Exercices IV

Manipulation de structure et fonction membre

1. Lecteur de fichier CSV

Les fichiers CSV (Comma Separated Value) représentent des tableaux dont la séparation entre chaque cellule est matérialisée par une virgule (“comma”) ou un point-virgule (“semicolon”) ou une tabulation ou quatre-espaces, etc ...

Voici un exemple de contenu de fichier CSV:

--- début du fichier sondage.csv ---
```
Prénoms,Numéros de téléphone,Réponses,Noms
Bastien,0612345678,4/18,Righi
Brandon,0656780612,12/18,Righi
Benjamin,0690127806,17/20,Jasse
Nicolas,0494568642,8/9,Righi
Dorian,0678903456,12/19,Jasse
Caroline,0494568642,8/12,Berlioz
Anthony,0676544321,11/19,Berlioz
Florent,0434560690,18/19,Colombier
Jean-Marc,0643219401,13/18,Colombier
Jeremy,0610901289,15/20,Jasse
Rémi,0680737812,20/20,Berlioz
Xavier,0683537812,17/20,Jasse
Barbara,0682631287,7/9,Righi
Fabien,0663736826,11/13,Berlioz
Cedric,0667836312,11/13,Colombier
Sébastien,0611896312,15/19,Righi
Valentin,0688634356,7/9,Jasse
Martinien,0649531265,8/9,Berlioz
Alexander,0680639531,17/20,Berlioz
Thomas,0680138353,14/15,Colombier
```
--- fin du fichier sondage.csv ---

1.a. Créez un répertoire de travail pour ce projet.
Dans ce répertoire, créez le fichier sondage.csv avec le contenu susmentionné.

On souhaite créer une `struct CSVParser` qui nous permettra de lire et d'exploiter plus facilement le contenu du fichier CSV.
Créez les fichiers CSVParser.hpp et CSVParser.cpp suivants:

```
/*
 * CSVParser.hpp
 *
 *  Note: Lecteur de fichier CSV
 */

struct CSVParser {


};
```

```
/*
 * CSVParser.cpp
 *
 */
#include "CSVParser.hpp"
```

Pour l'instant, les fichiers sont basiques. Créez le fichier CSVParserTest.cpp qui servira à effectuer les tests unitaires de la `struct CSVParser` avec le framework Catch (cf. dernière diapo du cours). Mettez-y un `TEST_CASE` vide pour l'instant. (On prépare ainsi tout l'environnement de travail.)
Comme à l'exercice 3, créez un fichier Makefile qui permet de compiler et générer un exécutable **test-csvparser** à partir des 3 fichiers C++. L'exécutable ne fait pas grand chose, c'est juste pour s'assurer que l'environnement de travail est prêt !

Si la compilation et l'exécution de test-csvparser se lancent avec succès, bravo! Votre environnement de travail est prêt.

1.b. On souhaite connaître de notre fichier CSV :

- le nombre de colonnes du tableau : `numberOfColumns`
- le nombre de lignes en dehors de l'en-tête du tableau (l'en-tête étant la 1ère ligne du tableau : `index == 0`) : `numberOfRows`
- l'en-tête du tableau : `header` (contenant un tableau de chaînes de caractères)

Quels sont les attributs membres à mettre dans notre structure CSVParser ?

Note: l'en-tête du tableau peut être un tableau de `string`.

Pour définir un tableau fixe de string

`std::string aLine[4];` // tableau de 4 strings
// (de même que l'on fait `char table[4]` quand on veut un tableau de 4 caractères)
`std::string aTmp = "Hello";`
`aLine[0] = aTmp;` // assigne le contenu de `aTmp` au 1er élément du tableau
etc...

Déclarez au sein de la `struct CSVParser` les attributs et une fonction `initWithFile(string path)` qui retourne un booléen (`true` si tout va bien et `false` en cas d'erreur de lecture du fichier et affiche un message sur la sortie standard d'erreur) :

```
/*
 * CSVParser.hpp
 *
 *  Note: Lecteur de fichier CSV
 */
#include <string>

struct CSVParser {
  string filepath;
  /*
   * TODO: déclarez les autres attributs ici
   */

  //@{
  // fonction membre: Initialise la structure à partir d'un fichier
  // Param path: chemin/vers/le/fichier.csv
  // Return:
  //    true : OK
  //    false : Erreur lecture fichier
```

```
  //@}
  bool initWithFile(string path);

};
```

Maintenant que vous avez déclaré la structure `CSVParser`, implémentez dans le fichier CSVParser.cpp (fichier d'implémentation) le contenu de la fonction membre `initWithFile`:

```
/*
 * CSVParser.cpp
 *
 */
#include "CSVParser.hpp"

bool CSVParser::initWithFile(string chemin_fichier) {
  // affectation d'une valeur à un attribut de la struct
  this->filepath = chemin_fichier;

  /*
   * TODO: implémentez la fonction ici
   *   retourne true si tout va bien ou false en cas d'erreur de lecture fichier
   */

  return true;
}
```

Note:

- Une fonction membre étant déclarée dans la `struct`, elle fait donc partie de l'espace de nommage de cette `struct`. D'où l'utilisation du préfixe `CSVParser::` pour implémenter le contenu de cette fonction membre dans le fichier CSVParser.cpp
- Le mot clé `this` désigne le pointeur vers l'objet courant (du type de la `struct`). Comme tout pointeur, on utilise l'opérateur `->` pour accéder aux attributs comme aux fonctions membres (publics).

