I31 – TP5

Contact : julien.seinturier@univ-tln.fr

Prérequis : Pour ce TP, se placer dans le répertoire ~/L2/I31/TP5. Si celui-ci n'existe pas, le créer.

Indication : Il est possible de réutiliser le code ou les fonctions des exercices précédents si nécessaire.

1. Calcul de statistiques simples

Cette partie de TP vise à réaliser des calculs de statistique simples (moyenne, variance, médiane) à partir de séries de valeurs de taille dynamique. Les pointeurs et l'allocation dynamique sont une bonne solution pour implanter ce genre de traitement.

Exercice 1.1.
Ecrire un programme `tp5-stat.c` contenant une fonction `int* creation_serie(unsigned int n)` qui crée dynamiquement un espace en mémoire permettant de stocker une série de valeurs de taille `n` et retourne un pointeur vers celui-ci. Le programme doit contenir également une fonction `main` qui permet de faire saisir à l'utilisateur une série de 5 valeurs.

Exercice 1.2.
Dans le programme `tp5-stat.c`, ajouter une fonction `void affiche_serie(int* s, unsigned int n)` qui affiche la série de valeurs pointée par la variable `s` sous la forme :

```
( s[0], …, s[i], …, s[n-1] )
```

Modifier la fonction `main` du programme pour afficher la série des 5 valeurs saisies.

Exercice 1.3.
Dans le programme `tp5-stat.c`, ajouter une fonction `void destruction_serie(int** ps)` qui désalloue l'espace mémoire contenant la série pointée par la variable `ps` et affecte la valeur `NULL` à la série pointée. Modifier la fonction `main` du programme pour détruire la série utilisée avant la fin de l'exécution.

Exercice 1.4.
Dans le programme `tp5-stat.c`, ajouter une fonction `float moyenne(int* s, unsigned int n)` qui calcule et retourne la moyenne de la série de `n` valeurs pointée par `s`. Si la série est vide, la fonction doit retourner la valeur spéciale `NAN` (*Not a Number*) de l'en-tête `math.h`. Modifier la fonction `main` afin de faire saisir à l'utilisateur le nombre de valeurs souhaitées dans la série, de saisir celle-ci et d'en afficher la moyenne.

RAPPEL : Soit $X = (x_1, \cdots, x_i, \cdots, x_n)$ une série de n valeurs. La moyenne empirique, notée $\bar{x}$, de la série X est définie par la somme des valeurs de X divisée par le nombre n de valeur. Plus formellement :

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Exercice 1.5.
Dans le programme `tp5-stat.c` ajouter une fonction `float variance(int* s, unsigned int n)` qui calcule et retourne la variance de la série de `n` valeurs pointée par `s`. Si la série est vide, la fonction doit retourner la valeur spéciale `NAN` (*Not a Number*) de l'en-tête `math.h`. Modifier la fonction `main` afin d'y ajouter l'affichage de la variance de la série saisie.

RAPPEL : Soit $X = (x_1, \cdots, x_i, \cdots, x_n)$ une série de n valeurs et soit $\bar{x}$ la moyenne empirique de X. La variance, notée V, de la série X est définie par la moyenne des carrés des écarts entre les valeurs de X et la moyenne empirique $\bar{x}$. Plus formellement :

$$V = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Exercice 1.6.
Dans le programme `tp5-stat.c` ajouter une fonction `int* tri_croissant(int* s, unsigned int n)` qui retourne un pointeur vers une série contenant les `n` valeurs pointée par `s` triées dans l'ordre croissant. La série `s` ne doit pas être modifiée, le choix de l'algorithme de tri est libre et si la série `s` est vide, la fonction doit retourner `NULL`. Modifier la fonction `main` afin d'y ajouter l'affichage de la série triée (penser à bien désallouer tous les pointeurs avant la fin de l'exécution).

Exercice 1.7.
Dans le programme `tp5-stat.c` ajouter une fonction `float mediane(int* s, unsigned int n)` qui calcule et retourne la médiane de la série de `n` valeurs pointée par `s`. Si la série est vide, la fonction doit retourner la valeur spéciale `NAN` (*Not a Number*) de l'en-tête `math.h`. Modifier la fonction `main` afin d'y ajouter l'affichage de la médiane de la série saisie.

