My Collection

Principe du TP à rendre

Avant d'aborder la programmation graphique interactive avec le projet de fin d'année, l'objectif de ce mini-projet est de mettre en œuvre toutes les notions d'algorithmique et de C présentées jusqu'à maintenant, sur le thème de la **gestion d'une collection**. L'application demandée sera en mode console « pure », c'est à dire qu'elle ne fera appel qu'aux entrées-sorties standardisées et portables du C, elle devra pouvoir compiler et fonctionner indifféremment sur Linux et Windows (pas de #include <windows.h>, pas de getch ni de kbhit ni de system("clear"); pas de couleurs etc...)

Le principe de gestion de **collection d'éléments de mêmes types** est un pilier de l'activité de l'informaticien, c'est un besoin omniprésent que ce soit pour une armada de soucoupes volantes sur lesquelles on tire dans un jeu vidéo, un ensemble de patients dans un hôpital, la flotte des véhicules d'une entreprise de transports, l'archive de tous les tweets sur un compte social... Différentes approches techniques existent pour répondre à ce besoin, selon le contexte : tables de bases de données, lignes/colonnes dans un tableur, fichiers CSV... Pour un programme, une collection doit souvent être mise en mémoire vive dans son intégralité. Ce besoin conduit à la notion de conteneur dans les langages orientés objet (C++ 1er semestre ING2). **A notre niveau en langage C nous avons comme principales approches à proposer les tableaux de structs (et variantes) et les listes chaînées de structs (et variantes).**

L'objectif est donc principalement de mettre en œuvre une de ces 2 approches, dans une application permettant interactivement de gérer des « fiches » dans une collection, en ajouter, en enlever, les voire toutes, faire des recherches etc... et d'assurer la persistance des données déjà saisies par l'utilisation d'un fichier de sauvegarde.

Vous choisirez un **thème de collection** librement, à condition qu'il permette naturellement d'intégrer au minimum dans chaque fiche le stockage :
- d'un champ chaîne de caractère (dans un tableau de char de taille fixe ou alloué dynamiquement)
- d'un champ numérique (entier ou à virgule)
- d'un champ structuré (lui même composé de plusieurs champs)

Exemples possibles :
- *Des aliments bons pour la santé (Nom, Calories par kg, Contenu en vitamines (vit A, vit. B, vit. C))*
- *Des personnages de manga (Nom, Age estimé des lecteurs, Contexte (œuvre, amis, ennemis))*
- *Des super-héros de comics (Nom, Vrai nom, Année création, Super-pouvoirs (principal, secondaire))*
- *Des films (Nom, Année, Réalisateur (nom, prénom))*
- *Des sports (Nom, Record du monde, Unité du record, Plage d'age pratiquants (age min, age max))*
- *Des comptes en banque (Nom, Prénom, Crédit en euros, Date ouverture (jour, mois, année))*

Etc...

Choisissez donc un thème de collection qui vous parle et trouvez les champs qui vont bien. Inutile de mettre 20 champs, les points ne s'attribuent pas sur ce critère : si vous savez mettre 3 champs de types différents alors vous saurez en mettre 20. Le problème intéressant pour l'ingénieur logiciel est celui de l'obtention des espaces de stockage (automatique ou dynamique ?), du découpage en sous-programmes, de l'échange des informations entre sous-programmes (paramètres), des algorithmes d'interaction (boucle de menu) et de traitement des données (tris, filtrages, sélections, recherches) en laissant les structures de données dans un état cohérent (par exemple pas de « trous » dans un tableau suite à l'effacement d'une fiche) et enfin l'aspect persistance (fichier).

Cahier Des Charges

Voici ce que souhaite avoir le client collectionneur :

Une application qui automatiquement quand on la lance charge les fiches déjà saisies lors des cessions précédentes et quand on la quitte sauve la collection dans son nouvel état (avec des fiches en plus ou en moins par rapport au lancement).

