# INFO-F105 – Langages de programmation I
# Projet - Simulation d'une Intelligence Artificielle Générative
# Phase II

**Youcef Bouharaoua**
youcef.bouharaoua@ulb.be

**Année académique 2023-2024**

## 1 Introduction

Après avoir posé les bases de notre festin numérique en préparant et organisant nos données textuelles, nous entrons dans une nouvelle étape de notre aventure culinaire dans l'intelligence artificielle générative. Pour cette seconde phase, il est important de noter que nous allons nous concentrer exclusivement sur le traitement du texte. Bien que l'univers des données puisse s'étendre aux images, vidéos et fichiers audio, notre trousseau de clés actuel en programmation et nos connaissances en traitement de ces types de données sont encore à enrichir. De plus, le développement d'algorithmes d'entraînement pour ces médias requiert une compréhension approfondie des mathématiques et des concepts avancés en IA que nous n'avons pas encore explorés.[1]

### Détails de soumission et critères d'évaluation

Durant cette deuxième phase du projet, les notions suivantes sont abordées :

- Les variables et la gestion de la mémoire.
- Les instructions basiques en C++ dont les conditionnelles et les boucles.
- Les types de données dont les pointeurs, les tableaux et les structures.
- Les fonctions dont la transmission par valeur et référence.

[1] Introduction reformulée en utilisant une IA

Il n'est pas nécessaire que vous maîtrisiez ces concepts dans les détails pour pouvoir faire cette phase. L'idée est de vous y familiariser de manière pratique. Ne vous inquiétez pas si le travail que vous rendez ne fonctionne pas parfaitement (tant que vous rendez votre travail), aucune pénalité ne sera appliquée. Pour rappel, votre travail pour la seconde phase doit être rendu au plus tard le 22 avril 2024 à 23h59. La remise est constitué d'un zip portant comme nom *phase2.zip* et contenant :

- Un dossier *TF IDF* contenant:
  - *tf_idf.hpp*
  - *tf_idf.cpp*
  - Makefile pour la partie TF-IDF.
- Un dossier *Analyse sentiments*
  - *analyse_sentiments.hpp*
  - *analyse_sentiments.cpp*
  - Makefile pour la partie analyse de sentiments.
- Un rapport d'analyse en format PDF.

En cas de non remise à temps, vous ne serez pas autorisés à rendre votre travail pour la troisième phase.

N'oubliez pas de consulter régulièrement les ressources mises à votre disposition pour vous aider tout au long de cette phase du projet. Voici quelques liens utiles pour faciliter votre progression :

- **Consignes Générales :** Lien vers les consignes générales du projet
- **Partie 1 du Projet :** Lien vers les détails de la Partie 1
- **Feedback sur la Partie 1 :** Lien vers le feedback de la Partie 1
- **Foire aux questions :** Lien vers la FAQ

Ces ressources sont conçues pour vous guider et vous fournir les informations nécessaires pour réussir cette phase du projet. Assurez-vous de les consulter pour mieux comprendre les attentes et pour tirer le meilleur parti des conseils et des instructions fournis.

# 2 La Phase II

## Objectif

Les textes offrent un terrain fertile pour appliquer et expérimenter avec des notions fondamentales en NLP (Natural Language Processing) de manière simplifiée. Dans cette phase, nous aborderons une version simplifiée de l'**analyse de sentiments** et de la méthode **TF-IDF (Term Frequency-Inverse Document Frequency)**, deux pierres angulaires dans le domaine du traitement du langage naturel. Ces techniques nous permettront d'extraire des informations et des enseignements précieux à partir de simples collections de textes, sans plonger dans les complexités des algorithmes d'entraînement ou des mathématiques avancées.

Cette exploration nous donnera un aperçu de comment les machines peuvent commencer à comprendre et à interpréter le langage humain, un ingrédient essentiel dans la recette de l'IA générative. Préparez-vous donc à manier vos ustensiles de programmation pour cette nouvelle étape de notre voyage culinaire numérique.

## 2.1 Implantation Simplifiée de TF-IDF

### 2.1.1 Principe de TF-IDF :

TF-IDF, pour Term Frequency-Inverse Document Frequency, est une mesure statistique utilisée pour évaluer l'importance d'un mot dans un document par rapport à un ensemble de documents. Le calcul se fait en deux étapes :

- **TF (Term Frequency)** indique la fréquence d'un mot dans un document.
- **IDF (Inverse Document Frequency)** diminue l'importance des mots qui apparaissent dans de nombreux documents, rendant les mots rares plus significatifs.

