Homework 1 - Listes de Course

1 Mise en situation :

En tant qu'étudiant, on connaît tous la problématique du passage à la caisse pendant les courses du mois et de l'embarras possible dans le cas ou l'argent serait insuffisant ! Dans ce premier devoir noté du cours, vous allez mettre en pratique tout ce que vous avez appris jusqu'ici pour construire un code capable de vous aider.

La situation est la suivante : on va vous fournir différents reçus de course, correspondant aux courses du mois, que vous allez devoir utiliser pour effectuer quelques rapides statistiques et surtout prédire comment il vous faut organiser votre argent avant vos prochaines courses.

L'exercice se structure en deux parties principales, chacune d'entre elles est subdivisée en petite tâche pour vous guider durant votre implémentation. Les deux parties s'organisent comme des "exercices tiroirs" il est donc bien important d'effectuer la première partie correctement avant de se lancer dans la seconde.

Quelques remarques générales

- Suivez bien les consignes ci-dessous, soyez sûres que votre dernière fonction s'exécute correctement avant de vous lancer dans l'implémentation de la deuxième.
- **Les types seront clairement spécifiés dans l'énoncé, ne les modifiez pas dans les fonctions que nous vous demandons d'écrire.** Vous êtes libres d'utiliser les fonctions intermédiaires que vous voulez tant que vous les écrivez vous-mêmes, mais si vous changez l'entête d'une des fonctions qu'on vous demande d'écrire, vous obtiendrez 0 point pour la fonction associée.
- Soyez particulièrement attentifs au format d'affichage présenté dans les exemples, des points de style seront attribués pour leur respect.

N'oubliez jamais les cas spéciaux :

- Pensez toujours aux cas limites! Lorsqu'un utilisateur entre des valeurs au clavier, il n'est pas garanti qu'il écrive toujours quelque chose de cohérent. Similairement, si une fonction attend un certain argument en paramètre, même si le type de l'argument est bon, il est de votre responsabilité de vérifier que l'argument est valide.
- On vous a fourni 3 listes pour tester votre code (`list1.bin`, `list2.bin`, and `list3.bin`). Tous les exemples d'output présentés dans les consignes qui suivent prennent pour référence la première liste `list1.bin`. En ce qui concerne les deuxième et troisième listes (`list2.bin`, `list3.bin`), nous vous avons fourni, pour chacune, un exemple d'output attendu à la fin des consignes (en annexes Sec.4). Avoir le même output que les sorties en annexes ne garantit pas que votre code est complètement correct, nos tests seront plus difficiles, testent des cas spéciaux.

- Oublier les cas limites peut vous coûter des points lors de la correction, ou nous testerons non seulement que votre programme fonctionne, mais aussi que les cas limites sont bien gérés!

Rendez un code propre : rendez un code clair, avec des noms de variables et de fonctions pertinents. Pensez à bien indenter votre code, et à le commenter raisonnablement. Des points de style peuvent être retirés même si votre code fonctionne, si ce dernier est incompréhensible ou inutilement complexe.

2 Première Partie :

Avant de commencer à implémenter votre partie du code, vous allez devoir importer les listes d'achats que nous vous avons fournies. Afin de garantir l'intégrité de ces listes nous vous les avons fournies au format *binaire* ce qui signifie que ça ne sert à rien d'essayer de les ouvrir avec des éditeurs de textes, ceux-ci seront incapable des les modifier ou des les lires correctement. Pour pouvoir importer et manipuler ces listes, nous vous avons fourni un code préliminaire contenant les 3 éléments suivant :

1. Il contient un `header` un peu particulier déclaré de la manière suivante :
   `#include "readBinary.h"`
   Ce `header` vous permet de faire appel à la fonction dont vous avez besoin pour importer la liste. Cette fonction est la suivante :
   `readBinary(const char* filename, char list[N_PRODUCTS][N_VALUES][MAX_CHAR])`.
   IMPORTANT : **La fonction est définie dans un fichier séparé intitulé `readBinary.c` de manière à pouvoir utiliser cette définition librement dans le code il vous faut dire au compilateur de la considérer en utilisant la commande suivante pour compiler votre code :**

