

## IFT1166 – TRAVAIL PRATIQUE #3 – 18 Mars 2K

### De la statistique au menu, et du bonus dans l'air!

"Selon les derniers chiffres, 43 % des statistiques sont fausses." [inconnu(e)]

Mohamed Lokbani

**LA DATE LIMITE EST LE 12 Avril 2K Minuit, avec une pénalité de 2 points par jours de retard.**

**À faire en équipe de deux.**

**1. BUT:** Ce TP a pour but de vous faire pratiquer les matières suivantes :

- \* Les patrons de fonctions et de classes.
- \* L'héritage simple.

#### Conseils

- \* Lisez l'énoncé de A à Z, puis en cas de problèmes contactez-nous.
- \* N'attendez pas la semaine précédant la remise avant de commencer ... vous n'aurez pas le temps!

**2. Description:** Dans ce travail pratique, vous allez concevoir un patron de classe permettant de stocker des données de types prédéfinis, de calculer des fréquences sur ces données, et finalement, de simuler une représentation graphique de ces données.

**3. Calcul des fréquences:** Le calcul de fréquence consiste à mesurer le nombre de fois qu'un élément apparaît dans un ensemble de données.

Si nous représentons cet ensemble de données par le tableau `data`, de taille `taille` et contenant des données de type entier, suivants:

```
data[8] = {1, 2, 1, 3, 4, 2, 6, 2}
```

les fréquences d'apparition des éléments dans le tableau `data` sont comme suit:

| élément du tableau data | fréquence |
|-------------------------|-----------|
| 1                       | 2         |
| 2                       | 3         |
| 3                       | 1         |
| 4                       | 1         |
| 6                       | 1         |

#### 4. Travail à réaliser

##### 4.1 Patron de la classe Enregistrement

**4.1.1 classe Enregistrement:** Les statistiques sont réalisées sur un ensemble de données préalablement stockées dans le tableau `data` de type `T` et de taille `taille`. Vous avez à écrire, en langage C++, le patron de la classe **Enregistrement**, contenant **au minimum** les membres suivants:

```
Enregistrement {
private:
    // tableau devant contenir les données originale du type T.
    T* data;
    // la taille du tableau data.
    int taille;
```

```

    // le nom de l'enregistrement.
    string nom;

    // vous pouvez ajouter d'autres membres (fonctions ou données) private.
public:
    // constructeur qui initialise les membres données privés à zéro.
    Enregistrement();

    // constructeur permettant de charger l'enregistrement avec
    // les données brutes, sur lesquelles des statistiques doivent
    // être calculées.
    Enregistrement(int nbre_elt,const T tab[],string nom);

    // destructeur
    ~Enregistrement();

    // constructeur de recopie
    Enregistrement(const Enregistrement&);

    // opérateur d'affectation
    Enregistrement& operator=(const Enregistrement&);

    // opérateur de sortie
    friend ostream& operator<<<>(ostream&,const Enregistrement<T>&);

    // vous pouvez ajouter d'autres membres (fonctions ou données) public.
};

```

En plus de cela, vous devez calculer la fréquence d'apparition des éléments du type **T**, contenus dans le tableau **data**. Pour ce faire, vous pouvez mémoriser le résultat du calcul de fréquence dans un autre tableau que vous déclarez comme membre données private de la classe **Enregistrement**. Sinon vous pouvez choisir une autre approche pour stocker les données traitées.

**4.1.2. Le paramètre T:** Le paramètre **T** peut prendre la place des trois types suivants: **int**, **char**, **string**.

**En Bonus** (voir les conditions dans le paragraphe 9): le traitement des données du type **float**. Pour plus de détails, voir **annexe -2-**.

**4.1.3 Fichier à remettre:** La classe **Enregistrement** doit être définie dans le fichier: **enr.h**

**4.2 Représentation graphique:** Pour ce travail pratique, nous allons **simuler** la sortie graphique sous les deux formes suivantes (**pour plus de détails, voir annexe -1-**):

**4.2.1 Histogramme:** Pour chaque élément, nous représentons, sa fréquence relative en pour-cent.

**4.2.2 Tarte (i.e. diagramme sectoriel ou camembert):** Pour chaque élément, nous représentons, sa fréquence relative en degrés.