**Formule :** Le score TF-IDF d'un mot est calculé comme suit :

$$\text{TF-IDF} = TF \times IDF$$

où,

$$TF(mot) = \frac{\text{Nombre de fois que le mot apparaît dans le document}}{\text{Nombre total de mots dans le document}}$$

$$IDF(mot) = \log_{10}\left(\frac{\text{Nombre total de documents}}{\text{Nombre de documents contenant le mot}}\right)$$

### 2.1.2 Tâches :

1. **Lire les Documents Textuels :** Fonction pour lire le contenu d'un fichier texte et le retourner sous forme d'une liste (`vector`) de mots.

Listing 1: Lecture d'un document

```
std::vector<std::string> readDocument(const std::string&
    filePath);
```

2. **Calculer le TF (Term Frequency) :** Fonction pour calculer la fréquence de chaque mot dans un document. La fréquence est calculée comme le nombre d'occurrences du mot divisé par le nombre total de mots dans le document.

Listing 2: Calcul du TF

```
std::map<std::string, double> computeTF(const std::vector<std
    ::string>& words);
```

3. **Calculer l'IDF (Inverse Document Frequency) :** Fonction pour calculer l'IDF pour chaque mot sur l'ensemble des documents. Cette fonction nécessite l'accès à tous les documents pour déterminer combien de documents contiennent chaque mot.

Listing 3: Calcul de l'IDF

```
std::map<std::string, double> computeIDF(const std::vector<std
    ::map<std::string, double>>& documentsTF, int
    totalDocuments);
```

4. **Calculer et Afficher le Score TF-IDF :** Fonction pour calculer le score TF-IDF pour chaque mot dans chaque document en utilisant les valeurs TF et IDF calculées précédemment. Cette fonction calcule ensuite le score TF-IDF en multipliant le TF par l'IDF pour chaque mot.

Listing 4: Calcul du TF-IDF

```
std::map<std::string, double> calculateTFIDF(const std::map<
    std::string, double>& tf, const std::map<std::string,
    double>& idf);
```

Fonction pour afficher en utilisant les flux standards (e.g., *cout*) les scores TF-IDF de chaque mot pour un document donné.

Listing 5: Affichage des scores TF-IDF

```
void displayTFIDFScores(const std::map<std::string, double>&
    tfidfScores);
```

#### 2.1.3 Livrables :

- **Code Source :** Un programme en C++ réalisant les calculs de TF et IDF, et déterminant le score TF-IDF pour les mots.

### 2.1.4 Ressources Fournies :

Ensemble de documents pour l'analyse (*medecine_1.txt*, *medecine_2.txt*, *medecine_3.txt*).

Ci-dessous, un exemple d'affichage avec les fichiers exemples fournis.

Listing 6: TF-IDF results

```
  [{'la': 0.4054651081081644,
 'medecine': 0.1013662770270411,
 'est': 0.1013662770270411,
 'science': 0.27465307216702745,
 'et': 0.0,
 'pratique': 0.27465307216702745,
 'du': 0.5493061443340549,
 'diagnostic,': 0.27465307216702745,
 'traitement': 0.27465307216702745,
 'de': 0.0,
 'prevention': 0.27465307216702745,
 'des': 0.1013662770270411,
 'maladies.': 0.27465307216702745},
{'les': 0.4054651081081644,
 'maladies': 0.4054651081081644,
 'cardiovasculaires': 0.5493061443340549,
 'representent': 0.5493061443340549,
 'une': 0.2027325540540822,
 'categorie': 0.5493061443340549,
 'majeure': 0.5493061443340549,
 'de': 0.0,
 'affectant': 0.5493061443340549,
 'le': 0.5493061443340549,
 'coeur': 0.5493061443340549,
 'et': 0.0,
 'vaisseaux': 0.5493061443340549,
 'sanguins.': 0.5493061443340549},
{'la': 0.4054651081081644,
 'chirurgie': 0.5493061443340549,
 'est': 0.2027325540540822,
 'une': 0.2027325540540822,
 'branche': 0.5493061443340549,
 'de': 0.0,
 'medecine': 0.2027325540540822,
 'qui': 0.5493061443340549,
 'traite': 0.5493061443340549,
 'les': 0.4054651081081644,
 'maladies': 0.2027325540540822,
 'et': 0.0,
 'blessures': 0.5493061443340549,
 'par': 0.5493061443340549,
 'des': 0.2027325540540822,
 'interventions': 0.5493061443340549,
 'manuelles': 0.5493061443340549,
 'instrumentales.': 0.5493061443340549}]
```

### 2.1.5 Note :

Vous pouvez légèrement modifier la signature des fonctions, mais votre choix doit être pertinent et justifié. Aussi, il est à noter que **les scores ci-dessus peuvent être différents de ce que vous obtenez**[2]

## 2.2 Implantation d'un système simplifié d'analyse de sentiments

### 2.2.1 Tâches

1. **Création d'un Dictionnaire de Sentiments :**
   Développez une structure pour représenter un dictionnaire contenant des listes de mots-clés associés à des sentiments positifs ou négatifs, en utilisant des tableaux dynamiques (`std::vector<std::string>`) pour stocker les mots-clés.

