Master Bio-informatique

UE : Programmation orientée objet

Projet de programmation : Recherche d'informations

Auteurs :
RudyAnne
Jeveta Ghanty
Amir Naar

Encadrants :
Benoit-Pinaud J.
Sandillon-Rezer N.F.

Mai 2014

TABLE DES MATIÈRES

INTRODUCTION

Dans le cadre de l'UE de Programmation Orientée Objet nous avons réalisé un programme permettant de charger une base de donnée minimaliste et de rechercher des informations à l'intérieur. Ce programme a été réalisé en C++, langage qui a été développé par Stroustrup Bjarne et normalisé pour la première fois en 1998. Ce langage est par ailleurs très utlisé de nos jours car il présente de nombreux avantages tels que sa performance, son grand nombre de fonctionnalités, sa portabilité des fichiers sources ou encore la gestion des exceptions.

Afin d'atteindre les objectifs souhaités dans ce projet nous avons établi une division logique dans notre démarche. Ce rapport s'efforcera de restranscrire celle-ci.
Dans un premier temps nous allons décrire brièvement le sujet du projet. Dans un second temps, nous établirons un cahier des charges. Puis nous expliquerons notre code source . Et enfin, nous terminerons par un test de notre programme. Nous tâcherons tout au long de ce projet de bien respecter les consignes et à mettre en exergue les perspectives ou axes d'améliorations de notre programme.

CHAPITRE 1

DESCRIPTION DU PROJET

Ce projet de programmation consiste à réaliser une mini base de donnée permmettant de stocker une liste de médicaments et leurs effets secondaires.

Ces informations sont contenues dans un fichier `liste.txt`. Il s'agit alors, dans un premier temps, de parser ce fichier afin d'en extraire des informations pertinentes.

Dans un second temps, à partir de ces informations que nous aurons stocker, nous devrons réaliser un histogramme représentant le nombre d'apparitions de tous les effets secondaires. Ensuite, nous devrons réalisé plusieurs algorithmes :

- le premier permettant de rechercher les médicaments ayant un effet secondaire donné.
- le suivant permettra de comparer un médicament entré par l'utilisateur à la liste des médicaments partageant un (ou plusieurs) effet(s) secondaire(s).

A partir de notre premier algorithme, notre programme sera en mesure de calculer la précision afin de déterminer l'efficacité de cet algorithme de recherche.

Enfin, nous réaliserons l'option facultative permettant à l'utilisateur d'ajouter un médicament dans la base de données sans avoir besoin de recharger le programme en saisissant une nouvelle entrée pendant le fonctionnement du programme.

CHAPITRE 2

CAHIER DES CHARGES

Notre programme devra respecter l'ensemble des normes décrites ci-dessous.

2.1 Chargement des données

Le chargement des données est une étape primordiale au bon fonctionnement du programme. Nous devons donc prendre en compte sous quel format se présente les données. Le fichier `.txt` contenant les informations présente un format particulier.
Il est composé de vingt cinq lignes contenant chacunes :

- le nom d'un médicaments,
- le nom de ses effets secondaires associés.
- des caractères de ponctuations et de séparation

Cependant, tous les médicaments n'ont pas le même nombre d'effets secondaires. Les lignes du fichier se présentent ainsi :

medicamentn : effet1, effet2, effetn.

De plus, nous pouvons noter, par endroit, que ce fichier peut ne pas respecter certaines normes typographiques comme le nombre d'espaces (cf. ligne 5 où il y a un espace superflu : espace en trop après "dizziness") :

Ketoprofène : vomiting, dizziness , hyperkalemia.

Nous allons devoir gérer cette problématique afin que notre base de donnée ne stocke pas deux fois le même caractère avec un espace en plus ou en moins.

La première étape importante consistera à parser ce fichier qui contient toutes les informations nécessaires à la réalisation de notre programme. C'est-à-dire que nous allons extraire une information structurée du flux de données. Ce processus nous permettra d'enlever les bruits (caractères de séparation) et nous ne stockerons que les informations nécessaires.

Le `main` du programme devra contenir une fonction usage qui sera appelé lorsque l'utilisateur appelera le programme avec des mauvais paramètres. De plus, il pourra également avoir accès à une aide lui donnant

une assistance pour utiliser le programme avec les options `-h` ou `-help`. Ainsi, le fait de rentrer un nom de fichier qui n'existe pas lancera une exception et retournera un messsage d'erreur.