RAPPEL : Soit $X = (x_1, \cdots, x_i, \cdots, x_n)$ une série de n valeurs triées par ordre croissant. La médiane, notée Me, de la série X est définie par la valeur située au milieu de la séquence. Plus formellement :

$$Me = \begin{cases} x_{\frac{n}{2}}, si\ n\ est\ pair \\ x_{\frac{n+1}{2}}, si\ n\ est\ impair \end{cases}$$

Suite en page suivante…

2. Utilisation de structures

Il apparait dans la partie 1 du TP qu'afin de gérer une série de valeur il est indispensable de connaitre :

- Le pointeur vers la série
- Le nombre de valeurs contenues dans la série

Une telle configuration alourdi les prototypes de fonction utilisant des séries et peut entrainer des erreurs si l'une des deux informations vient à être modifiée.

Afin de résoudre ce problème, une solution est de grouper les deux informations au sein d'une même structure.

Exercice 2.1.
Ecrire un programme `tp5-serie.c` contenant la déclaration d'un type permettant de représenter une série de valeurs. Ce type sera nommé `serie` et sera définit par une structure contenant les champs suivants :

- Une variable nommé `valeurs` de type `int*`.
- Une variable `taille` de type `unsigned int` représentant le nombre de valeurs de la série.

Le programme doit contenir également une fonction `main` qui déclare une valeur de type `serie`.

Exercice 2.2.
En vous inspirant de l'*exercice 1.1.* de la partie 1, dans le programme `tp5-serie.c`, ajouter une fonction `serie creation_serie(unsigned int n)` qui crée dynamiquement une série pouvant contenir `n` valeurs. Modifier la fonction `main` afin de faire saisir à l'utilisateur une série de 5 valeurs.

Exercice 2.3.
En vous inspirant de l'*exercice 1.2.* de la partie 1, dans le programme `tp5-serie.c`, ajouter une fonction `void affiche_serie(serie s)` qui affiche la série `s` sous la forme :

```
( s[0], …, s[i], …, s[n-1] )
```

Modifier la fonction `main` du programme pour afficher la série des 5 valeurs saisies.

Exercice 2.4.
En vous inspirant de l'*exercice 1.3.* de la partie 1, dans le programme `tp5-serie.c`, ajouter une fonction `void destruction_serie(serie* ps)` qui désalloue l'espace mémoire contenant la série pointée par la variable `ps` et affecte la valeur `NULL` à la série pointée et `0` à sa taille. Modifier la fonction `main` du programme pour détruire la série utilisée avant la fin de l'exécution.

AIDE : Soit `s` une structure contenant le champ `c` et soit `ps` une variable représentant un pointeur vers cette structure (`s*`). L'accès au champ `c` de `s` via la variable `ps` peut être fait de deux différentes manières :

- `ps->c`
- `(*ps).c`

Exercice 2.5.
En vous inspirant de l'*exercice 1.4.* de la partie 1, dans le programme `tp5-serie.c`, ajouter une fonction `float moyenne(serie s)` qui calcule et retourne la moyenne de la série `s`. Si la série est vide, la fonction doit retourner la valeur spéciale `NAN` (*Not a Number*) de l'en-tête `math.h`. Modifier la fonction `main` afin de faire saisir à l'utilisateur le nombre de valeurs souhaitées dans la série, de saisir celle-ci et d'en afficher la moyenne.

Exercice 2.6.
En vous inspirant de l'*exercice 1.5.* de la partie 1, dans le programme `tp5-serie.c`, ajouter une fonction `float variance(serie s)` qui calcule et retourne la variance de la série `s`. Si la série est vide, la fonction doit retourner la valeur spéciale `NAN` (*Not a Number*) de l'en-tête `math.h`. Modifier la fonction `main` afin d'y ajouter l'affichage de la variance de la série saisie.

Exercice 2.7.
En vous inspirant de l'*exercice 1.6.* de la partie 1, dans le programme `tp5-serie.c`, ajouter une fonction `serie tri_croissant(serie s)` qui retourne une série contenant mêmes valeurs que celles contenues dans la série `s` mais triées dans l'ordre croissant. La série `s` ne doit pas être modifiée, le choix de l'algorithme de tri est libre et si la série `s` est vide, la fonction doit retourner `NULL`. Modifier la fonction `main` afin d'y ajouter l'affichage de la série triée (penser à bien désallouer tous les pointeurs avant la fin de l'exécution).

Exercice 2.8.
En vous inspirant de l'*exercice 1.7.* de la partie 1, dans le programme `tp5-serie.c`, ajouter une fonction `float mediane(serie s)` qui calcule et retourne la médiane de la série `s`. Si la série est vide, la fonction doit retourner la valeur spéciale `NAN` (*Not a Number*) de l'en-tête `math.h`. Modifier la fonction `main` afin d'y ajouter l'affichage de la médiane de la série saisie.