Complétez la fonction `initWithFile` pour affecter la valeur calculée pour tous les attributs de la structure en fonction du contenu du fichier. (**Attention**: Modifiez le contenu du fichier mais

pas la signature de la fonction!) Pour valider que la structure `CSVParser` fait bien ce qu'on attend d'elle, implémentez les tests unitaires Catch correspondants.

2. Modèle de données

Dans le fichier CSV précédent, on se rend compte que le contenu correspond à des informations sur des personnes. Préparons donc ce qu'on appelle un "Business Object-Model" (BOM) ou modèles de données business.

Il s'agit de créer une structure `Person` qui peut contenir les informations de chaque ligne du tableau.

2.a. Quels sont les attributs de la structure `Person` ?
Créez dans le même dossier les fichiers Person.hpp et Person.cpp correspondant aux fichiers d'en-tête et d'implémentation de la structure `Person`.
Créez le fichier de test unitaire correspondant: PersonTest.cpp avec un `TEST_CASE` vide.

Attention: la fonction `main` d'un programme C++ est unique. Veillez à ce qu'elle soit "déclarée" dans un seul fichier test !

Vous avez maintenant 6 fichiers C++. Assurez-vous que le Makefile permet de compiler aussi les fichiers objets Person.o et PersonTest.o pour les ajouter à la construction de l'exécutable **test-csvparser**.

Si la compilation et l'exécution de test-csvparser se lancent avec succès, bravo! Votre environnement de travail est encore bon.

2.b. On souhaite vérifier que l'attribut membre contenant le numéro de téléphone contient bien une chaîne de 10 caractères numériques, commençant par un '0'.
Créez dans la `struct Person` la fonction membre `hasValidPhoneNumber()` qui retourne `true` si l'attribut membre pour le numéro de téléphone respecte bien les critères susmentionnés, ou `false` sinon.

Note: un exemple d'utilisation de la fonction membre d'un objet Person

```
Person toto;
if (toto.hasValidPhoneNumber()){
  ...
}
```

Pour valider que la fonction membre `hasValidPhoneNumber` fait bien ce qu'on attend d'elle, complétez le `TEST_CASE` Catch dans le fichier PersonTest.cpp (cf. Illustration plus bas)

```cpp
/*
 * PersonTest.cpp
 */
#include "Person.hpp"
#include "catch.hpp"

TEST_CASE( "Valid Phone Number", "[Person]" ) {
  Person toto;
  /*
   * TODO: affectez des valeurs à l'objet de type Person
   * (ou déclarez en affectant les valeurs directement)
   */
  REQUIRE( toto.hasValidPhoneNumber() == true );
  /*
   * TODO: affectez une autre valeur à un attribut de l'objet
   */
  REQUIRE( toto.hasValidPhoneNumber() == false );
}
```

Note: Pas besoin de lire le fichier CSV ici ! La structure `Person` n'a aucune connaissance de la notion de fichier. C'est juste une structure basique pour stocker des informations sur les personnes.

2.c. On souhaite vérifier que les attributs membres contenant les prénom et nom contiennent bien des caractères alphabétiques.
Créez la fonction membre `hasValidNames()` qui retourne :

- `true` si les noms et prénoms ne contiennent que des caractères alphabétiques.
  Note: on peut utiliser la fonction isalpha. Dans ce cas, n'hésitez à ajouter toutes les modifications nécessaires au fichier d'implémentation pour que ça compile.
- `false` sinon. (ne retourne pas `false` si seulement un des deux est vide)

Ajoutez le `TEST_CASE` Catch correspondant dans le fichier PersonTest.cpp pour valider que la fonction effectue bien ce qu'on attend d'elle.

3. Fraction

3.a. On remarque qu'on a des fractions au niveau de la colonne Réponses dans le fichier CSV. On souhaite donc mieux le matérialiser dans notre code.
Créez la `struct Fraction` avec les fichiers correspondants : Fraction.hpp, Fraction.cpp, FractionTest.cpp (comme effectué précédemment pour `Person`).
Quels sont les attributs de la structure `Fraction` ?
Note: une fraction est composée d'un numérateur entier et d'un dénominateur entier (n/d).

3.b. Ajoutez une fonction membre de type `string display()` qui permet de retourner une fraction sous le format textuel "3/2".
Ajoutez le `TEST_CASE` Catch correspondant dans le fichier FractionTest.cpp pour valider que la fonction effectue bien ce qu'on attend d'elle.

3.c. On souhaite pouvoir faire des calculs avec les objets de type `Fraction`.
Ajoutez à la `struct Fraction` une fonction membre de type `void operator+(int i)` qui permet d'additionner un entier avec une fraction.
Ajoutez le `TEST_CASE` Catch correspondant dans le fichier FractionTest.cpp pour valider que la fonction effectue bien ce qu'on attend d'elle. (**Attention** au division par zéro!)

