A picture containing icon

Description automatically generated **UE 159 Organisation et exploitation des données**

**HAUTE ÉCOLE DE NAMUR-LIÈGE-Luxembourg**

**Informatique B1**

**2023-2024**

Module 3 : Listes chainées

Objectifs

* Manipulation des listes chainées (triées ou non, avec ou sans blocs logiques) : écrire, comprendre/corriger ou mettre à jour des diagrammes d’actions/modules

**Références**

Cours de théorie OED

**Pour tous les exercices, ne traitez les cas d’erreur que s’ils sont explicitement demandés.**  
Partie 1

**Objectif : conception et manipulation des listes chainées simples**

Exercice 1

Un artisan qui fabrique régulièrement de nouveaux jouets, souhaite conserver leurs caractéristiques (référence, nom, prix, nombre d’exemplaires en stock, matériaux) pour une utilisation ultérieure.  
Parmi les structures que vous connaissez, quelle est la plus adéquate ? Justifiez votre réponse

Exercice 2

On décide de stocker, au fur et à mesure de leur fabrication, les jouets de cet artisan dans une liste chainée dont chaque chainon concerne un jouet et reprend :

* la référence (alphanumérique)
* le nom
* le prix
* le nombre d’exemplaires en stock.

A partir des informations obtenues pour les trois premiers jouets fabriqués par cet artisan, on vous demande de représenter graphiquement la liste chainée ainsi créée. Les chainons sont placés selon l’ordre du tableau (le premier jouet est le train) et la liste est pointée par **pDébutJouets**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| référence | nom | prix | nbExemplaires | adresse disponible pour ce jouet (en hexadécimale) |
| "BO172" | "Train en bois" | 47.50 | 6 | A41 |
| "BO613" | "Voiture en bois" | 51.50 | 13 | B18 |
| "BO727" | "Puzzle" | 9.25 | 30 | B35 |

Exercice 3

Soit la liste chaînée simple (pointée par **pDébutJouets**)de l’exercice 2 reprenant tous les jouets fabriqués par l’artisan durant l’année 2024.

Cette liste n’est pas triée.

On vous demande d’écrire :

1. Le module qui reçoit une référence de jouet et un prix et qui met à jour le prix (s’il est différent).   
   Traitez le cas d’erreur d’une référence non reprise dans la liste ;
2. Le module qui compte et renvoie le nombre de jouets de la liste qui coûtent plus de 50 euros ;
3. Le module qui affiche les références des jouets dont le prix est inférieur au prix reçu en argument, ainsi que la référence du jouet le moins cher (sans cas d’égalité).
4. Le module qui retourne les références des jouets dont le prix est inférieur au prix reçu en argument, ainsi que la référence du jouet le moins cher (sans cas d’égalité).
5. Le module qui compte le nombre de jouets dont le prix est respectivement inférieur à 10, 20, 30, 40 et 50 euros et qui affiche ces 5 quantités.

Exercice 4

Un pépiniériste a retenu, dans une liste chainée simple, la liste des articles achetés en 2024 par ses clients.

On vous demande d’écrire le module qui permet d’afficher pour chaque client :

* son numéro de carte
* pour chacun des articles achetés :
  + la référence de l’article
* la quantité totale achetée
* la référence de l’article qu’il a le plus acheté.

1. **CAS 1**: La liste est identifiée par **pDébutClients** dont chaque chainon concerne un client et retient :

* le numéro de carte du client (***numCarte***)
* le nombre d’articles achetés (***nbArticles***)
* un tableau **articles** reprenant pour chaque article acheté :
* la référence (***référence)***
* la quantité achetée (***quantité***)
* un pointeur vers le chainon suivant (***pSuiv***)

1. **CAS 2** : La liste est identifiée par **pDébutAchats** dont chaque chainon concerne un achat et contient :

* le numéro de carte du client (***numCarte***)
* la référence de l’article acheté (***référence***)
* la quantité achetée (***quantité***)
* un pointeur vers le chainon suivant (***pSuiv***)

Les chainons sont **classés par client**, pour un même client, **par référence d’article**. Pour un client donné, un article n’apparait qu’une seule fois.

Exercice 5

Reprenez la liste des trois jouets de l’exercice 2 et écrivez les instructions qui permettent d’ajouter un nouveau jouet dont on obtient les données via une fiche

1. en début de liste
2. en fin de liste
3. entre le premier et le deuxième chainon.

