TD 4 - Programmation modulaire, fonctions avec pointeurs, structures

Exercice 1 On souhaite écrire une fonction `saisieValIntervalle` qui réalise une saisie contrôlée d'un nombre n dans un intervalle $[a, b]$ donné. La fonction renvoie un booléen vrai si la valeur saisie est dans l'intervalle et faux sinon. Si la saisie est correcte, la valeur saisie est retournée dans un paramètre de sortie.

1. Donner la signature de cette fonction en langage algorithmique.
2. En déduire alors la déclaration `C` de cette fonction.
3. Compléter la définition de la fonction.
4. Écrire une fonction `main` qui utilise la fonction `saisieValIntervalle` pour récupérer 3 notes entre 0 et 20 puis affiche leur moyenne.
5. Représenter l'évolution de la pile lors de l'appel de l'exécution de ce programme.

Exercice 2 On veut réaliser un module `parfait` qui fournit les fonctions suivantes :

- teste si un entier est parfait ;
- calcule le nombre d'entiers parfaits dans un intervalle donné ;
- retourne le nième nombre parfait ;
- étant donnés deux entiers n et d, vérifie s'il y a un entier parfait dans l'intervalle $[n - d, n + d]$ (un booléen est retourné) et dans ce cas retourne dans un paramètre de sortie, le parfait le plus proche de n dans l'intervalle.

On rappelle qu'un nombre parfait est un nombre dont la somme de ses diviseurs est égale au double de ce nombre (ce problème a déjà été traité lors du TP2).

1. Écrire le fichier `parfait.h`.
2. Préciser le contenu du fichier `parfait.c` (la définition complète des fonctions sera finalisée en TP).
3. Une fonction `sommeDiv` sera utile pour la définition des différentes fonctions. Où est-il pertinent de la mettre ?

Exercice 3 On souhaite disposer d'une donnée structurée `temps` permettant de regrouper un nombre d'heures, un nombre de minutes, et un nombre de secondes (pouvant contenir des dixièmes, centièmes...). Une telle structure permettra à la fois de représenter un instant et une durée.

1. Proposer une définition de cette structure.
2. Proposer alors un module `temps` qui fournit les fonctions suivantes :
   - affiche un temps sous sa forme usuelle hh:mm:ss.dcm jusqu'au millième (on paramétrera les %d et %f pour qu'il affiche le temps sur exactement 12 caractères) ;
   - convertit un temps en secondes ;
   - convertit un nombre de secondes en temps ;
   - calcule une durée à partir de deux temps (un début et une fin) ;
   - calcule les heures d'arrivée (des temps) minimale et maximale à partir d'une heure de départ, d'une durée et d'un coefficient multiplicateur. L'idée est, dans le cas d'une course, de déterminer les heures min et max d'arrivée possibles étant donnés une heure de départ, une durée estimée pour le plus rapide, et un coefficient multiplicateur de la durée pour le dernier coureur. Par exemple : si l'heure de départ est 12:30:00.000, que la durée est 01:00:00.000 et que le coefficient est 2,5 (le dernier met 2,5 fois plus de temps que le premier) alors les heures min et max d'arrivée sont : 13:30:00.000 et 15:00:00.000.
     On proposera 3 versions de cette fonction : la V1 n'a pas de résultat et deux paramètres d'entrée-sortie; la V2 a un résultat et un paramètre d'entrée-sortie, la V3 n'a pas de paramètre d'entrée sortie mais retourne une structure intervalle à définir.

3. Écrire un programme de test de ce module qui :

   - affiche le nombre de secondes du temps 12:09:43.246 à l'écran comme suit :
     `Le temps 12:09:43.245 correspond à 43783.246094 secondes.`
   - affiche les heures min et max d'arrivée (avec les 3 versions de la fonction), en prenant comme heure de départ 12:09:43.246, comme durée 1h30 et comme coefficient 2,5.