v1.1 Projet C-Wildwater

FILIERE préING2 • 2025-2026
AUTEURS E.ANSERMIN – R.GRIGNON
E-MAILS eva.ansermin@cyu.fr – romuald.grignon@cyu.fr

DESCRIPTION GENERALE

- Ce projet vous demande de réaliser un programme permettant de faire la synthèse de données d'un système de distribution d'eau.
- Pour cela, vous avez à votre disposition un fichier *.csv* contenant un vaste ensemble de données détaillant la distribution d'eau en France, depuis les sources de captation, à travers les usines de traitement, les zones de stockage, jusqu'aux consommateurs finaux.

- Les données fournies sont factices mais les ordres de grandeurs sont équivalents à celles d'un pays comme **1/3 de celui de la France** métropolitaine, en termes de nombre d'installations et usagers, volumes de captation, traitement, stockage et acheminement.

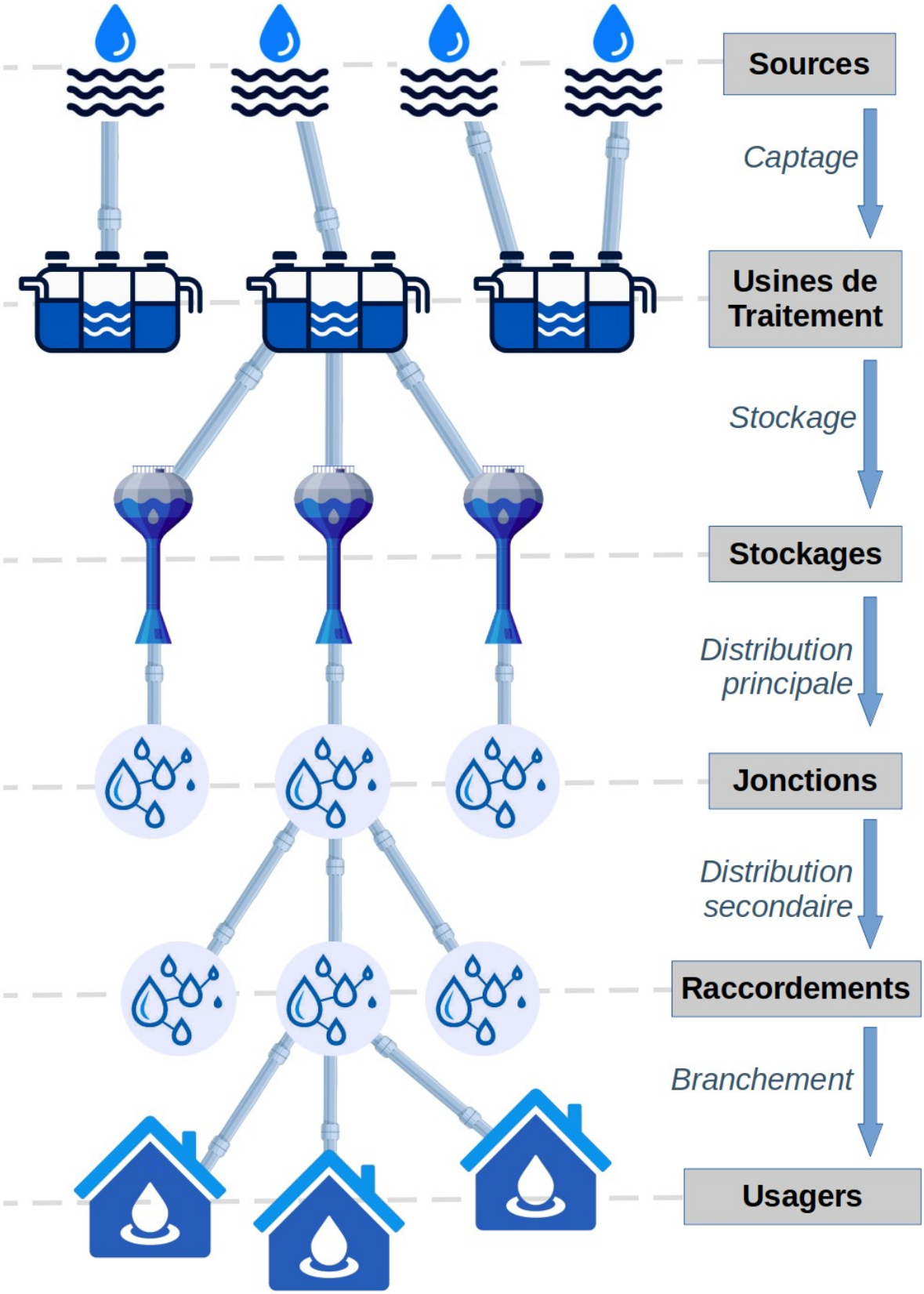
- Votre travail est de filtrer et traiter ces données au moyen d'un script shell et d'un programme écrit en langage C.

