Rapport Final du Projet

C-Wildwater

Réalisé par : Shahd, Kenza, Tenzin

21/12/2025-Informatique III,MEF1-2

Sommaire

I - Présentation des Objectifs

Ce document constitue le rapport final du projet informatique *C-WildWater*, dont l'objectif principal est la conception et le développement d'une application dédiée à l'analyse et à la synthèse de données issues d'un système de distribution d'eau potable. Ces données sont fournies sous la forme d'**un fichier CSV** de très grande taille, rendant tout traitement manuel impossible.

Le projet vise à exploiter efficacement ce volume important d'informations afin d'en extraire des indicateurs quantitatifs pertinents. Pour répondre aux contraintes de performance, l'application repose sur une architecture combinant deux technologies complémentaires :

- un **script Shell**, utilisé comme point d'entrée du programme, chargé de la gestion des commandes utilisateur et de l'orchestration des traitements ;
- un **programme en langage C**, responsable des calculs intensifs, afin d'assurer une exécution rapide et optimisée.

Les données étudiées décrivent l'ensemble des étapes du réseau de distribution d'eau potable, depuis les sources de captage jusqu'aux usagers finaux, en passant par les usines de traitement, les espaces de stockage et les différents niveaux de distribution.

Le travail demandé consiste à filtrer, structurer et analyser ces données afin de produire des résultats exploitables, notamment sous la forme d'histogrammes représentatifs des performances des usines et de calculs précis des pertes d'eau au sein du réseau. Ce projet met ainsi en œuvre des compétences en traitement de données, en algorithmique et en optimisation des performances dans un contexte réaliste et contraint.

II - Description de l'Application Développée :

L'application *C-WildWater* est une application en ligne de commande conçue pour analyser et exploiter un fichier de données volumineux décrivant un réseau de distribution d'eau potable. Elle permet à l'utilisateur de lancer différents traitements via un script Shell, qui constitue le point d'entrée du programme et assure la gestion des paramètres fournis.

L'architecture de l'application repose sur une séparation claire des responsabilités : le script Shell dirige l'exécution des traitements, tandis qu'un programme écrit en langage C réalise les calculs complexes afin de garantir des performances satisfaisantes malgré la taille importante du fichier de données.

Deux fonctionnalités principales ont été développées.

1. Génération d'histogrammes des usines de traitement

La première fonctionnalité consiste à produire des histogrammes permettant d'analyser les performances des usines de traitement de l'eau. Le programme est capable de calculer, selon le choix de l'utilisateur, les indicateurs suivants :

- la **capacité maximale** annuelle de traitement des usines ;

- le **volume total d’eau captée** par les sources alimentant chaque usine ;
- le **volume d’eau réellement traité**, après prise en compte des pertes dues aux fuites.

Les résultats de ces calculs sont d’abord exportés dans des fichiers intermédiaires au format CSV ou DAT, structurés et triés par identifiant d’usine. Ces fichiers sont ensuite exploités par le script Shell afin de générer automatiquement des graphiques représentant :

- les 50 usines de plus faible capacité ;
- les 10 usines de plus grande capacité.

Ces visualisations permettent une lecture synthétique et comparative des performances du réseau de traitement.

2. Calcul des pertes d’eau dans le réseau

La seconde fonctionnalité de l’application permet d’évaluer le **volume total d’eau perdu** par une usine donnée sur l’ensemble de son réseau de distribution aval. À partir de l’identifiant de l’usine fourni par l’utilisateur, le programme parcourt l’ensemble des tronçons du réseau, depuis la sortie de l’usine jusqu’aux usagers finaux.

Les pertes sont calculées en tenant compte des pourcentages de fuites sur chaque tronçon, en supposant une répartition équitable des flux d’eau entre les différents nœuds enfants. Le volume total de fuites est ensuite cumulé et exprimé en millions de mètres cubes par an.

Si l’identifiant de l’usine fourni ne correspond à aucune usine présente dans le fichier de données, le programme retourne la **valeur -1**, afin de distinguer ce cas d’erreur d’une situation où aucune fuite n’est constatée. Les résultats des calculs, y compris les éventuelles valeurs d’erreur, sont enregistrés dans un fichier historique pour assurer le suivi des analyses effectuées.

III- Méthodologie de Travail

Le développement de l’application *C-WildWater* a été mené selon une méthodologie progressive, organisée sur plusieurs semaines. Cette organisation a permis de structurer le travail, de valider chaque étape avant de passer à la suivante et d’assurer une bonne coordination au sein du groupe.

1. Répartition des tâches au sein du groupe

Afin d’optimiser l’efficacité du travail et de tirer parti des compétences de chacun, les tâches ont été réparties de la manière suivante :

- Shahd :
    - conception et implémentation des **structures** de données en langage C ;
    - implémentation des arbres **AVL** (usines et index des nœuds) ;
    - construction de l’**arbre de distribution** ;
    - implémentation des **calculs** (volumes, fuites) ;
    - gestion de la mémoire et optimisation des performances.

