Une application sécurisée

David Guennec

18 novembre 2025

Pourquoi ce projet ?

Le but de ce projet est dual. D'une part, il vise à vous faire travailler les mécanismes de **gestion dynamique de la mémoire** et donc les pointeurs. D'autre part, on veut ici vous pousser à mettre en pratique les principes de **C défensif** vus en cours : détection des erreurs, gestion fine de la mémoire et robustesse face à l'imprévu. Le projet vous poussera également à vous remémorer tout ce dont nous avons pu parler l'année dernière en C Sécurisé I (comme le découpage logique du code, la compilation d'un projet multi-fichiers et l'utilisation de *git*).

1 Contexte

Le peuple des Cyboulettes est en panique ! Les terribles Uiteurdizuiteur sont passés à l'attaque et menacent de dominer la nation Cyboulette nuit et jour et pour toute éternité !

Face à cet effroyable danger, les Cyboulettes s'organisent. Mais la défense du pays est périlleuse et les Cyboulettes finissent tailladés de toutes parts. Leur seule solution est alors d'utiliser leur matériel médical, qui va de la potion de soin ultime ($1^{\text{ère}}$ catégorie) au pansement Ecoprix (marque déposée) selon la gravité du problème (et selon les ressources disponibles). Afin d'organiser leur effort, les Cyboulettes ont mis en place le Bureau de Gestion des Ressources de Soin (le BGRS).

Et c'est là que vous entrez en jeu. Le BGRS a besoin d'une application ultra solide pour gérer les ressources et c'est à vous de la réaliser. Cependant, il arrive que des Uiteurdizuiteur réussissent à s'infiltrer pour tenter de faire des entrées invalides sur le système. Il se peut aussi que les opérateurs Cyboulettes, par la faute du stress, commettent des erreurs. Il faut donc que l'application réussisse à gérer tous ces problèmes.

Une dernière précision : la nation n'ayant pas trop de moyens, il a été décidé que les produits destinés à la réparation des outils utilisés par les Cyboulettes était, en fait, du matériel médical. Le BGRS se retrouve donc à gérer des produits à destination des Cyboulettes et du matériel visant à réparer des outils. Attention à ne rien mélanger !

2 Les types de matériels de soin gérés par l'application

2.1 Vue d'ensemble

Chaque entrée de l'inventaire représente un **produit médical-fantasy** utilisé par les Cyboulettes pour se soigner ou réparer leur matériel. Chaque produit est un enregistrement de type `Produit`, défini par plusieurs champs dynamiques.

2.2 Gestion des données

L'inventaire est ainsi le coeur de l'application. Il s'agira d'une base de données interne qui répertoriera tous les produits disponibles et en quelle quantité. Celle-ci sera représentée par une structure de données où tout sera alloué dynamiquement sur le tas. On veut donc gérer une structure de données où il serait facile de rechercher un produit, facile d'en supprimer et facile d'en ajouter ou encore de modifier un produit existant.

Pour mener à bien ce projet, de nombreuses structures de données pourraient être considérées. Mais nous allons ici profiter du fait que vous avez vu la notion de liste chaînée pour la mettre en application.

Cette liste chaînée sera pointée par un pointeur *head* qui renverra le 1^{er} maillon de la chaîne. Si vous le souhaitez, vous pouvez rechercher le concept de liste doublement chaînée et l'implémenter. Mais cela reste entièrement optionnel.

2.3 Contenu des maillons

Chaque maillon contiendra à minima les informations décrites dans la table 1 ci-après :

Champ	Description et contraintes
`id`	Identifiant unique attribué automatiquement par la bibliothèque (`uint32_t`).
`nom`	Nom court du produit, chaîne allouée dynamiquement (max. 64 caractères).
`description`	Description libre (max. 1024 caractères).
`quantite`	Nombre d'unités disponibles en stock.
`prix_unitaire`	Prix unitaire (flottant positif).
`categorie`	Type du produit (potion, pansement, etc.).
`date_peremption`	Date d'expiration (en `time_t`), 0 si non applicable.

