IFT 2015

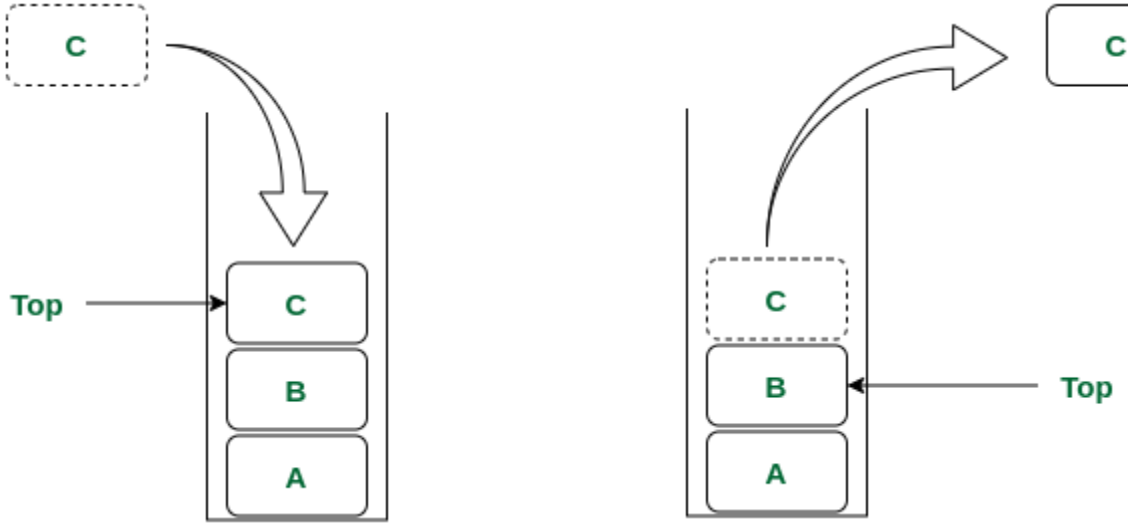
Démonstration 2 : Piles et début Listes

Piles

Rappel (1/5)

- Une pile (Stack) est une structure de données linéaire qui suit le principe **LIFO (Last-In-First-Out)**.

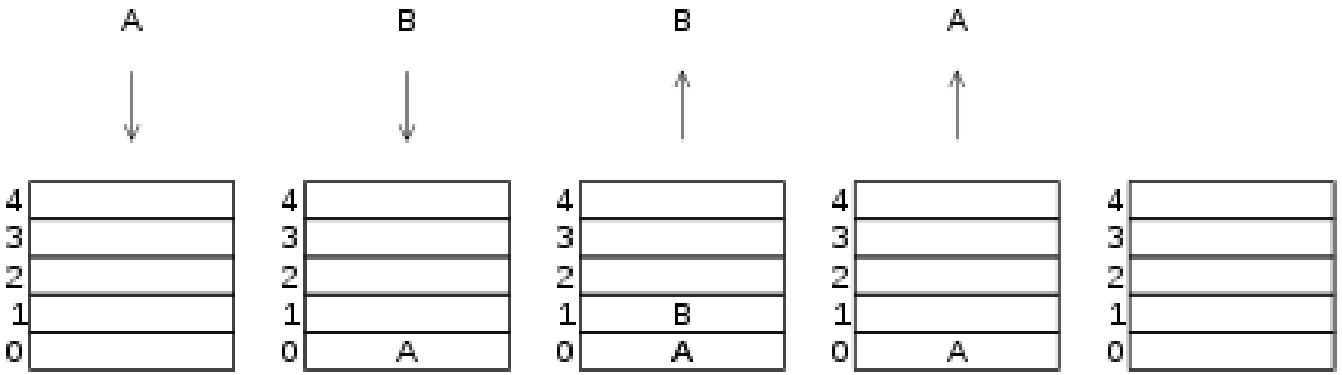
- Un seul pointeur pointant vers **l'élément le plus haut** de la pile. Chaque fois qu'un élément est ajouté dans la pile, il est ajouté en haut de la pile et l'élément ne peut être supprimé que de la pile.



