Projet : données libres

1. Prélude

Ce projet est volontairement long afin que chacun puisse s'exprimer en fonction de ses compétences, de son éventuelle expérience préalable et de ses goûts. La progression dans le projet est pensée en terme d'exercices à difficulté croissante.

Les exercices 1 à 5 sont accessibles à tous et vous permettent d'obtenir 12. Les exercices suivants demandent plus d'investissement personnel, à vous de voir jusqu'où vous voulez aller : **visez le maximum !** D'autre part, l'expérience que vous accumulerez au fur des premières questions vous fera paraître les questions ultérieures plus simples. Et vous aurez un vrai sentiment d'accomplissement en progressant dans le sujet.

Accrochez-vous, et demandez de l'aide aux autres étudiants (sans copier leur code bien évidemment, cela n'aurait aucun intérêt) ainsi qu'à vos chargés de TD.

Table des matières

2. Introduction

De plus en plus d'organisations, gouvernementales ou autres, mettent leurs données à la disposition du public, afin que celui-ci puisse se les approprier, en trouver des utilisations nouvelles et créatives, et aussi contrôler le fonctionnement de ces organisations sur la base de faits concrets.

Par exemple tout un chacun peut maintenant, grâce aux données mises en ligne récemment par la RATP, implanter une application adaptée à ses besoins spécifiques ; par exemple qui déclenche le réveil matin 25 minutes avant l'arrivée du RER, que celui-ci soit à l'heure ou pas ; ou d'un RER précédent en cas de gros retard annoncé sur la ligne. D'autres données incluent le cadastre ou le processus de rédaction des lois à l'Assemblée nationale, etc. Le citoyen peut ainsi consulter les amendements aux projets de lois auxquels son député a participé, et voir si ceux-ci sont conformes aux engagements de campagne de ce dernier. Voir par exemple les actions de l'association Regards Citoyens ; et en particulier cet article.

L'analyse de ce type de données a par exemple permis en janvier 2016 à des journalistes de mettre au jour le fait que des matchs de tennis de l'ATP avaient été truqués. Les données ainsi que leurs analyses sont disponibles sur `http://github.com/BuzzFeedNews/everything`. On y trouvera d'autres analyses de données tels que les tremblements de terre reliés à l'exploitation des gaz de schiste aux États-Unis, les mouvements des donateurs de la campagne présidentielle américaine lorsqu'un candidat sort de la course, ou une analyse du placement des enfants dans les crèches.

Comme les données peuvent être de grande taille, en extraire des informations intéressantes requiert la plupart du temps des traitements automatisés. Ce projet est l'occasion de

constater que vous avez dès à présent les moyens de commencer à traiter automatiquement ce type de données ; ou d'autres ! Par exemple des résultats d'expériences de physique.

À noter : en d'autres occasions, la bonne approche serait d'utiliser des bibliothèques existantes d'analyse de données et d'apprentissage automatique ; on peut par exemple penser à pandas et scikit-learn (pour python). Pour des traitements simples sur de petites données, un tableur suffirait.

Ici nous n'utiliserons que la bibliothèque standard de C++ (et éventuellement la SDL / libMLV pour les graphiques), quitte à réimplanter des outils de base ; en effet l'objectif est aussi de mettre en pratique ce que vous avez appris dans le cours Info 111 et de mieux comprendre comment fonctionnent ces outils.

Exercice 1 (Vive les mariés !)**.**
Dans ce premier exercice, nous ferons des statistiques simples sur un jeu de données contenant les mariages par jour de la semaine à Paris, entre 2010 et 2015.

`https://opendata.paris.fr/explore/dataset/statistiques-des-jours-des-mariages`

Pour simplifier, nous vous fournissons dans l'archive du projet un fichier `donnees/statistiques-des-jours-des-mariages.txt` contenant ces données dans un format texte simple : chaque ligne contient une année, un jour de la semaine (par exemple "Lundi") et le nombre de mariages célébrés.