Suite en page suivante…

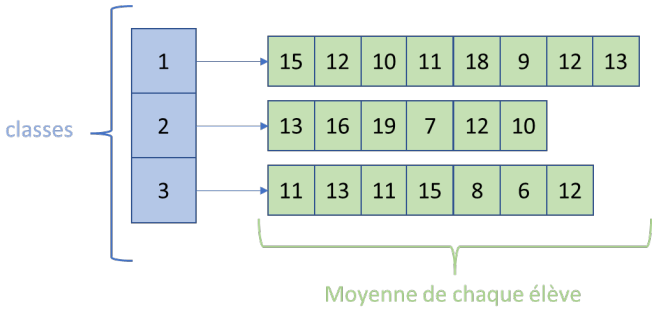
3. Gestion d'un ensemble de classes

Indication : Il est **fortement recommandé** de réutiliser du code et les fonctions écrites pour la **partie 2**.

Cette partie de TP vise à illustrer les capacités de calcul statistique implantées en première partie en simulant la gestion des notes d'un ensemble de classes dans une école primaire. Les moyennes seront représentées par des nombres entiers entre 0 et 20.

D'un point de vue statistique, une classe est représentée par une série de valeurs correspondant chacune à la moyenne individuelle d'un élève de la classe. Une école est un ensemble de classes chacune identifiée par un nombre entier allant de 1 (première classe) à n (dernière classe). Chaque classe peut être composée d'un nombre différent d'élèves et chaque école ayant un nombre variable de classes.

Par exemple, une école comportant 3 classes, la première composée de 8 élèves, la seconde de 6 élèves et la troisième de 7 élèves peut se représenter comme suit :



Dans la figure précédente, le premier élève de la classe 1 à une moyenne de 15, le premier élève de la classe 2 à une moyenne de 13, le troisième élève de la classe 3 à une moyenne de 11.

Exercice 3.1.
Ecrire un programme `tp5-ecole.c` contenant la définition d'un type `ecole` reposant sur une structure dont les champs sont :

- Un pointeur nommé **classes** vers un ensemble de `serie` (réutiliser le type définit pour l'exercice 2.1 de la partie 2)
- Une variable nommée **nb_classes** de type `unsigned int` qui représente le nombre de classes présentes dans l'école.

Le programme doit contenir également une fonction `main` qui déclare une variable **e** de type `ecole`.

Exercice 3.2.
Dans le programme `tp5-ecole.c`, ajouter une fonction `ecole creation_ecole(unsigned int m)` qui crée une école de `m` classes. Si la valeur de `m` est inférieure à 1, la fonction renvoie une variable de type `ecole` dont le champ `classe` est à `NULL` et le champ `nb_classes` à `0`. Modifier la fonction `main` afin de permettre à l'utilisateur de saisir un nombre de classes et de créer une variable `e` de type `ecole` avec le nombre de classes adéquat.

Exercice 3.3.
Dans le programme `tp5-ecole.c`, ajouter une fonction `void saisie_ecole(ecole e)` qui demande à l'utilisateur de saisir l'ensemble des notes d'une école. L'école utilisée doit avoir été créée avant d'être saisie. La fonction a pour algorithme :

```
pour chaque classe c de e
  demander le nombre d'eleves
  créer une serie à partir du nombre d'élèves
  pour chaque eleve e
    demander sa moyenne individuelle
    affecter la moyenne a la valeur e de la serie
  finpour
  affecter la serie saisie à la classe c
finpour
```

Exercice 3.2.
Dans le programme `tp5-ecole.c`, ajouter une fonction `void affiche_ecole(ecole e)` qui affiche l'école `e` sous la forme :

```
Classe 1 : ( s[0], …, s[i], …, s[n-1] )
…
Classe m : ( s[0], …, s[j], …, s[k-1] )
```

Modifier la fonction `main` du programme pour y ajouter l'affichage de l'école saisie.

Exercice 3.3.
Dans le programme `tp5-ecole.c`, ajouter une fonction `void destruction_ecole(ecole* pe)` qui désalloue toute la mémoire utilisée par l'école pointée par `pe`. Modifier la fonction `main` du programme pour détruire l'école utilisée avant la fin de l'exécution.

Exercice 3.4.
Dans le programme `tp5-ecole.c`, Modifier la fonction `main` du programme pour permettre à l'utilisateur :

- De saisir une école complète
- D'afficher la moyenne d'une classe
- D'afficher la variance d'une classe
- D'afficher la médiane d'une classe
- De quitter le programme