UE IG128
Organisation et exploitation des données

Année académique 2023-2024

Contenu

- Module 1 : Introduction
- Module 2 : Tableaux - Compléments
  - Traitement des tableaux triés
  - Bloc logique
  - Algorithmes de tri
- Module 3 : Listes chaînées
- Module 4 : Piles et files
- Module 5 : Arbres
- Module 6 : Tables de hachage

Module 3: Listes chainées

- **Préambule**: bilan concernant les tableaux
  - ❑ Avantages
    - Recherche (et/ou accès) rapide d'un élément
      - accès direct via son indice si on le connaît
      - par dichotomie sinon, si le tableau est trié
    - Algorithmes de tri
  - ❑ Désavantages
    - Allocation statique de la mémoire
      - si on prévoit **trop de cellules**, place mémoire perdue
      - si on prévoit **trop peu de cellules** => traitement impossible
    - En cas de suppression
      - décalage nécessaire
    - En cas d'ajout (si place disponible)
      - si trié, décalage nécessaire

Module 3: Listes chainées

3.1. Définition

Une **liste chainée** est une structure de données

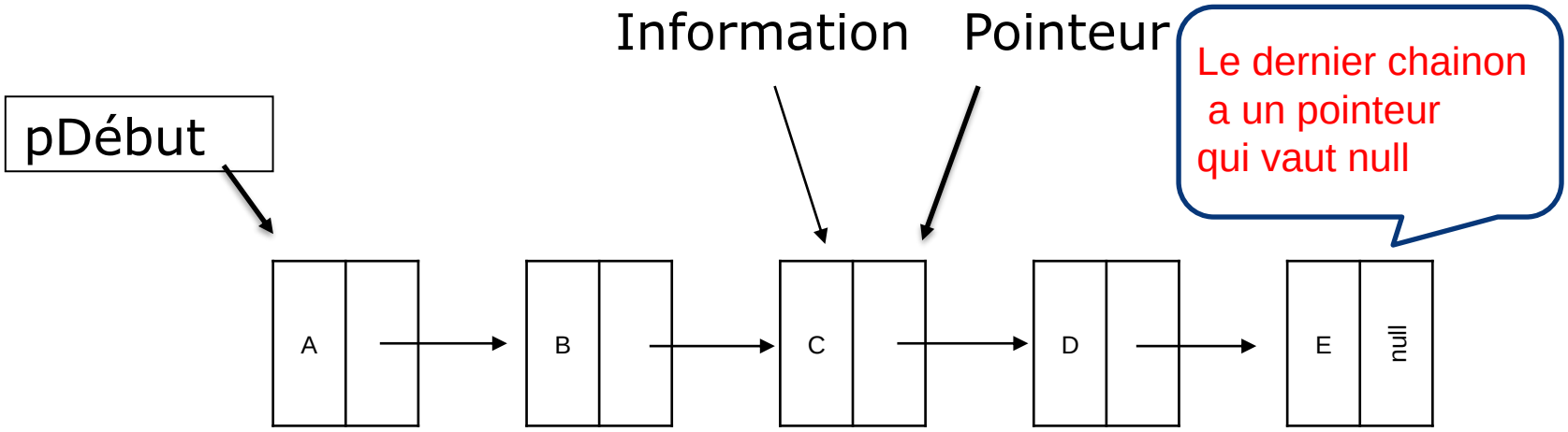
- homogène (tous les éléments sont de même type)
- constituée d'éléments ordonnés linéairement
- chaînés entre eux.

- Les éléments de la liste sont appelés **chainons** ;
- Chaque chainon peut comporter un certain nombre de **champs**;
- Chaque chainon comporte des champs de **données** et (au moins) un champ appelé **pointeur** contenant l'adresse du chainon suivant (ou null) ;
- Un champ de données peut être lui-même un tableau voire un pointeur vers une liste chainée ;

