RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
ÉCOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

MODULE : COMPILATION (COMPIL)

2ÈME ANNÉE CYCLE SUPÉRIEUR (2CS - OPTION SIL)

GROUPE : SIL 1

PROPOSITION DE LANGAGE SENTINEL *DSL d'Audit et de Sanitization de Données*

Réalisé par le binôme :

• ATTIA Oussama Abderraouf

Encadré par :

Mr. ABDMEZIEM Riyadh

Année universitaire : 2025 – 2026

Table des matières

1. INTRODUCTION GÉNÉRALE

L'informatique moderne opère une transition majeure vers l'utilisation massive de l'Intelligence Artificielle et le traitement de données à très grande échelle (*Big Data*). Cependant, la fiabilité de ces systèmes repose intégralement sur la qualité des données injectées. L'adage *"Garbage In, Garbage Out"* est plus vrai que jamais : un jeu de données "sale", biaisé ou contenant des informations personnelles (PII) peut compromettre un modèle d'IA entier ou entraîner des violations légales graves (RGPD).

Dans ce cadre, nous proposons de concevoir **SENTINEL**.

SENTINEL est un langage dédié (DSL - Domain Specific Language) dont l'objectif est l'automatisation sécurisée des pipelines de nettoyage de données. Il permet de définir des politiques de validation, de détection de toxicité et d'anonymisation dans un format formel et compilé.

Contrairement à l'approche classique consistant à écrire des scripts en Python ou Bash (difficiles à maintenir et potentiellement insécurisés), SENTINEL force le développeur à adopter une structure rigide. Le compilateur SENTINEL agit comme un "gardien" : il vérifie non seulement la syntaxe, mais aussi la cohérence sémantique des opérations de manipulation de données avant même l'exécution.

Le projet consiste en la réalisation de la chaîne de compilation complète (analyse lexicale, syntaxique et sémantique) ainsi que la génération de code intermédiaire pour ce langage.

2. POSITIONNEMENT TECHNIQUE : POURQUOI SENTINEL ?

La question de la pertinence d'un nouveau langage en 2025, face à la prédominance de Python et des LLMs (Large Language Models), mérite une réponse architecturale précise. SENTINEL répond à trois impératifs industriels critiques :

2.1. La Sécurité par le "Design" (Sandboxing)

Les langages généralistes comme Python sont trop permissifs pour des environnements critiques. Un script de nettoyage de logs mal conçu pourrait, par erreur, supprimer des fichiers système ou ouvrir des connexions réseaux non désirées. **SENTINEL applique le principe de moindre privilège.** Le langage est conçu comme un "Sandbox" (bac à sable) : il ne possède aucune instruction permettant l'accès arbitraire au système de fichiers ou au réseau. Le compilateur garantit que le programme ne fait *que* lire des entrées définies, les traiter, et écrire des sorties définies.

2.2. Le Déterminisme face au Probabilisme

Si les LLMs sont capables de nettoyer du texte, ils ne sont pas déterministes. Un même modèle peut nettoyer une donnée différemment à deux instants T, ou "halluciner" en supprimant des informations valides. Dans les secteurs bancaires, médicaux ou militaires, cette incertitude

est inacceptable. SENTINEL garantit un **comportement 100% reproductible** basé sur des règles logiques strictes.

2.3. Performance et "Green IT"

L'inférence d'un modèle d'IA pour vérifier la présence d'un mot interdit dans un log de 10 Go consomme une quantité d'énergie considérable (GPU). De même, lancer un interpréteur Python induit un coût mémoire (Overhead) significatif. En tant que langage compilé vers du code natif (via C/C++), SENTINEL offre une exécution ultra-rapide avec une empreinte mémoire minime, le rendant idéal pour l'informatique embarquée (*Edge Computing*) ou le traitement de flux massifs.

3. STRUCTURE GÉNÉRALE DU PROGRAMME

Afin d'imposer une rigueur logicielle, la structure d'un programme SENTINEL n'est pas libre. Elle suit le paradigme de "Configuration ségrégée de l'Exécution". Un fichier source se compose toujours d'un bloc unique `PROCESS`, subdivisé en deux sous-blocs obligatoires.

3.1. Le Bloc CONFIG

Ce bloc est statique. Il contient toutes les déclarations de ressources : fichiers, dictionnaires de mots-clés, seuils numériques et variables globales. Aucune instruction algorithmique (boucle, condition dynamique) n'y est autorisée. Cette séparation permet au compilateur d'optimiser l'allocation mémoire dès le départ.

3.2. Le Bloc PIPELINE

Ce bloc contient la logique impérative : lecture des flux, application des filtres et prise de décision. Les variables déclarées dans CONFIG y sont accessibles, mais de nouvelles variables locales temporaires peuvent y être définies.

