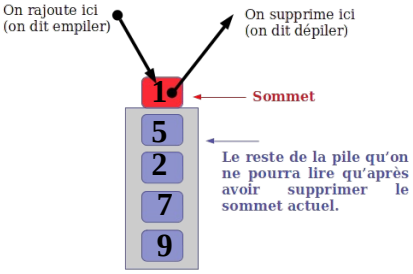
14 - Le type abstrait PILE / LIFO

I – Qu'est-ce que le type abstrait PILE ?

1.1 Idée générale

Une **pile** est un **type abstrait de données** ayant les propriétés suivantes :

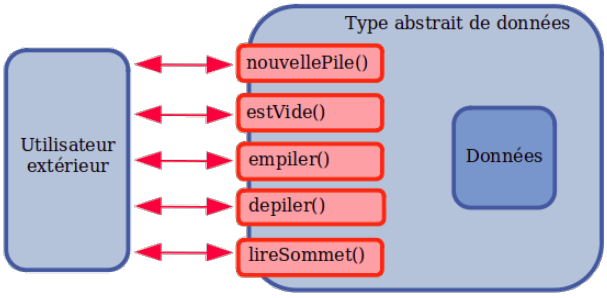
- elle est composée d'une séquence _____.
- on s'impose strictement de n'agir que sur l'une des extrémités qu'on nomme le **sommet** :
  - **insérer un élément au sommet** se nomme **l'empilement**
  - **supprimer un élément au sommet** se nomme **le dépilement**



Exemple : si on insère d'abord 9, puis 7 puis 2 puis 5, puis 1, on obtient :

LIFO : _____

1.2 Interface du type abstrait PILE



Prototype de la fonctions d'interface	Description (cas non mutable)
nouvellePile() -> Pile	On crée une nouvelle pile vide (). En anglais, cela pourrait donner newStack.
estVide(p:Pile) -> bool	Renvoie un booléen qui vaut True si la pile p transmise est une pile vide. En anglais, cela pourrait donner isEmpty ou isNull ou isNil
empiler(x:Elt, p:Pile) -> Pile	Renvoie une nouvelle pile où le sommet est maintenant l'élément x. En anglais, *push.*
depiler(p:Pile) -> Pile	Renvoie une nouvelle pile après avoir supprimé le sommet. En anglais, *pop.*
lireSommet(p:Pile) -> Elt	Renvoie le sommet de la pile p. Attention, on ne modifie pas la pile. En anglais, *peek ou top.*

Exercices 01° Fournir les contenus successifs de la pile p1 après exécution de chacune des instructions

```
1       p1 ← nouvellePile()
2       p1 ← empiler(5, p1)
3       p1 ← empiler(7, p1)
4       x ← lireSommet(p1)
5       p1 ← empiler(x, p1)
6       p1 ← empiler(4, p1)
7       p1 ← depiler(p1)
```

Exercices 02° On veut stocker séquentiellement 1, 2, 3 dans une pile. Donner l'algorithme à utiliser et fournir la représentation finale de la pile. Que va renvoyer la fonction lireSommet ? La pile est-elle modifiée après cette lecture ?

1.3 Type mutable

Pour les implémentations qui utilisent une structure mutable, on trouve souvent les fonctions **depiler** et **lireSommet** regroupées dans une seule et même fonction qui va modifier la pile sur place et qui renvoie le sommet qu'on vient de supprimer.

Les prototypes de ces fonctions d'interface sont alors :

- **`empiler`**`(x:Elt, p:Pile) -> None` : on rajoute un élément au sommet sur place
- **`depiler`**`(p:Pile) -> Elt` : on supprime le sommet sur place et on renvoie l'ancien sommet.