2.2 Fonctionnalités

Notre programme devra disposer des fonctionnalités suivantes.

Réalisation d'un histogramme

La première fonctionnalités dont disposera notre programme sera de réaliser un histogramme visualisable. En effet, il permettra de rechercher les occurences de chaques effets secondaires et les présentera sous le forme d'un histogramme représentant leur fréquence d'apparition.

Recherche d'informations

Plusieurs algorithmes seront réaliser afin de rechercher des informations dans notre base de donnée minimaliste conçue.

Notre premier algorithme de recherche permettra à l'utilisateur de trouver la liste des médicaments contenant un effet secondaire donnée.
La structure de donnée choisie (`map <string (médicaments), vector <string (effets secondaires)> >`) permettra de faciliter la recherche des médicaments ayant un effet secondaire donné. Ainsi, en entrant le nom d'un effet secondaire, l'utilisateur obtient la liste de médicaments ayant cet effet secondaire.

Nous coderons ensuite plusieurs algorithmes afin de comaparer les médicaments entre eux de différentes manières. L'un d'eux, notamment, permettra d'afficher la liste de médicament possédant le plus d'effets secondaires en commun, ou un autre d'afficher les deux premiers médicaments ayant au moins l'effet secondaire recherché. L'utilisateur pourra entrer un nom de médicament, le programme lui retournera la liste des médicaments adéquate.

Calcul de la précision

La précision est la mesure de l'efficacité de certains logiciels. Le rappel qui est couramment utilisé permet aussi de mesurer l'efficacité de certains logiciel.
La précision mesure la proportion des résultats qui sont considérés comme pertinents ou corrects. Par exemple, si un programme cherche des termes dans un document et trouve 100 candidats, dont 65 sont vraiment des termes (c'est-à-dire, 65 sur les 100 résultats corrects), la précision des résultats sera de 0.65 (les 35 autres résultats non-pertinents étant du bruit.)
Nous ramenerons ce résultat afin d'obtenir un pourcentage du calcul de la précision.

Le calcul de la précision dans notre programme s'effectuera grâce au premier algorithme de recherche. Il nous permettra ainsi de déterminer l'efficacité de celui-ci.

Ajout de médicaments dans la base de données

Nous avons décidé d'offir la possibilité à un utilisateur de rajouter des médicaments à la base de données sans pour autant recharger le programme. L'utilisateur pourra saisir une nouvelle entrée pendant le fonctionnement du programme et entrer le médicament qu'il souhaite à la condition qu'il n'existe pas déjà dans la base de données.

CHAPITRE 3

CONCEPTION : EXPLICATION DU CODE

Nous allons dans cette partie détailler comment nous avons développé notre programme. Ce-dernier utilise la programmation orienté objet. Il se découpe en 4 fichiers :

- un header `Pharmacie.hpp` contenant la déclaration de la classe Pharmacie
- `Pharmacie.cpp` contenant la définition et l'implémentation de méthodes dont le constructeur,
- `gestion.cpp` où se trouve la définition et l'implémentation de méthodes qui utilisent le flux d'entée standard : c'est grâce à ces méthodes que l'utilsateur interagira avec le programme,
- `parser.cpp` contenant le main du programme.

3.1 Algorithme de déroulement du programme

Le déroulement du programme est géré dans le fichier `parser.cpp`. En effet, il contient le `main` du programme. C'est ici que l'on instancie une `Pharmacie` en faisant appel au constructeur défini dans le fichier `Pharmacie.cpp`. Nous avons choisi d'établir un menu, dont la méthode se trouve dans le fichier `gestion.cpp`. L'utilisateur peut alors choisir de faire les recherches qui l'intéressent. Une boucle principale affiche ce menu, récupère le choix de l'utilisateur et affiche la fonction correspondante grâce à un branchement conditionnel ainsi :

- une fonction permettant d'afficher l'histogramme
- une fonction permettant de visualiser la map où sont stockés les médicaments
- une autre pour rechercher les médicaments ayant un effet secondaire donné
- une permettant de sélectionner les médicaments ayant le plus d'effets secondaires en commun avec un autre entré par l'utilisateur
- une autre encore afin d'avoir le calcul de la précision
- deux autres fonctions permettant de faire des recherches de médicaments différentes (recherche des deux premiers médicaments ayant des effets en commun, recherche des deux premiers médicaments aynt le(s) effet(s) entrées)