Partie 2

**Objectif : manipulation des listes chainées dont les chainons pointent sur des listes**

Exercice 6

Les informations relatives aux différentes œuvres exposées dans les salles d’un musée sont retenues dans des listes chainées simples.   
  
La liste chainée principale est identifiée par **pDébutSalles**.

Chaque chainon de la liste principale concerne une salle du musée et contient :

* le numéro de la salle (***numéro***),
* le nombre d’œuvres exposées dans la salle (***nbOeuvres***),
* un pointeur vers une sous-liste chainée simple (***pDébutOeuvres***) où chaque chainon concerne une œuvre exposée dans la salle et contient :
  + la référence de l’œuvre (***référence***),
  + le titre de l’œuvre (***titre***),
  + un pointeur vers l’œuvre suivante (***pOeuvreSuiv***),
* un pointeur vers la salle suivante (***pSalleSuiv***).

La liste principale est *triée par ordre croissant sur le numéro de salle* et les sous-listes *sont triées par ordre croissant sur la référence des œuvres*.

1. Critiquez l’utilité/la pertinence de la variable **nbOeuvres.**
2. Le musée prévoit d’agrandir sa collection. Ecrivez le module qui ajoute une salle vide dont le numéro sera le plus grand numéro de salle existant augmenté de 1.
3. Le musée a acquis une nouvelle œuvre. Ecrivez le module qui reçoit la référence de l’œuvre, son titre ainsi que le numéro de salle dans laquelle elle sera exposée et qui ajoute ces informations dans la liste. La liste des œuvres **doit rester triée**.   
   Prévoyez les cas d’erreur suivants : espace mémoire insuffisant, numéro de salle d’exposition inexistant, référence de l’œuvre déjà présente dans la salle prévue. Un message d’erreur adéquat sera renvoyé par le module.
4. Les œuvres d’une salle sont envoyées dans un autre musée pour une exposition temporaire. Écrivez le module qui reçoit le numéro de la salle et qui supprime cette salle de la liste ainsi que toutes les œuvres qui y sont exposées. Prévoyez le cas d’une erreur dans le numéro de la salle (message erreur renvoyé).

Exercice 7

Une liste chainée reprend les différentes destinations qu’offre une agence de voyage. Elle est triée par *ordre alphabétique sur la destination* et l’adresse du début de la liste est dans **pDébutDestinations**. Chaque chainon contient :

* **destination** : nom de la destination
* **nbHotels** : nombre d’hôtels de la destination
* **pDébutHotels** : pointeur vers une liste chainée contenant les informations sur les différents hôtels de cette destination. Chaque chainon de cette sous-liste contient :
  + **nom** : nom de l’hôtel ;
  + **prix** : prix d’une nuit en chambre double dans cet hôtel ;
  + **pHotelSuiv** : pointeur vers le chainon suivant des hôtels.

Cette sous-liste est *triée par ordre alphabétique sur le nom de l’hôtel*.

* **pDestSuiv** : pointeur vers le chainon suivant des destinations.

1. Complétez le module ci-dessous qui permet de rechercher et de renvoyer le nom de l’hôtel qui offre la chambre la moins chère ainsi que la destination correspondante.  
     
    o────────────────────o ↓ …

│ chambreMoinsChère │

o────────────────────o ↓ …

┌─── \*

│ prixMin = …

│ pDest = …

│

│╔══ while (pDest ≠ null)

│║

│║ pHotel = …

│║

│║╔══ while (pHotel ≠ null)

│║║

│║║┌── if(pHotel → prix < prixMin)

│║║│ prixMin = pHotel → prix

│║║│ nomHotelMoinsCher = …

│║║│ destinationMoinsChère = …

│║║└──

│║║

│║║ pHotel = …

│║╙──

│║

│║ pDest = …

│╙──

└──────────

1. Le module ci-dessous reçoit une nouvelle destination à ajouter à la liste. Le nombre d’hôtels de cette destination est initialisé à zéro. Si la destination existe déjà, il renvoie le message « destination déjà existante ». Il traite également le cas de la mémoire insuffisante.   
   Ce module contient des erreurs dans la signature et/ou le code, corrigez-les.

o──────────────────o ↓ destinationNouv

│ ajoutDestination │

o──────────────────o ↓

┌─── \*

│ pDest = pDebut

│╔══ while (pDest ≠ null AND destinationNouv ≠ pDest → destination)

│║ pDestPrécédent = pDest

│║ pDest = pDest → pSuiv

│╙──

│ erreur = "ajout effectué"