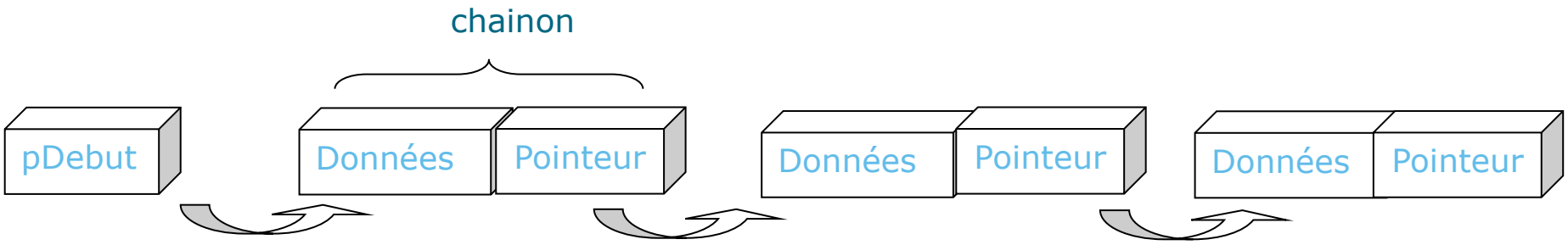
Le début de la liste est déterminé par **un pointeur de début de liste** retenant l'adresse du premier chainon de la liste.

Module 3: Listes chainées

La forme la plus simple est la **liste chainée simple**.



Module 3: Listes chainées

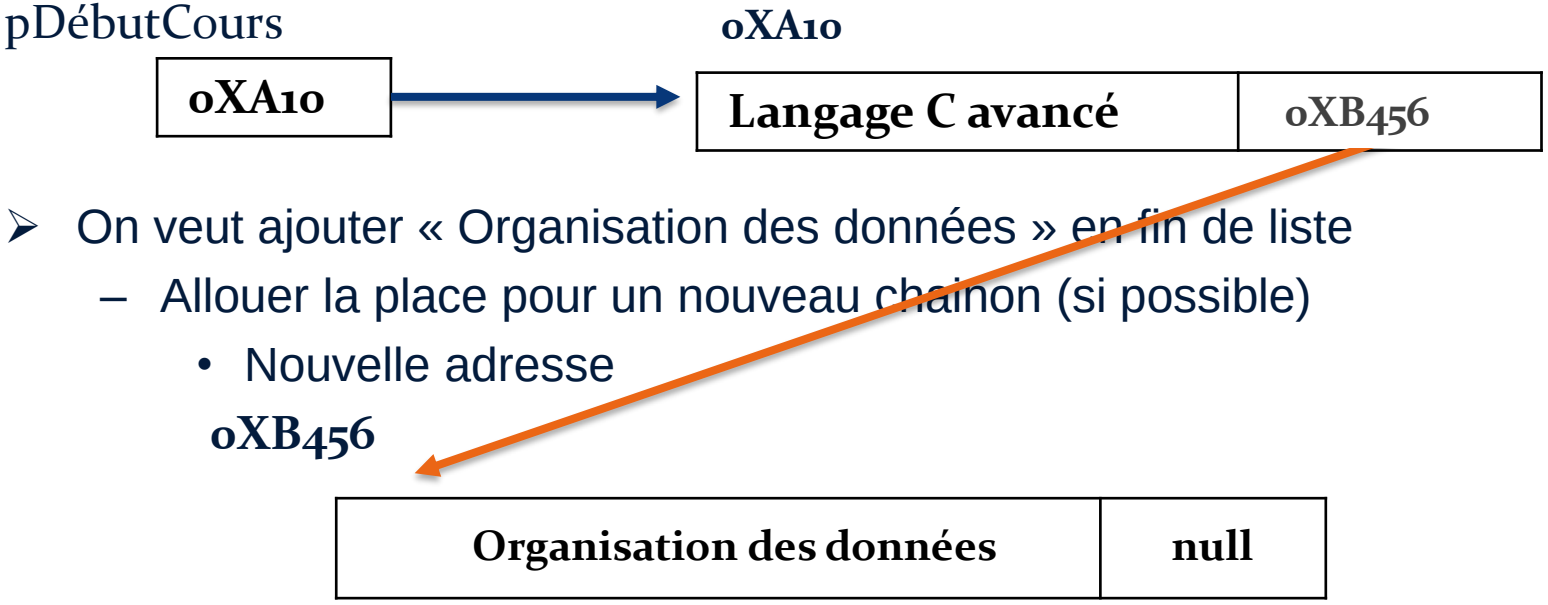


Les listes chainées permettent de gérer la mémoire de manière **dynamique** et sans déplacement de chainons.