Squelette syntaxique :

```
PROCESS "Nom_Processus" {

    CONFIG {
        // D claration des constantes, fichiers et structures
        // Allocation statique m moire
    }

    PIPELINE {
        // Instructions de traitement s quentiel
        // Logique m tier (Boucles, Tests, E/S)
```

```
    }
}
```

4. SYSTÈME DE TYPES ET STRUCTURES DE DONNÉES

L'innovation de SENTINEL réside dans son typage fort orienté "Métier".

4.1. Types Scalaires

1. `integer` : Entier signé classique sur 32 bits. Utilisé pour les boucles et les compteurs simples.
2. `text` : Chaîne de caractères. Contrairement au C, c'est un type primitif géré dynamiquement.
3. `probability` : Type flottant strictement borné `[0.0, 1.0]`.
   — *Règle Sémantique :* Toute tentative d'affectation hors bornes déclenche une erreur de compilation (si constante) ou d'exécution.
4. `verdict` : Type énumératif système à trois états possibles :
   — `VALID` : Donnée saine.
   — `SUSPECT` : Nécessite une revue humaine.
   — `REJECT` : Donnée bloquée automatiquement.

4.2. Types Structurés et Collections

Pour le traitement par lots (*Batch Processing*), le langage intègre des structures optimisées :

— `dictionary` : Une liste en lecture seule de chaînes de caractères (Mots interdits, whitelist).

— `batch<T>` : Une file dynamique typée.

4.3. Déclaration de Variables

Les variables sont déclarées avec la syntaxe `TYPE identificateur = VALEUR`. L'initialisation est obligatoire dans le bloc CONFIG.

```
CONFIG {
    integer threshold = 80
    probability confidence = 0.99
    text file_path = "/data/logs.txt"
    dictionary stop_words = ["error", "fatal", "null"]
}
```

5. INSTRUCTIONS DU LANGAGE

Les instructions sont conçues pour être expressives et concises.

5.1. Abstraction des Entrées / Sorties (I/O)

Le langage masque la complexité des pointeurs de fichiers (FILE*).

— `INGEST <var> FROM <source>` : Lit une entrée (ligne par ligne pour les fichiers texte). Retourne `EOF` ou `NULL` à la fin.

— `DISPATCH <var> TO <dest>` : Écrit la variable dans une destination sécurisée.

— `LOG "Message" LEVEL <int>` : Instruction système d'audit. Le niveau (1-5) détermine la gravité.

5.2. Primitives de Sanitization

Ces instructions modifient directement le contenu des variables `text`.

— `NORMALIZE <var>` : Applique une mise en minuscules (to_lower) et un nettoyage des espaces (trim).

— `MASK <var> WITH <char>` : Remplace tout le contenu (ou une sous-partie ciblée) par le caractère donné. Utilisé pour l'anonymisation.

5.3. Instruction de Détection de Patterns (SCAN)

L'instruction `SCAN` est une structure de contrôle itérative spécifique. Elle permet de parcourir un texte pour chercher des occurrences issues d'un `dictionary`.

```
SCAN <texte> WITH <dictionnaire> {
    // Ce bloc s'ex cute SI un mot est trouv
    // 'MATCH' est une variable implicite contenant le mot trouv
}
```

6. OPÉRATEURS, PRIORITÉS ET ASSOCIATIVITÉ

SENTINEL introduit des opérateurs sémantiques (`CONTAINS`, `IN`) qui s'ajoutent aux opérateurs standards.

Règles d'associativité : De Gauche à Droite (L-to-R) pour tous les opérateurs, sauf l'opérateur d'affectation (=) qui est associatif à droite.

TABLE 1 – Table des priorités des opérateurs (du plus fort au plus faible)

backcolour **Priorité**	**Opérateurs**	**Description**
1	`( )`	Regroupement explicite.
2	`NOT`, `-`	Opérateurs unaires (Négation logique, signe).
3	`IN`, `CONTAINS`	**Opérateurs Sémantiques**. Vérifie l'inclusion dans une liste ou une sous-chaîne.
4	`*`, `/`, `%`	Multiplications et division entière.
5	`+`, `-`	Additions (ou concaténation texte) et soustractions.
6	`<`, `>`, `<=`, `>=`	Comparaison numérique.
7	`==`, `!=`	Test d'égalité et d'inégalité.
8	`AND`	Conjonction Logique (ET).
9	`OR`	Disjonction Logique (OU).
10	`=`	Affectation de variable.

7. RÈGLES DE SÉMANTIQUE STATIQUE

Le compilateur doit implémenter un ensemble de règles strictes lors de l'analyse sémantique. Tout manquement entraîne une erreur de compilation et l'arrêt du processus.