3.d. Ajoutez à la `struct Fraction` une fonction membre de type `void operator+(const Fraction a)` qui permet d'additionner une fraction avec une autre fraction.
Ajoutez le `TEST_CASE` Catch correspondant dans le fichier FractionTest.cpp pour valider que la fonction effectue bien ce qu'on attend d'elle.

3.e. Ajoutez à la `struct Fraction` les fonctions membres opérateurs suivantes :

- *bool operator<(const Fraction a)* qui définit l'opérateur inférieur pour les fractions.
- *bool operator>(const Fraction a)* qui définit l'opérateur supérieur pour les fractions.

Ajoutez les `TEST_CASE` Catch correspondants dans le fichier FractionTest.cpp pour valider que ces fonctions effectuent bien ce qu'on attend d'elles.

4. Retour à la structure Person

Il s'agit maintenant qu'on a créé une `struct Fraction` qui fonctionne, de l'utiliser dans la `struct Person`.

4.a. Remplacez le type de l'attribut membre de la `struct Person` qui représente le champ "Réponse" du fichier CSV et corrigez toutes les erreurs de compilation suite à ce changement. Exécutez les tests unitaires et corrigez toutes les erreurs également.

4.b. Le fichier CSV est en fait le résultat d'un sondage, et la colonne "Réponses" correspond au nombre de réponses positives sur le nombre de réponses données.
Ajoutez à la `struct Person` les fonctions membres de type `int getNumberOfPositiveResponses()` and `int getNumberOfGivenResponses()` qui retournent les valeurs susmentionnées.
Ajoutez les `TEST_CASE` Catch correspondants dans le fichier PersonTest.cpp pour valider que les fonctions effectuent bien ce qu'on attend d'elles.

5. Retour à la structure CSVParser

5.a. Ajoutez dans la `struct CSVParser` une fonction membre `getLine(int rowNum)` qui retourne le contenu de la ligne rowNum (rowNum == index + 1) du fichier CSV sous forme de tableau de `string`.
Ajoutez le `TEST_CASE` Catch correspondant dans le fichier CSVParserTest.cpp pour valider que la fonction effectue bien ce qu'on attend d'elle.
Note: Ici dans la structure `CSVParser`, la notion de fichier est connue. N'hésitez pas à créer d'autres fichier.csv inspirés du fichier initial afin de tester différents cas possibles (Cas aux limites: autres `TEST_CASE`)

5.b. Ajoutez une autre fonction membre `getLineWithHighestRateOfResponses()` qui retourne le contenu de la ligne avec la réponse (fraction) la plus élevée, sous forme de tableau de `string`.
Ajoutez le `TEST_CASE` Catch correspondant dans le fichier CSVParserTest.cpp pour valider que la fonction effectue bien ce qu'on attend d'elle.

5.c. Ajoutez une autre fonction membre `getLineWithLastname(string name)` qui retourne un tableau de tableaux de `string` de contenus des lignes avec les personnes de même nom que la valeur du paramètre `name`.
Ajoutez le `TEST_CASE` Catch correspondant dans le fichier CSVParserTest.cpp pour valider que la fonction effectue bien ce qu'on attend d'elle.

5.e. Ajoutez une autre fonction membre `getPersonWithLastname(string name)` qui retourne un tableau de `Person` avec les personnes de même nom que la valeur du paramètre `name`.
Ajoutez le `TEST_CASE` Catch correspondant dans le fichier CSVParserTest.cpp pour valider que la fonction effectue bien ce qu'on attend d'elle.

6. Détection de séparateur

On souhaite maintenant que la `struct CSVParser` reconnaisse automatiquement le séparateur utilisé dans le fichier CSV.

6.a. Déclarez à l'intérieur de la `struct CSVParser`, un type `enum Sep_t` qui a les valeurs :

- `CSV_COMMA = 0`
- `CSV_SEMICOLON`
- `CSV_TABULATION`
- `CSV_4SPACES`

Ajoutez une fonction membre de type `Sep_t getSeparatorType()` qui retourne la valeur correspondante au séparateur détecté dans le fichier CSV.
Ajoutez le `TEST_CASE` Catch correspondant dans le fichier CSVParserTest.cpp pour valider que la fonction effectue bien ce qu'on attend d'elle.
Note : N'hésitez pas à créer d'autres fichier.csv inspirés du fichier initial, un par type de séparateur, afin de tester les différents cas possibles.
Note 2 : Dans Vim, la commande pour remplacer toutes les occurrences d'un caractère par un autre est la suivante (Faites d'abord ":" pour entrer en mode commande):

- `s/,/;/ga` ==> remplace toutes les virgules par des points-virgules
- `s/;/\t/ga` ==> remplace tous les points-virgules par des tabulations
- etc ...

6.b. Dans le cas ou le séparateur est 4-espaces, comment résoudre le cas où le nom ou le prénom d'une personne contient un espace.
Ajoutez le `TEST_CASE` Catch correspondant dans le fichier CSVParserTest.cpp pour valider ce cas de figure.