Listing 7: Structure SentimentDictionary

```
struct SentimentDictionary {
    std::vector<std::string> positiveWords;
    std::vector<std::string> negativeWords;
};
```

2. **Fonction de Chargement du Dictionnaire :**
   Écrivez une fonction pour initialiser le dictionnaire de sentiments à partir de fichiers texte séparés `positive.txt` et `negative.txt`. Les fichiers sont fournis et se trouvent dans le *.zip* des ressources de cette seconde phase.

Listing 8: Fonction de chargement du dictionnaire

```
void loadSentimentDictionary(SentimentDictionary&
    dictionary, const std::string& positivePath, const std
    ::string& negativePath);
```

[2] C'est dû à l'utilisation de librairies externes et à des concepts plus évolués (e.g., Tokenization.)

3. **Résultat de l'analyse :**
   Créez une structure pour stocker le résultat de l'analyse, incluant le nombre de mots positifs, le nombre de mots négatifs, et une conclusion générale sur le sentiment du texte. Les attributs *positiveCount* et *negativeCount* représentent respectivement le nombre total de mots positifs et négatifs trouvés dans le texte tandis que l'attribut *overallSentiment* représente l'évaluation générale du sentiment du texte, basée sur l'analyse des mots positifs et négatifs. En fonction des valeurs de *positiveCount* et *negativeCount*, le sentiment global pourrait être défini comme "Positif", "Négatif" ou "Neutre"

Listing 9: Structure AnalysisResult

```
struct AnalysisResult {
    int positiveCount;
    int negativeCount;
    std::string overallSentiment;
};
```

4. **Analyse de sentiments Simplifiée :**
   Implémentez une fonction qui prend en entrée un chemin vers un fichier texte contenant un avis ou un commentaire, analyse chaque mot du texte en utilisant le dictionnaire de sentiments, et détermine si le texte général est plutôt positif, négatif, ou neutre.

Listing 10: Fonction d'analyse de sentiment

```
AnalysisResult analyzeSentiment(const SentimentDictionary&
    dictionary, const std::string& filePath);
```

### 2.2.2 Livrables

- **Code Source C++ :** Comportant les fonctions originales pour la gestion du dictionnaire de sentiments et l'analyse de sentiment. Affichez simplement le nombre de mots positifs et négatifs à la fin de l'analyse de chaque fichier.

Ci-dessous, un exemple de trois textes:

- Un texte négatif

Listing 11: text_negatif.txt

```
Malheureusement, le produit est mauvais.
```

- Un texte neutre

Listing 12: text_neutre.txt

```
Ce document presente une analyse objective des faits
    sans exprimer une opinion personnelle ou emotion.
```

- Un texte positif

Listing 13: text_positif.txt

```
Le produit est utile, il est excellent !
```

# 3 Rapport d'analyse

Dans le cadre de la seconde phase du projet, incluant à la fois le travail sur le TF-IDF et l'analyse de sentiments, vous êtes invités à rédiger un rapport succinct qui évaluera plusieurs aspects de votre implantation. Ce rapport devra couvrir les points suivants :

1. **Choix des Types :** Expliquez pourquoi vous avez choisi certains types de données.
2. **Utilisation des Boucles :** Discutez de l'utilisation des différentes structures de boucles dans votre code et justifiez vos choix.
3. **Passage de Paramètres :** Décrivez les différentes manières de passage de paramètres en C++ que vous avez utilisées et expliquez pourquoi vous avez préféré une méthode à une autre dans certains cas.
4. **Comparaison avec Python :** Discutez des différences que vous auriez rencontrées si le projet avait été réalisé en Python, en termes de syntaxe, de structures de données disponibles, de gestion de la mémoire, et de passage de paramètres.

Votre rapport devra non seulement refléter une compréhension technique des aspects abordés mais aussi présenter une réflexion critique sur vos choix de conception et d'implantation. Ce document représente une opportunité de démontrer votre capacité à analyser et à justifier les décisions prises durant le développement du projet.

Le rapport ne doit pas dépasser deux pages et doit être soumis au format PDF en même temps que votre code source, dans le fichier zip nommé *phase2.zip*, avant la date limite du 22 avril 2024 à 23h59.