DISTRIBUTION D'EAU

- En France, la quantité d'eau captée dans les différentes sources, et qui doit servir pour alimenter les usagers en eau potable, est d'environ :
  $5{,}5 \times 10^9\ m^3$ ($1m^3$ éq. à 1000 litres)

- Parmi cette quantité d'eau prélevée, une partie est perdue dans le transport entre les différents acteurs (conduites, raccordement, usines de traitements, stockages, …). Le pourcentage d'eau perdue est non négligeable et de l'ordre de 20 % sur l'ensemble du réseau.
  Pour ce projet, la topologie fournie a été simplifiée pour des raisons pédagogiques mais les ordres de grandeurs des quantités d'eau qui transitent d'un acteur à un autre sont respectées.

- Le circuit classique de l'eau potable qui parvient au logement d'un usager est illustré sur le schéma page suivante :


Sources
Captage
Usines de Traitement
Stockage
Stockages
Distribution principale
Jonctions
Distribution secondaire
Raccordements
Branchement
Usagers

➢ SOURCES

Que l'eau provienne d'un champ captant, ou d'une petite source naturelle, l'ensemble des points de captages sont dénommés ici *« sources »*. On en recense plus d'une trentaine de milliers en France avec chacune son volume annuel qui diffère suivant la taille et les conditions météorologiques chaque année. Ici chaque source est reliée à **une seule** usine de traitement pour simplifier la topologie.

➢ USINES DE TRAITEMENTS

Chaque installation technique qui doit effectuer un traitement pour rendre l'eau potable est dénommée *« usine de traitement »*. Son nombre est d'environ la moitié du nombre de sources. Chacune ayant sa capacité maximale de traitement annuel.

Chaque usine peut traiter **plusieurs** sources, et est reliée en sortie à **plusieurs** espaces de stockage. Le nombre de sources et d'espaces de stockage reliés à une usine n'est pas limité spécifiquement.

➢ ESPACES DE STOCKAGES

Classiquement on pense aux chateaux d'eau pour les espaces de stockage mais il existe des réservoirs enterrés, semi-enterrés, des cuves surélevées plus petites que les chateaux d'eau, des bassins inclus aux usines de traitement, …. Ces espaces varient en taille et en nombre en fonction de la densité de population. On peut simplifier le nombre de points de stockage comme équivalent au nombre de sources même si il n'y a pas de corrélation directe entre les deux.

Un espace de stockage reçoit l'eau d'une seule usine, et distribue l'eau vers une jonction unique.

➢ DISTRIBUTION

Par ce terme, on regroupe toutes les infrastuctures qui distribuent l'eau depuis les espaces de stockages vers les usagers. Cette distribution est loin d'être directe, on a donc simplifié la topologie en 3 parties :

- **distribution principale :** qui achemine l'eau des espaces de stockage vers des branchements/répartiteurs dénommés ici *« jonctions »*.
- **distribution secondaire :** qui achemine l'eau dans les différents quartiers/arrondissements, au plus près des usagers vers des entités dénommées ici *«raccordements »*
- **branchement** : qui permet de relier le réseau avec le compteur d'eau de l'usager

➢ USAGERS

Le nombre de compteurs d'eau en France est de plus d'une vingtaine de millions environ. Les usagers ne sont pas décrits dans ce projet : seuls les tronçons de raccordement seront utilisés par votre programme.

