UE 159
Organisation et exploitation des données

Année académique 2023-2024

Contenu

- Exercice récapitulatif sur les tableaux de structures
- Module 1: Introduction
- Module 2: Tableaux - Compléments
  - Traitement des tableaux triés
  - Bloc logique
- Module 3: Listes chaînées
- Module 4: Piles et files
- Module 5: Arbres
- Module 6: Tables de hachage

Module 1: Introduction

Organisation

Le principe de base d'une *structure de données* est de stocker en mémoire des données auxquelles le programmeur veut pouvoir accéder plus tard et sur lesquelles il veut pouvoir effectuer des opérations (lecture, mise à jour, suppression, insertion…).

Exploitation

Notion de coût (= complexité) d'un algorithme

- Complexité en temps
- Complexité en espace

Module 1: Introduction

Exemple :

Considérons un ensemble de données concernant des clients, réparties en

- nom
- rue
- ville
- profession
- téléphone

…

A chaque usage, correspond une structure appropriée

=> un traitement plus rapide et plus efficace.

Utiliser une structure de données appropriée permet de diminuer :

- **La complexité d'une application informatique,**
- **Le taux d'erreurs.**

Types de structures de données(collections)

- Séquentielles (ou linéaires) : on peut y ranger les objets dans un ordre arbitraire
  - tableaux ;
  - listes chainées ;
  - piles ;
  - files ;
- Mappées : on y range les objets en fonction d'une clé
  - tables de hachage ;
  - arbres binaires de recherche ;
  - B-trees;
- Autres : les graphes

Module 2

Tableaux : Compléments

2.1. Définition

Structure de données (ou collection)

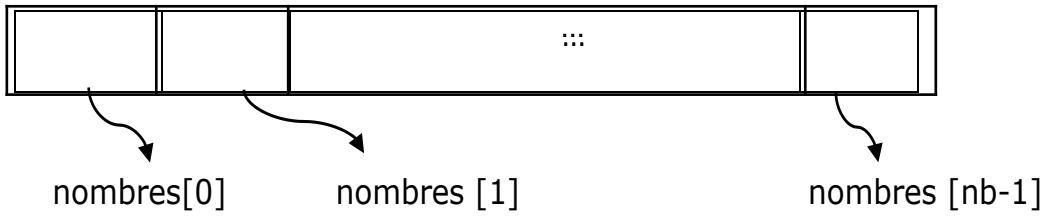
- homogène
  - tous les éléments sont de même type
- séquentielle
  - tous les éléments sont consécutifs en mémoire
- indicée
  - accès direct à un élément via un indice
- de taille fixe

- triée ou non selon les besoins

2.1.1. Tableau à une dimension

- La forme de tableau la plus simple est le vecteur.
- On accède à chaque élément du tableau (cellule) via son *indice*
- L'espace mémoire est alloué de façon statique car la longueur du tableau est fixe.
- Chaque cellule peut contenir un ou plusieurs champs dont certains peuvent être eux-mêmes des structures de données

nombres : Tableau de nb cellules, chacune contenant un nombre



2.1.1. Tableau à une dimension

Exemples:

- **nombres** | cellule (**nb***) | chaque cellule contient un nombre
- **achats** | cellule (**nbAchats***) | date, nomClient, montant
- **étudiants** | cellule (100*) | nom, section, année, **cotes** | cellule (5*) | intituléCours, points

2.1.2. Tableau à une dimension trié

- Les éléments (cellules) sont ordonnés suivant une relation d'ordre (alphabétique, croissant, chronologique…) **sur un champ**

- Les algorithmes de recherche vont varier selon que le tableau est trié ou non

- Les opérations d'**insertion** nécessiteront, si la taille du tableau le permet, une recherche de la position puis un décalage des cellules dans le cas d'un tableau **trié** alors que dans le cas d'un tableau **non trié**, l'ajout peut se faire en fin de tableau

- En cas d'ajout d'un paquet de données assez conséquent dans un tableau trié, on peut ajouter les données en fin de tableau puis trier le tableau

