ChenYI-TECH

Rapport de projet de programmation en langage C

Abbadi Anass
Eish Ahmed
Ferrand Hugo

PréING1 – CY Tech
Année académique 2024–2025

Table des matières

1 Présentation générale du projet

Le projet *ChenYI-TECH* a été développé dans le cadre du module de programmation encadrée en première année de la prépa intégrée à CY Tech pour l'année 2024–2025. Son objectif était de mettre en œuvre une application complète en langage C, capable de gérer un chenil virtuel à travers une interface en ligne de commande. L'application devait permettre d'ajouter des animaux, de les rechercher selon différents critères, de les supprimer (dans le cadre d'une adoption), et d'afficher des statistiques variées, comme les besoins alimentaires ou l'inventaire par tranches d'âge.

Ce projet nous a permis d'appliquer les connaissances vues en cours de manière concrète, tout en nous confrontant aux problématiques réelles de conception logicielle : modularité du code, gestion des erreurs, interaction utilisateur robuste et maintenabilité.

2 Constitution du groupe et répartition des tâches

Notre groupe était composé de trois étudiants : Anass, Hugo et Ahmed. Dès le début, chacun s'est orienté naturellement vers les tâches qui lui correspondaient le mieux. Anass s'est principalement chargé de la gestion des fichiers, en s'occupant de la lecture, l'écriture et la sauvegarde des données des animaux. Il a aussi mis en place le système d'identifiants uniques. Hugo s'est quant à lui consacré à l'interface utilisateur, aux menus, aux interactions, ainsi qu'à la gestion du buffer d'entrée et à l'intégration des couleurs ANSI pour améliorer la lisibilité. Enfin, Ahmed s'est occupé des fonctionnalités avancées, comme le calcul des besoins alimentaires journaliers, l'inventaire par tranches d'âge, et la robustesse générale de l'application.

Cette répartition, bien que claire au départ, a évolué au cours du projet selon les besoins et les difficultés rencontrées. Nous avons travaillé en bonne intelligence, en nous entraidant régulièrement.

3 Compréhension du sujet et approche choisie

Le cahier des charges imposait plusieurs contraintes importantes. Le programme devait fonctionner exclusivement dans un terminal, gérer un maximum de 50 animaux, ne pas utiliser de variable globale, et être suffisamment robuste pour qu'aucune mauvaise saisie de l'utilisateur ne puisse provoquer un plantage.

Nous avons opté pour une structure de projet modulaire, avec un fichier `.c` et un `.h` dédiés à chaque fonctionnalité majeure : gestion des animaux, interface utilisateur, lecture/écriture des fichiers, fonctions avancées et utilitaires d'affichage. Cela nous a permis de développer et tester chaque module de manière indépendante avant de les intégrer ensemble. Pour le stockage, nous avons choisi un fichier texte classique dans lequel chaque ligne correspond à un animal, avec des champs séparés par des espaces. La simplicité de

ce format a facilité le parsing.

4 Développement du programme

Le développement a commencé par la création de la structure `Animal`, comprenant un identifiant, un nom, une espèce, une année de naissance, un poids, et un commentaire optionnel. À partir de cette base, nous avons implémenté les fonctions principales : ajout, recherche, suppression et sauvegarde.

La fonction d'ajout comprend une validation stricte de chaque champ : le nom et l'espèce doivent uniquement contenir des lettres (avec tolérance pour les apostrophes et les tirets), l'année doit être numérique et inférieure ou égale à 2025, et le poids doit être un réel strictement positif. Pour garantir la cohérence, chaque nouvel animal reçoit un identifiant auto-incrémenté calculé en fonction des identifiants existants.

La recherche permet de filtrer les animaux par nom, par espèce ou encore par tranche d'âge (jeune, adulte, senior). Un effort particulier a été fait pour que la recherche soit insensible à la casse et suffisamment tolérante sans perdre en précision.

La suppression d'un animal se fait exclusivement par identifiant, afin d'éviter les erreurs en cas de doublons dans les noms. Une fois l'identifiant saisi, un contrôle est effectué et l'animal est retiré du tableau avant de sauvegarder le nouveau fichier.

Enfin, les fonctions avancées permettent d'afficher un inventaire du chenil réparti par tranches d'âge, ainsi que de calculer les besoins alimentaires journaliers pour les chiens, chats, hamsters et autruches. Les espèces non reconnues sont listées à part dans un avertissement.

5 Difficultés rencontrées et solutions apportées

Tout au long du projet, nous avons été confrontés à de nombreuses difficultés, à la fois techniques et liées à l'ergonomie de l'interface utilisateur. Ces problèmes ont fait l'objet d'une réflexion approfondie et ont été résolus au fil du développement.