(1) Téléchargez les fichiers du projet, typiquement avec la commande `info-111` :

```
info-111 fetch Projet-DonneesLibres
```

(2) Dans le dossier `donnees`, ouvrez le fichier `statistiques-des-jours-des-mariages.txt`. **Ne le modifiez pas : c'est le fichier que votre programme va analyser.** Regardez et comprenez les informations décrites par le fichier. Par exemple, la première ligne indique 756 mariages célébrés un jeudi en 2015.

(3) Ouvrez le fichier `mariage-total.cpp` et complétez la fonction `main` qui calcule et affiche le nombre de total de mariages célébrés à Paris entre 2010 et 2015. **La bonne réponse est 55342. Vérifiez que c'est ce que vous obtenez.**

(4) Ouvrez le fichier `mariage-samedi.cpp` et complétez la fonction `main` qui calcule et affiche le nombre de mariages célébrés un samedi à Paris entre 2010 et 2015. **vous devez trouver 31418, vérifiez que c'est bien le cas**.

Exercice 2 (Les prénoms)**.**
Sur le même site, on trouve aussi les statistiques des prénoms donnés aux enfants nés chaque année, entre 2004 et 2019 :

`https://opendata.paris.fr/explore/dataset/liste_des_prenoms/`

Complétez le programme `prenoms.cpp` qui demande une année à l'utilisateur puis qui utilise le fichier simplifié fourni dans l'archive `donnees/liste_des_prenoms.txt` pour calculer l'ensemble des naissances de l'année ainsi que le prénom le plus donné. Vous devez obtenir l'affichage suivant :

```
Entrez une année entre 2004 et 2019 : 2015
En 2015, il y a eu 31970 naissances
Le prénom le plus donné a été : Adam (donné 356 fois)
```

Remarque : si votre programme ne fonctionne pas, vous pouvez utiliser le fichier `donnees/donnees-test.txt` pour effectuer des tests sur un plus petit fichier.

Exercice 3.
Cet exercice n'est pas nécessaire pour la suite, si vous êtes bloqués, passez à l'exercice d'après !

On souhaite effectuer **dans un même programme** différents calculs sur le nombre de mariages. Pour cela, on va traiter les données à l'aide de tableaux.

(1) Dans le fichier `mariage-complet.cpp`, compléter la fonction `litTableauAnnee`. La fonction doit lire le fichier **une seule fois** et renvoyer un tableau de 6 cases (une pour chaque année de 2010 à 2015) qui contiendra dans chaque case le nombre total de mariages pour l'année (c'est-à-dire la somme de toutes les lignes du fichier qui concernent cette année). L'indice d'une année est : `annee - 2010` (0 correspond à 2010, 1 à 2011, etc).

Testez votre fonction en appelant `testLitTableauAnnee` dans la fonction `main`.

(2) Complétez la fonction `litTableauJours`. La fonction doit lire le fichier **une seule fois** et renvoyer un tableau de 7 cases qui contiendra dans chaque case le nombre de mariages pour le jour donné. Vous pouvez utiliser la fonction `indiceJour` pour obtenir l'indice du tableau correspondant à un jour en chaîne de caractères.

Testez votre fonction en appelant `testLitTableauJours` dans la fonction `main`.

(3) Complétez les fonctions `somme`, `moyenne` et `indiceMax` et lancez les tests correspondants (en plus des tests précédents) dans la fonction `main`.

(4) Enfin, complétez votre programme pour que la fonction `main` appelle l'ensemble des tests puis effectue les calculs suivants en utilisant les fonctions écrites précédemment :
— nombre de mariages au total sur toutes les années
— nombre de mariages en moyenne par an
— l'année où l'on a célébré le plus de mariages (et le nombre de mariages célébrés)
— le jour où l'on a célébré le plus de mariages (et le nombre de mariages célébrés)
— le pourcentage de mariages célébrés le samedi.

L'affichage doit être le suivant

```
Le nombre de total de mariages célébrés entre 2010 et 2015
est de 55342
Le nombre de mariages célébrés en moyenne par an est de 9223
L'année où l'on a célébré le plus de mariages est 2014 avec
9866 mariages
Le jour de la semaine où l'on a célébré le plus de mariages est
le Samedi avec 31418 mariages
Le pourcentage de mariages célébrés le samedi est de 56.7706%
```

Exercice 4 (Vers une bibliothèque générique)**.**
On souhaite à présent pouvoir analyser différents fichiers de données sans réécrire le même code plusieurs fois. Le but est d'obtenir une mini-bibliothèque de fonctions réutilisables.

