette fonction parcourt l'ensemble des villes (complexité de **O(M)**) de la liste et appelle la fonction deplacerVille (complexité de **O(M+N)**) pour chaque ville de la liste. On obtient donc une complexité de **$O(M^2+M*N)$**.
- *calculTotalVaccinsVille* :
  Cette fonction parcourt l'ensemble des semaines planifiées de la liste passée en paramètre : on a donc une complexité de **O(N)** avec N le nombre de semaines planifiées de la liste.