— et une dernière fonction afin d'ajouter un médicament à la base de données
— enfin, un message d'erreur s'affichera si le choix est invalide

3.2 Le header

Le fichier `.hpp` va donc contenir la déclaration de la classe avec les attributs et les prototypes des méthodes. Nous séparerons nos méthodes et attributs en `private` pour celles qui sont internes au programme et qui ne doivent pas être accessible de l'extérieur ; et en `public` pou les méthodes et attributs accessibles.

3.3 Création de l'objet `Pharmacie`

C'est dans le fichier `Pharmacie.cpp` que nous allons implémenter une partie de nos méthodes dont le constructeur Chacunes de ces méthodes devant être précédées par le nom de la classe suivie de l'opérateur de résolution de portée ainsi :

```
Pharmacie::<méthode>
```

Cela permet au compilateur de savoir à quelle classe se rapporte cette méthode. En l'occurence, ici nous n'avons qu'une seule classe mais un des atouts de la programmation orienté objet permettra d'ajouter d'autres classes tout en maintenant la pérennité du programme.

3.3.1 Le constructeur

Il prend en argument un paramètre de type `string` correspondant au nom du fichier à traiter. Il vérifie d'abord que le fichier est bien valide et quand c'est le cas il fait appel à la méthode `parsePharma()` sinon il renvoie une exception indiquant que le fichier n'est pas bon. Ainsi, lorsquu'un ojet de type `Pharmacie` sera créé, le parsing se fera en même temps.

3.3.2 Parsing du fichier `.txt`

Le parsing du fichier `liste.txt` se fait via les 3 méthodes suivantes :
— `trim()`
— `parseMed()`
— `parsePharma()`

`trim()` est utile pour prendre un caractère et en supprimer les espaces superflus qui peuvent l'entourer.

`parseMed()` traite une ligne type en la lisant caractère par caractère. Lorsque le caractère de séparation " : " est reconnu, le nom du médicament étant le caractère situé juste avant va être stocké dans une variable.

Ensuite, la ligne continue d'être lue et dès qu'une virgule est reconnue le caractère situé avant est stocké dans un vecteur contenant les

effets secondaires. Ces instructions sont contenues dans une boucle dont la condition d'arrêt est qu'il n'y ait plus de virgule reconnue : ce qui veut dire qu'il n'y a plus d'effets secondaires sur cette ligne. Enfin, le médicament est stocké dans une table associative de type `map` avec son vecteur d'effets secondaires.

`parsePharma()` va permettre d'étendre ce traitement à toutes les lignes du fichier `.txt`.

3.4 Développement des algorithmes

3.4.1 Histogramme

`afficheHisto()` parcourt la map obtenue précédemmet. Il s'agit alors de stocker dans une liste tous les effets secondaires. On parcourt alors cette liste nouvellement créée. Parallèlement, il a été nécessaire de coder une méthode `rechercheEffet(string)` retournant un vecteur contenant les noms de médicaments provoquant un effet secondaire donné. Ainsi, le parcours de la liste permet pour chaque effets secondaires de trouver les médicaments le provoquant. On obtient le nombre de ces médicaments grâce à la fonction `size()`. On affiche alors autant d'étoiles `"*"` qu'il y a de médicament par effets.

3.4.2 Recherche d'informations

Recherche des médiaments ayant un effet secondaire en commun

Cet algorithme (`rechercheEffet(string)`) consiste à chercher le(s) médicament(s) dans la map en prenant en paramètre un effet secondaire entré par l'utilsateur. Cette méthode retourne un vecteur de string comprenant le nom des médicaments provoquant cet effet. Après avoir déclaré notre vecteur, il s'agit de parcourir la `map` instanciée par l'objet `Pharmacie`. Ensuite pour chacun des éléments de la map, l'effet est comparé à celui recherché. S'il correspond alors le médicament correspondant est ajouté dans le vecteur qui est retourné en résultat.