(1) Commencez par compléter la fonction `afficheTableau` du fichier `prenoms-tableau.cpp` et vérifier en appelant la fonction de test depuis la fonction `main`.

La fonction doit afficher les valeurs du tableau ligne par ligne en les séparant par un espace :

```
M 2011 Bubulle 3
F 2012 Bichette 4
F 2011 Babouche 7
F 2011 Ziboulette 1
```

(2) Dans le fichier `prenoms-tableau.cpp`, complétez la fonction `litTableau` qui transforme un fichier dont on connaît le nombre de colonnes en un tableau de données à deux dimensions (de chaînes de caractères). Testez votre fonction avec la fonction de test proposée.

Aide 1 : on ne connaît pas à l'avance le nombre de lignes du tableau. Stratégie : faire une boucle `while(fichier)`, lire la ligne avec une boucle for et l'ajouter au tableau avec `push_back` si la lecture a fonctionné.

Aide 2 : Si les tests ne passent pas, essayez à partir du fichier plus simple `donnees/donnees-test.txt` et affichez le résultat avec `afficheTableau` pour comprendre d'où vient l'erreur.

(3) Complétez la fonction `colonne` dont la documentation est donnée.

(4) Copiez les fonctions `somme` et `indiceMax` que vous avez déjà écrites. Dans la fonction `main`, utiliser les fonctions `litTableau`, `colonne`, `somme` ainsi que la fonction fournie `conversionInt` pour afficher le nombre total de naissances enregistrées (511485).

(5) Complétez la fonction `selectLignes` qui permet de sélectionner et renvoyer des lignes d'un tableau de données, en fonction d'un critère sur une colonne (par exemple, pour sélectionner les lignes contenant un prénom donné). Testez votre fonction avec la fonction de test proposée.

(6) Complétez la fonction `main` pour que, en plus des tests, le programme demande à l'utilisateur d'entrer un prénom et affiche :
— le nombre total de garçons et filles à qui on a donné ce prénom
— l'année où le prénom a été le plus donné, et le nombre d'occurences

Remarque : si un prénom est donné à la fois à des filles et des garçons, cela correspondra à des lignes différentes dans le tableau, par exemple `"M 2005 Camille 29"` et `"F 2005 Camille 238"`.

Voilà des exemples d'exécution du programme pour comparer avec vos résultats :

```
Nombre total de naissances : 511485
Choisissez un prénom : Octave
Le prénom Octave a été donné à 680 garçons entre 2004 et 2019
L'année la plus forte est 2019 avec 79 enfants
Le prénom Octave n'a été donné à aucune fille entre 2004 et 2019
```

```
Nombre total de naissances : 511485
Choisissez un prénom : Agathe
Le prénom Agathe n'a été donné à aucun garçon entre 2004 et 2019
Le prénom Agathe a été donné à 1301 filles entre 2004 et 2019
L'année la plus forte est 2017 avec 111 enfants
```

```
Nombre total de naissances : 511485
Choisissez un prénom : Camille
Le prénom Camille a été donné à 1018 garçons entre 2004 et 2019
L'année la plus forte est 2019 avec 97 enfants
Le prénom Camille a été donné à 3731 filles entre 2004 et 2019
L'année la plus forte est 2006 avec 287 enfants
```

Exercice 5.
Pour éviter la duplication de code, nous allons maintenant regrouper dans différents fichiers les fonctions que nous avons écrit précédemment, afin de créer une bibliothèque et utiliser la compilation séparée.

(1) Compléter les fichiers `tableau-lecture.cpp` et `tableau-donnees.cpp` (avec les fonctions déjà écrites à l'exercice précédent). La documentation se trouve dans les fichiers `tableau-lecture.h` et `tableau-donnees.h`.

(2) Compléter les tests dans `tableau-donnees-test.cpp`.