Au lancement l'application offre un menu interactif permettant de :
- quitter l'application (en sauvant l'état actuel de la collection)
- voir la totalité des fiches de la collection (si il y en a beaucoup ça défile sur la console : acceptable)
- ajouter une nouvelle fiche (dont le contenu est à saisir immédiatement)
- désigner une fiche selon une clé unique pour pouvoir la supprimer (par exemple un champ *nom* sert de clé)
- faire une recherche par valeur sur un champ au choix et afficher la/les fiche(s) correspondante(s) (par exemple je peux chercher toutes les fiches avec nom=Superman ou vrai nom=Kal-El ou Année création=1933 ou super-pouvoir principal=voler etc...)
- faire un tri stable[1] de la collection sur un champ au choix[1.5], en ordre croissant ou décroissant (par exemple je peux trier selon le nom en ordre alphabétique inverse puis refaire un tri selon l'année de création en ordre croissant et enfin demander l'affichage de la collection : je vois alors les fiches rangées par années croissante, et à année égale par ordre alphabétique inverse du nom)

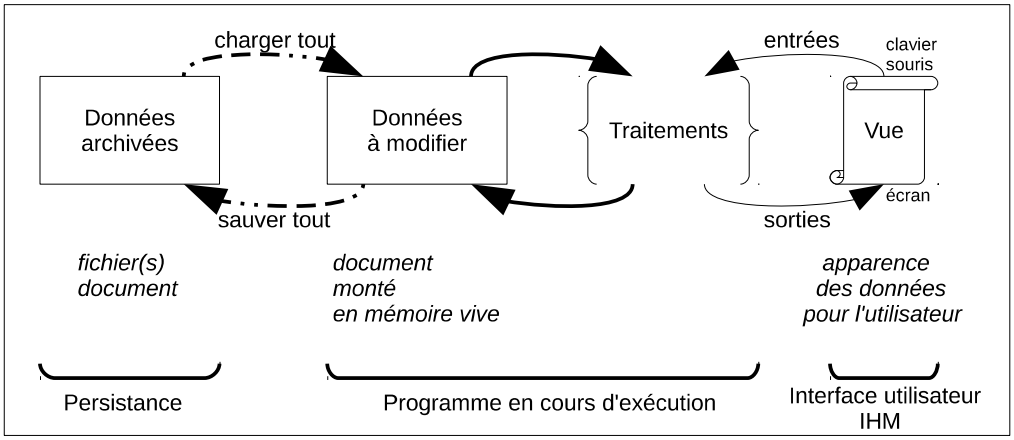
1. Voir Wikipedia sur ce qu'est un « tri stable ». 1.5. Sauf champ structuré si il ne s'y prête pas

L'application garantie de ne pouvoir introduire de doublons selon le champ qui sert de clé (par exemple on ne peut pas avoir 2 fiches « Superman »). Si l'application a une limite dans le nombre de fiches qu'elle peut gérer, elle gère correctement cette limite en refusant poliment l'ajout d'une fiche excédentaire (plutôt qu'en crashant). Le tri est sauvegardé : si un tri a été fait avant de quitter l'application, lors de la prochaine ouverture de l'application on trouve les fiches rangées selon cet ordre.

Le fichier de sauvegarde des données sera au même niveau que l'application[2]. En cas d'absence de ce fichier de sauvegarde des données, l'application démarre correctement (naturellement avec 0 fiche). Si le fichier est absent lors de la sauvegarde (à la fermeture de l'application) celui-ci est correctement (re)créé avec les fiches présentes dans l'application au moment de la fermeture.

Le client collectionneur souhaite avoir l'application livrée avec un fichier de sauvegarde comportant déjà entre 10 et 100 fiches (selon vos préférences). Les fiches comporteront des données permettant de tester les fonctionnalités de manière convaincante, en particulier les tris. Par exemple il y aura une fiche Wonder Woman année 1941 ainsi qu'une fiche Captain America 1941 permettant de voir qu'à année égale un tri préalable par nom (en ordre alphabétique direct ou inverse) place Wonder Woman respectivement après ou avant Captain America. *Le cas d'une collection de super-héros est utilisé à titre d'exemple, encore une fois : vous êtes libre du thème.*