Sélection des médicaments ayant le plus d'effets secondaires en commun

Il s'agit ici de trouver la liste des médicaments avec le plus d'effets secondaires en commun. Plusieurs algorithmes ont été réalisés afin de réaliser cette tâche. Dans un premier temps, la méthode `statMedParEffets (vector<string>)` prend en paramètre un vecteur d'effet secondaire et retourne une map stockant pour chaque effet secondaire leur nombre d'apparition. Dans un second temps, `estpresent(list<string>, string)` vérifie la présence d'un élément dans une liste.

Enfin, `afficherPlusCommuns(list<string>, int)` permet d'afficher le premier élément d'une liste quand l'entier associé n'est pas nul, c'est-à-dire, ici quand l'élément apparaît au moins une fois.
La méthode `rechercheCommun(vector<string>, int)` utilise les trois algorithmes précédents. Elle parcourt tous les médicaments et sélectionne ceux qui ont le plus d'effets secondaires communs. Celle-ci prend en paramètre un vector de string contenant les effets secondaires et un entier correspondant au nombre d'occurences trouvées. En faisant appel à la première méthode `statMedParEffets (vector<string> )`, celle-ci stocke dans une map chaque effets avec leur occurence.
N.B. : Pour la suite, il a été nécessaire de stocker ce nombre d'apparition en `double` afin de pouvoir faire des calculs.
La `map` est parcourue : si l'élément n'est pas déjà présent dans la liste et si le score est supérieur à 0. En utilisant le deuxième algorithme (`estpresent(list<string> , string nom)`) une variable, initialisée à 0, récupère ce score et ajoute le médicament correspondant en tête de liste (`push front()`).
Finalement, le dernier algorithme (`afficherPlusCommuns(list<string>, int)`) permet d'afficher le premier résultat de cette liste. On retourne alors un vecteur de ces résultats.

Deux premiers médicaments trouvés avec au moins un effet secondaire en communs

On utilise le même algorithme de recherche que dans la fonction `rechercheCommuns()`. Une fois cet algorithme effectué, les deux premiers éléments de la `map "medicament" -> "nombre d'effets communs"` sont renvoyés.

Sélection de médicament au hasard

Le même algorithme de recherche que `rechercheCommuns()` est encore utilisé. Seul l'ordre de consultation des élements de la `map` change. Au lieu de les parcourir un a un dans l'ordre, l'algorithme passe d'un élément à un autre de façon aléatoire.
Le premier médicament rencontré qui comporte un effet secondaire commun est alors renvoyé.

3.4.3 Calcul de la précision

La méthode `precisionMed(vector<string> )` prend en paramètre un vector d'effet secondaire.
Elle fait appel à `statMedParEffets(vector<string> )` qui permet de récupérer une `map` contenant les médicaments et le nombre d'effets secondaires qu'ils ont en commun avec ceux entrés par l'utilisateur. Il s'agit alors de parcourir cette `map` et de calculer le rapport : (nombre d'effets dans ce médicament/ nombre total d'effets entrés)*100.

N.B. : Multipier par 100 permet d'obtenir un pourcentage de notre précision.

On retourne alors cette `map` qui contient maintenant pour chaque médicament un pourcentage de l'effet retrouvé. Comme nous l'avons énoncé précédemment, la nécessité d'avoir une valeur de type `double` est essentiel ici afin de faire le calcul sans quoi le résultat serait erroné. `PrecisionAlgo()` et `afficherPresisionMed (map <string, double> )` permettent d'afficher le résultat du calcul de la précision.

3.4.4 Option facultative : ajout de médicament

Nous avons codé cette fonctionnalité en utilisant les trois méthodes suivantes :

- `medicamentExiste(string)`
- `ajouterMedicament (string, vector<string>)`
- `AjouteMed()`

`medicamentExiste(string)` renvoit un booléen. Cette méthode teste si le nom de médicament entré existe et renvoie vrai ou faux en fonction. Ce test est nécessaire pour ne pas ajouter dans notre base de donnée minimaliste un médicament qui existe déjà.

`ajouterMedicament (string, vector<string>)` ne fait qu'ajouter le couple médicament/effets secondaires dans la base de donnée.