La première source d'erreurs concernait la lecture des entrées utilisateur. Le recours initial à `scanf` posait de nombreux problèmes, notamment la présence de caractères résiduels dans le tampon d'entrée (`stdin`), ce qui perturbait les interactions suivantes. Cela rendait l'interface instable, voire inutilisable. Nous avons résolu ce problème en remplaçant la plupart des `scanf` par des `fgets`, associées à une suppression explicite du saut de ligne et une vérification du contenu saisi caractère par caractère. Cela nous a permis de contrôler finement la validité des saisies tout en assurant une interaction plus fluide.

Un second problème récurrent était lié à la validation des champs saisis. Par exemple, dans la fonction `saisirAnimal`, l'utilisateur pouvait entrer des noms vides, des chaînes composées uniquement d'espaces ou même des caractères spéciaux. Ces saisies rendaient

incohérente la base de données. Pour y remédier, nous avons mis en place des boucles de validation, contrôlant que la chaîne contient au moins une lettre alphabétique, et ne comporte que des caractères autorisés (lettres, tirets, apostrophes ou espaces).

De même, lors du filtrage par nom, espèce ou tranche d'âge dans `rechercherAnimaux()`, une simple erreur de frappe comme "oui" ou "n" au lieu de "O" ou "N" pouvait casser le flux du programme. Désormais, seule une saisie exacte de "O" ou "N" (en majuscule ou minuscule) est acceptée. Un message d'erreur explicite est affiché en cas d'entrée invalide.

Le filtre par tranche d'âge a également nécessité une double validation. Initialement, un caractère ou un chiffre non prévu (ex. 5 ou une lettre) pouvait provoquer une boucle infinie ou une rupture dans la logique. Nous avons d'abord vérifié que la saisie était un entier valide, puis qu'il appartenait à l'ensemble des choix autorisés (1, 2 ou 3).

Lors de la suppression d'un animal dans `executerCommande()`, la saisie d'un caractère non numérique provoquait également des comportements erratiques. Ce cas a été sécurisé en passant par `fgets`, puis en vérifiant que la chaîne contenait uniquement des chiffres avant de la convertir.

En ce qui concerne la robustesse mémoire, nous avons été confrontés à quelques erreurs de segmentation dues à l'oubli de vérifications lors des allocations dynamiques ou de l'accès à des éléments invalides. Nous avons donc intégré des tests de nullité et des conditions limites pour sécuriser toutes les opérations critiques.

Des erreurs de compilation sont également apparues lors du linkage. Certaines fonctions étaient bien définies dans les fichiers sources, mais pas déclarées dans les headers. Cela a été corrigé en harmonisant tous les `.h` avec les `.c` correspondants.

Un problème spécifique concernait la fonction `afficherBienvenue()`, chargée d'afficher un chien en ASCII avec un message d'accueil. Elle avait bien été codée, mais n'était jamais appelée dans le `main`. Ce bug a été corrigé en appelant explicitement la fonction dès le lancement du programme, avant l'affichage du menu principal.

Enfin, dans la fonction `afficher_NourritureJournaliere()`, les espèces non reconnues n'étaient ni affichées ni prises en compte. Désormais, elles sont listées séparément à la fin avec leur nom, identifiant et espèce, pour informer l'utilisateur que ces cas ne sont pas inclus dans les calculs.

Toutes ces difficultés ont été identifiées au fur et à mesure grâce à des tests rigoureux et une communication efficace au sein de l'équipe. Leur résolution nous a permis de livrer une application stable, robuste, et agréable à utiliser.

6 Évaluation du résultat obtenu

Le programme final compile sans erreur sur macOS et Ubuntu, respecte toutes les contraintes fixées dans le cahier des charges, et propose une expérience fluide et agréable via une interface en ligne de commande. Tous les cas d'usage ont été testés, y compris les

scénarios les plus improbables, et aucun plantage n'a été observé.

Le programme est modulaire, bien commenté, et chaque fonction remplit son rôle précisément. L'utilisateur est guidé à chaque étape par des messages clairs et des codes couleur qui facilitent l'usage, même pour quelqu'un qui n'est pas habitué à ce genre d'interface.

7 Conclusion

Ce projet a été pour nous une occasion concrète de mettre en œuvre les concepts fondamentaux de la programmation en langage C. Il nous a appris à structurer un programme complet, à anticiper les erreurs, à communiquer efficacement en équipe et à produire un livrable fonctionnel et robuste. Le rendu visuel et le soin apporté à l'interface témoignent de notre implication et de notre volonté de livrer un projet soigné.

Nous sommes fiers du travail accompli, et ce projet restera pour nous un excellent souvenir de notre première année à CY Tech.