   ```
   gcc -Wall homework1.c readBinary.c -o homework1
   ```

   Remarquez que le fichier **"readBinary.c"** a été inclus et placé juste avant l'argument de sortie (toujours placé après l'option "-o"). Comme vous le voyez, cette fonction prend deux arguments :

   - **le nom du fichier** contenant la liste d'achats que vous devez importer (vous avez le choix entre 3 listes différentes : `list1.bin`, `list2.bin`, `list3.bin`).
   - **le tableau** que cette fonction va remplir contenant tous les achats de la liste que vous aurez choisie et que vous allez devoir manipuler `char list[N_PRODUCTS][N_VALUES][MAX_CHAR]`. Comme vous l'aurez certainement remarqué, il s'agit d'un de tableau de caractères multidimensionnel. Ce tableau possède `N_PRODUCTS` entrées, chacune correspondant à un achat, à chacun des achats sont associées `N_VALUES`=4 valeurs : la catégorie, le nom du produit, la quantité de produits achetés et le prix *unitaire* du produit. Finalement chacune de ces valeurs représente un tableau de `MAX_CHAR` caractères (au maximum).

   Pour vous aider un peu voila à quoi ressemble graphiquement la liste une fois qu'elle a été remplie par la fonction :

N_PRODUCTS = 28

Catégorie	Nom du produit	Quantité	Prix unitaire
3	Oranges Blondes - 2.50 kg	1	6.5
MAX_CHAR = 100	MAX_CHAR	MAX_CHAR	MAX_CHAR

N_VALUES = 4

2. Le tableau de caractères qui sera rempli par la fonction a aussi déjà été déclaré pour vous, et il s'agit du tableau de `char` : `char list[N_PRODUCTS][N_VALUES][MAX_CHAR]`. Comme vous pouvez le constater, les dimensions de ce tableau sont définies par des `macros` en début de code. Ces `macros` sont les suivantes :

   - `#define N_PRODUCTS 28` représente le nombre maximal d'achats présents dans une liste.
   - `#define N_VALUES 4` représente le nombre de valeurs/propriétés associées à chaque achat.
   - `#define MAX_CHAR 100` représente le nombre maximal de caractères possible pour chacune des valeurs.

   Enfin nous avons aussi défini pour vous une `macro` restreignant le nombre maximal de catégories possible pour toutes les listes :
   `#define MAX_CATEG 10`, vous indiquant qu'il ne peut y avoir que 10 catégories au maximum. Vous pourrez donc, à chaque fois que nécessaire, utiliser cette `macro` pour définir la taille des tableaux qui contiennent autant d'éléments que de nombre de catégories.

3. Finalement le code que l'on vous a fourni contient aussi un `header` supplémentaire `<string.h>`. Il s'agit d'une librairie standard utilisée pour manipuler les variables de type `string` (ou tableau de caractères). Si vous souhaitez en apprendre plus sur cette librairie, nous vous invitons à aller faire un tour sur ce lien. Dans le cadre de cet exercice, vous n'aurez besoin que de deux fonctions particulières :

   - **int atoi(const char *string)** cette fonction vous permet de convertir un tableau de caractères oui `string` en un entier. Elle prend donc en argument un tableau de caractères et renvois l'entier (num.) correspondant.

     Exemple : int num = atoi("abcd12") => num = 12
     Exemple : int num = atoi("abcd12.65") => num = 12
     Exemple : int num = atoi("12") => num = 12

   - **int atof(const char *string)** De manière similaire cette fonction vous permet de convertir un tableau de caractères en valeur numérique à virgule flottante `float`.

