

## Complément Projet (3)

Vous disposez maintenant de la [donnée](#), de la [démonstration](#), d'un [fichier source de départ](#) et de [fichiers de tests](#).

### Par quoi commencer ? Comment s'organiser ?

#### Bonnes pratiques :

- d'abord prendre le temps de lire la donnée *tout en exécutant le programme de démonstration* sur les fichiers tests fournis.
- N'hésitez pas à commencer à *créer vos propres fichiers de tests* simplement en les écrivant avec `geany`.
  - Le programme de démonstration devrait répondre à la plupart de vos questions sur le comportement attendu pour toutes sortes de scénarios.
- **Tenez un journal** documentant l'avancement du projet, les questions en suspens, les choix effectués, les tests effectués, les bugs rencontrés. Ce document sera très utile dans vos interactions avec les assistants en cas de blocage.
- Continuez à faire les séries ; elles apportent une expérience précieuse et parfois même des éléments directement ré-utilisables dans le projet.

**Mauvaise pratique :** commencer à coder tête baissée.

#### Étapes conseillées :

- identifier la nature des données manipulées et comment elles sont organisées. Étant donné l'avancement du cours, une approche exploitant des **vector** est suffisante et sera acceptée pour le rendu final.
- identifier les **actions** sur ces données, soit selon la stratégie top-down ou bottom-up.
- dans votre journal, dessiner une vue d'ensemble de la résolution en déléguant les sous-problèmes à des fonctions jusqu'à atteindre un niveau suffisamment simple.

Remarque : dans le programme final la taille d'une fonction ne devra pas dépasser une taille écran (**40** lignes) et une ligne ne doit pas dépasser **87** caractères.

- effectuer la mise au point fonction par fonction, en validant chaque fonction avec un petit programme de test, avant de passer à la suivante.

C'est une erreur grossière d'écrire tout le code source d'un seul coup puis d'appuyer sur le bouton « build »...

- Vous pouvez commencer à tester la fonction `main()` en utilisant des fonctions vides ou très simplifiées (appelées « *stub* ») pour les fonctions traitant les sous-problèmes et qui ne sont pas encore finalisées. Cela permet de valider certains aspects comme la lecture des données et la transmission de données entre les différents niveaux de fonctions.