(3) Pour compiler un programme dépendant de plusieurs fichiers, on doit compiler chaque fichier séparément. Pour automatiser les compilations, on utilise un fichier `Makefile` qui permet de sauvegarder les dépendances de fichiers et les lignes de compilations associées. Lancez la commande `make tableau-donnees-test` dans le terminal puis exécutez le fichier `tableau-donnees-test` créé pour tester vos fonctions. De même avec `tableau-lecture-test`.

(4) Complétez le fichier `prenoms-tableau-2.cpp` en y recopiant la fonction main que vous aviez déjà écrite dans `prenoms-tableau.cpp` (sans les tests). Compilez le fichier avec la commande `make prenoms-tableau-2` puis exécutez-le. Vous devez avoir le même résultat que celui que vous aviez obtenu avec `prenoms-tableau`

(5) Complétez les fonctions `creeTableauAnnee` et `creeTableauJours` du fichier `mariage-complet-2.cpp`. Ces fonctions sont similaires à `litTableauAnnee` et `litTableauxJours` mais prennent cette fois **un tableau de données** en entrée et non plus un fichier.

On utilisera les fonctions `selectLignes`, `colonne` et `somme` pour faire les calculs : pour chaque année du tableau `annees` (ou chaque jour), on fera un `selectLignes` puis une somme de colonnes sur le tableau `data`

Remarque : cette méthode n'est pas très efficace ; dans les exercices suivants, on écrira des meilleures fonctions.

!! Important !! L'exercice 6 est indépendant des exercices 7 et 8.
— **Si vous voulez faire des traitements de tableau plus avancés, commencez par l'exercice 6.**
— **Si vous voulez plutôt travailler de façon plus poussée sur les fichiers, commencez par les exercices 7 et 8.**

Exercice 6 (Des requêtes plus difficiles)**.**
On cherche à effectuer des requêtes (c'est à dire des interrogations et sélections de données) plus complexes sur les tableaux de données.

(1) Complétez les fonctions `chercheIndice` et `distinct` du fichier `tableau-donnees-avance.cpp`. La documentation se trouve dans `tableau-donnees-avance.h` et les tests dans `tableau-donnees-avance-test.cpp`. Vous pouvez compiler le fichier avec la commande `make tableau-donnees-avance-test`.

Indication : la fonction `distinct` permet de récupérer une colonne du tableau de données en supprimant les doublons : par exemple, on peut obtenir les années qui apparaissent dans le fichier, ou les prénoms différents, etc. Pour cela, on lira le tableau ligne par ligne, et on utilisera la fonction `chercheIndice` pour savoir si l'élément a déjà été ajouté au tableau à renvoyer.

(2) Complétez la fonction `conversionDouble` en vous inspirant du fonctionnement de `conversionInt` donné dans `tableau-donnees.cpp`, avec utilisation des string stream.

(3) On cherche à écrire une fonction classique des requêtes sur base de données : la fonction `groupBy`. L'idée est de calculer des sommes récapitulatives à partir d'une colonne, en regroupant selon des données qui se trouvent dans une autre colonne. Par exemple, sur le tableau suivant :

M	2011	Bubulle	3
F	2012	Bichette	4
F	2011	Babouche	7
F	2011	Ziboulette	1

Si nous regroupons selon les années de la 2ème colonne, en sommant les valeurs de la dernière colonne, nous obtenons :

2011	11
2012	4

Complétez la fonction `groupByInt` de `tableau-donnees-avance.cpp`. En plus du tableau de données, cette fonction prend en paramètre :
- les valeurs distinctes à prendre en compte pour le `groupBy` (par exemple `{ "2011", "2012" }` pour la somme regroupée ci dessus)
- le numéro de la colonne d'où sont tirées les valeurs selon lesquelles doit se faire le regroupement (dans l'exemple ci dessus, le regroupement se fait selon les années qui sont dans la 2ème colonne, c'est à dire la colonne d'indice 1)
- le numéro de colonne des valeurs à sommer (dans l'exemple ci dessus, il s'agit de la 4ème colonne, ayant pour indice 3).

La fonction renvoie un tableau d'entier qui correspond aux sommes obtenus (dans l'exemple ci dessus on obtient `{ 11, 4 }`). Vous aurez besoin d'appeler la fonction `chercheIndice`.