     Exemple : int num_dec = atof("abc12.65") => num_dec = 12.65
     Exemple : int num_dec = atof("12.65") => num_dec = 12.65

Avant de vous lancer dans l'implémentation de la suite du code il vous faudra donc faire usage de la fonction décrite précédemment en l'appelant au début de votre code pour importer la liste d'achats que vous aurez choisis dans le tableau qui a été prévu a cet effet, comme suit :

```
readBinary("filename.bin",list);
```

Où il vous faudra remplacer `"filename.bin"` par le nom du fichier de l'une des listes fournies : `list1.bin,list2.bin,list3.bin`
Pour la suite vous n'avez plus qu'à suivre les indications qui vous sont données.

2.1 Créer votre structure

Votre première tache sera de définir/déclarer une `structure` appelée `struct item` représentant une ligne dans la liste d'achats. Cette structure doit contenir les 4 données définies pour chaque achat en utilisant le bon `type` de variable. La catégorie et la quantité devraient correspondre à des variables de type `int` (`category, quantity`), le prix un `float` (price) et enfin le nom devrait rester un `tableau de caractères` (name[MAX_CHAR]).
Pour pouvoir représenter la liste d'achats complète avec nos nouvelles structures, il faut aussi définir/déclarer un tableau de structures `struct item` de taille `N_PRODUCTS` appelé `list_items`. Vous allez remplir ce tableau de structure au prochain point.

2.2 Créer votre tableau de structure

Votre seconde tâche est d'implémenter la fonction suivante:

```
void process_list(char list_char[N_PRODUCTS][N_VALUES][MAX_CHAR], struct item
    list_struct[N_PRODUCTS])
```

Cette fonction prendra deux arguments : le premier sera le tableau de caractères représentant la liste initiale, le second est un tableau de structures. La fonction doit remplir le tableau de structures avec chacun des achats présents dans la liste initiale définie par les 4 valeurs qui lui sont associées. **Attention : souvenez-vous que pour convertir une variable de type `string` en une variable d'un autre type il vous faudra utiliser les 2 fonctions mentionnées en début de première partie : `atoi() and atof()`.**
Après avoir fini l'implémentation de cette fonction, vous devez l'appeler dans le programme principal et initialiser le tableau `list_items`.

2.3 Déterminer le nombre de catégorie

Comme indiqué précédemment, chacun des achats qui sont présents dans la liste appartient à une catégorie. Avant de pouvoir poursuivre, vous avez besoin de connaître le nombre total de catégories. Vous devez donc implémenter la fonction suivante :

```
int find_categories(struct item list_struct[N_PRODUCTS])
```

Cette fonction prendra en argument le tableau de structure et retournera le nombre de catégories distinctes présentent dans la liste. De plus, cette fonction doit afficher le nombre des catégories dans le format suivant:

> This list contains **x** categories !

Petit indice : les catégories sont des ***int*** *ou la première est libellée "0", et chaque catégorie suivante a un ID plus grand de 1 que le précédent. Vous pouvez vous référer à l'exemple fourni au tableau 1.*
Après avoir implémenté la fonction, vous devez l'appeler dans le programme principal et stocker le résultat dans une variable nommée `int num_categories`. Une fois que vous aurez trouvé le nombre de catégories dans liste vous pourrez utilisez cette variable dans toutes les fonctions qui en

ont besoin. **En particulier il vous sera possible de restreindre les boucles concernant les tableaux de dimensions `MAX_CATEG`, à ne considérer que les `num_categories` premiers éléments.**

2.4 Vérification du tableau et quelques statistiques

Maintenant que vous avez construit votre tableau de structure contenant chacun de vos achats, vous allez créer la fonction suivante :

```
void print_list_chunk(struct item list_struct[N_PRODUCTS], int n_categ)
```

Cette fonction prend deux arguments: le tableau de structure que vous avez rempli ainsi que le nombre de catégories présente dans la liste. Une fois appelée, la fonction doit demander à l'utilisateur d'entrer deux nombres représentant les index qui forment l'intervalle des lignes du tableau que la fonction devra afficher. En particulier, si cet intervalle (index de début ou index de la fin) n'est pas consistent avec la taille de votre tableau, votre fonction devra afficher un message d'erreur et demander de nouveau un index, jusqu'à ce que l'intervalle donné soit consistant avec votre tableau. Dans le cas défavorable, l'output devrait être le suivant :

```
Enter the index of the first item you want to print : -1
Enter the index of the last item you want to print : 0
Invalid Range !
Enter the index of the first item you want to print :
```

ou encore :

```
Enter the index of the first item you want to print : 6
Enter the index of the last item you want to print : 1
Invalid Range !
Enter the index of the first item you want to print :
```

