TP 1

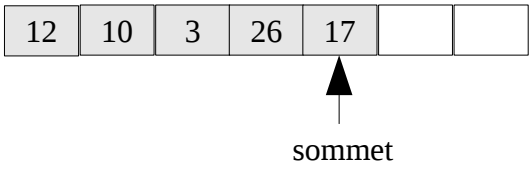
Structures de données élémentaires

Récupération du code source
Téléchargez le code source utilisé pour ce TP à partir de l'adresse suivante : http://esisar.net/cs316/tp1.tar.gz
Décompressez le fichier téléchargé en utilisant la commande suivante : `tar -xzvf tp1.tar.gz`
Un squelette de fichiers est proposé pour chaque exercice. Utiliser la commande `make` pour compiler chaque exercice.
A rendre
A la fin de la séance, le code source développé durant la séance de TP (zippé) doit être déposer sur *chamilo*.
La solution finale est à déposer sur *chamilo* dans une semaine (Date limite : Dimanche 8 décembre à 23h59).

http://esisar.net/cs316/chamilo dossier Séance 1 – Groupe TP*

Exercice 1

Une pile met en œuvre le principe *dernier entré, premier sorti*, ou *LIFO* (Last-In, First-Out).



Soit la structure de données suivante qui représente une *pile* d'entiers *positifs* (voir `tp1/exo1/pile.h`) :

```
#define MAX 7

struct pile {
   int valeurs[MAX]; /* les donnees stockees dans la pile */
   int sommet;       /* le sommet courant de la pile */
};
typedef struct pile Pile;
```

a) Écrire en C l'implémentation des fonctions suivantes (dans le fichier `tp1/exo1/pile.c`) :

- `void init(Pile **p)` initialiser la pile.
- `int empiler(Pile *p, int d)` insérer une nouvelle valeur dans la pile. Retourner -1 si débordement, ou 0 dans le cas normal.
- `int depiler(Pile *p)` supprimer une valeur de la pile et la retourner. Retourner -1 si la pile est vide.
- `int est_vide(Pile *p)` retourner 1 si la pile est vide, -1 sinon.
- `int est_plein(Pile *p)` retourner 1 si la pile est pleine, -1 sinon.

- `Int sommet(Pile *p)` retourner la valeur en sommet de la pile.
- `void afficher(Pile *p)` afficher sur le terminal le contenu de la pile.

En cas de débordement de la pile, des messages appropriés doivent être affichés à l'utilisateur.

b) Écrire un programme qui test l'ensemble des fonctions de la pile (dans le fichier `tp1/exo1/exo1.c`).

c) Ajouter une fonction qui permet de savoir si un entier est dans une pile. Si l'entier n'est pas dans la pile, la fonction retourne -1, sinon elle retourne le nombre d'extractions nécessaires pour obtenir sa sortie. S'il existe plusieurs occurrences du même entier dans la pile, on prend la première trouvée.

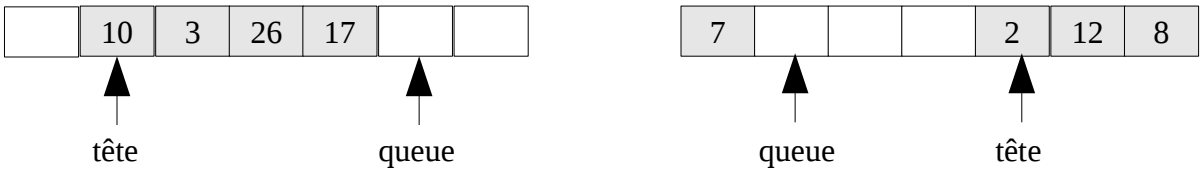
```
int existe(Pile *p, int d)
```

d) Ajouter une fonction qui permet de supprimer la première occurrence d'un entier dans une pile (utiliser une autre pile pour des opérations temporaires).

```
void supprimer(Pile *p, int d)
```

Exercice 2

Une file est accessible par une tête et une queue. Les éléments sont ajoutés en queue et retirés en tête (si elle n'est pas vide).



a) Proposer une implémentation en C d'une file d'entiers avec une représentation sous forme d'un tableau de taille suffisante (`tp1/exo2/file.h`). Les fonctions possibles (définies dans `tp1/exo2/file.h` et à implémenter dans `tp1/exo2/file.c`) sont :

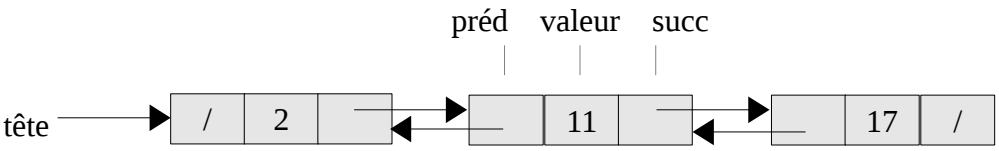
- `void init(File **f)` initialiser la file.
- `int taille(File *f)` retourner la taille de la file.
- `void enfiler(File *f, int d)` enfiler un entier.
- `int defiler(File *f)` défiler, retourner l'élément en tête de file.
- `int est_vide(File *f)` retourner 1 si la file est vide, 0 sinon.
- `int est_plein(File *f)` retourner 1 si la file est pleine, 0 sinon.
- `void afficher(File *f)` afficher sur le terminal le contenu de la file.

c) Écrire un programme qui test l'ensemble des fonctions de la file (dans le fichier `tp1/exo2/exo2.c`).

d) [*Bonus*] Proposer une nouvelle implémentation de la file à l'aide de deux piles.

Exercice 3

Contrairement au tableau, pour lequel l'ordre linéaire est déterminé par les indices, une liste chaînée est une structure de données où l'ordre est déterminé par un pointeur dans chaque objet. La figure suivante montre un exemple d'une liste doublement chaînée. Elle contient un ensemble d'*éléments*, où chaque *élément* contient les attributs *préd* (pointeur vers l'élément prédécesseur), *succ* (pointeur vers l'élément successeur) et *valeur* (la valeur stockée).



a) Proposer une structure de données en C pour une liste d'entiers doublement chaînée et triée.

b) Implémenter les fonctions suivantes :

- INSERER_ELEMENT(L, e) qui permet l'insertion d'un élément *e* déjà initialisé dans la bonne place de la liste chaînée triée.
- INSERER(L, i) qui permet d'ajouter un entier *i* à la liste.
- RECHERCHE(L, i) qui trouve le premier élément de valeur *i* dans la liste *L* et retourne un pointeur sur cet élément. Si aucun objet de valeur *i* n'apparaît dans la liste, la procédure retourne *NULL*.
- SUPPRIMER_ELEMENT(L, e) qui permet de supprimer l'élément *e* de la liste chaînée *L*.
- SUPPRIMER_TOUT(L, i) qui permet de supprimer toutes les occurrences de l'entier *i* dans la liste *L*.

c) Écrire un programme de test.