│┌── if(pDest ≠ null)

││

││ erreur = " destination déjà existante "

││

│├── else

││

││ pNouvDest = adresse mémoire nouveau chainon destination

││

││┌── if (pNouvDest != null)

│││

│││ erreur = "mémoire insuffisante"

│││

││├── else

│││

│││ pNouvDest  → destination = destinationNouv

│││ pNouvDest  → nbHotels = 0

│││

│││

│││┌── if (pDest == pDeb) // ajout liste vide ou début

││││ pDeb = pNouvDest

││││ pNouvDest  → pSuiv = pDeb

││││

││││

│││├── else // ajout milieu ou fin

││││

││││ pNouvDest  → pSuiv = pDest

││││

│││└──

││└──

│└──

└──────────

1. Le module ci-dessous reçoit une destination à supprimer de la liste. Ce module supprime aussi tous les hôtels qui s’y trouvent (et libère la mémoire correspondante). On suppose que la destination est présente dans la liste.   
   Ce module contient des erreurs dans la signature et/ou le code, corrigez-les.

o────────────────────────o ↓ destinationSup

│ SuppressionDestination │

o────────────────────────o ↓

┌─── \*

│ pDest = pDébut

│╔══ while ( destinationSup ≠ pDest → destination)

│║

│║ pDest = pDest → pSuiv

│╙──

│ pHotel = pDebHotels

│╔══ do while (pHotel ≠ null)

│║ libérer mémoire pointée par pHotel

│║ pHotel = pHotel → pSuiv

│║

│╙──

│

│┌── if(pDest == pDébut)

││ pDébut = pDébut → pSuiv

│├── else

││ pDest → pSuiv = pDest→ pSuiv

│└──

│

│ libérer mémoire pointée par pDest

└──────────

Exercice 8

Dans le but de gérer les emprunts des livres de la bibliothèque d’une haute école, on dispose d’une liste chainée **simple** dont chaque chainon concerne un emprunteur. Cette liste est *rangée par ordre alphabétique sur l’identité de l’emprunteur* et est pointée par **pDébutEmprunteurs.**

Chaque chainon de la liste contient les informations suivantes :

* les nom et prénom de l’emprunteur (***emprunteur***) ,
* un pointeur vers une sous-liste chainée simple (***pDebLivres***), ordonnée chronologiquement *sur la date de fin d’emprunt*. Elle retient les renseignements sur les livres empruntés par la personne, à savoir :
* le titre du livre (***titre***)
* la date de fin de l’emprunt sous la forme AAAAMMJJ (***dateFinEmprunt***)
* un pointeur vers le livre suivant (***pLivreSuiv***).
* un pointeur vers l’emprunteur suivant (***pEmprunteurSuiv***).

Écrivez le diagramme d’actions qui, via un menu, permet de faire face aux situations décrites ci-dessous :

1. Une personne rapporte un livre. Le module correspondant recevra l’identité de l’emprunteur et le titre du livre.
2. Une personne (déjà emprunteur ou pas encore) emprunte un livre. Le module correspondant recevra son identité, le titre du livre et la date de fin d’emprunt.
3. Une personne prolonge l’emprunt d’un livre. Le module correspondant recevra les mêmes informations que ci-dessus.   
   Remarque : ne déplacez le chainon que si cela est vraiment nécessaire.
4. À partir de la date du jour encodée au clavier, le module affichera la liste des emprunteurs ayant des livres en retard avec, pour chacun :
   * le nom de l’emprunteur,
   * le nombre de livres en retard,
   * la liste des titres de ces livres.

Cette liste sera affichée par ordre décroissant sur le nombre de livres en retard.

1. Quitter l’application.  
     
     
   **Prévoyez les erreurs suivantes**:
   * Emprunt d’un livre déjà emprunté par l’emprunteur (code erreur 1),
   * Restitution d’un livre non emprunté par l’emprunteur (code erreur 2),
   * Prolongation d’un livre non emprunté par l’emprunteur (code erreur 3),
   * Date de prolongation d’un emprunt inférieure à la date initiale (code erreur 4),
   * Mémoire insuffisante (code erreur 5).

Afin de séparer le traitement et l’affichage, l'affichage du message d'erreur se fera dans un module spécifique (**afficherMessageErreur)**.  
Ce module reçoit un code d'erreur numérique, code renvoyé par les modules apparaissant dans le menu des situations à traiter.