2.2. Algorithmes de recherche

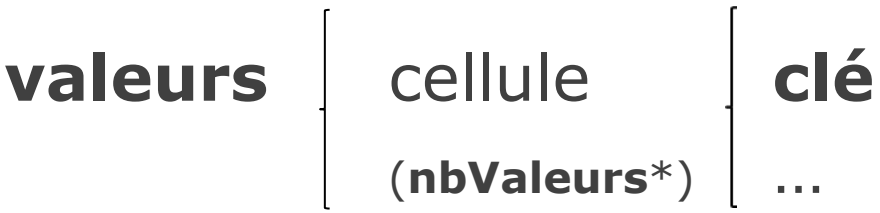
2.2.1. Recherche dans un tableau non trié

Dans ce type de tableau, la seule recherche possible est une recherche séquentielle.

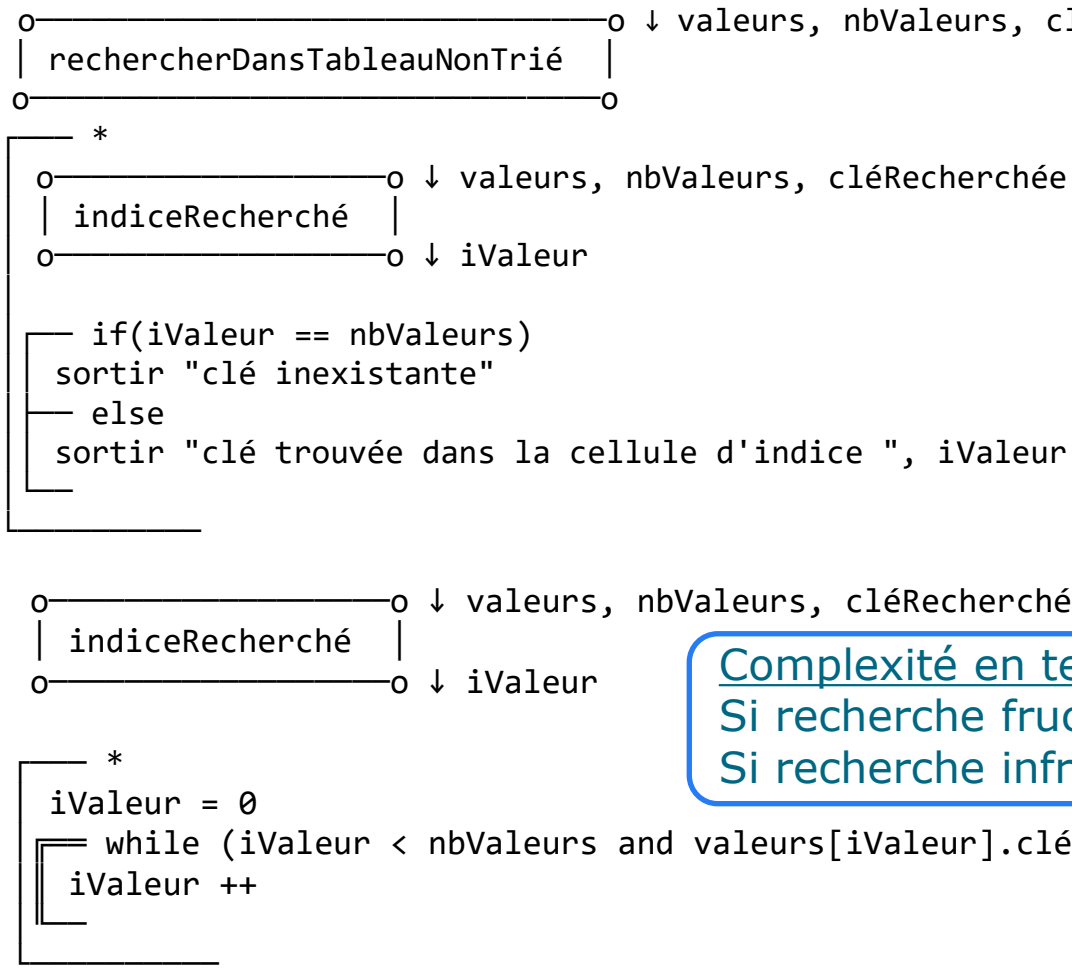
Soit un tableau **valeurs** de **nbValeurs** cellules dont un champ s'appelle **clé**.