FORMAT DES DONNEES

- Le fichier CSV fourni contient toutes les informations de chaque tronçon (captage, stockage, distribution principale et secondaire, branchement), ainsi que les informations de chaque usine de traitement (capacité maximale annuelle de traitement de l'eau).

  Comme ce fichier représente environ 1/3 de la topologie de distribution d'eau de la France, le nombre de lignes peut monter jusqu'à plus de 8 millions. Le fichier pèse plus de 500Mo.
  Il n'est donc pas possible de traiter ce fichier manuellement, même avec l'aide d'un tableur. Un programme informatique pour fournir les données demandées sera donc nécessaire.

- Chaque ligne du fichier contient 5 colonnes, séparées par le caractère point-virgule ' ; ' et représente un tronçon du graphe de distribution.
  - La colonne #2 contient l'identifiant unique de l'un des acteurs de la distribution. C'est l'identifiant '***amont***' du tronçon.
  - La colonne #3 contient l'identifiant unique de l'acteur '***aval***' du tronçon actuel.
  - La colonne #4 contient le volume d'eau qui sort de l'acteur '***amont***' du tronçon (si c'est une source, alors on parle d'un voume d'eau captée, si c'est une usine, alors on parle d'un volume maximal de traitement).
  - La colonne #5 contient un pourcentage de fuites dans ce tronçon. Il faudra connaître le volume d'eau qui passe dans ce tronçon pour récupérer le volume des fuites.
  - La colonne #1 contient l'identifiant unique de l'usine qui a traité l'eau potable passant dans ce tronçon.

- Certaines lignes ne possèdent pas toujours ces 5 valeurs. Une valeur manquante est indiquée par un tiret '-'.

- Détail des différentes lignes du fichier :

SOURCE → USINE (*captage*)

```
-;
Spring #MQ001991L;
Facility complex #RH400057F;
20892;0.997
```

On retrouve l'identifiant unique de la source en 2ère colonne, puis l'identifiant unique de l'usine en 3ème colonne.
Ensuite on retrouve la quantité annuelle captée (**en milliers** de m^3), ainsi que le pourcentage de fuites dans ce tronçon (en %).
La colonne 1 n'est pas utilisée.
Il n'existe qu'une ligne comme celle-ci par source. Plusieurs de ces lignes peuvent indiquer une même usine car plusieurs sources peuvent être traitées par la même usine.

USINE

```
-;
Facility complex #RH400057F;
-;
4749292;-
```

Ces lignes décrivent l'usine (donc un nœud du graphe contrairement à toutes les autres lignes qui décrivent des arêtes)
On retrouve l'identifiant unique de l'usine en 2ère colonne,
et sa capacité maximale de traitement en 4ème colonne (en milliers de m^3).
Les colonnes 1, 3 et 5 ne contiennent pas d'information.
Il n'existe qu'une seule ligne comme celle-ci par usine.

USINE → STOCKAGE (*stockage*)

```
-;
Facility complex #RH400057F;
Storage #13178;
-;3.777
```

On retrouve l'identifiant unique de l'usine en 2ère colonne, puis l'identifiant unique de l'espace de stockage en 3ème colonne.
En colonne 5 se trouve le pourcentage de fuites dans ce tronçon (en %).
Les colonnes 1 et 4 ne sont pas utilisées.
Il n'existe qu'une seule ligne comme celle-ci par espace de stockage. Plusieurs de ces lignes peuvent contenir le même identifiant d'usine puisqu'une usine peut avoir plusieurs espaces de stockages en sortie.