TABLE 1 – Contenu d'un maillon de la liste chaînée.

2.4 Catégories principales

L'inventaire gère plusieurs **catégories de produits**, chacune correspondant à un type d'objet particulier utilisé par les Cyboulettes.

- **Potions et Élixirs** (❀) : liquides magiques de soin ou de régénération.
- **Pansements et Compresses** (■) : objets de premiers secours.
- **Médicaments et Sérums** (❑) : produits chimiques plus puissants, parfois instables.
- **Outils médicaux** (✍) : instruments de soin, scalpel, seringue, défibrillateur artisanal...
- **Consommables divers** : tout ce qui ne rentre pas ailleurs (gants, masques, gel hydroalcoolique...).

2.5 Exemples de produits initiaux du BGRS

Pour tester la bibliothèque, le BGRS vous confie un petit stock de base. Ces objets sont à intégrer dans vos premiers tests (chargement, sauvegarde, recherche...).

Potion de Soin Ultime (❀)

Catégorie : Potion
Description : Restaure immédiatement 100% des PV à tout Cyboulette et guérit toute altération d'état. Légèrement pétillante.
Prix : 300.0 `Crédits`
Quantité : 3
Note : Produit mythique, souvent falsifié par les Uiteurdizuiteur.

Duct tape(■)

Catégorie : Consommable
Description : Tout équipement brisé redevient utilisable et récupère 50% de ses PV. Colle aux doigts.
Prix : 10.0 `Crédits`
Quantité : 20

Pansement Écoprix (■)

Catégorie : Pansement
Description : Un pansement basique, parfois déjà utilisé. Réduit les dégâts superficiels.
Prix : 1.0 `Crédit`
Quantité : 342
Note : 1% de chances d'avoir été empoisonné par un Uiteurdizuiteur, auquel cas l'utilisateur perd 20 PV.

Sérum de Vitalité Bêta (❑)

Catégorie : Médicament
Description : Réactif expérimental améliorant temporairement la vigueur des Cyboulettes.
Prix : 25.0 `Crédits`
Quantité : 5

WD-40 (❀)

Catégorie : Consommable
Description : Restaure immédiatement 100% des PV à tout équipement. Fait perdre 5 PV si ingéré par un être vivant.
Prix : 10.0 `Crédits`
Quantité : 20
Note : Potion hautement efficace en toute situation. Toujours en avoir dans son inventaire.

Scalpel Photonique (✍)

Catégorie : Outil
Description : Instrument de précision alimenté par énergie solaire. Peut aussi griller du pain.
Prix : 120.0 `Crédits`
Quantité : 2

Gel Hydroalcoolique divers

Catégorie : Consommable
Description : Élimine 99.99% des virus, y compris ceux imaginaires. Brûle un peu les tentacules.
Prix : 5.0 `Crédits`
Quantité : 20

2.6 Conseils pour les étudiants

Astuce RPG du Dev

— Créez une fonction d'ajout automatisé de ces objets de départ : c'est votre "loot de départ".

— Vous pouvez enrichir la base avec vos propres objets (par ex. "Potion de Debug", "Marteau de Valgrind", etc.).

— Testez des cas limites : nom très long, quantité nulle, prix négatif, description absente. Pensez à toutes les entrées absurdes que pourraient faire les Uiteurdizuiteur !

— Imaginez tous les problèmes auxquels votre code peut être soumis.

3 Application à réaliser

4 Description de l'application à réaliser

Objectifs de conception

L'application à concevoir est un **programme en ligne de commande (CLI)** permettant de gérer dynamiquement l'inventaire médical du peuple des Cyboulettes. Elle devra être robuste, sécurisée, et parfaitement stable, même en cas d'entrées utilisateur invalides ou d'erreurs mémoire.