Module 3: Listes chainées

Exemple:

Soit une liste chainée dont chaque chainon contient le libellé d'un cours



- On veut ajouter « Organisation des données » en fin de liste
  - Allouer la place pour un nouveau chainon (si possible)
    - Nouvelle adresse

Module 3: Listes chainées

- Avantages
  - place mémoire réservée et libérée au fur et à mesure des besoins (allocation dynamique de la mémoire)
  - suppression et ajout plus simples (il n'y a plus de décalage)
- Inconvénients
  - il n'y a plus d'indice => parcours de la liste depuis le début pour accéder à des données
  - un chainon prend plus de place mémoire qu'une cellule (présence du pointeur)

Module 3: Listes chainées

Autres types

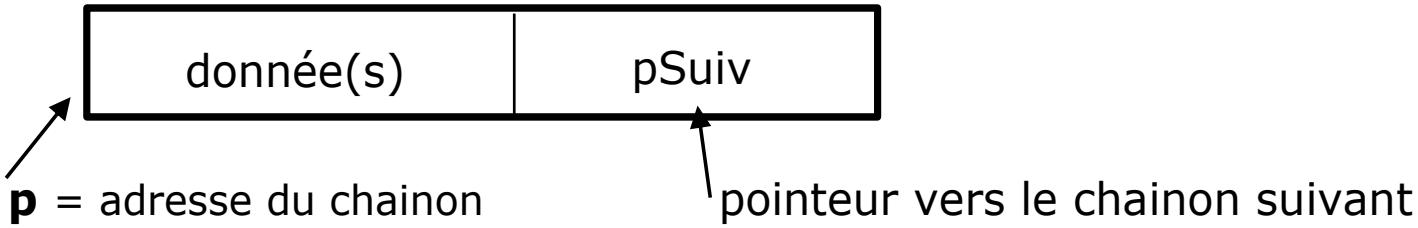
- Liste doublement chainée
  - Un chainon contient l'adresse du suivant et l'adresse du précédent
  - L'adresse du précédent du premier chainon vaut null.

- Liste circulaire
  - Le pointeur du suivant du dernier chainon est l'adresse du premier chainon

Module 3: Listes chainées

3.2. Notations et opérations

Pour la suite du cours, nous schématiserons un chainon d'une liste simple de la manière suivante :



Nous noterons :

- **p** : pointeur vers un chainon quelconque ;
- **pNouv :** pointeur vers un nouveau chainon **;**
- **p -> donnée(s) :** donnée(s) du chainon pointé par p ;
- **p -> pSuiv :** adresse du chainon qui suit celui pointé par p ;
- **pDébut :** pointeur vers le premier chainon de la liste.

3.3. Algorithmes

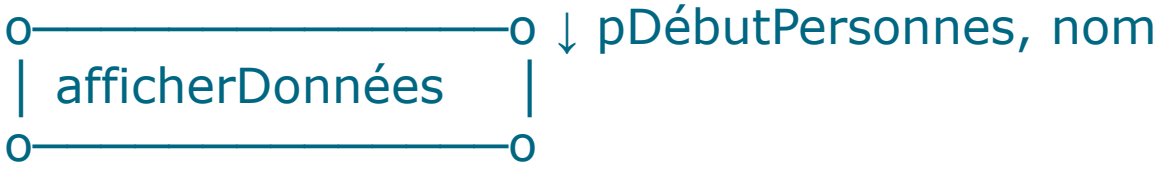
3.3.1. Recherche dans une liste simple non triée

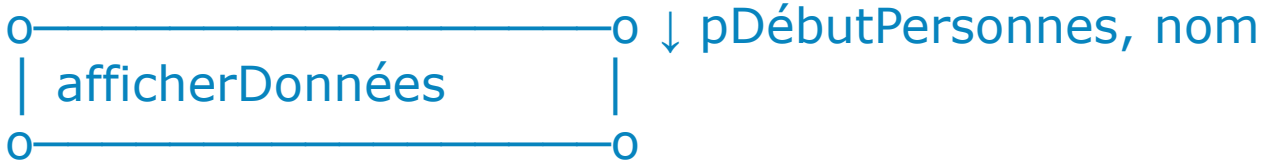
Exemple

Soit une liste chainée dont chaque chainon concerne une personne inscrite à une formation et contient le nom (**nom**), la date de naissance (**ddn**) et la profession de la personne (**profession**) ainsi qu'un pointeur vers le chainon suivant (**pSuiv**).