STOCKAGE → JONCTION (distribution principale)

```
Facility complex #RH400057F;
Storage #13178;
Junction #TM12995S;
-;3.308
```

On retrouve l'identifiant unique de l'espace de stockage en 2ère colonne, puis l'identifiant unique de la jonction en 3ème colonne.
En colonne 5 se trouve le pourcentage de fuites dans ce tronçon (en %).
La colonne 1 contient l'identifiant unique de l'usine qui a traité l'eau passant dans ce tronçon. La colonne 3 n'est pas utilisée.
Il n'existe qu'une seule ligne comme celle-ci par espace de stockage (ou par jonction, car il n'y a qu'un seul tronçon de distribution principale entre 1 stockage et 1 jonction)

JONCTION → RACCORDEMENT (distribution secondaire)

```
Facility complex #RH400057F;
Junction #TM12995S;
Service #LD204279R;
-;3.735
```

On retrouve l'identifiant unique de la jonction en 2ère colonne, puis l'identifiant unique du raccordement en 3ème colonne.
En colonne 5 se trouve le pourcentage de fuites dans ce tronçon (en %).
La colonne 1 contient l'identifiant unique de l'usine qui a traité l'eau passant dans ce tronçon. La colonne 3 n'est pas utilisée.
Il n'existe qu'une seule ligne comme celle-ci par raccordement. Plusieurs de ces lignes peuvent contenir le même identifiant de jonctions puisqu'une jonction peut avoir plusieurs raccordements en sortie.

RACCORDEMENT → USAGER (branchement)

```
Facility complex #RH400057F;
Service #LD204279R;
Cust #TJ8256677Z;
-;7.273
```

On retrouve l'identifiant unique du raccordement en 2ère colonne, puis l'identifiant unique de l'usager en 3ème colonne.
En colonne 5 se trouve le pourcentage de fuites dans ce tronçon (en %).
La colonne 1 contient l'identifiant unique de l'usine qui a traité l'eau passant dans ce tronçon. La colonne 3 n'est pas utilisée.
Il n'existe qu'une seule ligne comme celle-ci par usager. Plusieurs de ces lignes peuvent contenir le même identifiant de raccordement puisqu'un raccordement peut avoir plusieurs usagers en sortie.

- Dans ce fichier, il est important de noter que pour **chaque ligne**, quelle que soit l'endroit où elle se situe dans le fichier, l'identifiant '*amont*' a obligatoirement été décrit plus tôt comme identifiant '*aval*'.
  ex : une ligne qui décrit un tronçon avec une jonction en '*aval*', est obligatoirement située plus tôt dans une ligne qui décrit ce même tronçon avec la jonction en *'amont'*:

```
Storage #13178;Junction #TM12995S;-;3.308
…
Junction #TM12995S;Service #LD204279R;-;3.735
```

- Cela signifie que tous les ancêtres d'un nœud de notre arbre sont forcément décrits en amont dans le fichier : la première ligne de données du fichier est donc un tronçon « *source → usine* » et la dernière ligne un tronçon « raccordement → usager ».
  Attention toutefois : « en amont » ne signifie pas que le parent est situé **immédiatement** avant le nœud fils. Globalement les données sont mélangées entre elles et vous ne pouvez pas préjuger de l'ordre des lignes du fichier. Vous pouvez avoir une usine décrite au début du fichier, et son premier nœud de stockage décrit très loin après.

OBJECTIFS

- Le but de ce projet est de créer un programme Shell qui va réaliser des opérations de calculs sur le fichier de données. Le script Shell est un langage interprété qui sera beaucoup trop lent pour effectuer énormément de calculs, et pour accroitre les performances, ce script appellera un programme en langage C qui fera les traitements et renverra ses résultats.

Histogramme de performance des usines

Chaque usine de traitement de l'eau voit passer chaque année un volume considérable d'eau captée depuis les différentes sources. Il est demandé dans ce projet de créer un histogramme avec comme valeur pour chaque usine, une seule des informations suivantes (au choix, **exclusif**). L'utilisateur choisira ce traitement en entrant des arguments différents dans le script Shell (cf. §Script Shell) :

- Capacité maximale de traitement :
  C'est la quantité d'eau que l'usine **peut traiter** par an. L'unité fournie en sortie doit être en millions de m^3 ($M.m^3$).
- Volume total capté depuis les sources :
  C'est la somme des volumes d'eau **prelevés** par an par toutes les sources qui alimentent une usine. L'unité fournie en sortie doit être en millions de m^3. Cette somme est normalement inférieure à la capacité maximale de traitement.
- Volume total traité par les usines :
  C'est la somme pondérée des volumes captés par les sources multipliés par le pourcentage d'eau qui parvient aux usines. Cette somme est inférieure au volume total capté car il y a des fuites dans les tronçons entre les sources et les usines. L'unité sera également en millions de m^3.