Rappel : Opérations (2/5)

- **push(e)** : ajoute l'élément e sur le haut de la pile
- **pop( )** : retire et retourne l'élément sur le haut de la pile, null si vide
- **top( )** : retourne l'élément sur le haut de la pile, null si vide
- **size( )** : retourne le nombre d'éléments dans la pile
- **isEmpty( )** : retourne un booléen indiquant si la pile est vide

Rappel : Opérations (3/5)



Rappel : Opérations (4/5)

Method	Return Value	Stack Contents
push(5)		
push(3)		
size( )		
pop( )		
isEmpty( )		
pop( )		
isEmpty( )		
pop( )		
push(7)		
push(9)		
top( )		
push(4)		
size( )		
pop( )		
push(6)		
push(8)		
pop( )		

Rappel : Opérations (5/5)

Method	Return Value	Stack Contents
push(5)	–	(5)
push(3)	–	(5, 3)
size( )	2	(5, 3)
pop( )	3	(5)
isEmpty( )	false	(5)
pop( )	5	( )
isEmpty( )	true	( )
pop( )	null	( )
push(7)	–	(7)
push(9)	–	(7, 9)
top( )	9	(7, 9)
push(4)	–	(7, 9, 4)
size( )	3	(7, 9, 4)
pop( )	4	(7, 9)
push(6)	–	(7, 9, 6)
push(8)	–	(7, 9, 6, 8)
pop( )	8	(7, 9, 6)

Exercices

Exercise 1 :

Supposons qu'une pile S initialement vide ait effectué un total de 25 opérations push, 12 opérations top et 10 opérations pop, dont 3 ont renvoyé null pour indiquer une pile vide.

Quelle est la taille actuelle de S ?

Exercise 2 :

Supposons qu'une pile S initialement vide ait effectué dans l’ordre les opérations suivantes : 1 pop ; 3 push ; 1 len ; 2 pop ; 1 isEmpty ; 5 push ; 2 top ; 7 pop ; 2 push ; 3 top.

Quelle est la taille actuelle de S ?

Exercise 3 :

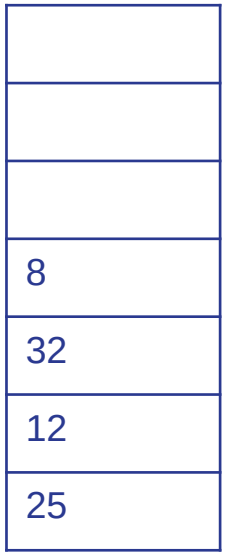
Quelles valeurs sont renvoyées lors de la série suivante d'opérations de pile?

push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop().

Exercise 4 :

Montrez l'état de la pile S après les opérations suivantes :

S.pop() -> S.top() -> S.pop() -> S.push(45) -> S.top() -> S.pop() -> S.push(99) -> S.push(18) -> S.top() -> S.top() -> S.push(55) -> S.push(83) -> S.pop()



Exercise 5 :

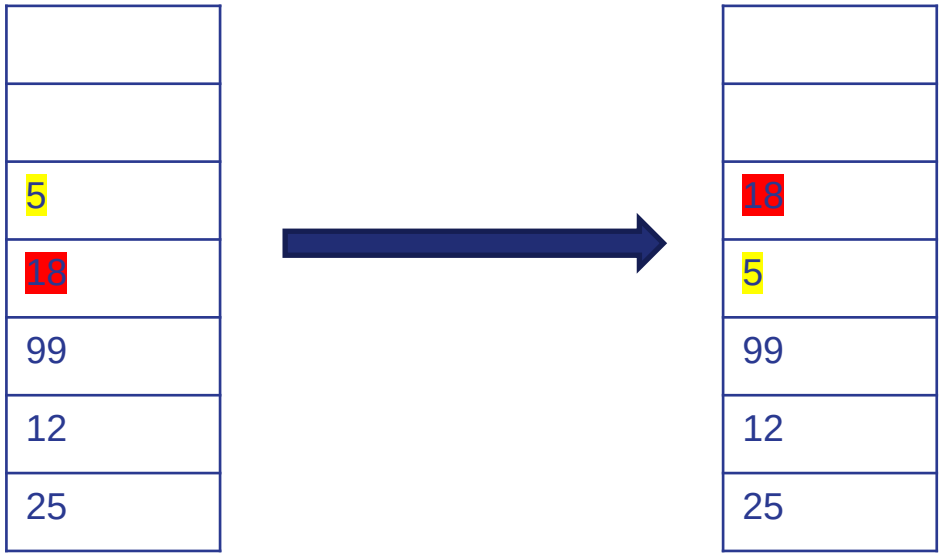
Donnez un pseudo code avec la signature transfer(S,T) qui transfère tous les éléments de la pile S vers la pile T, de sorte que l'élément qui commence en haut de S soit le premier à être inséré sur T et l'élément en bas de S se termine en haut de T

