ECE Lyon - ING1 - Informatique 2 - Partiel

Partie 1 - Lecteur de musique *(13 points)*

Dans cette partie, vous allez devoir développer les bases de ce qui pourrait servir à une plateforme de streaming musical en ligne type Deezer, Spotify, etc.

Exercice 1 - Structures *(2 points)*

Chaque morceau est représenté sous la forme d'une structure de données permettant à la plateforme de proposer un moteur de recherche, une discographie, des playlists, etc.

Donnez le code définissant la structure "`Track`" contenant :

- Un titre : `title` (50 caractères maximum)
- Le nom de l'artiste : `artiste` (50 caractères maximum)
- La durée du morceau : `duration` (exprimée en millisecondes, toujours positive)
- La note moyenne donnée par les utilisateurs de la plateforme : `score`, comprise entre `0.0` et `5.0` (ex : `4.65` étoiles sur `5`)

Donnez le code définissant la structure "`Album`" contenant :

- 12 morceaux : `tracks` (`Track`)
- Le nom de l'album : `name` (100 caractères maximum)

Exercice 2 - Listes chaînées *(3 points)*

En soirée, de nombreuses personnes souhaitent ajouter/retirer/modifier l'ordre des morceaux à la playlist en cours. La gestion sous forme de tableau n'est donc pas très appropriée pour ce genre d'utilisation.

Reprenez le code de la structure "`Track`" définie dans l'exercice 1 afin d'en faire un maillon de liste chaînée (vous choisirez entre liste *simplement* chaînée, et liste *doublement* chaînée, en justifiant votre choix).

Donnez le code du sous-programme "`deleteTrack(...)`" qui reçoit en paramètres le titre du morceau à supprimer, ainsi que la playlist dans laquelle il doit être supprimé (la liste chaînée), puis le supprime. Vous veillerez à ne provoquer aucune fuite mémoire, et à ne perdre aucun morceau.

Exercice 3 - Mode aléatoire *(2 points)*

Donnez le code du sous-programme "`shuffleTrack(...)`" qui reçoit en paramètres le nombre de morceaux présents dans la playlist, et la position (l'indice) du morceau en cours de lecture.

Ce sous-programme devra effectuer un tirage aléatoire ne dépassant pas le nombre de morceaux et renvoyer l'indice du morceau à jouer ensuite. Si le numéro tiré est égal à la position du morceau en cours de lecture, un nouveau numéro devra être tiré.

Les nombres tirés aléatoirement par le sous-programme "`shuffleTrack()`" seront-ils différents à chaque lancement du programme ? Si oui, expliquez pourquoi. Si non, que faut-il faire pour ne jamais tirer la même combinaison à chaque lancement du programme ?

Exercice 4 - Mode hors ligne *(2 points)*

La plupart des plateformes de streamming musical permettent de télécharger les morceaux en local afin de s'affranchir du besoin d'une connexion à internet. Ce service est très utile dans le cas de déplacements.

Donnez le code du sous-programme "`saveTrack(...)`" qui reçoit en paramètre le morceau à sauvegarder et qui le sauvegarde dans un fichier binaire sur le disque dur (dans le dossier courant (`./`)). Chaque fichier créé devra porter le nom du titre du morceau qu'il contient, suivi de l'extension `.datatrack` (si le fichier existe déjà, il sera écrasé).

```
1 fwrite(a, sizeof(int), n, pFichier); // Ajoute n éléments de type int dans le fichier
      pointé par pFichier, à partir de l'adresse "a". Retourne le nombre d'éléments ajoutés.
```

Exercice 5 - Interface graphique *(4 points)*

Nous souhaitons créer, sous Allegro 5, l'interface graphique suivante :

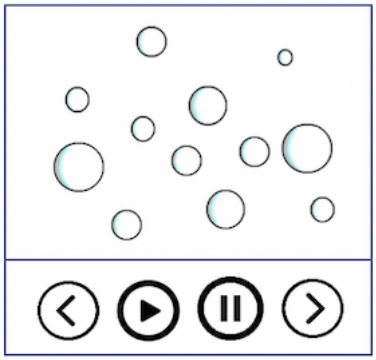

FIGURE 1 – Interface de l'application

En étant le plus précis possible, indiquez comment vous procéderiez pour :

- détecter un clic de la souris sur l'un des boutons.
- alterner entre lecture et pause en appuyant sur la barre d'espace.
- passer au morceau précédent/suivant lors de l'appui sur les flèches de gauche ou de droite.
- dessiner la partie visuelle principale de l'interface, composée de bulles de tailles différentes, se déplaçant dans des directions rectilignes aléatoires, rebondissants sur les bords de l'interface et disparaissant en se percutant.