Exercices 03° Fournir les contenus successifs de la pile `p3` après exécution de chacune des instructions

```
1       p3 ← nouvellePile()
2       empiler(5, p3)
3       empiler(7, p3)
4       x ← depiler(p3)
5       empiler(x, p3)
6       empiler(4, p3)
7       depiler(p3)
```

1.4 Fonctions d'interface optionnelles

Prototype de la fonctions d'interface	Description (cas non mutable)
taille(p:Pile) -> int :	On renvoie le nombre d'éléments dans la Pile. Cela peut être pratique pour éviter de dépasser la taile limite
vider(p:Pile) -> Pile	On vide la pile. Pour une pile non-mutable, on renvoie donc une pile vide. Cela peut être utile pour signaler qu'on se sépare d'éléments pourtant non traités. En anglais, ça se nomme clear.
echanger(p:Pile) -> Pile	*On inverse le sommet et l'élément juste en dessous. En anglais, ça se nomme swap. On est déjà à la limite limite de l'utilisation d'une PILE puisqu'on accède au second élément.*

Exercices 04° Imaginer l'algorithme gérant la réception et l'émission de paquets IP par un routeur utilisant une mise en mémoire des requêtes via une pile unique. On considérera un routeur dont la pile ne peut contenir que 5 paquets IP. Le routeur doit pouvoir vider sa mémoire si il sature : de toutes manières, avec le protocole TCP, les paquets disparus seront émis à nouveau par l'émetteur. Processus monotâche, pas d'interruption. Rudimentaire, rudimentaire, tout en séquentiel.

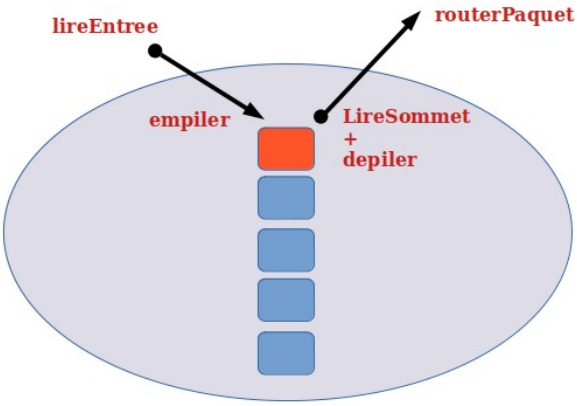
On considère qu'on peut

* Lire le paquet IP d'entrée avec la fonction

```
lireEntree() -> PAQUET_IP
```

* Traiter le routage d'un paquet IP avec la fonction

```
routerPaquet(paquet:PAQUET_IP) -> None
```



II – Choix d'utilisation d'une PILE / LIFO

Exercices 05° Dire si l'utilisation d'une pile est adaptée, ou pas. Si la pile n'est pas adaptée, expliquer le problème qu'on va rencontrer en résolvant le problème avec ce type de gestion.

A - Annulation des actions dans un logiciel.
On stocke les actions avec un empilement. On utilise un dépilement pour savoir ce qu'on doit annuler.

B - Réception et gestion des requêtes HTTP sur un serveur
La réception sera gérée par un empilement. La gestion se fait progressivement avec un dépilement.

C - Réception et gestion des demandes d'impression sur une imprimante partagée
La réception sera gérer par un empilement. La gestion se fait progressivement avec un dépilement.

D - Inversion des éléments d'un tableau
On stocke les éléments avec un empilement en commençant par l'index 0. Une fois tout le tableau lu, on dépile pour créer le nouveau tableau en commençant par l'index 0.

E - Classement des produits les plus vendus
On désire obtenir tous les mois une mise à jour des produits les plus vendus dans un magasin.

F - Gestion des clients arrivant dans un magasin
On empile le client lorsqu'il rentre dans la boutique. On dépile pour savoir qui servir.

G - Stockage des fonctions lors des appels
On stocke en mémoire les fonctions avec un empilement lorsque le programme en a besoin. On dépile lorsqu'elles répondent.

H - Questions et réponses d'un prof lors d'un cours
Les questions sont stockées avec un empilement. La réponse à la question se fait progressivement avec un dépilement.

I - Gestion des stocks
On empile un produit lorsqu'il rentre en magasin. On doit le supprimer lorsqu'il sort.

III – Pile d'exécution et récursivité

Voir le site