L'architecture document/application



Nous souhaitons vous voir maîtriser cette architecture avant de vous lancer dans des systèmes plus complexes (bases de données, cloud...). Le principe à respecter est le suivant : un fichier ne reste jamais ouvert longtemps, et on ne l'ouvre que pour le lire ou l'écrire en totalité, pas pour le bidouiller. Soit on l'ouvre en mode « r » et on lit **tout** depuis le fichier vers la mémoire vive (dans les espaces de stockage prévus, tableaux ou listes chaînées) et on le ferme. On dit alors que le document (correspondant au fichier) est monté en mémoire vive. Le travail avec le document se fait alors complètement avec ces données en mémoire vive et il n'est pas question d'aller déranger le fichier à ce stade. **Ce serait par exemple un contre-sens d'ouvrir le fichier en mode « a » pour compléter le fichier sous prétexte qu'on ajoute une fiche, une fiche s'ajoute à la collection en mémoire vive.** Seulement quand on veut sauver alors on sauve l'intégralité des données depuis la mémoire vive vers

2. Donc vous ne devez pas avoir de chemin d'accès absolu dans vos fopen, pas de "c:/etc.../heroes.txt" mais juste "heroes.txt", dans le répertoire courant.

le fichier. La version précédente du fichier est totalement écrasée : on ouvre en mode « w » on écrit **toutes** les données depuis la mémoire vive (espaces de stockage prévus, tableaux ou listes chaînées) vers le fichier, et on le ferme.

Le planning de développement

Il est bon de rappeler que la chronologie de présentation du CDC par le client n'est pas forcément la meilleure chronologie pour votre planning prévisionnel de développement : réfléchissez soigneusement à l'ordre dans lequel les fonctionnalités peuvent être complétées et **testées**. N'hésitez pas à écrire des *main* temporaires ou des fonctions auxiliaires pour tester. Par exemple un ordre de développement pourrait être (sans obligation aucune) :

1. Définir le type struct d'une fiche avec juste un seul champ (le numérique, le plus simple)
2. Ecrire un sous-programme permettant de saisir les données d'une struct passée par adresse
3. Ecrire un sous-programme permettant d'afficher les données d'une struct passée par adresse
4. Tester ! (vous voyez, il n'est pas question de fichier ici)
5. Mettre en place la structure d'accueil de la collection (tableau ou liste chaînée)
6. Ecrire un sous-programme d'ajout d'une fiche à la collection
7. Ecrire un sous-programme d'affichage de toutes les fiches de la collection
8. Tester ! (il n'est toujours pas question de fichier, bien qu'il apparaisse en 1er dans le CDC !)
9. Mise en place menu (encore incomplet) : Quitter / Ajouter un / Afficher tous
10. Tester !
11. Avant d'alourdir le projet et les protocoles de test, cloner le projet et essayer d'adopter une approche plus sophistiquée pour la collection (par exemple avec allocation dynamique...)
12. Tester ! C'est à votre portée ou pas → adopter ou pas l'approche plus sophistiquée
13. Compléter la structure avec les champs prévus, compléter les sous-progs concernés
14. Tester ! (il n'est toujours pas question de fichier)
15. Commencer à mettre en place le fichier ? Sous-progs charger_tout et sauver_tout...
16. Tester ! Commencer à mettre en place une base de fiches de test...
17. Maintenant qu'on a le chargement de 10 fiches au démarrage on peut tester plus facilement des fonctionnalités de recherche sans avoir à passer du temps à taper à la main...
18. *Etc... selon le même principe stratégique de pouvoir valider régulièrement une appli encore incomplète mais cohérente et d'optimiser les temps de test*

Cahier Des Charges étendu

A ces fonctionnalités de base, prioritaires, le client souhaiterait voir ajouté (harmonieusement dans le thème de la collection) les fonctionnalités suivantes :