Ecrire le module qui reçoit une valeur de clé (**cléRecherchée**) et qui affiche le message adéquat selon que la clé a été trouvée ou non.

Description



2.2. Algorithmes de recherche



Complexité en temps :
Si recherche fructueuse : (nbValeurs + 1) / 2
Si recherche infructueuse : nbValeurs

2.2. Algorithmes de recherche

2.2.2. Recherche séquentielle dans un tableau trié

Soit un tableau **valeurs** de **nbValeurs** cellules dont un champ s'appelle **clé**, ce tableau étant **trié en ordre croissant sur la clé**.

Ecrire le module qui reçoit une valeur de clé (**cléRecherchée**) et qui affiche le message adéquat selon que la clé a été trouvée ou non.

Description

valeurs [cellule (**nbValeurs***) [clé (↗) …]

2.2. Algorithmes de recherche

```
o──────────────────────────o  ↓ valeurs, nbValeurs, cléRecherchée
| rechercherDansTableauTrié |
o──────────────────────────o
 ┌── *
 | o──────────────────o ↓ valeurs, nbValeurs, cléRecherchée
 | | indiceRecherché  |
 | o──────────────────o ↓ iValeur
 |
 | ┌── if(iValeur == nbValeurs or cléRecherchée ≠ valeurs[iValeur].clé )
 | | sortir "clé inexistante"
 | ├── else
 | | sortir "clé trouvée dans la cellule d'indice ", iValeur
 | └──
 └──────────
```

```
o──────────────────o ↓ valeurs, nbValeurs, cléRecherchée
| indiceRecherché  |
o──────────────────o ↓ iValeur

 ┌── *
 | iValeur = 0
 | ╔══ while (iValeur < nbValeurs and cléRecherchée > valeurs[iValeur].clé)
 | ╚═| iValeur ++
 |   └──
 └──────────
```

Complexité en temps :
Si recherche fructueuse : (nbValeurs + 1) / 2
Si recherche infructueuse : (nbValeurs + 1) / 2

2.2. Algorithmes de recherche

2.2.3. Recherche dichotomique dans un tableau trié

Soit un tableau **valeurs** de **nbValeurs** cellules dont un champ s'appelle **clé**, ce tableau étant **trié en ordre croissant sur la clé**.

Ecrire le module qui reçoit une valeur de clé (**cléRecherchée**) et qui affiche le message adéquat selon que la clé a été trouvée ou non.

2.2. Algorithmes de recherche

```
o──────────────────────────────o ↓ valeurs, nbValeurs, cléRecherchée
│ rechercherMéthodeDichotomique │
o──────────────────────────────o
┌── *
│ borneInf = 0
│ borneSup = nbValeurs - 1
│
│ iMilieu= [(borneInf + borneSup)/2]ENT
│
│╔═ while (borneInf ≤ borneSup and valeurs[iMilieu].clé ≠ cléRecherchée)
│║
│║ ┌── if (cléRecherchée < valeurs[iMilieu].clé ))
│║ │ borneSup = iMilieu - 1
│║ ├── else
│║ │ borneInf = iMilieu + 1
│║ └──
│║ iMilieu= [(borneInf + borneSup)/2]ENT
│╚═══
│
│ ┌── if(borneInf > borneSup)
│ │ sortir "clé inexistante"
│ ├── else
│ │ sortir "clé trouvée dans la cellule d'indice ", iMilieu
│ └──
└──────────
```

Complexité en temps :
Si recherche fructueuse : $\log_2 (nbValeurs - 1)$
Si recherche infructueuse : $\log_2 nbValeurs + 1$
si nbValeurs = 1000 -> 10 et 11

Ajout dans un tableau trié: Exercice

Un tableau **étudiants** est constitué de **nbEtud** cellules.
Chaque cellule concerne un étudiant et contient son **nom**, son **groupe** et son **pourcentage** à l'issue de la session d'examens de janvier.