L'adresse du premier chainon se trouve dans **pDébutPersonnes**.

Ecrire le module qui reçoit le nom d'une personne et qui affiche ses données ou, si elle n'est pas présente dans la liste, le message d'erreur « personne non inscrite »






```
*
pPersonne = pDébutPersonnes
while (pPersonne ≠ null AND nom ≠ pPersonne → nom)
  pPersonne = pPersonne → pSuiv

if ( pPersonne == null)
  sortir "personne non inscrite"
else
  sortir pPersonne → nom
  sortir pPersonne → ddn
  sortir pPersonne → profession
```

Module 3: Listes chainées

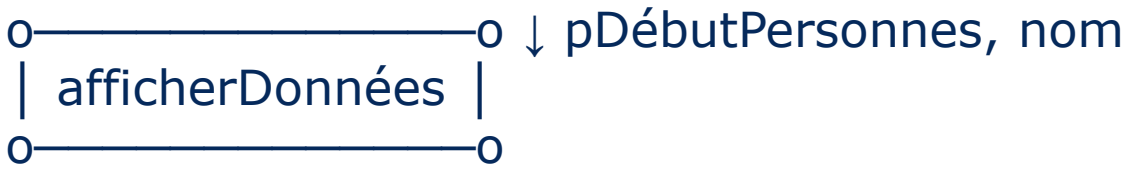
3.3.2. Recherche dans une liste simple triée