Pour générer cet histogramme, le programme devra d'abord générer un fichier de données au **format CSV** contenant une première ligne indiquant les colonnes utilisées (identifiant de l'usine, volumes, …) ainsi que les données. Ce fichier sera stocké sur le disque dur avec un nom qui permet d'identifier l'information choisie

- ex: ***vol_max.dat***, ***vol_captation.txt***, ***vol_traitement.tmp***

Quels que soient les noms des fichiers en sortie que vous choisirez, vous devrez faire en sorte que chacun des 3 fichiers possède un nom unique pour bien les distinguer les uns des autres.
Ce fichier contiendra l'ensemble des usines, triées par identifiant dans l'ordre alphabétique inverse.

A partir des données fichiers, le script va générer une image regroupant les 50 plus petites usines ET **une autre image** avec les 10 plus grandes usines (la capacité maximale est la référence), y ajouter la légende correcte, les titres et les unités, et les sauvegarder au **format PNG**. Les noms des fichiers images seront les mêmes que le nom du fichier de données généré précédemment.

Rendement de distribution d'une usine

Il y a énormément de fuites sur l'ensemble du réseau d'eau et il est demandé de récupérer des éléments quantitatifs précis pour évaluer les coûts de maintenance. Pour cela on souhaite obtenir le **volume d'eau perdue** par une usine dans **tout son réseau aval**.

Votre programme prendra en **paramètre** un **identifiant unique** d'usine de traitement, va parcourir l'ensemble des tronçons situés en aval, jusqu'aux usagers finaux, et faire la somme des pertes rencontrées dans chaque tronçon.

- On part du principe que pour chaque nœud de distribution, la quantité d'eau qui le traverse est répartie équitablement par les nœuds enfants.

- La sortie attendue est une valeur cumulée des **volumes d'eau potable perdus** (volumes **après traitement** de l'usine) en millions de m^3 ($M.m^3$).
  Si l'identifiant de l'usine n'est pas trouvé, le résultat retourné devra être la valeur -1 (pour ne pas confondre avec un 0 qui signifierait qu'il n'y a strictement aucune fuite).

- Un fichier contenant l'historique des rendements calculés devra être complété avec l'identifiant de l'usine choisie précédemment et son volume de pertes. Si le fichier n'existe pas, il devra être créé. Si il existe déjà, une nouvelle ligne sera rajoutée. Le nom du fichier sera suffixé '***.dat***'.

SCRIPT SHELL

- Votre application sera constituée d'un script shell qui va séquencer toutes les actions demandées par l'utilisateur : c'est donc le point d'entrée du projet.
  Ce script Shell va prendre en arguments plusieurs valeurs permettant à l'utilisateur d'effectuer l'un des traitements décrits dans la section précédente.
  - Le tout premier argument sera le **chemin du fichier** de données

  - ***histo*** : cet argument va demander au script Shell d'effectuer un histogramme des usines comme indiqué précédemment. Cet argument prend une valeur obligatoire pour spécifier le traitement parmi :
    - ***max*** : volume maximale de traitement de l'usine
    - ***src*** : volume total capté par les sources
    - ***real*** : volume total réellement traité

    Pour ce traitement, le fichier de sortie contiendra en première ligne les noms des colonnes suivantes en fonction du traitement demandé : identifier / max volume ($k.m^3.year^{-1}$) / source volume ($k.m^3.year^{-1}$) / real volume ($k.m^3.year^{-1}$)