Ce tableau est **trié** par *ordre décroissant sur le pourcentage*.

Ecrivez le module permettant d'obtenir le nom, le groupe et le pourcentage d'un nouvel étudiant et d'insérer ces données au bon endroit dans le tableau.

Ajout dans un tableau trié: Exercice

Structure du tableau :

$$\text{étudiants} \begin{cases} \text{cellule} \\ (\text{nbÉtud} \ *) \end{cases} \begin{cases} \text{nom} \\ \text{groupe} \\ \text{pourcentage} \searrow \end{cases}$$

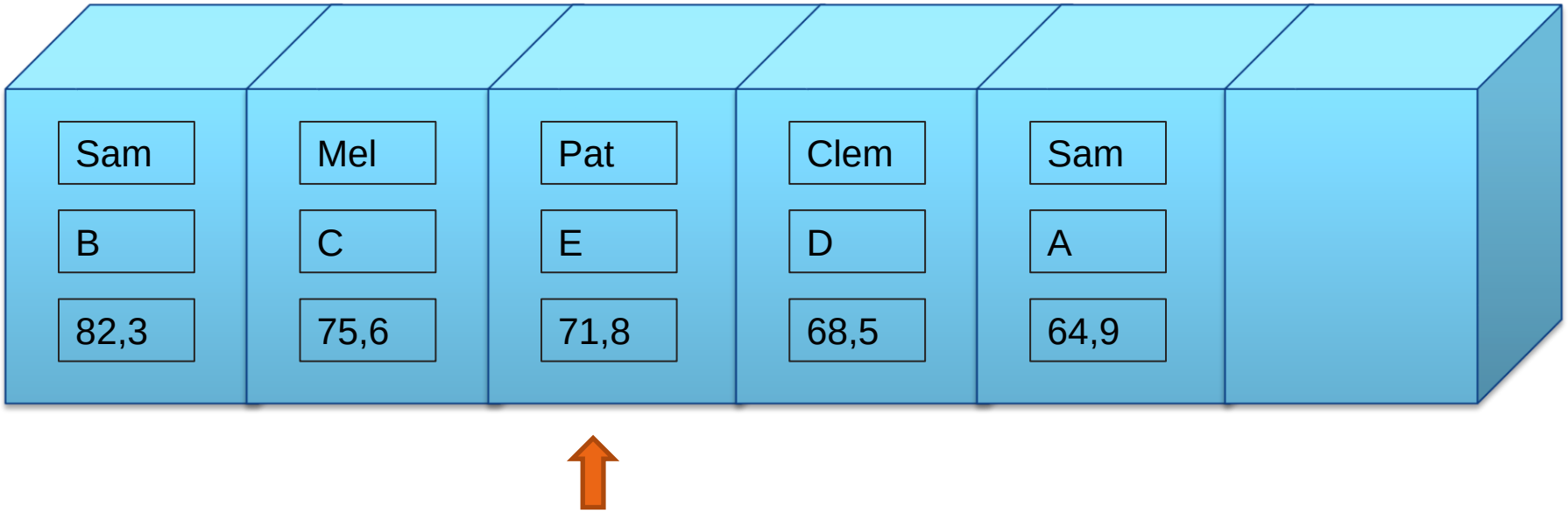
Entrées

- nom
- groupe
- pourcentage

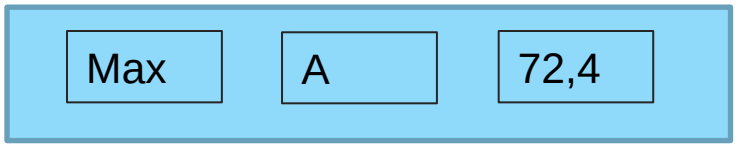
Ajout dans un tableau trié: Exercice

étudiants

nbÉtud : 5



Entrées

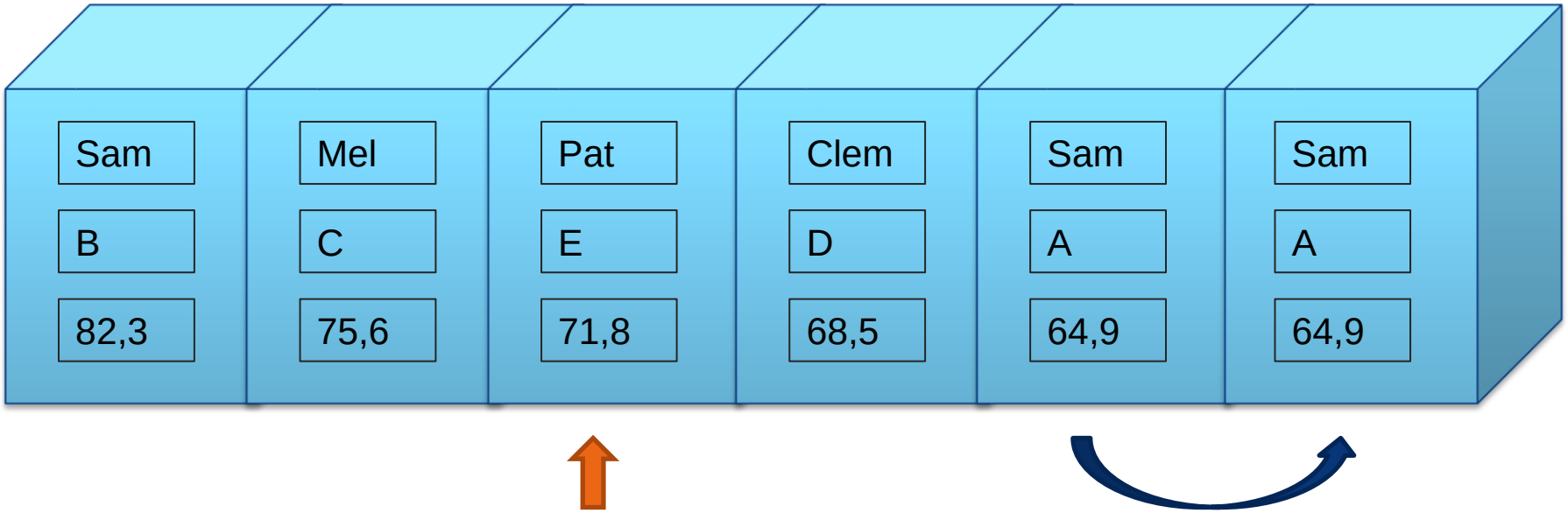


iNouvÉtud : 2

Ajout dans un tableau trié: Exercice

étudiants

nbÉtud : 5



Entrées

Max	A	72,4

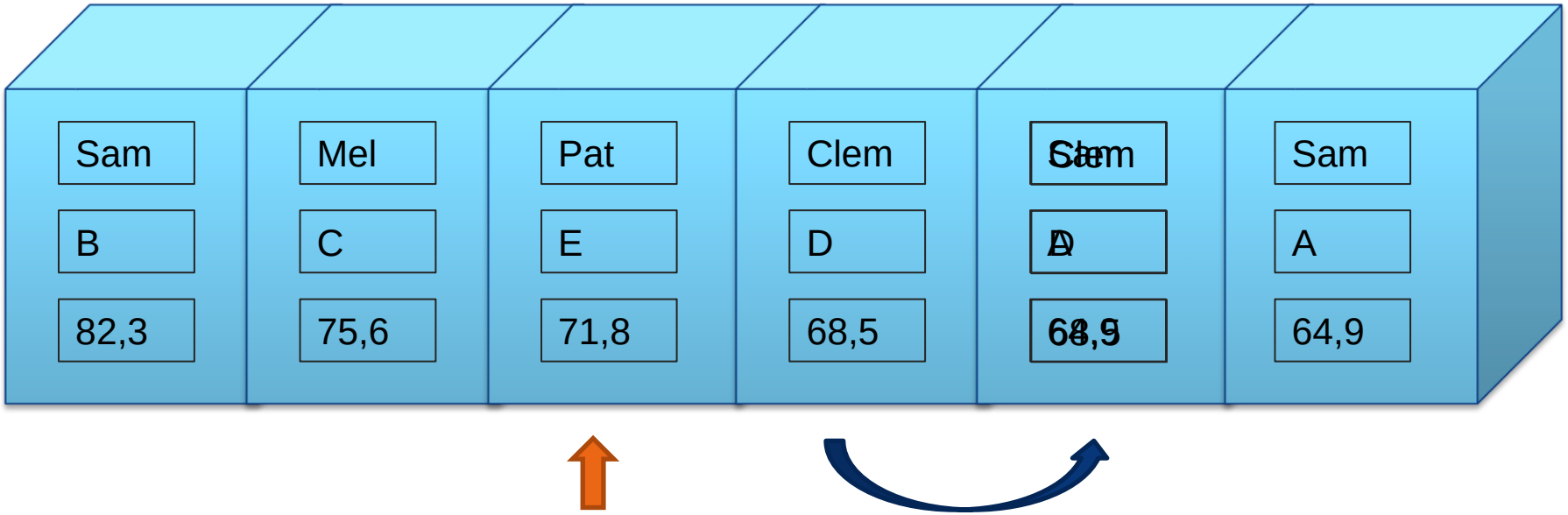
étudiants[5] = étudiants[4]