Partie 2 - Prise de décisions : coder mieux *(5 points)*

Exercice 6 - Recensement de valeurs *(2 points)*

Soit un tableau `values` de 1000 entiers aléatoires compris entre 0 et 10 (inclus).

Soit un tableau `count` visant à recenser le nombre de 0, de 1…, de 10 présents dans le tableau `values` (si `count[3] == 12`, alors il y a `12` "`3`" dans le tableau `values`).

Donnez un programme qui déclare et initialise (si besoin) les deux tableaux, puis qui dans un deuxième temps, remplit le tableau `count` d'après le tableau `values`. Enfin, affichez le nombre d'apparitions de chaque nombre dans le terminal.

Vous veuillerez à créer un programme efficace, de complexité minimale.

Exercice 7 - Choix de structures de données *(3 points)*

D'après le cahier des charges ci-dessous, indiquez quelles structures de données vous utiliseriez, en prenant soin de justifier vos choix.

Un magasin de meubles voudrait un logiciel permettant, au moment d'une vente, d'indiquer à son entrepôt quels cartons préparer pour qu'ils puissent être retirés par le client.

- Les clients sont servis dans la règle du premier arrivé, premier servi.
- Si un client achète plusieurs meubles, chaque meuble est traité individuellement (2 meubles achetés = 2 demandes de préparation envoyées à l'entrepôt).
- La personne effectuant la vente devra pouvoir demander à ce qu'une préparation soit effectuée en priorité.
- Les préparateurs devront avoir les informations suivantes pour chaque préparation à effectuer :
  - L'identifiant code-barre du carton à préparer (entier positif)
  - L'allée où se trouve le carton (lettre comprise entre `A` et `J`)
  - Le numéro (dans l'allée) où se trouve le carton (de `1` à `37`)
  - L'étage où se trouve le carton (valeur allant de `0` à `5`)
  - Le nom du client à qui donner le carton.
- Chaque préparateur peut traiter jusqu'à 4 demandes de préparation à la fois. Au moment d'entrer dans l'entrepôt, il doit alors savoir quels sont les potentiels 4 cartons à récupérer. Lorsqu'il revient à l'accueil, sa liste doit alors être vidée, puis remplie à nouveau avec de nouvelles préparations à effectuer.

Partie 3 - Arbres Binaires de Recherche *(4 points)*

Règle de base de construction d'un Arbre Binaire de Recherche :

$$maximum(\text{valeurs sous-arbre gauche}) < valeur(\text{noeud}) < minimum(\text{valeurs sous-arbre droit})$$

Exercice 8 - Création d'un arbre binaire de recherche *(1 point)*

Soit un arbre binaire de recherche vide. Dessinez-le en y ajoutant les valeurs suivantes dans l'ordre donné :

50, 60, 34, 42, 57, 64, 46, 62, 28, 40, 93, 61, 12.

Donnez la **taille** et la **hauteur** de cet arbre en prenant soin de définir ces deux termes.

Exercice 9 - Parcours préfixé, infixé et postfixé *(3 points)*

Donnez l'affichage produit par les parcours **préfixé**, **infixé** et **postfixé** sur l'arbre précédemment construit. Si vous n'avez pas réussi à construire l'arbre dans l'exercice précédent, inventez-en un, si possible équilibré, de `taille >= 10` et de `hauteur >= 3`.

Enfin, répondez aux questions suivantes :

- Au niveau du code, qu'est-ce qui varie entre les trois types de parcours ?
- Quel parcours permet la bonne libération mémoire d'un arbre binaire ? Pourquoi ?
- Selon vous, quelle structure de données est la moins **complexe*** lors de l'ajout ou de la recherche d'une valeur dans une structure triée : une liste chaînée ou un arbre binaire de recherche ? Expliquez pourquoi, en comparant par exemple l'ajout d'une grande valeur dans une liste chaînée et un arbre binaire de recherche triés par ordre croissant et contenant déjà beaucoup de valeurs.

La **complexité d'un algorithme est une estimation du nombre d'opérations de base (comparaison, calcul, affectation...) effectuées par l'algorithme en fonction de la taille des données qu'il reçoit en entrée.*