Exercise 6 :

Donnez une méthode **récursive** pour supprimer tous les éléments d'une pile.

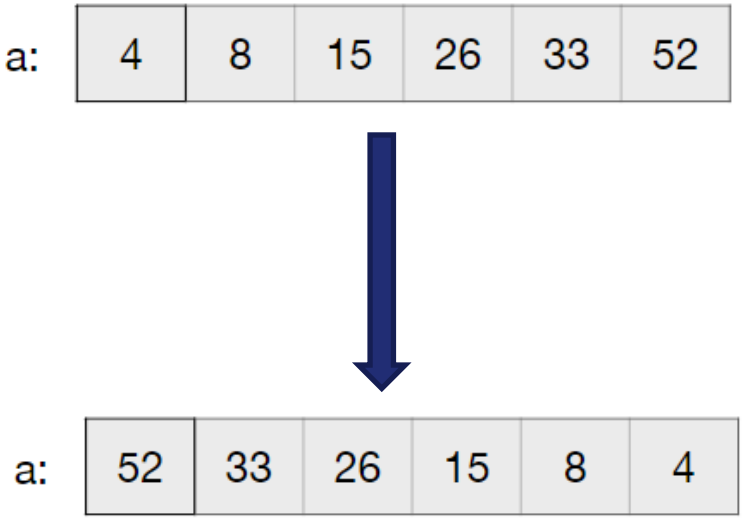
Exercise 7 :

Donnez une méthode “swap” qui échange les deux premiers éléments du haut de la pile.



Exercise 8 :

Donnez une méthode qui renverse les éléments d'un tableau en utilisant une pile.



Exercise 9 :

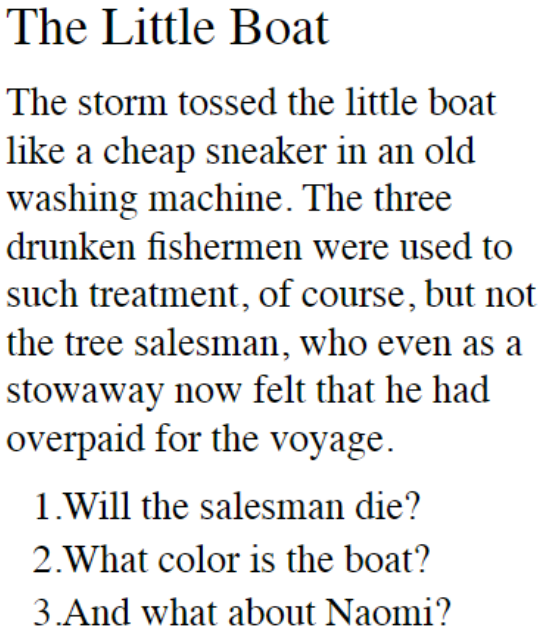
Implémentez une méthode qui permet de vérifier le balancement de parenthèses

- Chaque "(", "{", ou "[" doit être associé à ")", "}" ou “]”
- correct : ( )(( )){([( )])}
- correct : ((( )(( )){([( )])}))
- Incorrect :)(( )){([( )])}
- Incorrect : ({[ ])}
- Incorrect : (

Exercise 10 :

Implémentez une méthode qui permet de vérifier le balancement de “tags” HTML

```
<body>
<center>
<h1> The Little Boat </h1> </center>
<p> The storm tossed the little boat like a cheap
sneaker in an old washing machine. The three drunken
fishermen were used to such treatment, of course, but
not the tree salesman, who even as a stowaway now
felt that he had overpaid for the voyage. </p>
<ol>
<li> Will the salesman die? </li>
<li> What color is the boat? </li>
<li> And what about Naomi? </li>
</ol>
</body>
```



Exercise 11 :

Supposons qu'Alice ait choisi trois entiers distincts et les ait placés dans une pile S dans un ordre aléatoire.