- Possibilité de modifier une fiche (à désigner) sur un de ses champs (au choix) en ressaisissant l'info
- Un champ correspondant à une quantité variable d'informations, par exemple N super-pouvoirs
- Un champ correspondant à une relation, par exemple un champ binôme, le binôme de Batman serait Robin. En affichant les infos d'un super-héro on affiche les infos de son binôme si il existe. Attention on ne veut ni re-rentrer ni dupliquer les infos du binôme (référencer par pointeur ou clé)
- Système sélection/action, à partir d'une recherche on n'obtient pas juste un affichage des fiches correspondantes mais un marquage (sélection) de celles-ci. L'action d'effacement peut alors porter sur plusieurs fiches : les fiches sélectionnées, de même pour l'action afficher_tout (filtrage)

Conditions du travail à faire

Le TP est à présenter en binôme (un dépôt pour 2) ou en monôme. Les binômes sont à constituer au sein d'un même groupe de TP. La période de réalisation s'étend du Lundi 30/01 (la séance de TP de cette semaine étant consacrée au démarrage de ce mini-projet) au **Lundi 20/02 à 10H du matin, date à laquelle vous devrez avoir déposé votre code de mini-projet** (l'adresse du dépôt campus vous sera communiquée ultérieurement). Le dépôt sera une archive .zip ou .rar comportant l'ensemble des codes sources permettant de compiler le projet ainsi que le (ou les) fichier(s) de vos fiches déjà enregistrées (une dizaine ou plus, voir ci-dessus). On ne veut pas avoir plusieurs projet : l'archive déposé doit correspondre à un seul exécutable qui intègre les différentes fonctionnalités demandées. Les pénalités de retard usuels s'appliquent : -2 points par période de retard de 24H entamée.

Les règles usuelles d'un travail évalué s'appliquent, en particulier l'équipe encadrante sera extrêmement attentive aux plagiats et ceux-ci seront sévèrement sanctionnés. L'emprunt d'exemples donnés par l'encadrement, de même que l'emprunt de courts blocs de lignes de code ou de fonctions accessibles publiquement sur le web est accepté **mais doit être clairement spécifié dans le code source et dans le rapport** (voir ci-dessous). La copie pure et simple d'un code fonctionnel de 50 lignes ou plus, qu'il vienne d'un prof, d'une autre équipe, du Web, même suivie d'une « customisation » ou d'une ré-écriture n'est pas acceptable. Vous devez avoir tapé l'essentiel de votre application vous même à partir de votre compréhension des mécanismes de la programmation, pas à partir de copier-coller d'un code tiers, ni en recopiant massivement à la main un code tiers ce qui revient au même qu'un copier-coller. Soyez sûrs que nous savons faire la différence.

Nouveauté : rapport de validation des fonctionnalités/techniques, et auto-évaluation

A la période de réalisation avec dépôt avant le Lundi 20/02 à 10H succède une période d'**auto-évaluation du travail réalisé** : vous avez une semaine supplémentaire pour valider les différentes fonctionnalités du projet et **déposer un rapport de fonctionnalité avant le Lundi 27/02 à 10H du matin, au format .pdf**. (pour rappel tous les traitements de texte moderne tel que Word ou LibreOffice Writer permettent d'exporter au format pdf).

→ Fonctionnalités

Une grille détaillée est indiquée ci-après : vous testerez chacune des fonctionnalités correspondantes au CDC. Cette grille devra figurer **en tête** de votre rapport[3], complétée. Pour preuve du bon fonctionnement de **chacune** des fonctionnalités revendiquée, vous aurez sur le rapport une ou plusieurs **captures écran** détaillées sur les pages en annexe. Le nombre de pages en annexe nécessaire à démontrer le bon fonctionnement de tous les points opérationnels de votre projet n'est pas limité. Il est possible qu'il vous faille 50 pages, alors mettez 50 pages. Dans la grille d'auto-évaluation, en face de chaque fonctionnalité revendiquée (pour laquelle la réponse est **OUI**), vous préciserez la(les) page(s) du rapport où trouver les captures qui prouvent le bon fonctionnement de cette fonctionnalité. Pour les fonctionnalités non revendiquées (pour laquelle la réponse est **NON**) il n'est évidemment pas nécessaire de les prouver par captures.