`AjouteMed()` constitue la partie du code intégrant le flux d'entrée standard afin d'intéragir avec l'utilisateur. Il va ainsi pouvoir ajouter un nom de médicament à la condition qu'il n'existe pas déjà auquel cas un message le lui indiquant est renvoyé. Sinon il peut l'ajouter à la base de donnée et entrer des effets secondaires qui seront stocké dans un vecteur. L'utilisateur peut à tout moment cesser la saisie en tappant `"exit"`.
A la fin de la saisie le couple médicament/effets secondaires est ajouté à la map grâce à la méthode `ajouterMedicament (string, vector<string>)`.

3.5 Architecture

La programmation orienté objet impose un code très structuré, c'est ce qui le rend souple, robuste et réutilisable. Voici une représentation schématique de notre programme :

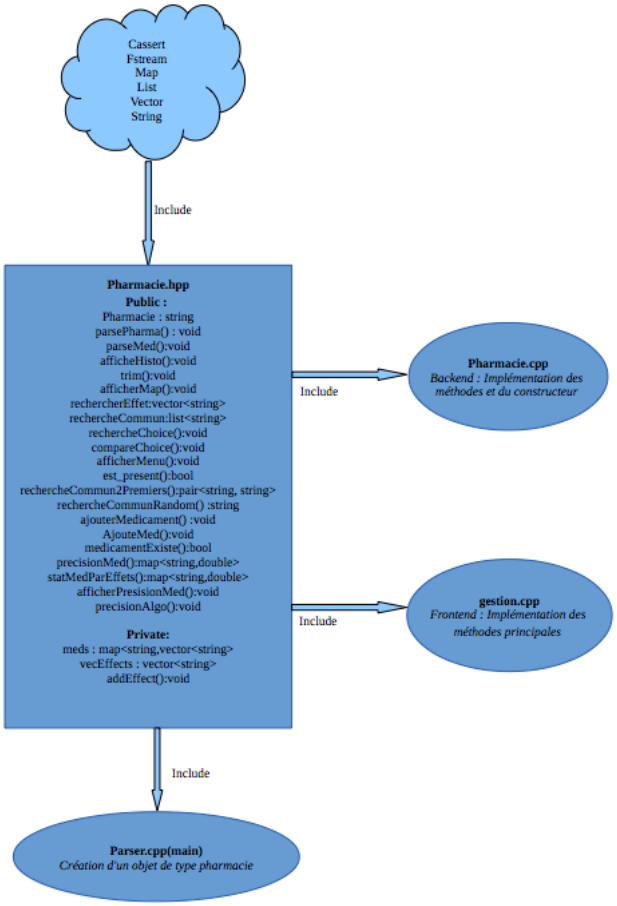

FIGURE 3.1 – Schéma de l'architecture du programme

CHAPITRE 4

RÉALISATION : TEST DE L'APPLICATION

Dans cette partie, nous nous attacherons à réaliser un test de notre programme.

4.1 Lancement du programme

4.1.1 Compilation

Avant de lancer l'application, il faut bien s'assurer de disposer de tous les fichiers nécessaires afin de convertir notre code source en fichiers binaires (éxecutable). Ainsi, après les étapes de pré-compilation et de compilation, un lien d'édition doit être créé afin d'obtenir un éxécutable, c'est au cours de cette étape que le compilateur agrège chaque fichier `.o` avec les fichiers binaires des librairies utilisées.
Le fichier `Makefile` fournit avec le projet permet de réaliser ces étapes en lançant la commande `"make all"` directement depuis le terminal. On peut aussi nettoyer le répertoire de travail avec les commandes `"make clean"` ou `"make mrproper"`. En outre, il est possible d'obtenir une aide concernant l'utilisation de ce programme en utilisant les paramètres -h ou –help au lancement du programme.

4.1.2 Menu principal

Le lancement du programme s'accompagne du lancement d'un menu principal. L'utilisateur entre un entier de 0 à 9 correspondant à l'option qu'il souhaite. Le programme gère le cas où l'utilisateur entre une option non conforme grâce à un contrôle qui lui renvoie un message d'erreur sans sortir du programme et lui laisse la possibilité de choisir de nouveau une option. Ce type d'interface assez intuitive (4.1) a été choisi afin de faciliter le déroulement du programme par l'utilisateur.