- ***leaks*** : cet argument va demander au script Shell d'effectuer la mesure des pertes d'une usine sur l'ensemble de son réseau aval. Cet argument prend un identifiant d'usine pour pouvoir faire le travail. Cet identifiant doit être complet et respecter la casse.
  Pour ce traitement, les noms des colonnes en sortie sont : identifier / Leak volume ($M.m^3.year^{-1}$)

- Exemples de commandes Shell correctes :
  - myScript.sh *wildwater.dat* **histo max**
  - myScript.sh *wildwater.dat* **histo src**
  - myScript.sh *wildwater.dat* **histo real**
  - myScript.sh *wildwater.dat* **leaks "Facility complex #RH400057F"**

- Toute commande incorrecte ou incomplète doit générer un message d'erreur par le script Shell et stopper le traitement.
  Ex: Une commande leaks sans identifiant est une commande incomplète.
  Ex: Une commande leaks avec un identifiant inexistant, doit générer le traitement et cet identifiant sera ajouté au fichier de sortie avec le volume -1 comme expliqué dans la section précédente.

- Toute commande avec des **arguments supplémentaires** inattendus doit générer un message d'erreur et stopper le traitement.

- Les traitements pour remplir les fichiers de sortie (histogrammes, volume de fuites) doivent être **obigatoirement** effectués par un programme écrit en **langage C**. Si ce n'est pas le cas, les points accordés sur cette partie ne pourront pas être comptabilisés dans la note finale.

- La compilation du programme C doit être vérifiée par le script Shell. Si l'exécutable n'existe pas, le script doit permettre de le recréer.
  Le code retour du programme C doit être vérifié par le script Shell pour décider de la suite du traitement.

- A la fin du script Shell, la durée totale du script doit être affichée **en millisecondes**, quel que soit le résultat obtenu en sortie (traitement correct ou erreur). Une précision supérieure à la seconde n'est pas obligatoire dans cet affichage.

PROGRAMME C

- Le but de ce programme C est de réaliser les histogrammes des usines de traitements et le calcul des pertes d'eau pour une usine donnée. **Un seul** exécutable permettra de réaliser tous ces traitements.
- Dans le cas de l'histogramme, votre programme C doit récupérer l'ensemble des sources et des usines. Un AVL prenant les identifiants de chaque usine comme valeur permettra un gain de temps considérable. Il suffira de construire cet AVL, sommer les quantité de volumes d'eau captées, acheminées et traitées, pour chaque usine, afin d'être capable de renvoyer l'ensemble des informations.
- Dans le cas du calcul de fuites, il faudra très certainement construire un arbre classique dont le nombre d'enfants n'est pas connu à l'avance : une liste chainée des nœuds enfants pourrait être une proposition suffisante pour répondre à la problématique.
  Pour retrouver le noeud parent lors de l'ajout d'un nouveau nœud, faire une recherche exhaustive sera très mauvais en terme de complexité temporelle. Un AVL supplémentaire contenant l'ensemble des noeuds déjà connus permettra de connaître avec une complexité logarithmique, l'adresse du noeud parent et résoudre ce problème.
- A chaque fois que le projet vous demnde d'implémenter un AVL, si une autre structure est utilisée à la place, les points relatifs à cette fonctionnalité ne seront pas comptabilisés.
- Le format des données d'entrée du programme C est laissé libre. A vous de voir si vous passez tout le fichier de données, ou si vous voulez le filtrer (lignes / colonnes) à l'aide du script Shell avant de passer les informations au programme C.
- Le format des données de sortie du programme C est également laissé libre du moment qu'il vous permet de récupérer les informations nécessaires à votre traitement.
- Le programme C devra retourner un code d'erreur avec une valeur strictement positive si un problème est rencontré, et 0 sinon. Ce programme ne devra jamais s'arrêter de manière inattendue : c'est à vous de bien vérifier que toutes les données que vous traitez sont correctes. Si vous détectez un problème, vous devez stopper le programme et renvoyer un code d'erreur.
  Le code du programme C devra donc être robuste.
- Il vous est également demandé de limiter au mieux la taille mémoire utilisée. Pour ce faire, vous devrez définir des variables dans vos structures ayant le moins d'empreinte mémoire possible, tout en garantissant le fonctionnel demandé bien sûr.
  De plus toutes les allocations dynamiques doivent être libérées avant de terminer le programme C correctement. Par contre si le programme C rencontre une erreur, il n'est pas demandé de libérer toute la mémoire explicitement.
- Enfin, la compilation du programme C devra se faire avec l'utilitaire 'make' en utilisant un 'Makefile'. La seule instruction de compilation attendue par votre script Shell est l'appel à l'exécutable 'make'. Pas d'appel direct à gcc depuis votre script.