3 Une version .docx et une autre .odt vous seront fournies en temps utile pour vous permettre de l'intégrer

→ Choix de conception et technique de programmation

En plus des fonctionnalités (ce que l'utilisateur peut faire ou pas avec votre application) nous sommes également attentifs aux façons dont votre projet traite les problèmes, avec quels choix de conception et quelles techniques de programmation. Certains choix sont mutuellement exclusifs, vous ne pouvez donc a priori pas cocher simultanément plusieurs cases dans le groupe indiqué « **Un au choix** ». Si vous avez le sentiment que votre conception ou solution technique n'entre pas dans les cases, contactez votre chargé de TP et précisez vos remarques explicitement sur votre rapport, en complément de la grille d'auto-évaluation. Sont valorisées (ou non pénalisées) les techniques qui d'une part sont généralement considérées comme de bonnes pratiques (paramètres plutôt que variables globales, sous-programmes de taille raisonnable, passages par adresse plutôt que par valeur pour les structs) et d'autre part les conceptions considérées comme plus complexes mais offrant plus de souplesse (dans l'idée où l'application est amenée à évoluer) telles que l'allocation dynamique plutôt que l'allocation automatique, les liste chaînées plutôt que les tableaux. Cependant rappelez vous qu'il est préférable de réussir une approche plus basique que de rater une approche complexe. Une approche par tableau de structs plutôt que liste chaînée est également correcte même si elle comporte des limites. Vous préciserez en face de chaque technique/conception revendiquée une ou des lignes du code déposé (nom du fichier et numéro de ligne dans le fichier source) représentatives du point.

→ Barème

Le barème est précisé et la somme des points validés donne une note sur 20. La note ne peut dépasser 20 même si le barème totalise plus, il n'est donc pas attendu que vous traitiez tous les points. **Si** vous jouez le jeu sérieusement, votre note d'auto-évaluation sera reprise et validée telle quelle.

→ Arbitrage

L'encadrement vérifiera tout ou partie des fonctionnalités et choix et techniques revendiquées. Un défaut manifeste (cas particuliers mals gérés, fonctionnalité implémentée trop « à l'arrache »...) conduira à une neutralisation du point correspondant. **Une disparité entre ce qui est revendiqué et ce qui est effectivement opérationnel dans le projet conduira à des points de pénalité**, par exemple si vous revendiquez une fonctionnalité qui vaut 1 points mais qu'il s'avère que votre code est incapable de réellement assurer cette fonctionnalité, alors vous avez -1 au lieu de +1. Si nous détectons trop de disparité entre les points revendiqués et le code déposé nous pouvons décider de simplifier le problème en attribuant nous même une mauvaise note, un 0 le cas échéant, voire ajouter une observation en cas de falsification systématique et avérée. Dans tous les cas l'encadrement a le dernier mot sur la note finale.

Il va de soi que les fonctionnalités sont à valider sur le **code du dépôt en l'état** (tel que déposé avant le Lundi 20/02) et qu'il n'est pas question de retoucher ce code ou de valider des fonctionnalités sur des « améliorations » ou des « petites corrections de bugs » qui interviendraient après. Au dépôt le code est figé, la fusée est lancée, on ne peut plus qu'observer (scrupuleusement et attentivement) son bon ou son mauvais fonctionnement.