iNouvÉtud : 2

Ajout dans un tableau trié: Exercice

étudiants

nbÉtud : 5



Entrées

Max	A	72,4

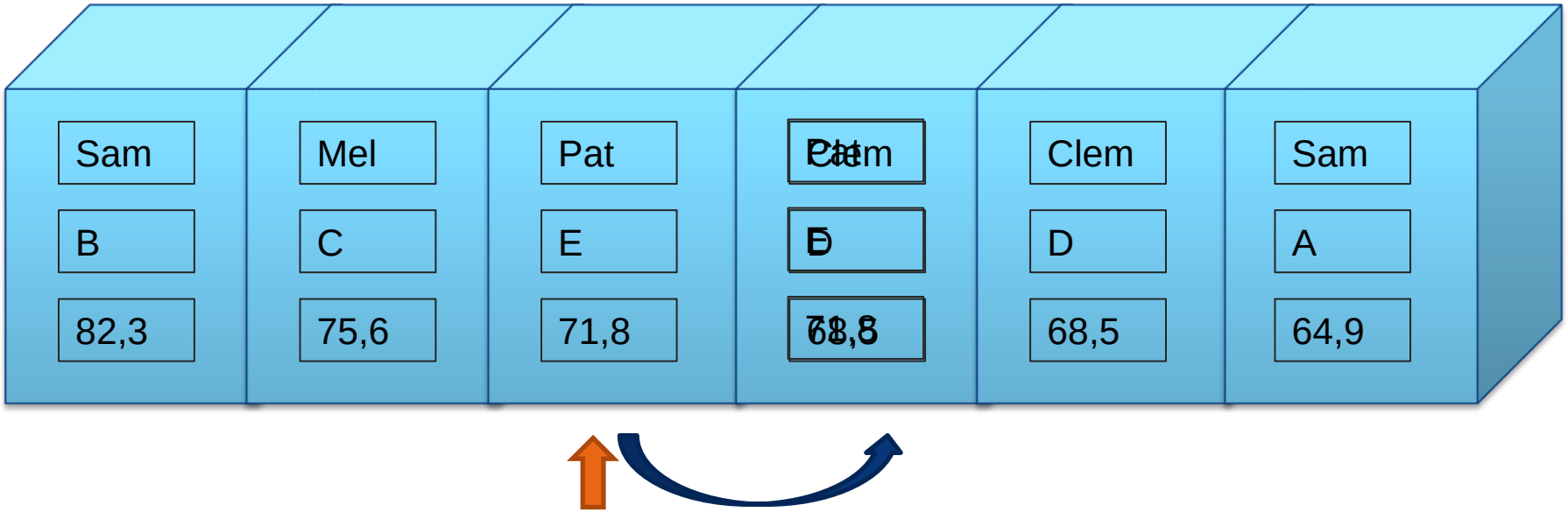
```
étudiants[4] = étudiants[3]
```

iNouvÉtud : 2