Et dans le cas où l'intervalle est consistant :

```
Enter the index of the first item you want to print : 2
Enter the index of the last item you want to print : 3
This is an extract of the grocery list from item-2 to item-3
Item No 2:
- Category: 1
- Name: Longobardi Sauce de tomates au basilic - 700.00 g
- Quantity: 2
- Unitary Price: 5.10
----------
Item No 3:
- Category: 3
- Name: Oranges Blondes - 2.50 kg
- Quantity: 1
```

- Unitary Price: 6.50

De plus, vous devrez aussi créer une petite fonction appelée :

```
void print_stats(struct item list_struct[N_PRODUCTS], int n_categ)
```

Cette fonction vous permettra d'afficher les statistiques suivantes par rapport à votre liste :

- le nombre total de produits achetés par catégorie (pour chacune des catégories)
- le prix total des achats

De plus si lors du calcul des statistiques la fonction rencontre un achat appartenant à une catégorie en dehors de l'ensemble défini par `int n_categ` la fonction devra se terminer en affichant le message suivant : *"A category is missing or has not been properly set !"*.

Si cette fonction a été correctement implémentée et qu'il n'y a pas des catégories qui manquent, la fonction devrait afficher un output ressemblant à ceci :

```
Number of articles found in each category :
7 articles found in category 0
6 articles found in category 1
9 articles found in category 2
17 articles found in category 3
For a total price of : 174.10
```

Après avoir fini l'implémentation de ces deux fonctions, vous devez les appeler dans le programme principal.

Faites particulièrement attention au format d'affichage de la liste et ceux des différentes variables a afficher. Des points de styles seront attribués pour les formats d'affichage, par format il s'agit avant tout des **formats d'affichage des variables**, de plus l'aspect général de l'output devrait être respecté (l'ordre de sortie et les retours à la ligne p. ex. mais pas les whitespace).

3 Deuxième Partie :

Maintenant que vous êtes sûres que votre tableau de structures a été déclaré et initialisé correctement, vous allez pouvoir faire quelques prédictions.

3.1 Dépenses par catégorie

Votre première tâche dans cette deuxième partie sera d'implémenter une fonction appelée:

```
float calculate_ratio(struct item list_struct[N_PRODUCTS], int category_id)
```

Cette fonction prendra en argument un tableau de structures ainsi que l'identifiant d'une des catégories. La fonction doit retourner le ratio d'argent dépensé pour cette catégorie (c-à-d le rapport entre la quantité d'argent dépensée pour cette catégorie et la quantité totale d'argent dépensé).
Après avoir fini l'implémentation de cette fonction, vous devez implémenter une nouvelle fonction appelée :

```
void calculate_all_ratios(struct item list_struct[N_PRODUCTS], int n_categ, float
    ratios[MAX_CATEG])
```

Cette fonction doit se servir (faire appel) de la fonction `calculate_ratio`. Vous pourrez ainsi calculer les ratios pour chacune des catégories dans la liste et stocker chacun des ratios dans un tableau nommé `float ratios[MAX_CATEG]` (4ème argument de la fonction).
De plus, cette fonction doit également vérifier que les ratios sont bien calcules, c-a-d la fonction doit vérifier que la somme de tous les ratios est bien égale à 1. **N'oubliez pas que la variable `int n_categ` vous permet de retreindre le nombre de variables utilisées dans le tableau de dimension `MAX_CATEG`.**
Une fois que ces valeurs sont calculées, vous devez, à la fin de votre fonction, les afficher (à la suite du dernier affichage) selon l'output suivant :

```
Checking ratios by category : 0.140 | 0.223 | 0.245 | 0.392, total = 1.000
```

Après avoir fini l'implémentation de cette fonction, vous devez l'appeler dans le programme principal.