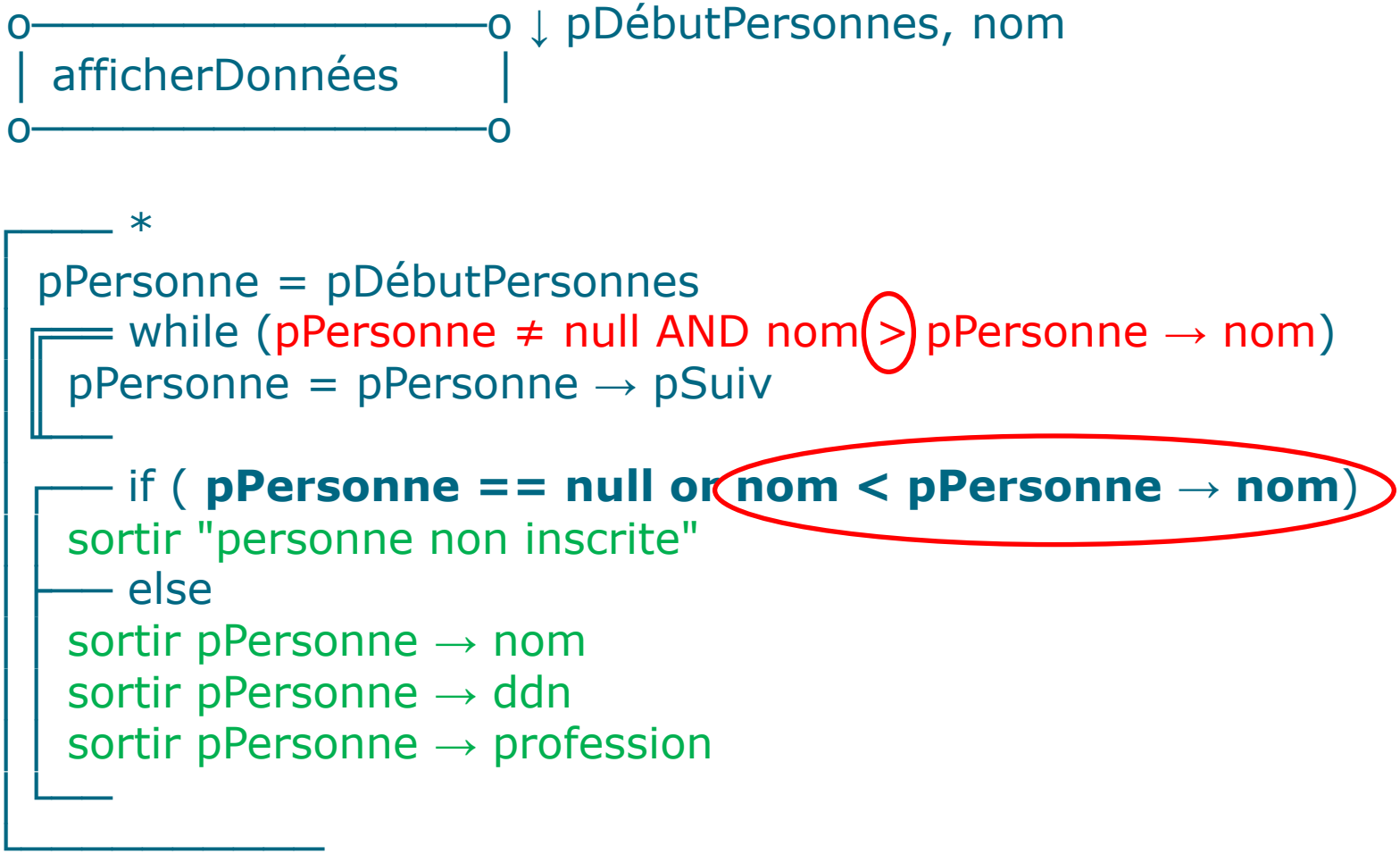
Exemple

Soit une liste chainée dont chaque chainon concerne une personne inscrite à une formation et contient le nom, la date de naissance et la profession de la personne ainsi qu'un pointeur vers le chainon suivant (**pSuiv**)..
Les chainons sont ***triés par ordre alphabétique sur le nom.***

L'adresse du premier chainon se trouve dans **pDébutPersonnes**.

Ecrire le module qui reçoit le nom d'une personne et qui affiche ses données ou, si elle n'est pas présente dans la liste, le message d'erreur « personne non inscrite »




afficherDonnées
↓ pDébutPersonnes, nom
*
pPersonne = pDébutPersonnes
while (pPersonne ≠ null AND nom > pPersonne → nom)
pPersonne = pPersonne → pSuiv
if (pPersonne == null or nom < pPersonne → nom)
sortir "personne non inscrite"
else
sortir pPersonne → nom
sortir pPersonne → ddn
sortir pPersonne → profession

Module 3: Listes chainées

Exercice 1

Soit une liste chainée simple pointée par **pDébutParties** et dont chaque chainon concerne une partie d'un jeu jouée par un joueur et contient

- le nom du jeu (**nomJeu**)
- le nom du joueur (**nomJoueur**)
- son score (**score**)
- un pointeur vers le chainon suivant (**pSuiv**).

La liste est triée par ***ordre alphabétique*** sur le **nom du jeu** et pour un même jeu, par ***ordre alphabétique*** sur le **nom du joueur**.

Ecrire le module qui, à partir de la liste, affiche

- pour chaque jeu: le nom du jeu et le nombre de joueurs
- le nombre de jeux différents

Module 3: Listes chainées

3.3.3. Ajout d'un nouveau chainon dans une liste chainée simple

- Ajout en début de liste
- Ajout en fin de liste
- Insertion dans une liste triée