BONUS

- Pour la génération d'histogramme, la commande *myScript.sh* **histo all** pourra générer un histogramme cumulé des 3 valeurs (capacité, captage, traitement). Chaque ligne du fichier de sortie contiendra donc 4 colonnes au lieu de 2.
  Chaque barre de l'histogramme contiendra 3 valeurs imbriquées pour visualiser avec 3 couleurs différentes, en vert la quantité de volume encore possible de traiter (max – source) et en rouge la quantité de volume perdu après captage (source – real) et en bleu la quantité réellement fournie en sortie de l'usine. L'unité sera la même que définie précédemment.

- Pour la génération des fuites par usine, un bonus demandé est de fournir les identifiants amont et aval du tronçon qui perd le plus d'eau (non pas en pourcentage mais en valeur absolue : en effet certains tronçons proches des usagers perdent parfois de l'eau avec un pourcentage à 2 chiffres, mais en quantité absolue cela reste négligeable comparé à des fuites sur de grosses conduites de distribution). L'unité sera la même que définie précédemment.

- La livraison de bonus ne sera comptabilisée que si les fonctionnalités de base ont été implémentées (peut importe si elles sont totalement fonctionnelles, il faut un minimum de traitement pour chaque fonctionnalité avant que votre groupe s'attèle aux fonctionnalités bonus.

CRITERES DE NOTATION

- Le rendu du travail sera un **lien github** menant au projet. Inutile d'envoyer les fichiers par email/Teams ou toute autre moyen : le seul livrable attendu et qui sera évalué sera le lien du dépôt git dans lequel doivent se trouver tous les fichiers de votre projet. Avant la date de rendu vous pouvez configurer ce dépôt en « privé » pour ne pas laisser d'autres personnes vous plagier.
  A la date de rendu, ce dépôt devra être **visible publiquement** pour que vos chargés de projet puissent y accéder librement. Tout retard d'accès à ce dépôt public aura un impact négatif sur votre note finale.

- **Aucun rendu en dehors du dépôt ne sera pris en compte !**

- Le dépôt de code contiendra, en plus des fichiers de code, un fichier texte **ReadMe** contenant les instructions pour compiler et pour utiliser votre application.

- Il contiendra aussi un document au format **PDF** présentant la répartition des tâches au sein du groupe, le planning de réalisation, et les limitations fonctionnelles de votre application (la liste de ce qui n'est pas implémenté, et/ou de ce qui est implémenté mais qui ne fonctionne pas correctement/totalement).

- Il vous est demandé d'effectuer une livraison sur le dépôt 1 fois **après chaque séance de TD au minimum**, même si le dépôt de code n'est pas fonctionnel.
  Si vous avez des évolutions au niveau de votre code entre deux séances, vous devrez avoir effectué **un commit la veille** de la séance, pour bien fixer l'historique de vos modifications, et pour pouvoir travailler dessus avec votre chargé de TD.
  De fait, au minimum il y aura un commit juste avant et juste après la séance obligatoirement. Ces dates de commit seront utilisées pour votre évaluation.

- Si vous avez des modifications, même non fonctionnelle elles ne doivent pas rester plusieurs jours sans être stockées sur le dépôt de code, et éviter ainsi une perte catastrophique dans l'hypothèse où votre machine viendrait à tomber en panne. Il est de **votre responsabilité** d'archiver régulièrement vos modifications sur le dépôt. **Aucune perte de code ne pourra être prise en compte** si vous n'utilisez pas votre dépôt.
  Cela vous forcera également à développer en équipe avec un dépôt central que vous vous partagez et qui contient les dernière modifications du projet.