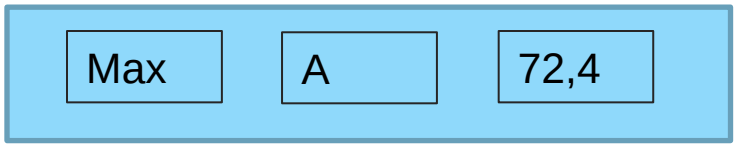
Ajout dans un tableau trié: Exercice

étudiants

nbÉtud : 5



Entrées



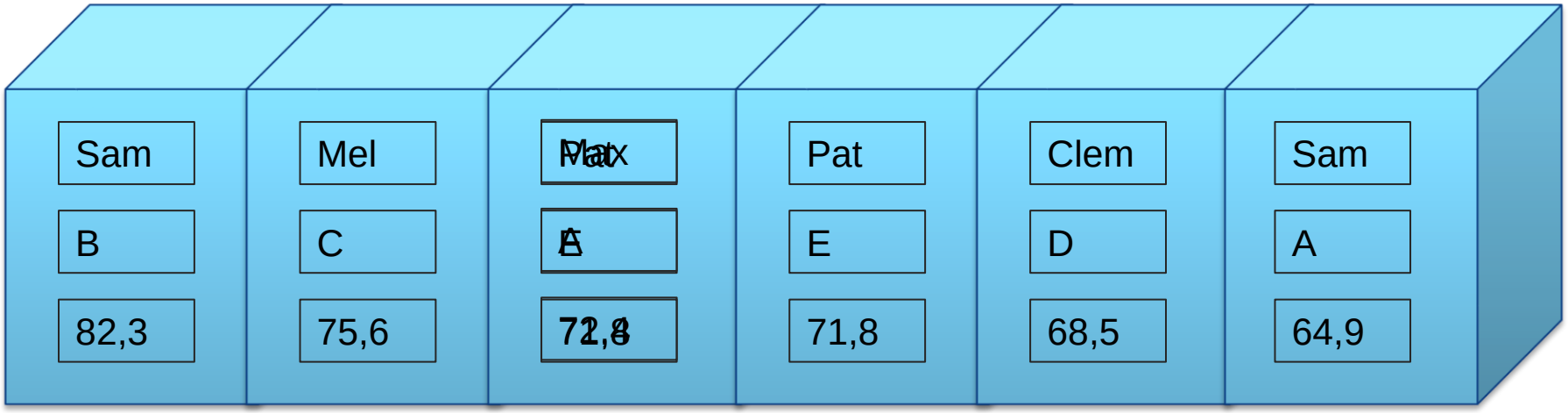
étudiants[3] = étudiants[2]

