Briques de base pour l'algorithmique

1] Représentation des séquences de données
↳ notion d'ordre _ notion de position.

2 grandes familles → listes et les tableaux

types de données ≠ structures de données
↳ des objets munis d'opérations ↳ comment on les implémente sur une machine.

ex: type Ensemble d'entiers
op: - union de 2 ens.
- le test d'appartenance
- intersection
- ajouter un élt de E.
- supprimer un élément de E

Ex: Type entier
. repres. des nb.
. +, /, *, −, ...

- Listes: (type de données)
  - Ajouter un élément, supprimer un élément,
  - longueur, concaténer 2 listes
  - accéder au $i^{ème}$ élément. $1 \leq i \leq |l|$
  - insérer un n^{el} élément en position i.
  - parcourir la liste
  - créer une liste, détruire.

ex: Pile → . empiler (ajouter au début)
. dépiler (supprimer le début ~~et le récupérer~~).
LIFO . Tete, sommet
. test du vide

(Last In First Out)

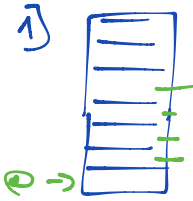
- créer une file vide.

File:
- ajout (en début de liste)
- suppression en fin

FIFO
First In ...

- la tête de file
- test du vide
- File vide.

Pour tous ces objets, on peut considérer
2 repres. possibles → représentation contiguës (tableau)
représentation par chaînages



→ 1 case = pour stocker 1 élé :

Avantage tab:
- accès en tps cstant à toutes les cases du tableau/liste
- bénéficier du cache.
- compact en espace mémoire.

Désavantages:
- taille limitée, FIXÉE.
- insertion d'un élément peut être coûteuse (car nécessitant un décalage)

Représnt. par chaînage :

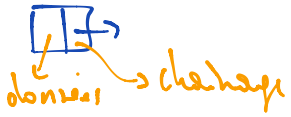
Liste chaînée:

Avantages:
- pas de limite de taille (autre que celle de l'ordi.).
- l'ajout, l'insertion, la suppression → facile

Désavantages - pas d'accès direct au $i^{ème}$ élément
- espace mémoire perdu par les pointeurs

Créer et manipuler des SdD chaînées:

- accéder aux champs de données des $\mathcal{C}$.
- créer et détruire des $\mathcal{C}$.



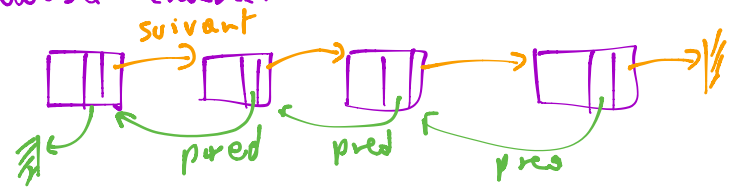
Soit c l'adresse d'une $\mathcal{C}$.
c->d : accès au champ d de la cellule.

Alloc(type cellule) -> allouer de la mem. pour stocker une $\mathcal{C}$ de ce type. Retourne l'@ de la zone

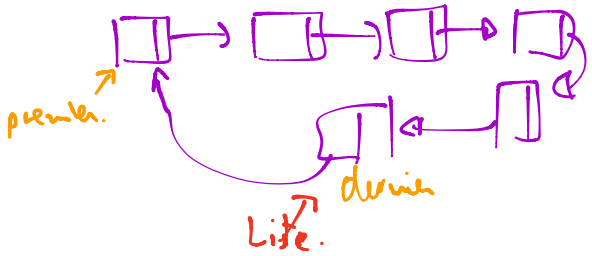
Elim(c) : désallouer l'esp. mémoire.

↑ liste = l'adresse de la première cellule.

- liste doublement chaînée:



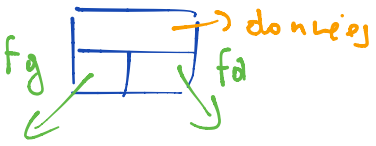
- liste circulaire.



Structures arborescentes :

→ Arbre binaire avec chaînages.

1 arbre = l'@ de la racine.



fg = fils gauche.
fd = fils droit

Mêmes op. : champ de données $x \to d$.
@ d'un nœud.
0 si arbre vide.
+ les liens vers les sous-arbres.

Algorithme de parcours :

Visit(x) ← @ d'un nœud

Si $x == 0$ Alors "traitement arbre vide"
Sinon : traitement de x n°1
Visit ($x \to fg$)
traitement de x n°2
Visit ($x \to fd$)
traitement de n°3