Testez votre fonction à l'aide de `tableau-donnees-avance-test.cpp`. Ensuite, compilez avec `make mariage-complet-3` le fichier `mariage-complet-3.cpp` (après l'avoir complété avec la fonction `main` de `mariage-complet-2.cpp`) et vérifiez en l'exécutant que vous obtenez les mêmes résultats que dans `make mariage-complet-2`.

(4) En vous inspirant de l'utilisation de `distinct` et `groupBy` dans `mariage-complet-3.cpp`, répondez aux questions ci-dessous en complétant la fonction `main` de `prenoms-tableau-avance.cpp` :
- Combien y-a-t il eu au total de naissances de garçons ? De filles ?
- Quelle année a-t-on eu le plus de naissances (garçons et filles confondus) ?
- Combien a-t-on de naissances en moyennes par an ?
- Combien y-a-t il de prénoms féminins différents ? Et masculins ?
- Quel est le prénom féminin le plus populaire ? masculin ?

Pour toutes ces questions, il faudra utiliser une combinaison d'appels à `distinct` et / ou `groupByInt` avec parfois en plus une sélection `selectLignes` à effectuer. Voici les réponses que vous devez obtenir :

```
Il y a eu 263757 naissances de garçons et 247728 naissances de filles
L'année qui a eu le plus de naissances est : 2010 avec 33259 naissances
En moyenne, naissent 31967 enfants par an
Il y a eu 1360 prénoms de filles différents et 1268 prénoms de garçons différents
Le prénom féminin le plus populaire est Louise avec 4350 naissances
Le prénom masculin le plus populaire est Gabriel avec 5460 naissances
```

(5) Complétez la fonction `groupByDouble` et lancez les tests correspondants. La fonction est la même que `groupByInt` mais on doit maintenant fabriquer un tableau de `double`

(6) ♣ Version avancée : éviter la duplication pour les fonctions `conversion` et `groupBy` en utilisant un template.

Exercice 7 (Un premier fichier CSV)**.**
L'objectif est de reprendre l'exercice 2, mais en partant du fichier de données tel qu'il est fourni par `opendata.paris.fr` : `donnees/liste_des_prenoms.csv`

Ce fichier est en format CSV (Comma Separated Values). C'est un format texte répandu pour représenter des données tabulaires. D'ailleurs, vous pouvez charger ce type de fichier directement dans un tableur, par exemple pour vérifier à la main que vos résultats sont corrects.

Chaque ligne de texte représente une ligne du tableau, et les cellules sont séparées par des point-virgules. La première ligne d'un fichier CSV est une ligne d'entête qui correspond au nom des colonnes du tableau.

(1) Consulter et essayer le programme `getline-exemple` fourni dans l'archive. Il utilise la fonction `getline`, dont la documentation est donnée ci-dessous.

```
/** @file **/
/** Lit une ligne d'un flux et la stocke dans 'resultat'
 * @param f un flux entrant
 * @param resultat une chaîne de caractères
 * @return le flux f
 **/
istream& getline(istream &f, string &resultat);

/** Lit une chaîne de caractères depuis un flux jusqu'au séparateur
    et la stocke dans 'resultat'
 * @param f un flux entrant
 * @param resultat une chaîne de caractères
 * @param separateur un caractère
 * @return le flux f
 **/
istream& getline(istream &f, string &resultat, char separateur);
```

Vous constaterez que cette fonction *modifie* la chaîne de caractère 'resultat' passée en argument ! De fait, le symbole '&' indique que 'resultat' est passé *par référence*. Vous verrez les détails au second semestre.

Attention ! Vous remarquez que le fichier csv contient une colonne supplémentaire au début de chaque ligne avec un nombre qui est, presque tout le temps, le même qu'à la fin de la ligne (mais pas toujours !). Cette colonne n'est pas expliquée sur le site : dans la vraie vie, les fichiers de données ont souvent des défauts de ce genre. Nous considérons que le nombre correct est celui en fin de ligne.

(2) À l'aide de la fonction `getline`, implanter le programme `prenoms-csv.cpp` et qui fait le même calcul que `prenoms.cpp` mais à partir du fichier CSV.

Remarque : pour vos tests et le debuggage, vous pouvez utiliser le fichier `donnees/donnees-test.csv` qui utilise le même format mais avec moins de données.

