Q : Diriez-vous que la structure de données imposée dans ce laboratoire est adaptée au problème de la gestion d'une animation graphique? Parlez de modifications que vous apporteriez à la structure de données pour l'améliorer. (15)

A : Oui, car nous ne connaissons pas la taille de l'animation dès le début et devons donc faire de nombreuse allocation de mémoire. Un tableau dynamique aurait pu suffire, mais les nombreux redimensionnements serait très couteux, surtout lors de la copie des éléments vers le nouveau tableau. La liste chainée est donc adaptée car les nouveaux éléments peuvent être joints à la fin de la structure sans avoir à réallouer la structure au complet.

J'ai modifié la structure de manière à la rendre plus modulaire et réutilisable. J'ai séparé les listes chainées double et simple de leur contenue et en ai fait des templates afin de pouvoir m'en resservir dans le futur. De plus, les frames et les instructions sont des classes distinctes de la classe animation. Ceci m'a permis d'adopter une stratégie « diviser pour mieux régner » pour charger le fichier.

Q : On désire ajouter de nouvelles fonctionnalités au programme de gestion d'animations. On veut avoir la possibilité de rajouter un frame à partir de l'interface graphique (quand l'animation est déjà chargée). De plus, on voudrait avoir accès à une barre de défilement pour arrêter ou repartir l'animation à partir de n'importe quel frame. Est-ce que la structure de données actuelle peut s'adapter à ce genre de spécification? Si oui, dîtes comment, si non proposez une structure de données alternatives. (10)

A : Oui, les listes chainées sont justement parfaites pour de l'insertion au milieu de la structure. Ma structure list (liste chainée double) comprend déjà une méthode pour l'insertion au milieu. Cette méthode se nomme insert_after.

Q : Comment serait-il possible d'accélérer le temps de chargement du fichier d'animation par une modification à la structure de données ? (5)

A : Éviter les allocations et les copies inutiles en utilisant des move operators plutôt que des copy operator. list<T>& operator=(list<T>&& x); plutôt que list<T>& operator=(const list<T>& x);