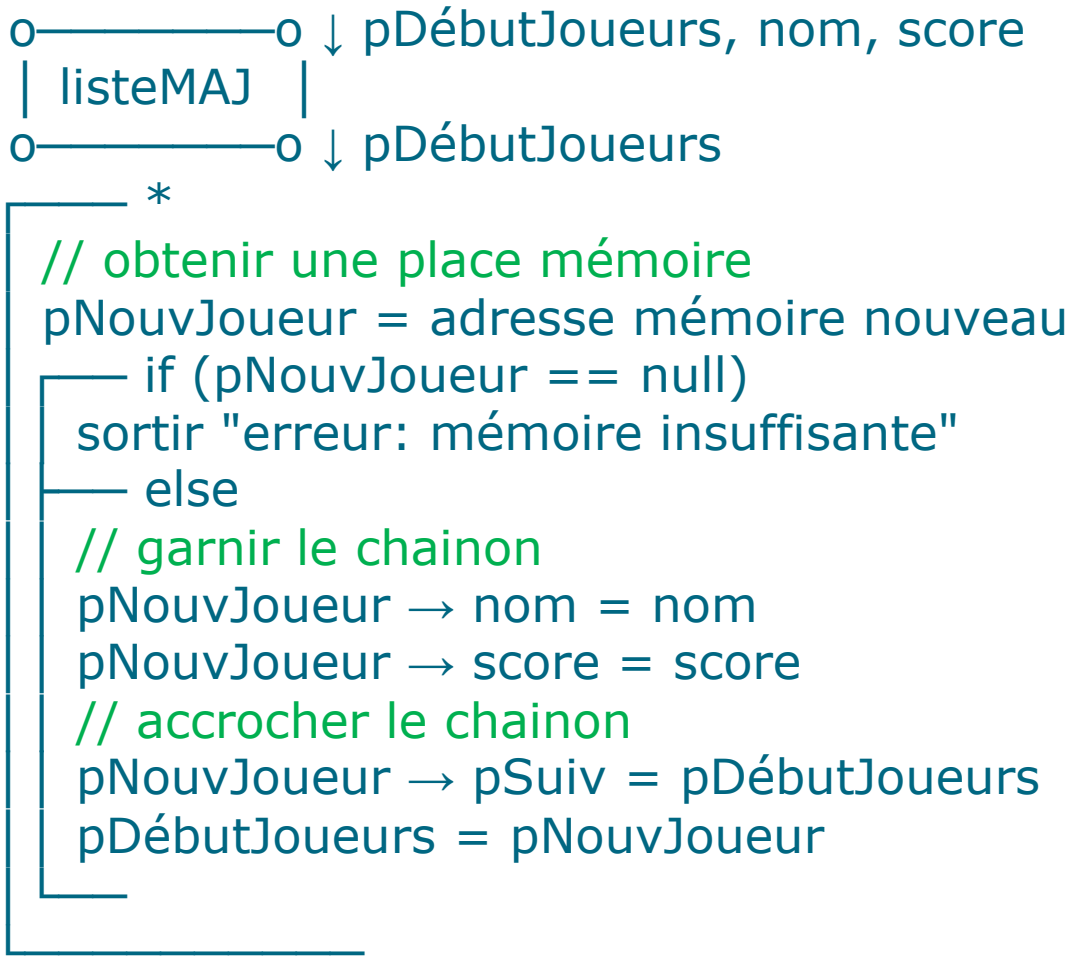
Ajout en début de liste

Soit une liste chainée simple pointée par **pDébutJoueurs** et dont chaque chainon concerne un joueur et contient

- le nom du joueur (**nom**)
- son score (**score**)
- un pointeur vers le chainon suivant (**pSuiv**).

Ecrire le module qui reçoit les données d'un nouveau chainon et l'ajoute **en début de liste**.

Traiter le cas de la mémoire insuffisante en affichant le message adéquat.



Ce module est-il correct dans le cas où la liste est vide au départ?

Ajout en fin de liste

Soit une liste chainée simple non vide pointée par **pDébutJoueurs** et dont chaque chainon concerne un joueur et contient

- le nom du joueur (**nom**)
- son score (**score**)
- un pointeur vers le chainon suivant (**pSuiv**).

Ecrire le module qui reçoit les données d'un nouveau chainon et l'ajoute en fin de liste.

↓ pDébutJoueurs, nom, score

```
*
// obtenir une place mémoire
pNouvJoueur = adresse mémoire nouveau chainon
if (pNouvJoueur == null)
  sortir "erreur: mémoire insuffisante"
else
  // garnir le chainon
  pNouvJoueur → nom = nom
  pNouvJoueur → score = score
  // parcourir la liste pour se placer en fin de liste
  pJoueur = pDébutJoueurs
  while (pJoueur ≠ null)
    pJoueurPrécédent = pJoueur
    pJoueur = pJoueur →  pSuiv

  // accrocher le chainon
  pJoueurPrécédent → pSuiv = pNouvJoueur
  pNouvJoueur → pSuiv  = null
```

Ce module est-il correct dans le cas où la liste est vide au départ?

