ING1
Semestre 1 - 2025/2026

Contrôle continu - Programmation C-avancé
Durée : 1h30

Tous documents et outils autorisés
Aucune question aux surveillants n'est autorisée

Les exercices nécessitent d'avoir le code des TDs fonctionnel. C'est à dire être capable de charger en mémoire les informations des étudiants contenues dans le fichier `data.txt`. Le TD sur les bibliothèques, ainsi que la partie avec les fonctions d'interfaces (API) ne sont pas obligatoires.

L'ensemble du code évalué devra être situé dans de nouveaux fichiers distincts de ceux déjà présents dans votre environnement, pour identifier facilement le code ajouté du code déjà existant.

Si vous avez besoin de modifier des fichiers des TDs précédents, il conviendra de l'indiquer à minima en commentaires dans votre fichier principal.

Exercice 1 (*Recalcul*) [6.0 pts]

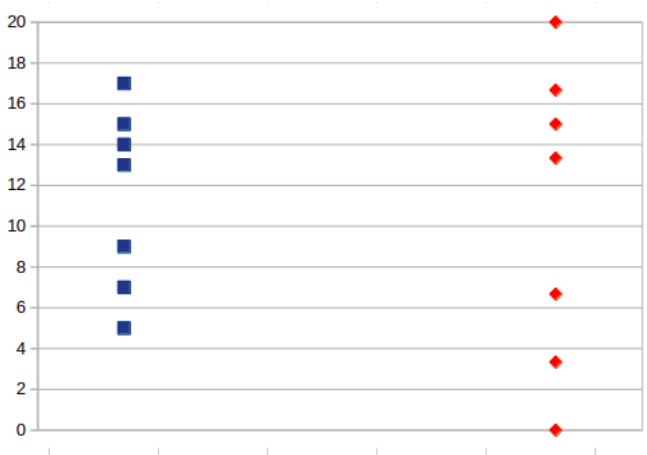
On souhaite effectuer un recalcul des moyennes générales pour toute la promotion, pour compenser les différences de notation entre matières et/ou entre correcteurs. Un recalcul est une opération qui va prendre toutes les moyennes générales des étudiants, relever la plus petite et la plus grande valeur, et modifier les notes pour les ramener dans un nouvel intervalle.

Comme l'opération de recalcul est dépendante de beaucoup de paramètres extérieurs, on ne peut pas coder un seul algorithme. Pour éviter de modifier toute notre application, nous allons passer par un pointeur de code :

1. Définir un pointeur de procédure `Recalcul` qui va prendre en paramètres un étudiant, une valeur minimale de note, et une valeur maximale de note.
2. Créer une fonction `... miseAJourNotes(...)` qui va prendre en paramètres une promotion et un pointeur de type `Recalcul`. Cette fonction va :
   — rechercher les moyennes générales extrêmes,
   — modifier la moyenne générale de chaque étudiant, l'une après l'autre, en utilisant le pointeur `Recalcul`,
   — renvoyer un booléen pour indiquer si le traitement a été terminé avec succès ou non.

   On ne cherche qu'à modifier les moyennes générales (qui ne seront donc plus cohérentes avec les autres données des étudiants mais ce n'est pas un souci dans notre contexte d'évaluation).
3. Ecrire une fonction `... normalisation(...)`, qui aura un prototype compatible avec le type `Recalcul`.
   Cette fonction de normalisation va prendre la moyenne générale de l'étudiant, et la modifier de telle sorte que la plus petite moyenne se retrouve à 0, et la plus grande moyenne se retrouve à 20.
   Si cette fonction rencontre un problème, le programme doit s'arrêter.
   Dans l'illustration tableau ci-dessous, on voit les moyennes avant et après normalisation. La plus petite moyenne se retrouve à 0, la plus grande se retrouve à 20. Les autres sont placées proportionnellement entre les 2.

	Avant	Après
Etudiant 1	5	0.00
Etudiant 2	13	13.33
Etudiant 3	7	3.33
Etudiant 4	14	15.00
Etudiant 5	15	16.67
Etudiant 6	17	20.00
Etudiant 7	9	6.67
Intervalle	[5, 17]	[0, 20]



4. Dans un programme principal, utiliser la fonction `... miseAJourNotes(...)` et la fonction `... normalisation(...)` pour afficher les nouvelles moyennes générales de chaque étudiant.

Exercice 2 (*Tableau de prénoms*) **[8.5 pts]**

Créer une fonction `... listePrenoms(...)` qui doit prendre en paramètre une promotion d'étudiants, et qui doit retourner en sortie un tableau de chaines de caractères ainsi que sa taille.

Respecter les conditions suivantes pour réaliser cette fonction :
— les données de la structure de la promotion ne doivent pas être modifiées par cette fonction,
— toutes les chaines du tableau de sortie doivent être situées les unes à la suite des autres dans la mémoire,
— chaque chaine du tableau de sortie contiendra le prénom d'un étudiant, en lettres majuscules (si des caractères accentués composent le prénom, on autorise que ces caractères ne soient pas modifiés),
— la valeur NULL sera renvoyée si un problème survient.

Dans un programme principal, utiliser cette fonction et afficher chaque prénom du tableau de sortie à l'écran.

Inutile de libérer la mémoire allouée dans cet exercice.

Exercice 3 (*Endianness*) **[5.5 pts]**

Suivant les technologies utilisées (matérielles) l'ordre des données en mémoire n'est pas forcément le même que celui sur nos machines. Pour s'adapter à toutes les situations on propose de créer une fonction pour modifier l'ordre des octets dans la mémoire.

1. Créer une fonction `... inverser_ordre(...)` qui va prendre en paramètres un pointeur générique, une taille de données (en octets), ainsi que le nombre de données à traiter, et qui va renvoyer les données modifiées.

   Cette fonction va (entre autres) :
   — conserver le contenu des données en entrée totalement intact
   — vérifier la taille de chaque donnée (2, 4 et 8 sont des valeurs acceptées)
   — prendre chaque donnée d'entrée et inverser l'ordre de ses octets en mémoire avant de les renvoyer
   — renvoyer l'adresse NULL si un problème est rencontré

2. Dans un programme principal, prendre une structure `Etudiant` déjà allouée en mémoire et remplie (prendre un étudiant **au hasard dans votre liste**) et la sauvegarder dans un nouveau fichier au format binaire en respectant les conditions suivantes :
   — sauvegarder à la suite et dans l'ordre le nom, le prénom, l'identifiant, l'age, la moyenne générale, puis chaque matière (le nom suivi de la note) sans aucun autre octet supplémentaire.
   — les chaines de caractères doivent conserver leur ordre naturel
   — chaque variable primitive doit être sauvegardée en inversant les octets en fonction de sa taille.

   Le programme principal affichera les octets de la structure en Hexadécimal avant le traitement, et affichera les octets écrits dans le fichier pour vérifier que l'inversion, quand elle doit avoir lieu, est correcte.

   Le programme principal s'assurera que les données allouées temporairement sont bien désallouées avant la fin du programme.