iNouvÉtud : 2

Ajout dans un tableau trié: Exercice

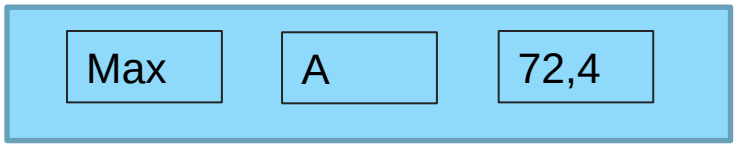
étudiants

nbÉtud : 6



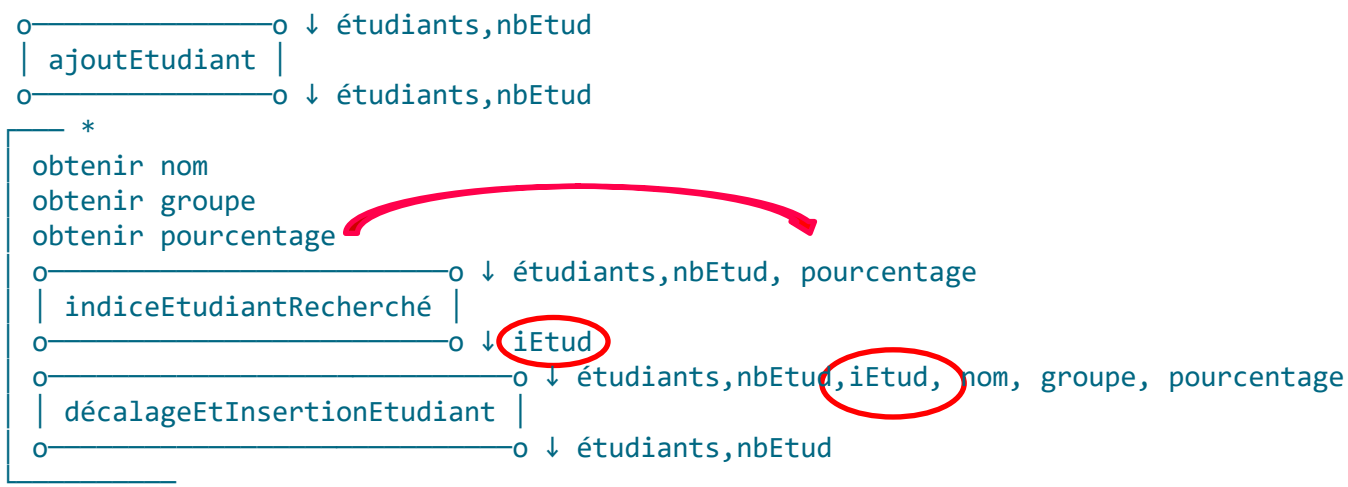

```
étudiants[2].nom = nom
étudiants[2].groupe = groupe
étudiants[2].pourcentage = pourcentage
```

Entrées



iNouvÉtud : 2

Ajout dans un tableau trié: Exercice



Ajout dans un tableau trié: Exercice

```
o──────────────────────────o ↓ étudiants,nbEtud, pourcentage
│ indiceEtudiantRecherché │
o──────────────────────────o ↓ iEtud
┌── *
│ iEtud = 0
│╔═ while (iEtud < nbEtud and pourcentage < étudiants[iEtud].pourcentage)
│╚═ iEtud ++
└──────────
```

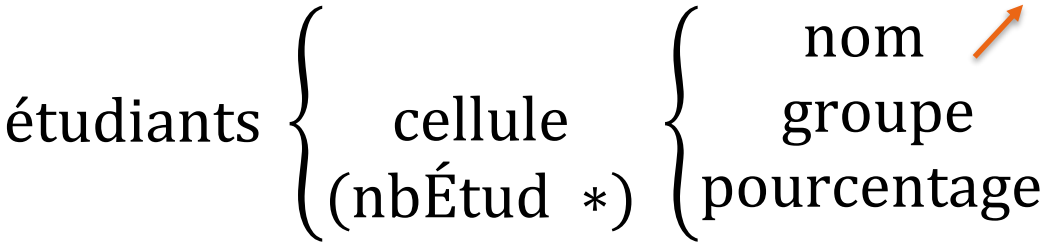
```
o──────────────────────────────o ↓ étudiants,nbEtud,iEtudNouv, nom , groupe, pourcentage
│ décalageEtInsertionEtudiant │
o──────────────────────────────o ↓ étudiants,nbEtud
┌── *
│ iEtud = nbEtud
│╔═ while (iEtud > iEtudNouv)
│╚═ étudiants[iEtud] = étudiants[iEtud - 1]
│   iEtud --
│
│ étudiants[iEtud].nom = nom
│ étudiants[iEtud].groupe = groupe
│ étudiants[iEtud].pourcentage = pourcentage
│ nbEtud ++
└──────────
```

Ou iEtudNouv
car ici iEtud == iEtudNouv

Suppression dans un tableau trié : Exercice

Ecrire le module qui reçoit le nom d'un étudiant et qui supprime cet étudiant du tableau **étudiants** décrit ci-dessous.

Prévoir le cas d'erreur où l'étudiant n'est pas présent dans le tableau en affichant un message d'erreur.



Entrées (param): nom de l'étudiant à supprimer

Suppression dans un tableau trié : Exercice