3.2 Prévoyez vos prochaines dépenses

Pour la prochaine tache, vous devrez implémenter la fonction suivante :

```
void print_budget_per_categ(float ratios[MAX_CATEG], int n_categ, float total_solde)
```

Cette fonction vous permettra d'afficher la quantité d'argent que vous devrez réserver pour chaque catégorie sur la base des ratios précédemment calculés. Comme pour la tâche précédente votre tableau de dimension `MAX_CATEG` est probablement plus grand que le nombre de catégories présentent dans votre liste. La variable `int n_categ` correspond au nombre de catégories présentent dans votre liste actuelle et permet de restreindre les variables utilisées dans votre tableau.

Votre fonction devrait afficher les données dans le format suivant :

```
Next time you should spend : 14.01 CHF for category 0
Next time you should spend : 22.29 CHF for category 1
Next time you should spend : 24.47 CHF for category 2
Next time you should spend : 39.23 CHF for category 3
```

Après avoir fini l'implémentation de cette fonction, vous devez premièrement créer une nouvelle variable de type float qui représente la quantité d'argent que vous souhaitez dépenser pour vos

prochains achats. Vous pouvez l'initialiser à `100 CHF`. En utilisant cette variable et le tableau des ratios calculés précédemment, vous devez appeler cette fonction dans le programme principal.

3.3 Augmentation des prix

Imaginez maintenant que le supermarché dans lequel vous allez faire vos courses met un point d'honneur à respecter la transparence et vous avertit que tous les articles des catégories `0 et 2` augmenteront de **10%**.
Vous devez à présent implémenter la fonction suivante :

```
float calculate_excess_after_rising(float old_ratios[MAX_CATEG], float
    percent_raise, int for_category[MAX_CATEG], float new_ratios[MAX_CATEG], int
    n_categ, int n_changed, float total_solde)
```

Cette fonction devra donc contenir les arguments suivants :

- un tableau contenant les ratios avant l'augmentation des prix, précédemment calculés avec la fonction `calculate_all_ratios`
- la hausse des prix exprimée en pourcentage (pour tous les catégories concernées par l'augmentation, l'augmentation des prix est le meme)
- les catégories concernées par l'augmentation contenues dans un tableau `for_category[MAX_CATEG]` et identifiées par leurs valeurs
- un entier représentant le nombre de catégories concernées par la hausse des prix (`int n_changed`), c-a-d un entier qui spécifie le nombre d'éléments dans `for_categories[MAX_CATEG]`
- un entier représentant le nombre de catégories dans la liste (`int n_categ`)
- un nouveau tableau capable d'accueillir les nouveaux ratios calculés par la fonction, évalués à la suite de la hausse des prix
- le solde total a la disposition

Cette fonction se chargera de prendre en considération l'augmentation des prix dans les catégories concernées, et affichera la quantité d'argent qu'il vous manquera si vous conservez le même budget et les memes ratios calculés précédemment (`old_ratios[MAX_CATEG]`). De plus cette fonction prendra aussi en compte ces ratios calculés précédemment et remplira un nouveau tableau (`new_ratios[MAX_CATEG]`) avec les ratios mis à jour apres l'augmentation des prix. La fonction doit retourner la somme manquante et mettre à jour le tableau des nouveau ratios. De plus, la fonction doit afficher la somme manquante de la manière suivante :

```
Considering now that the prices of category 0 and 2 have been raised by 10%
You are exceeding your budget by 3.85 CHF
```

Après avoir fini l'implémentation de cette fonction, vous devez l'appeler dans le programme principal.