— **Contrôle de Type (Type Safety)** : Il est interdit d'effectuer des opérations arithmétiques entre des types incompatibles (ex : `integer + text`). La concaténation n'est explicite que via des fonctions dédiées ou contextes précis.

— **Contrainte Probability** : Les littéraux affectés à un type `probability` doivent être analysés syntaxiquement. Si une valeur comme `1.5` ou `-0.1` est détectée, le compilateur lève une erreur : *"Probability Out of Bounds"*.

— **Scope (Portée)** : Les variables définies dans `PIPELINE` ne peuvent pas porter le même nom qu'une variable de `CONFIG` (Shadowing interdit pour éviter les erreurs d'audit).

— **Immuabilité des Dictionnaires** : Une structure `dictionary` ne peut être modifiée (ajout/-suppression) à l'intérieur du bloc `PIPELINE`. Elle est en lecture seule (*Read-Only*).

8. EXEMPLE COMPLET DE SCRIPT SENTINEL

Le programme ci-dessous illustre un pipeline complet : ingérer un flux de messages de chat, calculer un "score de risque", bloquer les contenus haineux, et anonymiser les adresses emails avant de sauvegarder.

```
1 PROCESS "Chat_Moderator_V1" {
2
3     CONFIG {
4         // --- RESSOURCES STATIQUES ---
```

```
        integer max_risk = 50
        integer count = 0

        // Fichiers simul s
        text src = "/input/stream.log"
        text db_clean = "/output/validated.csv"
        text db_quarantine = "/output/toxic.log"

        // Listes de mots
        dictionary toxic_words = ["idiot", "scam", "hate"]
        dictionary alerts = ["help", "admin", "sos"]
    }

    PIPELINE {
        // --- LOGIQUE D'EXECUTION ---
        LOG "Demarrage du traitement..." LEVEL 1

        text msg = ""
        integer risk_score = 0
        verdict status = VALID

        // Lecture du flux entrant
        INGEST msg FROM src

        WHILE (msg != NULL) {

            // Reinitialisation
            risk_score = 0
            status = VALID
            NORMALIZE msg

            // Etape 1 : Analyse de Toxicite
            SCAN msg WITH toxic_words {
                // Penalite forte pour chaque mot interdit
                risk_score = risk_score + 25
            }

            // Etape 2 : Decision et Routage
            IF (risk_score >= max_risk) {
                status = REJECT
                LOG "Message rejete (High Risk)" LEVEL 3
```

```
                DISPATCH msg TO db_quarantine
            }
            ELSE {
                // Etape 3 : Anonymisation conditionnelle
                IF (msg CONTAINS "@") {
                    MASK msg WITH "*"
                    LOG "PII detectee et masquee" LEVEL 2
                }

                status = VALID
                DISPATCH msg TO db_clean
            }

            count = count + 1
            INGEST msg FROM src
        }

        LOG "Traitement termine avec succes." LEVEL 1
    }
}
```

Listing 1 – Programme Data_Sanitizer.sent

9. FEUILLE DE ROUTE TECHNIQUE

Pour mener à bien l'implémentation de SENTINEL dans le cadre de ce projet académique, nous utiliserons les outils standards de compilation UNIX.

1. **Analyse Lexicale (FLEX)** : Reconnaissance des mots-clés `PROCESS`, `INGEST`, des littéraux flottants pour les probabilités, et gestion des chaînes de caractères.
2. **Analyse Syntaxique (BISON)** : Définition de la grammaire (Grammaire LALR) pour structurer l'arbre syntaxique abstrait (AST). Gestion des blocs imbriqués et des priorités d'opérateurs via les directives `%left` et `%right`.
3. **Table des Symboles et Sémantique** : Implémentation en C d'une double table des symboles :
   - *Table Globale* pour la section CONFIG.
   - *Table Locale* pour le scope courant du PIPELINE.

   C'est à cette étape que seront vérifiées les compatibilités de types et les contraintes (ex : variable `text` utilisée comme un `integer`).
4. **Génération de Code** : Le projet aboutira à la génération de Quadruplets (Code intermédiaire), prêt à être exécuté par une machine virtuelle simple ou traduit en langage C.

10. CONCLUSION

Ce document a présenté la spécification formelle du langage **SENTINEL**. En choisissant de s'éloigner des projets de manipulation de fichiers génériques, nous proposons une solution orientée vers les besoins réels de l'industrie : l'intégrité des données à l'ère de l'Intelligence Artificielle.

Avec une syntaxe claire, des contraintes de typage fortes et une architecture "secure-by-design", SENTINEL constitue un sujet de compilation riche. Il permettra de démontrer, à travers la mise en œuvre de Flex et Bison, comment les concepts théoriques de la compilation s'appliquent pour résoudre des problèmes complexes de sémantique, d'optimisation et de sécurité logicielle.