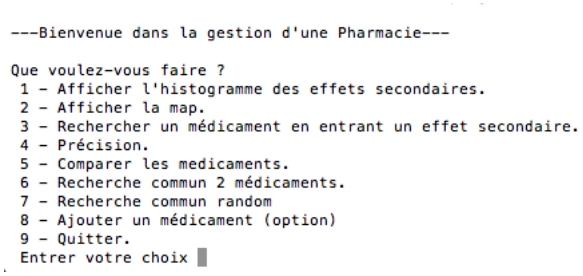

FIGURE 4.1 – Affichage du menu

4.2 Execution du programme

4.2.1 Affichage de l'histogramme

Lorsque l'utilisateur choisi l'option 1, le programme fait appel à la fonction `afficheHisto()`. Ainsi, l'utilisateur pourra voir les fréquences d'apparition de chaque effets secondaire dans le fichier `liste.txt`. Pour chaque effet secondaire nous l'indiqueront par une étoile '*'. Le programme affiche l'histogramme suivant :

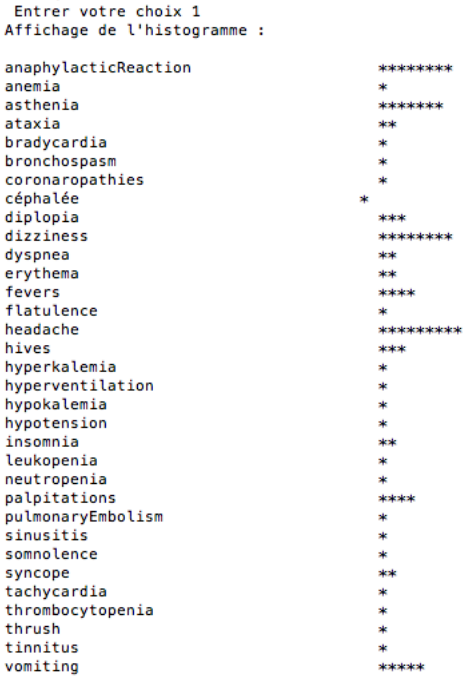

FIGURE 4.2 – Affichage de l'histogramme

4.2.2 Visualisation des médicaments

Il est possible d'afficher la liste des médicaments (4.3) en sélectionnant l'option 2. Nous avons choisi d'ajouter la visualisation de leurs effets secondaires qui sera utile par rapport à l'option choisie. Ainsi, il sera possible de vérifier si l'utilsateur a bien ajouté son médicament dans la base de donnée minimaliste.

```
 Entrer votre choix 2
Medicament : Airomir
Effet(s) secondaire(s) : palpitations
Medicament : Aspirine
Effet(s) secondaire(s) : anaphylacticReaction hyperventilation dizziness bronchospasm
Medicament : Betanol
Effet(s) secondaire(s) : asthenia insomnia
Medicament : Bipreterax
Effet(s) secondaire(s) : headache asthenia vomiting dizziness
Medicament : Carbaglu
Effet(s) secondaire(s) : bradycardia vomiting fevers
Medicament : Cilest
Effet(s) secondaire(s) : coronaropathies headache
Medicament : Dafalgan
Effet(s) secondaire(s) : anaphylacticReaction erythema leukopenia neutropenia
Medicament : Epinitril
Effet(s) secondaire(s) : hypotension syncope
Medicament : Epitomax
Effet(s) secondaire(s) : anemia dizziness tinnitus
Medicament : Eptativ
Effet(s) secondaire(s) : anaphylacticReaction hives
Medicament : Ketoprofène
Effet(s) secondaire(s) : vomiting dizziness hyperkalemia
Medicament : Lexomil
Effet(s) secondaire(s) : headache ataxia diplopia asthenia
Medicament : Loratadine
Effet(s) secondaire(s) : dizziness palpitations anaphylacticReaction
Medicament : Loxen
Effet(s) secondaire(s) : headache palpitations tachycardia
Medicament : Phénergan
Effet(s) secondaire(s) : dizziness anaphylacticReaction thrombocytopenia insomnia
Medicament : Reactine
Effet(s) secondaire(s) : dizziness headache hives anaphylacticReaction
Medicament : Riastrap
Effet(s) secondaire(s) : anaphylacticReaction dyspnea pulmonaryEmbolism
Medicament : Seretide
Effet(s) secondaire(s) : thrush hypokalemia palpitations sinusitis
Medicament : Smecta
Effet(s) secondaire(s) : vomiting flatulence
Medicament : Tetagrip
Effet(s) secondaire(s) : erythema fevers asthenia dyspnea headache
Medicament : Tiorfast
Effet(s) secondaire(s) : asthenia fevers headache
Medicament : Tranxène
Effet(s) secondaire(s) : diplopia asthenia
Medicament : Voltaren
Effet(s) secondaire(s) : vomiting hives céphalée dizziness somnolence
Medicament : Xanax
Effet(s) secondaire(s) : headache ataxia asthenia diplopia
Medicament : Xolair
Effet(s) secondaire(s) : anaphylacticReaction fevers syncope headache
```