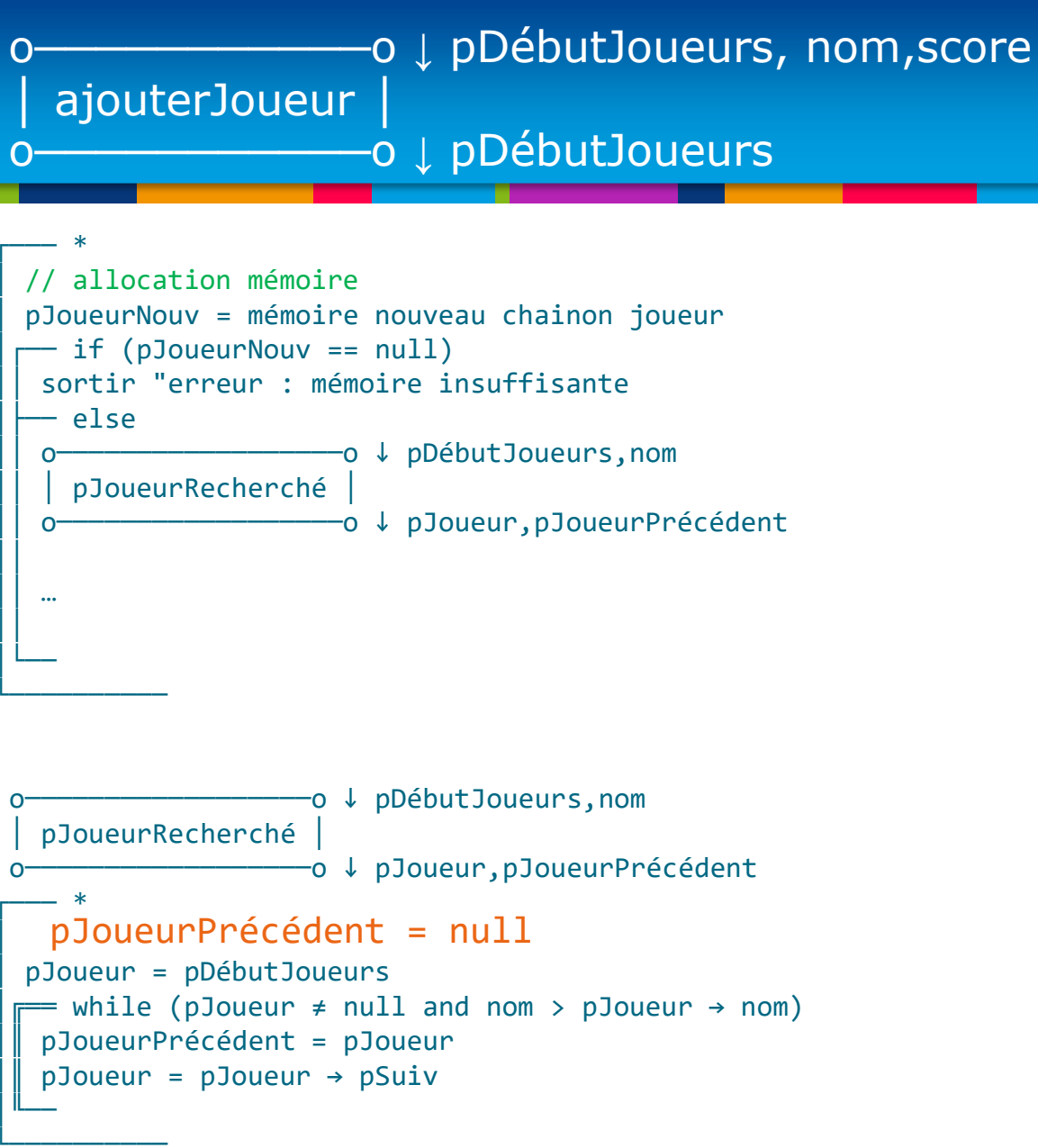
Ajout dans une liste triée

Soit une liste chainée simple pointée par **pDébutJoueurs** et dont chaque chainon concerne un joueur et contient

- le nom du joueur (**nom**)
- son score (**score**)
- un pointeur vers le chainon suivant (**pSuiv**).

Cette liste est **triée par ordre alphabétique** sur le nom du joueur.