- Votre rendu contiendra des exemples d'exécution de votre application. Vous mettrez dans un dossier '*tests*', des images, des fichiers qui ont été générés avant votre rendu, et vous présenterez ces résultats dans le document PDF. Ces éléments doivent donc être reproductibles par vos chargés de TD.

- Le rendu est un travail de groupe : si des similitudes entre groupes sont trouvées, et/ou si des exemples disponibles sur Internet sont découverts sans être sourcés, une procédure de fraude à un examen pourra être envisagée. Le but pédagogique de ce projet est que vous réalisiez par vous-même ce programme, et que vous maitrisiez l'ensemble du code fourni.

- L'équipe pédagogique se réserve le droit de convoquer un groupe après la date de rendu afin de discuter du contenu du code fourni et vérifier que l'équipe est bien à l'origine de cette implémentation. Typiquement si du code paraît d'un niveau technique beaucoup trop élevé par rapport à ce que l'on pourrait attendre classiquement d'un groupe d'étudiants, il est probable que vous soyiez amenés à expliquer ce code devant l'équipe pédagogique. En cas de non maitrise de ce code, une pénalité sur la note finale pourra être appliquée.

- Le code en langage C sera séparé en **modules** (fichiers .c et .h, sous-dossiers). Il ne faut pas que votre programme C soit d'un seul tenant.
- Un fichier **Makefile** sera présent et il permettra de compiler l'exécutable. La première cible permettra de compiler le projet. Ce fichier inclura entre autres une cible '**clean**' qui permet d'effacer les fichiers générés.

- Votre code sera **commenté** (modules, fonctions, structures, constantes, …) et correctement **indenté**. Ces commentaires ne doivent pas être des commentaires ligne à ligne comme le ferait des outils de type LLM: cela n'a aucun d'intérêt professionnel : les commentaires donnent des informations fonctionnelles par blocs de code.
- Les **symboles** du code (variables, fonctions, types, fichiers, …) seront dans la **même langue** que les **commentaires** (soit tout en anglais, soit tout en français, mais pas de mélange entre les langues utilisées).
- Le programme C et le script Shell doivent respecter toutes les consignes décrites dans ce document.

- Le programme C et le script Shell ne doivent en aucun cas générer d'erreur inattendue. Il ne doit y avoir aucune **erreur de segmentation**, **erreur de syntaxe**, ou de nom de **commande inconnue**, etc.
  Ce critère sera extrêmement punitif sur la note finale : il est donc de votre responsabilité de tester votre programme correctement pour pouvoir corriger ces situations avant la livraison.
- Si une erreur est détectée par votre programme/script, un message d'erreur doit s'afficher pour indiquer la cause à l'utilisateur, et un code retour avec une valeur strictement positive doit être retournée.

- Les structures allouées temporairement dans votre programme doivent être désallouées explicitement avant la fin. La quantité de mémoire non libérée à la fin de l'exécution, sera évaluée. Pensez à appeler la fonction ***free(…)*** quand il faut !
- De plus, la quantité de mémoire vive consommée par votre programme, sera également notée : pensez donc à limiter votre empreinte mémoire. Attention tout de même à faire en sorte que votre programme fonctionne : l'optimisation mémoire reste un plus. La première étape étant de réaliser un projet fonctionnel.

- Vous disposez d'un fichier de données CSV qui est figé. Il est donc possible pour un groupe d'étudiants de « coder en dur » les résultats attendus. Pour éviter ce cas de triche, il est possible que l'évaluation de votre programme se fasse avec un fichier de données CSV différent du votre (mais similaire en terme de structure, de taille, …). Pensez donc à faire un programme véritablement générique pour éviter une mauvaise surprise lors de l'évaluation.

RESSOURCES UTILES

GitHub
- site Web : https://github.com/

Format CSV
- site Web : https://fr.wikipedia.org/wiki/Comma-separated_values

GnuPlot
- site Web : http://gnuplot.info/