- Kenza :
    - développement du script **Shell** (`script.sh`) ;
    - gestion des arguments utilisateurs et des messages d'erreur ;
    - intégration de **GnuPlot** pour la génération des histogrammes ;
    - rédaction du **Makefile** ;
    - réalisation des tests fonctionnels.
- Tenzin :
    - lecture et interprétation du **fichier CSV** ;
    - gestion des cas particuliers liés au format des données ;
    - participation à la documentation du projet (README et rapport PDF).

Cette organisation a permis de travailler en parallèle sur les différentes composantes du projet tout en assurant une cohérence globale grâce à des échanges réguliers et à l'utilisation du dépôt GitHub.

2. Planning de réalisation

Le projet s'est déroulé sur trois semaines, chacune correspondant à une phase distincte du développement.

Semaine 1 : Analyse, structures de données et lecture du fichier CSV

Jour	Tâches principales
Jour 1	Analyse du sujet et des exigences Étude du format CSV Conception des structures de données
Jour 2	Rédaction du Makefile Initialisation GitHub Squelette du programme C
Jour 3	Implémentation de l'AVL Tests unitaires
Jour 4	Lecture du CSV Stockage des données Tests sur extrait
Jour 5	Construction de l'arbre de distribution Gestion des relations parent/enfants
Jour 6	Gestion des cas particuliers Corrections Commit intermédiaire
Jour 7	Documentation Préparation des histogrammes Tests finaux

Semaine 2 : Calculs et génération des fichiers de sortie

Jour	Tâches principales
Jour 8	Calcul des volumes par usine (capacité max, volume capté, volume réel)
Jour 9	Tri des identifiants d'usines Génération des fichiers de données histogrammes
Jour 10	Implémentation du calcul des fuites Parcours DFS de l'arbre aval
Jour 11	Application des pourcentages de fuites Calcul du volume cumulé (M.m³)
Jour 12	Développement du script Shell Gestion des commandes histo et leaks
Jour 13	Gestion des erreurs Passage des paramètres au programme C
Jour 14	Tests sur jeu de données réduit Commit GitHub

Semaine 3 : Intégration, visualisation et finalisation

Jour	Tâches principales
Jour 15	Génération des images PNG avec GnuPlot Intégration des scripts GnuPlot
Jour 16	Création des graphiques (50 plus petites / 10 plus grandes usines)
Jour 17	Gestion du fichier historique des fuites (.dat) Optimisation mémoire
Jour 18	Gestion des codes de retour Messages d'erreur du programme C
Jour 19	Tests finaux avec le fichier complet (~500 Mo) Analyse des performances
Jour 20	Rédaction du README.md Finalisation du rapport PDF
Jour 21	Préparation du dossier tests Commit final Ouverture du dépôt GitHub

IV- Analyse des Défis Rencontrés et des Solutions Apportées

1. Problème majeur de performances

La principale difficulté rencontrée lors du développement du projet a été le temps d'exécution excessif du programme. Lors des premiers tests sur le fichier CSV complet (environ 500 Mo), le programme pouvait dépasser deux minutes d'exécution, ce qui ne respectait pas les objectifs de performance attendus.

Ce ralentissement était principalement dû :

- au parcours intégral du fichier CSV à chaque traitement ;
- à des opérations trop lente sur les chaînes de caractères ;
- à une gestion inefficace de la lecture et du filtrage des données.

Pour résoudre ce problème, plusieurs optimisations ont été mises en place, à la fois dans le programme en langage C et dans le script Shell.

2. Optimisation du traitement des chaînes de caractères en C

Une grande partie du temps d'exécution était liée au traitement des lignes du fichier CSV. Afin d'améliorer les performances, nous avons étudié et utilisé plusieurs fonctions de la bibliothèque standard <string.h>.

- **strtok**
  Cette fonction a été utilisée pour découper chaque ligne du fichier CSV en colonnes, en utilisant le caractère ; comme séparateur.
  Dans le cadre de notre projet, strtok permet de parser efficacement les lignes sans avoir à recopier inutilement les chaînes, ce qui réduit le nombre d'allocations mémoire et améliore la vitesse de traitement.

- **strcspn**
  Cette fonction a servi à nettoyer les chaînes de caractères, notamment pour supprimer les retours à la ligne (\n) en fin de champ. Cela permet d'éviter des erreurs lors des comparaisons d'identifiants et garantit que les chaînes stockées en mémoire sont correctement formatées.

- **strcmp**
  Il a été utilisée pour comparer les identifiants (types de nœuds, identifiants d'usines, choix de traitement). Son utilisation est nécessaire pour déterminer rapidement le type de ligne (source, usine, stockage, etc.) et orienter le traitement sans recourir à des comparaisons plus coûteuses.

- **stderr**
  Les messages d'erreur du programme C ont été envoyés vers la sortie d'erreur standard (stderr). Cela permet de séparer clairement les messages d'erreur des données de sortie, ce qui est nécessaire pour que le script Shell puisse rediriger correctement les résultats tout en

conservant des logs (messages enregistrés par un programme pour garder une trace de ce qu'il fait pendant son exécution) exploitables.