Ecrire le module qui reçoit les données d'un nouveau chainon et l'ajoute au bon endroit dans la liste.


o——————o ↓ pDébutJoueurs, nom,score
| ajouterJoueur |
o——————o ↓ pDébutJoueurs
*
// allocation mémoire
pJoueurNouv = mémoire nouveau chainon joueur
if (pJoueurNouv == null)
sortir "erreur : mémoire insuffisante
else
o——————o ↓ pDébutJoueurs,nom
| pJoueurRecherché |
o——————o ↓ pJoueur,pJoueurPrécédent
…
o——————o ↓ pDébutJoueurs,nom
| pJoueurRecherché |
o——————o ↓ pJoueur,pJoueurPrécédent
*
pJoueurPrécédent = null
pJoueur = pDébutJoueurs
while (pJoueur ≠ null and nom > pJoueur → nom)
pJoueurPrécédent = pJoueur
pJoueur = pJoueur → pSuiv


```
┌── *
│ // allocation mémoire
│ pJoueurNouv = mémoire nouveau chainon joueur
│┌── if (pJoueurNouv == null)
│ sortir "erreur : mémoire insuffisante
│├── else
│ o──────────────────o ↓ pDébutJoueurs,nom
│ │ pJoueurRecherché │
│ o──────────────────o ↓ pJoueur,pJoueurPrécédent
│ // garnir le chainon
│ pJoueurNouv → nom = nom
│ pJoueurNouv  → score = score
│ // attacher le chainon
│
│ …
│
│└──
└──────────
```


```
┌─── *
│ // allocation mémoire
│ pJoueurNouv = mémoire nouveau chainon joueur
│ ┌── if (pJoueurNouv == null)
│ │ sortir "erreur : mémoire insuffisante
│ ├── else
│ │ pDébutJoueurs,nom
│ │ pJoueurRecherché
│ │ pJoueur,pJoueurPrécédent
│ │ // garnir le chainon
│ │ pJoueurNouv → nom = nom
│ │ pJoueurNouv  → score = score
│ │ // attacher le chainon
│ │ pJoueurNouv → pSuiv = pJoueur
│ │ ┌── if (pJoueur == pDébutJoueurs) // ajout devant ou liste vide
│ │ │ pDébutJoueurs = pJoueurNouv
│ │ ├── else     //ajout milieu ou fin
│ │ │ pJoueurPrécédent → pSuiv = pJoueurNouv
│ │ └──
│ └──
└──────────
```

Module 3: Listes chainées

Exercice 2

Soit une liste chainée simple pointée par **pDébutJeux** et dont chaque chainon retient

- le nom d'un jeu (**nomJeu**)
- un pointeur vers une liste chainée simple de joueurs à ce jeu (**pDébutJoueurs**)
- un pointeur vers le chainon jeu suivant (**pSuiv**).

La liste est triée par *ordre alphabétique* sur le nom du jeu.

Chaque chainon des listes des joueurs retient

- le nom d'un joueur (**nomJoueur**)
- le score du joueur (**score**)
- un pointeur vers le joueur suivant (**pSuiv**).

Ces listes sont triées par *ordre alphabétique* sur le nom du joueur.

Module 3: Listes chainées

Exercice 2

Soit une liste chainée simple pointée par **pDébutJeux** et dont chaque chainon retient

- le nom d'un jeu (**nomJeu**)
- un pointeur vers une liste chainée simple de joueurs à ce jeu (**pDébutJoueurs**)
- un pointeur vers le chainon jeu suivant (**pSuiv**).

La liste est triée par *ordre alphabétique* sur le nom du jeu.

Chaque chainon des listes des joueurs retient

- le nom d'un joueur (**nomJoueur**)
- le score du joueur (**score**)
- un pointeur vers le joueur suivant (**pSuiv**).

Ces listes sont triées par *ordre alphabétique* sur le nom du joueur.

1. Ecrire le module qui reçoit un nom de jeu et qui affiche les noms des joueurs à ce jeu.
2. Ecrire le module qui reçoit un nom de jeu et qui ajoute ce jeu à la liste;
   NB: ne pas traiter le cas de la mémoire insuffisante.
3. Compléter le module précédent en lui fournissant en plus le nom et le score d'un joueur et en ajoutant ce joueur dans la liste.

3.3.4. Suppression d'un chainon dans une liste chainée simple

Soit une liste chainée simple pointée par **pDébutDevis** et dont chaque chainon concerne un devis pour un travail de peinture et retient

- le nom de l'entreprise (**nom**)
- Le montant du devis (**montant**)
- un pointeur vers le chainon suivant (**pSuiv**).

Cette liste est **triée par ordre alphabétique** sur le nom de l'entreprise.

Une entreprise a fait faillite. Ecrire le module qui reçoit le nom de cette entreprise et qui supprime son devis de la liste.