**4.2.3 Représentation en C++:** Vous devez écrire ce qui suit:

-a- une classe de base abstraite **Graphe** ayant 3 fonctions membres:

- 1) constructeur: **Graphe()** ;
- 2) destructeur: **~Graphe()** ;
- 3) une fonction virtuelle pure, permettant de représenter les statistiques sous la forme de Histogramme ou Tarte: **Dessine()** ;

- b- une classe dérivée **Histogramme**, qui hérite avec le mode `public`, de la classe `Graphe`.
  - 1) constructeur: `Histogramme()`;
  - 2) destructeur: `~ Histogramme()`;
  - 3) une fonction permettant de représenter les statistiques mémorisées dans la classe `Enregistrement<T>`, sous la forme de `Histogramme`:  
`Dessine(const Enregsitrement<T>&);`
- c- une classe dérivée **Tarte**, qui hérite avec le mode `public`, de la classe `Graphe`.
  - 1) constructeur: `Tarte()`;
  - 2) destructeur: `~ Tarte()`;
  - 3) une fonction permettant de représenter les statistiques mémorisées dans la classe `Enregistrement<T>`, sous la forme de `Tarte`:  
`Dessine(const Enregsitrement<T>&);`

Les fonctions `Dessine` des classes `Histogramme` et `Tarte` diffèrent par leur mode de calcul des fréquences relatives (l'un est en pour-cent, l'autre est en degrés).

**4.2.3 Fichier à remettre:** Les classes `Graphe`, `Histogramme` et `Tarte` doivent être définies dans le fichier: **graphe.h**

**5. Outils fournis:** Nous mettons à votre disposition 5 fichiers, ces fichiers sont disponibles sur la page web du tp3 @:

<http://www.iro.umontreal.ca/~dift1166> sinon `/u/dift1166/Sujet/H2K/tp3`

Ces fichiers contiennent ce qui suit:

**5.1 fichier programme:** `prgtest.cpp` contient un programme de simulation du fonctionnement des classes `Enregistrement`, `Graphe`, `Histogramme` et `Tarte`. Il vous permettra de tester vos programmes.

**5.2 fichiers de sortie:** (pour les options de compilation, voir le paragraphe 7)

`sortie.all` est le résultat obtenu sans option de compilation.  
`sortie.enr` est le résultat obtenu avec l'option de compilation **Enr**.  
`bonus.enr` est le résultat obtenu avec l'option de compilation **Bonus** et **Enr**.  
`bonus.all` est le résultat obtenu avec l'option de compilation **Bonus**.

Votre programme devra fournir **les mêmes résultats** que ceux enregistrés dans les fichiers de sortie.

**6. Programmes à remettre:** Les patrons de classes et de fonctions doivent être déclarés et définis dans un fichier d'en-tête. Vous devez écrire la description complète des classes `Enregistrement`, `Graphe`, `Histogramme` et `Tarte`, en tenant compte de l'organisation suivante:

| fichier en-tête       | Classe(s) défini(e)s                                                 |
|-----------------------|----------------------------------------------------------------------|
| <code>enr.h</code>    | <code>Enregistrement</code>                                          |
| <code>graphe.h</code> | <code>Graphe</code> , <code>Histogramme</code> et <code>Tarte</code> |

Vous devez remettre les 2 fichiers: `enr.h` et `graphe.h`, voir le paragraphe 8 pour la procédure de remise.

**7. Compilation:** Dans ce qui suit, Tdb est le nom du programme exécutable (vous pouvez choisir un autre nom).

**7.1. Compilation sans bonus**

-a- test du fonctionnement de la classe Enregistrement

```
g++ -Wall -DEnr -o Tdb testprg.cpp
```

-b- test du fonctionnement de toutes les classes avec tous les types (int, char, string).

```
g++ -Wall -o Tdb testprg.cpp
```

**7.2. Compilation avec bonus**

-a- test du fonctionnement de la classe Enregistrement, avec le type float

```
g++ -Wall -DBonus -DEnr -o Tdb testprg.cpp
```

-b- test du fonctionnement de toutes les classes avec tous les types (int, char, strong, float).

```
g++ -Wall -DBonus -o Tdb testprg.cpp
```

**8. Remise:** Vous devez faire deux types de remise :

- Papier (à glisser sous la porte de mon bureau, Pavillon André-Aisenstadt, local 2249)

**bien identifier sur le listing le cours (IFT1166) et vos noms (inclure la page de garde du tp3).**

- Électronique : dans le répertoire où sont vos programmes, tapez:

```
remise ift1166 tp3 enr.h graphe.h
```

**9. Barème:** Ce tp3 est noté sur 15 points (+2 points de bonus, au total 17 points). Les points seront répartis comme suit :

Exactitude : 7 points

un programme avec des avertissements "warnings" mais qui fait des choses non prévues dans la spécification 0/7,

un programme qui ne compile pas 0/7,

un programme avec des avertissements ("warnings") et qui donne les résultats escomptés 5/7.