FIGURE 4.3 – Visualisation des médicaments et ses effets secondaires

4.2.3 Recherche de médicaments

L'option 3 permet à l'utilisateur de rechercher un ou des médicaments en entrant un effet secondaire. D'abord, un message indique d'entrer l'effet secondaire que l'utilisateur souhaite tester. Le programme affiche alors la liste des médicaments contenant cet effet. Ainsi, quand on teste l'effet `fevers`, le programme affiche ceci :

```
 Entrer votre choix 3
Veuillez entrer un nom d'effet secondaire recherche : fevers
Medicaments trouves :
Carbaglu
Tetagrip
Tiorfast
Xolair
```

FIGURE 4.4 – Exécution de l'algorithme de recherche de médicaments

4.2.4 Calcul de la précision

L'option 4 de notre programme permet de calculer la précision. L'utilisateur doit saisir le nombre puis le nom des effets secondaires qui l'intéressent. Il obtient alors la liste des médicaments et le pourcentage recherché. Un exemple d'utilsation de cette option est illustrée ci-dessous :

```
 Entrer votre choix 4
Veuillez entrer le 1er effet du médicament :
fevers
Veuillez entrer le 2ème effets de ce médicament ou sinon tappez exit :
insomnia
Veuillez entrer le 3ème effets de ce médicament ou sinon tappez exit :
anaphylacticReaction
Veuillez entrer le 4ème effets de ce médicament ou sinon tappez exit :
exit
Précision de l'algorithme de recherche :
Airomir : 0
Aspirine : 33.3333
Betanol : 33.3333
Bipreterax : 0
Carbaglu : 33.3333
Cilest : 0
Dafalgan : 33.3333
Epinitril : 0
Epitomax : 0
Eptativ : 33.3333
Ketoprofène : 0
Lexomil : 0
Loratadine : 33.3333
Loxen : 0
Phénergan : 66.6667
Reactine : 33.3333
Riastrap : 33.3333
Seretide : 0
Smecta : 0
Tetagrip : 33.3333
Tiorfast : 33.3333
Tranxène : 0
Voltaren : 0
Xanax : 0
Xolair : 66.6667
```

FIGURE 4.5 – Exécution du calcul de la précision

4.2.5 Recherche d'informations

Les options 5 à 7 permettent de rechercher des informations selon différentes conditions. Par exemple, quand l'utilsateur choisit l'option 5, le programme lui demande d'abord combien d'effets secondaires il veut tester ; ensuite il peut entrer les effets qu'il souhaite. Le programme lui affiche alors la liste des médicaments ayant le plus d'effets en commun avec ceux entrés :

```
 Entrer votre choix 5
Combien d'effets voulez-vous comparer ?
2
Entrez l'effet 1
fevers
Entrez l'effet 2
insomnia
Medicament ayant le plus d'effets communs avec ces effets secondaires  :
Xolair
Tiorfast
Tetagrip
```

FIGURE 4.6 – Comparaison de médicaments

On peut aussi retouver 2 médicaments en commun avec l'option 6 ou faire une recherche de médicaments au hasard avec l'option 7.

4.2.6 Ajout d'un médicament dans la base

Enfin, la dernière option, en entrant l'entier 8 permet à l'utilisateur d'ajouter des médicaments dans la base de donnée minimaliste sans recharger le programme. Ainsi quand l'utilisateur choisit cette option, un message lui deande d'abord combien de médicaments il veut ajouter.