4.1 Organisation générale du projet

Le projet devra être composé de plusieurs fichiers source et d'un fichier principal.
Le code doit être :

— **clairement commenté** : chaque fonction doit comporter une brève description de son rôle, de ses paramètres et de ses valeurs de retour ;

— **structuré** : pas de fonctions "monstres" de 200 lignes, mais un découpage cohérent ;

— **propre et compilable sans avertissements** avec les options `-Wall -Wextra -Werror`. Notez que ces options représentent le minimum de ce qui est nécessaire. Nous avons parlé d'autres options en cours.

4.2 Interface utilisateur attendue

Interface en ligne de commande (CLI)

L'application fonctionne uniquement en **ligne de commande (CLI)**. Au démarrage, elle affiche un menu principal proposant les actions suivantes :

```
=== Bureau de Gestion des Ressources de Soin (BGRS) ===

1. Afficher l'inventaire complet
2. Ajouter un produit
3. Supprimer un produit
4. Modifier un produit
5. Rechercher un produit
6. Sauvegarder l'inventaire
7. Charger un inventaire
8. Quitter

Choix :
```

Chaque saisie utilisateur doit être validée. Aucune entrée incorrecte (nombre hors plage, chaîne vide, caractères invalides) ne doit faire planter le programme. En cas d'erreur, un message explicite doit être affiché et la saisie répétée.

4.3 Fonctionnalités à implémenter

Les fonctionnalités à réaliser sont ici décrites en terme de ce qu'attend l'utilisateur. Dans la pratique, le fonctionnement dans le code peut être découpé de la manière qui vous semble la plus logique. Par exemple, si vous pensez que l'action **Ajouter un produit** nécessite de rechercher au préalable si le produit n'est pas déjà présent dans la base, vous pouvez tout à fait faire des appels à rechercher **Rechercher un produit** avant ou pendant l'ajout. En d'autres termes, le fonctionnement interne de l'application peut tout à fait ne pas correspondre directement aux différentes opérations décrites ci-après (du moment que le travail est fait) :

F1. **Afficher l'inventaire** : Affiche tous les produits sous forme de tableau formaté (id, nom, catégorie, quantité, prix).

F2. **Ajouter un produit** : L'utilisateur saisit les champs nécessaires (nom, description, catégorie, quantité, prix). Le programme alloue dynamiquement la mémoire pour les chaînes et vérifie les limites de taille.

F3. **Supprimer un produit** : Suppression à partir de l'`id`. La mémoire associée doit être libérée proprement sans fuite ni double libération.

F4. **Modifier un produit** : Permet de modifier un champ (par exemple, changer la quantité ou le prix). Les nouvelles valeurs sont validées avant remplacement.

F5. **Rechercher un produit** : Recherche par nom (insensible à la casse). Affiche le ou les résultats correspondants.

F6. **Sauvegarde de l'inventaire** : Sauvegarde des données dans un fichier texte ou binaire. **Vous avez le droit de réutiliser le code de lecture/écriture que vous avez**

développé l'année dernière dans le cadre du projet de *C Sécurisé I*. Vous pouvez également le réécrire si vous le souhaitez.

F7. Chargement de l'inventaire : Lecture d'un fichier précédemment sauvegardé, en reconstituant les structures dynamiques correspondantes. Les anciennes données doivent être libérées (i.e. détruites) avant rechargement.

4.4 Exigences techniques et sécuritaires

Il vous est demandé de respecter les exigences suivantes :

Règles impératives de sécurité

- Toute allocation mémoire (`malloc/realloc`) doit être immédiatement vérifiée ;
- Aucun dépassement de tampon n'est toléré : utilisez `fgets()` au lieu de `gets()` ;
- Les chaînes doivent toujours être terminées par `'\0'` ;
- Aucun `free()` ne doit être appelé deux fois sur la même zone mémoire ;
- Le programme ne doit pas fuiter de mémoire (tests sous `Valgrind/AddressSanitizer` très conseillés).