Modularité : 6 points ; la manière avec laquelle vous allez coder votre programme!

Style : 2 points ; le code doit être lisible, bien indenté et commenté.

Bonus : 2 points pour le traitement du type float. **Le bonus sera pris en compte que si la date limite de remise aura été respectée.**

**10. Des questions à propos de ce TP?**

l'adresse email: [dift1166@iro.umontreal.ca](mailto:dift1166@iro.umontreal.ca)

pour faciliter le traitement de votre requête, inclure dans le sujet de votre email, au moins la chaîne: [IFT1166]

Les questions posées (ainsi que les réponses) durant la durée de ce tp seront archivées sur la page web du tp3. Vous pouvez les consulter on-line pour voir si votre question y figure déjà.

**11. Mise à jour**

20-03-2K paragraphe 4.2.3.a classe abstraite au lieu de "virtuelle pure" (sinon pas de sens!)

18-03-2K diffusion

**bon courage!**

## Annexe -1-

### Exemple d'un calcul de fréquence

Si le tableau `data` est un ensemble de données du type `int`:

```
data[8] = {1, 2, 1, 3, 4, 2, 6, 2}
```

Nous obtenons les résultats suivants:

| éléments distincts du tableau data | fréquence (F) | Histogramme (H en %) | Tarte (T en degrés) |
|------------------------------------|---------------|----------------------|---------------------|
| 1                                  | 2             | 25                   | 90                  |
| 2                                  | 3             | 37.5                 | 135                 |
| 3                                  | 1             | 12.5                 | 45                  |
| 4                                  | 1             | 12.5                 | 45                  |
| 6                                  | 1             | 12.5                 | 45                  |

où:

\* **fréquence (F)**: indique le nombre de fois qu'un élément apparaît dans `data`. Par exemple, dans `data`, le "1" apparaît **deux** fois.

\* **fréquence totale (FT)**: est la somme des fréquences de chaque élément distinct de `data` (i.e. la somme des FQ). Par exemple,  $FT = 2+3+1+1+1+1 = 8$

\* **représentation en Histogramme (H)**: une règle de trois nous permet de calculer pour chaque élément, la fréquence relative, en pour-cent (H), par rapport à la fréquence totale. Par exemple, pour l'élément "1", cette valeur est calculée comme suit:

$$F = 2, FT = 8 \Rightarrow H = 2 \cdot 100 / 8 = 25\%$$

\* **représentation en Tarte (T)**: une règle de trois nous permet de calculer pour chaque élément, la fréquence relative, en degrés (T), par rapport à la fréquence totale. Par exemple, pour l'élément "1", cette valeur est calculée comme suit:

$$F = 2, FT = 8 \Rightarrow H = 2 \cdot 360 / 8 = 90^\circ$$

## Annexe -2-

### les données du type float

**Égalité entre deux nombres réelles**: Pour ce travail pratique, deux nombres réels seront considérés identiques s'ils sont les mêmes à deux chiffres prêt après la virgule, i.e. vous tronquez (donc pas d'approximation) après la 3<sup>e</sup> décimale. Par exemple, 2.34 et 2.349 seront considérés comme étant les mêmes, alors que 2.34 et 2.38 seront considérés comme différents.

**L'élément distinct**: Pour deux nombres réelles "identiques", l'élément distinct sera celui qui est rencontré en premier lors du calcul de fréquence. Par exemple, à deux décimales prêt, 2.34 est identique à 2.349. L'élément distinct dans ce cas, est 2.34, car il était "lu" en premier.

`setprecision(n)` pour fixer le nombre de décimales après la virgule, par défaut `n=6`.

`setiosflags (ios::fixed | ios::showpoint)` pour un affichage en virgule fixe, avec la virgule (point) incluse.

`cout.unsetf( ios::fixed | ios::showpoint )` pour réinitialiser la valeur des drapeaux (`ios::fixed` et `ios::showpoint`).