Ces optimisations ont permis de réduire significativement le temps de traitement et d'améliorer la robustesse globale du programme.

3. Utilisation des pipes dans le script Shell

Une autre optimisation majeure a été l'utilisation des **pipes (|)** dans le script Shell afin d'éviter que le programme C relise systématiquement l'intégralité du fichier CSV.

Le principe consiste à :

- filtrer les lignes pertinentes directement avec **grep** (commande Unix/Linux qui sert à rechercher du texte dans un fichier) dans le script Shell ;
- envoyer le résultat via un pipe au programme C ;
- indiquer au programme C de lire depuis l'entrée standard (stdin) grâce au symbole -.

Dans notre projet :

- pour les histogrammes, seules les lignes nécessaires (sources, usines) sont envoyées ;
- pour le calcul des fuites, seules les lignes qui contiennent l'identifiant de l'usine demandée sont envoyées.

Cette technique permet :

- de réduire le volume de données traité par le programme C ;
- d'éviter des accès disque inutiles ;
- de diminuer fortement le temps d'exécution global.

La gestion du retour d'erreur via **PIPESTATUS** permet également de vérifier que le programme C s'est correctement exécuté avant de continuer le traitement.

4. Gestion du tri des données

La gestion du tri des usines a également constitué une difficulté importante. Initialement, le tri était envisagé directement en langage C, ce qui s'est révélé complexe et coûteux en temps de développement.

La solution retenue a été de :

- laisser le programme C produire des fichiers de données structurés (.csv ou .dat) ;
- laisser le tri au script Shell en utilisant la commande **sort**.

Le tri est effectué :

- numériquement (-g) ;
- sur la colonne correspondant au volume étudié ;
- avec le séparateur ;.

Une fois les données triées, les commandes **head** et **tail** permettent d'extraire facilement :

- les 50 plus petites usines ;
- les 10 plus grandes usines.

Cette séparation entre le C (calculs lourds) et le Shell (manipulation de fichiers et tri) a permis de simplifier le code tout en conservant de bonnes performances.

Ces difficultés ont permis de mieux comprendre l'impact des choix d'implémentation sur les performances, ainsi que l'intérêt de lier le langage C et les outils Shell.

V- Limitations fonctionnelles de l'application :

Malgré la mise en œuvre des fonctionnalités principales demandées dans le sujet, certaines limitations reste dans l'application *C-WildWater*, tant sur le plan fonctionnel que technique. Ces limitations sont détaillées ci-dessous.

1. Fonctionnalités non implémentées

- Histogramme "all" (bonus) :
  La génération de l'histogramme prenant la capacité maximale, le volume capté et le volume réellement traité n'a pas été implémentée.
  Le programme prend uniquement en charge les modes **max, src et real**, conformément aux fonctionnalités de base.
- Identification du tronçon le plus fuyard (bonus) :
  Lors du calcul des pertes d'eau d'une usine, l'application ne fournit pas l'identifiant du tronçon amont/aval présentant la plus grande perte en valeur absolue.
  Seul le volume total des pertes est actuellement calculé et retourné.

2. Fonctionnalités partiellement implémentées ou perfectibles

- Calcul des fuites aval :
  Le calcul des pertes d'eau repose sur l'hypothèse d'une répartition équitable des volumes entre les nœuds enfants, comme autorisé par l'énoncé.
  On a utilisé cela pour simplifier le modèle réel de distribution ce qui peut entraîner une approximation des volumes de fuites, notamment dans des sous-réseaux très ramifiés.
- Gestion des données incomplètes du CSV :
  Certaines lignes comportant des valeurs manquantes (-) sont ignorées ou traitées de manière simplifiée. Même si cela n'entraîne pas de crash du programme, certaines données partielles peuvent ne pas être prises en compte dans les calculs finaux.

VI-Bilan final :

Le projet *C-WildWater* nous a permis d'appliquer les notions d'algorithmique, de programmation en langage C et de scripting Shell à travers un sujet réaliste portant sur le traitement de données massives.

L'application repose sur une architecture combinant un script Shell comme point d'entrée et un programme C pour les gros calculs ce qui assure à la fois performance et séparation claire des responsabilités. L'utilisation de structures de données adaptées, notamment les arbres AVL et les listes chaînées, a renforcé notre compréhension des enjeux d'optimisation.

Les fonctionnalités principales (génération d'histogrammes et calcul des pertes d'eau par usine) ont été implémentées et validées. Certaines simplifications ont été faites afin de respecter les contraintes de temps, et sont clairement documentées dans les limitations fonctionnelles.

Ce projet a également mis en évidence l'importance de la robustesse du code, de la gestion de la mémoire, de la documentation, ainsi que du travail collaboratif via Git.
Malgré l'absence de certaines fonctionnalités avancées, le projet constitue une base solide et fonctionnelle, susceptible d'évolutions futures.