n°	Point fonctionnel	pts	OUI/ NON	pris	Page démo
1	Chaque fiche comporte au moins un champ chaîne de caractère	1			
2	Chaque fiche comporte au moins un champ numérique	1			
3	Chaque fiche comporte au moins un champ composé de champs !	1			
4	Chargement automatique de la collection complète à chaque démarrage appli. à partir d'un fichier de sauvegarde	1			
5	Sauvegarde automatique de la collection complète à chaque fermeture appli. vers un fichier de sauvegarde	1			
6	Menu interactif, l'appli reste ouverte jusqu'à ce qu'on choisisse quitter	1			
7	Affichage de tous les éléments de la collection	1			
8	Ajouter une fiche à la collection	1			
9	Supprimer une fiche de la collection (désignation par valeur unique)	1			
10	Recherche/affichage sur au moins un champ particulier	1			
11	Recherche/affichage sur n'importe quel champ au choix (cocher cette case vous permet de cocher la précédente, total 2 pts)	1			
12	Trier les fiches sur au moins un champ particulier, un seul ordre (croissant) → 0.5 ordre au choix (croissant ou décroissant) → 1	0.5 ou 1			
13	Trier les fiches sur n'importe quel champ au choix, un seul ordre (croissant) → 0.5 ordre au choix (croissant ou décroissant) → 1 (cocher cette case vous permet de cocher la précédente, total doublé)	0.5 ou 1			
14	Garantie de ne pas pouvoir faire de doublons sur le champ servant de clé de désignation de fiche (par exemple pour désigner fiche à effacer)	0.5			
15	Le tri est sauvé (on retrouve le dernier tri au démarrage de l'appli)	0.5			
16	Le tri est stable (tris successifs selon des champs différents...)	0.5			
17	Gestion correcte de la limite d'ajouts (refus d'ajouter une fiche quand la limite est atteinte). Il est autorisé de modifier le code pour tester cet aspect, par exemple baisser artificiellement la taille du stockage...	0.5			
18	L'appli est livrée (dépôt campus au 20/02) avec un fichier de données correspondant à 10 à 100 fiches permettant le test des fonctionnalités	1			
19	Pouvoir modifier une fiche, un champ particulier → 0.5, champ au choix → 1	0.5 ou 1			
20	Champ à quantité variable d'information, taille max → 0.5, alloc. dyn. → 1	0.5 ou 1			
21	Champ « relationnel » : une fiche peut désigner d'autre(s) fiche(s) → 0.5 cette désignation est robuste à modif. de tout champ de fiche(s) désignée(s) → 1	0.5 ou 1			
22	Système sélection/action : on peut sélectionner des fiches (recherche), permet d'avoir une action d'afficher_tous_selects (0.5) ou d'effacer_selects (0.5)	0.5 ou 1			
	Choix conception & technique	**pts**	**OUI/ NON**	**pris**	**fic. & lig. code src.**
23	Tableau surdimensionné de taille fixe genre [100], variable nb_fiches qui bouge Tableau surdimensionné de taille fixe genre [100], valeur spéciale « case vide » Liste chaînée à simple chaînage **(groupe 23 : un au choix)** Liste chaînée à double chaînage	→ 0 → 0 → 1 → 2	____ ____ ____		
24	Encapsulation de la collection dans un type t_collection, par exemple une struct contenant tableau de structs et champ nb_fiches, ou struct contenant ancre(s)...	1			
25	Allocation dynamique sur mesure des chaînes de caractère (& free...)	1			
26	Allocation dynamique des structs correspondant aux fiches (tableau de pointeur sur structs, ou maillon avec pointeur sur data) (& free...)	1			
27	Passage systématique par adresse des structs (aucun passage par valeur struct)	0.5			
28	Aucun sous-programme de plus de 50 lignes (hors lignes vides et commentaires) Indiquer fichier et ligne du plus gros. Attention ne pas tricher, une ligne par . . . ;	0.5			

	Pénalités techniques Si vous répondez NON à une question, la pénalité s'applique	pts	OUI/ NON	per du	fic. & lig. code src.
29	Chaque sous-programme est commenté au dessus avec explication de son rôle et du rôle de ses paramètres et retour éventuel, les commentaires sont fidèles	-2			
30	Il n'y a pas de Warnings (0 errors / 0 warnings) ni en Debug ni en Release	-1			
31	Il y a au moins un sous-programme par fonctionnalité, pas juste un gros *main*{ … }	-2			
32	Les informations principales (collection, fiches...) circulent de sous-programme en sous-programme par l'intermédiaire de paramètres et retours éventuels, on n'utilise pas des variables importantes (la collection, le nombre de fiches) sous forme de globales (variables déclarées en dehors de tout sous programme)	-4			
33	Les variables locales sont bien toutes déclarées en début de chaque sous programme avant toute action (consigne à respecter en ING1)	-1			
					← TOTAL

Robin Fercoq et toute l'équipe encadrante,

30/01/2017

Version 1, sujette à ajustements mineurs en cas de besoin.

En 1ère page : image de collection d'insectes, citation académique,

https://beetlesinthebush.wordpress.com/collection/