3.4 Évitons l'embarras

Pour finir, puisque vous ne pouvez pas dépenser plus d'argent que les 100 CHF prévus initialement, vous devez implémenter une dernière fonction appelée :

```
void redistribute(float old_ratios[MAX_CATEG], float new_ratios[MAX_CATEG], int
    n_categ, float total_solde)
```

Cette fonction prendra 4 arguments : les premiers ratios calculés (`old_ratios`), les ratios mis à jour suite à l'augmentation des prix (`new_ratios`), le nombre de catégories présentes dans votre liste (`n_categ`) et enfin votre budget (`total_solde`), qui n'a pas changé. Votre fonction tiendra compte des anciens et des nouveaux ratios pour vous prédire les corrections qu'il faudra effectuer sur votre première répartition du budget (Sec. 3.2). Cette fonction affichera donc la quantité d'argent, pour chacune des catégories, qu'il vous faudra enlever ou rajouter pour maintenir les anciens ratios. Votre fonction devra afficher le résultat comme suit :

```
Conserving a total cash-bill of 100.00 CHF.
Because of the raise you should add 0.83 CHF. from category 0 expenses
Because of the raise you should remove 0.83 CHF. from category 1 expenses
Because of the raise you should add 1.45 CHF. from category 2 expenses
Because of the raise you should remove 1.45 CHF. from category 3 expenses
```

4 Annexes

Ci-dessous la sortie attendue de votre code si vous utilisez la deuxième liste:

```
This list contains 4 categories !
Enter the index of the first item you want to print : 2
Enter the index of the last item you want to print : 3
This is an extract of the grocery list from item-2 to item-3
Item No 2:
- Category: 1
- Name: Persil
- Quantity: 1
- Unitary Price: 1.75
_________-
Item No 3:
- Category: 3
- Name: Valais - Eau minérale naturelle - non gazeuse
- Quantity: 2
- Unitary Price: 7.20
_________
Number of articles found in each category :
7 articles found in category 0
5 articles found in category 1
12 articles found in category 2
13 articles found in category 3
For a total price of 414.83
Checking ratios by category : 0.443 | 0.128 | 0.268 | 0.161, total = 1.000
Next time you should spend : 44.30 CHF. for category 0
Next time you should spend : 12.80 CHF. for category 1
Next time you should spend : 26.83 CHF. for category 2
Next time you should spend : 16.08 CHF. for category 3
Considering now that the prices of category 0 and 2 have been raised by 10%
You are exceeding your solde by 7.11 CHF.
Conserving a total cash-bill of 100.00 CHF.
Because of the raise you should add 1.19 CHF. from category 0 expenses
Because of the raise you should remove 0.85 CHF. from category 1 expenses
Because of the raise you should add 0.72 CHF. from category 2 expenses
Because of the raise you should remove 1.07 CHF. from category 3 expenses
```

Ci-dessous la sortie attendue de votre code si vous utilisez la troisième liste:

```
This list contains 4 categories !
Enter the index of the first item you want to print : 2
Enter the index of the last item you want to print : 3
This is an extract of the grocery list from item-2 to item-3
Item No 2:
- Category: 1
- Name: Le Saunier de Camargue - Nature Sauvage - Fleur de Sel
- Quantity: 1
- Unitary Price: 6.95
----------
Item No 3:
- Category: 3
- Name: Valais - Eau minérale naturelle - non gazeuse
- Quantity: 2
- Unitary Price: 7.20
----------
Number of articles found in each category :
8 articles found in category 0
5 articles found in category 1
11 articles found in category 2
13 articles found in category 3
For a total price of 232.08
Checking ratios by category : 0.272 | 0.108 | 0.238 | 0.383, total = 1.000
Next time you should spend : 27.17 CHF for category 0
Next time you should spend : 10.79 CHF for category 1
Next time you should spend : 23.78 CHF for category 2
Next time you should spend : 38.26 CHF for category 3
Considering now that the prices of category 0 and 2 have been raised by 10%
You are exceeding your solde by 5.10 CHF.
Conserving a total cash-bill of 100.00 CHF.
Because of the raise you should add 1.27 CHF. from category 0 expenses
Because of the raise you should remove 0.52 CHF. from category 1 expenses
Because of the raise you should add 1.11 CHF. from category 2 expenses
Because of the raise you should remove 1.86 CHF. from category 3 expenses
```