Exercice 8 (Généralisation)**.**
L'objectif de cet exercice est d'ajouter à notre bibliothèque des fonctionnalités génériques pour les fichiers CSV. On implantera les fonctions dans `tableau-lecture-csv.cpp` . Des tests sont fournis dans `tableau-lecture-csv-test.cpp`

(1) Complétez la fonction suivante dans le fichier `tableau-lecture-csv.cpp` et lancez les tests correspondant dans `tableau-lecture-csv-test.cpp`

```
/** Construction d'un tableau 2D de chaînes lu depuis un fichier CSV
 * @param fichier le nom d'un fichier contenant un nombre fixe
  * d'entiers par ligne, séparés par des espaces
 * @param nb_colonnes le nombre de colonnes du fichier
 * @return un tableau d'entiers à deux dimensions
 **/
vector<vector<string>> litTableauCSV(string fichier, int nb_colonnes);
```

Si vous rencontrez des difficultés, n'hésitez pas à utiliser les données de test `donnees/donnees-test.csv` ainsi que la fonction `afficheTableau`.

Si malgré ça, vous n'y arrivez pas, vous pouvez copier-coller l'implantation fournie dans le fichier `en_cas_d_urgence_briser_la_glace.cpp`. On vous demandera tout de même de savoir expliquer comment elle marche !

(2) ♣ Même chose, mais sans avoir besoin de spécifier le nombre de colonnes

```
/** Construction d'un tableau 2D de chaînes lu depuis un fichier CSV
 * @param fichier le nom d'un fichier contenant un nombre fixe
  * d'entiers par ligne, séparés par des espaces
 * @return un tableau d'entiers à deux dimensions
 **/
vector<vector<string>> litTableauCSV(string fichier);
```

Exercice 9 (Applications)**.**

(1) (`arbres-hauteur.cpp`) Quel est le genre et l'espèce de l'arbre le plus haut de Paris ? `https://opendata.paris.fr/explore/dataset/arbresremarquablesparis/` (Note, on vous fournit le fichier des arbres remarquables, mais vous pouvez aussi tester vos programmes sur l'ensemble des arbres de Paris, soit plus de 200 000 entrées : `https://opendata.paris.fr/explore/dataset/les-arbres/`)

(2) (`arbres-platanus.cpp`) Combien a-t-on d'arbres de Genre "Platanus" et combien d'espèces différentes pour les Platanus ?

(3) (`actes-civil.cpp`) En quelle année a-t-on eu le plus de déclarations de naissances ? de mariages ? `https://opendata.paris.fr/explore/dataset/statistiques_de_creation_d_actes_d_etat_civil_par_arrondissement/`

Exercice 10.

(1) En utilisant la bibliothèque graphique du TP 8, dessiner la carte de toutes les communes de France à partir du fichier CSV fourni sur :
`https://www.data.gouv.fr/fr/datasets/decoupage-administratif-communal-francais-issu-`

On pourra par exemple représenter chaque commune par un cercle centré sur ses coordonnées géographiques et de rayon proportionnel à la population.

(2) Positionner sur cette carte d'autres éléments intéressants.

(3) Positionner sur cette carte plusieurs barycentres de la France (population / surface / ...)

3. Pour aller plus loin

Choisir une ou plusieurs de ces extensions :

Exercice ♣ 11.

(1) Quelle est la voiture commercialisée en France qui consomme le plus ? Le moins ?
`https://www.data.gouv.fr/fr/datasets/emissions-de-co2-et-de-polluants-des-vehi`

(2) Quelle marque, en moyenne sur toute la gamme commercialisée en France, produit les véhicules qui consomment le plus ? le moins ?

Exercice ♣ 12.
Lister les stations de métros à Paris où ont été retrouvés des lecteurs MP3.

Exercice ♣ 13.
Implanter l'application mentionnée dans l'introduction (RéveilRER) sous forme d'une application Android.

Exercice ♣ 14.
À vous de trouver des applications intéressantes des données ouvertes !

4. Références

— `http://opendata.paris.fr`
— `http://data.gouv.fr`
— `http://www.ideeslibres.org/`