Exigences supplémentaires

En outre, un `Makefile` fonctionnel sera exigé pour compiler votre programme. Vous devrez avoir au moins les cibles all et clean (cf. dev sécurisé 1) dans ce Makefile. **Ne l'oubliez pas.**

4.5 Tests et validation

L'application devra être la plus robuste possible. Par exemple, pensez à passer les scénarios suivants sans erreur :

- ✔ ajout puis suppression répétée de produits ;
- ✔ modification d'un produit inexistant (message d'erreur attendu) ;
- ✔ sauvegarde puis rechargement de l'inventaire complet ;
- ✔ gestion d'une description très longue (proche de 1000 caractères) ;
- ✔ saisies incorrectes (prix négatif, caractères invalides, quantités trop grandes).
- ✔ et tout ce que vous pouvez imaginer...

5 Phase 1 : Développement Sécurisé et Robuste (8 heures)

Dans cette première phase, pour laquelle 8h vous sont attribuées, vous devez implémenter votre programme en C en utilisant les préceptes de programmation défensive vus en cours. L'usage de tous les documents et d'outils de développement (i.e. IA génératives, IDE, outils de debug, etc.) est autorisé.

L'application sera à rendre **le mardi 9 décembre 2025 à 23h59**, c'est à dire le soir qui suit la dernière séance encadrée correspondant à la phase 1. Le document à rendre sur Moodle sera le suivant :

— Un questionnaire nommé "dossier de dépôt de prototype" qui vous sera adressé par le BGRS. Ce questionnaire vous sera fourni **durant la dernière heure du projet**.

— Un lien vers le git où réside votre code (il faudra donc le rendre public ou à minima visible pour vos enseignants (c'est à dire pour les utilisateurs **guennec** et **houssou**).

— Une archive de votre dépôt au moment de la soumission (qui sera utilisée en cas de problème avec votre git).

5.1 Règles de programmation défensive

Gardez les différentes idées en tête au cours du développement. La programmation défensive est un état d'esprit permanent, pas une considération de dernière minute que l'on adopte juste avant de rendre le projet.

— **Validation d'entrée :** toutes les entrées doivent être vérifiées (longueur, plage numérique, NULL checks). Ne jamais faire confiance à l'utilisateur ou au fichier d'entrée.

— **Allocation minimale :** n'allouer que la mémoire nécessaire (pas de champs statiques surdimensionnés).

— **Vérifications systématiques :** vérifier le résultat de `malloc`/`realloc`/`fopen`/`read`/`fwrite` et réagir proprement en cas d'erreur.

— **Pas de fonctions dangereuses :** interdiction d'utiliser `gets`, `strcpy`, `sprintf` non sécurisé. Préférer `fgets`, `snprintf`, `strncpy` avec soin, ou mieux `getline` si disponible.

— **Overflow integers :** avant toute multiplication ou addition qui calcule une taille à allouer, vérifier les `UINT_MAX` / overflows.

— **Propriétés atomiques :** les opérations complexes (p.ex. mise à jour) doivent laisser la structure dans un état cohérent même en cas d'échec.

— **Libération systématique :** fournir une fonction `inventory_destroy` qui libère tout proprement. Aucun memory leak toléré.

— **API clairement documentée :** chaque fonction doit documenter les préconditions, postconditions, et codes d'erreur.

6 Phase 2 : que va-t-il arriver à votre code ? (4 heures)

Les modalités de la 2ème partie du projet seront fournies en temps voulu. Sachez cependant ceci :

— Vous devrez donner accès à votre code (i.e. à votre git) à quelqu'un.

— Votre code a tout intérêt à faire moins de chose mais bien plutôt que beaucoup mais mal.

Vous verrez quand on y sera...

— Fin du sujet —