Ensuite si celui-ci existe déjà un message est d'erreur est renvoyé et il est impossible d'ajouter des effets secondaires à ce médicament sinon l'utilsateur peut en ajouter et cesse sa saisie en entrant `"exit"` :

```
 Entrer votre choix 8
Combien de médicaments voulez-vous ajouter ?
2
Veuillez entrer le nom du médicament 1 : Xolair
Le médicament Xolair existe déjà dans la base de données
Veuillez entrer le nom du médicament 2 : Efferalgan
Veuillez entrer le 1er effet du médicament :
fevers
Veuillez entrer le 2ème effets de ce médicament ou sinon tappez exit :
palpitations
Veuillez entrer le 3ème effets de ce médicament ou sinon tappez exit :
exit
```

FIGURE 4.7 – Ajout d'un médicament dans la base de données

Enfin, l'utilisateur pourra quitter proprement le programme en entant l'entier 9.

4.3 Discussion

4.3.1 Problèmes rencontrés

Lors de la modélisation du programme, nous avons pu faire faire face à quelques doutes. En particulier sur la manière de comprendre certaines consignes afin de pouvoir les implémenter correctement.
Notamment pour le calcul de la précision, il a fallu s'accorder sur la manière d'obtenir un résultat afin d'implémenter un algorithme adéquat. En effet, nous nous sommes demandé, s'il s'agissait d'un calculer d'une information "globale" indépendante des entrées faites par l'utilsateur ou bien si pour chaque médicament on pouvait faire ce calcul. Il a été plus logique d'implémenter cette deuxième option pour permettre à l'utilisateur d'interagir avec le programme.

De plus, il a fallu permettre au programme de réaliser des calculs corrects chose que les variables de type `integer` ne permettait pas étant donné que nous devions faire un rapport entre deux résultats. Il a juste fallu prendre en compte cela et stocker les variables, contenues dans les `maps` correspondantes, en `double`.
N'ayant jamais programmé auparavant en C++, notre manque de connaissance de toutes les oppportunités qu'offrent ce langage nous a quelques peu restreint dans nos choix. Nous aurions pu optimiser notre programme en utilsant des `templates` qui aurait permis par exemple de définir un modèle de fonction dont le type d'arguments et de retour n'est pas défini. Cette alternative aurait permis au programme de gagner en clarté et en temps d'exécution bien que le temps d'exécution ne pose pas de problème ici.

Enfin, afin d'offrir une utilisation plus ergonomique et intuitive du programme, une interface graphique aurait pu être réalisé avec `QtCreator`.

4.3.2 Modularité du programme

La modularité est importante afin d'avoir un code source plus claire mais aussi plus souple. La séparation des méthodes en prototype et définition permet une modularité du programme. Le header contient le nombre minimal d'information permettant de compiler le fichier `.cpp` correspondant. Il permet aussi de réduire de façon drastique le temps de compilation. En outre, il permet aussi de ne pas inclure plus d'une fois un même fichier grâce aux gardes anti inclusion multiples. Ainsi, il est possible de réutiliser les méthodes définies dans un autre fichier `.cpp`.

CONCLUSION

Ce projet nous a permis de mieux cerner les attentes liés à la programmation orientée objet et d'aquérir des notions dépassant le cadre du projet.

En effet, à travers la lecture d'un fichier `.txt`, l'extraction et le traitement de ces informations grâce au langage `C++`, nous avons pu constater l'étendue et les nombreuses fonctionnalités qu'offrent ce langage.

L'utilisation de la programmation orientée objet apporte de nombreux avantages. En effet, tant pour ce qui est de l'encapsulation, de la réutilisation de code ou encore pour éviter les redondances ce type de programmation est un atout indéniable.

Par ailleurs, nous avons pu voir les vastes possibilités qu'offrent le langage `C++` en travaillant les différents concepts algorithmiques. Ceci a été très enrchissant dans le cadre de notre parcours en bioinformatique.

Dans une perspective d'améliaration, le programme pourrait s'étendre à d'autres types de fichiers en entrée et il serait envisageable de traiter une plus grande quantité de données.

BIBLIOGRAPHIE

[1] Bjarne STROUSTRUP. *LE LANGAGE C++*, Pearson Education ; Édition : Revue et corrigé. 16 juin 2004.

[2] Claude DELANNOY. *Apprendre le C++*. édition EYROLLES

[3] Mathieu NEBRA, Mathieu SCHALLER. *PROGRAMMEZ AVEC LE LANGAGE C++*, éditions Le livre du zéro.

[4] `http://www.cplusplus.com/`