Ecrivez une fonction en pseudo-code (sans boucles ni récursivité) qui utilise une seule comparaison et une seule variable x, avec une probabilité de 2/3.

Expliquez pourquoi votre méthode est correcte.

Exercise 12 :

La notation postfixée est une manière non ambiguë d'écrire une expression arithmétique sans parenthèses. Il est défini de sorte que si "(exp1)op(exp2)" est une expression normale entièrement entre parenthèses dont l'opération est op, la version postfixée de celle-ci est "pexp1 pexp2 op".

A B C + * D / (postfixée) (gauche, droite, racine)

((A (B C +) *) D /)

Décrivez une manière non récursive d'évaluer une expression en notation postfixée **en utilisant une pile**. Déterminer la complexité.

Exercise 13 :

Implémentez un algorithme qui permet de convertir une expression infixée en une expression postfixée **en utilisant une pile**.
Supposons que l'expression infixée est une string sans espace.

Par exemple,

Input: A*B+C
Output: AB*C+

Input: (A+B)*(C/D)
Output: AB+CD/*

Exercise 14 :

Implémentez un algorithme qui permet de convertir une expression prefixée en une expression postfixée **en utilisant une pile**.

Par exemple,

Input : Prefix : *+AB-CD
Output : Postfix : AB+CD-*

Lists

Array Lists

Un choix évident pour implémenter la liste ADT est d'utiliser un tableau A, où A[i] stocke (une référence à) l'élément d'indice i.

Implementer la class ArrayList qui implémente l'interface List de Java avec capacité = 16, ainsi que les méthodes suivantes:

- **Public int size():** Renvoie le nombre d'éléments dans ArrayList.
- **Public Boolean isEmpty():** Retourne si ArrayList est vide.
- **Public E get(int i):** Renvoie (mais ne supprime pas) l'élément à l'index i
- **Public E set(int I , E e) :** Remplacer l'élément à l'index i par e ,et renvoie l'élément remplacé.
- **Public void add(int i, E e):** Insérer l'élément e pour qu'il soit à l'index i, en décalant tous les
- éléments suivants plus tard
- **Public E remove(int i):** Supprime et renvoie l'élément à l'index i, décalant les éléments suivants plus tôt

Discuter la complexité de ces méthodes.

```java
public class ArrayList<E> implements List<E> {

    private static final int CAPACITY = 16;
    private int size = 0;
    private E[] data;

    public ArrayList(final int capacity) {
        data = (E[]) new Object[capacity];
    }

    public ArrayList() {
        this(CAPACITY);
    }
```

```java
@Override
public int size() {
    return this.size;
}

@Override
public boolean isEmpty() {
    return this.size == 0;
}
```

```
protected void checkIndex(int i) {
    if (i < 0 || i > data.length) {
        throw new IndexOutOfBoundsException("Invalid index.");
    }
}

@Override
public E get(int i) throws IndexOutOfBoundsException {
    checkIndex(i);
    return data[i];
}

@Override
public E set(int i, E e) throws IndexOutOfBoundsException {
    checkIndex(i);
    E removed = data[i];
    data[i] = e;
    return removed;
}
```

```java
@Override
public void add(int i, E e) throws IndexOutOfBoundsException, IllegalStateException {
    checkIndex(i);

    if (size == data.length) {
           throw new IllegalStateException("ArrayList is full!");

    }

    for (int k = size - 1; k >= i; k--) {
        data[k+1] = data[k];
    }

    data[i] = e;
    size++;
}
```

```java
@Override
public E remove(int i) throws IndexOutOfBoundsException {
    checkIndex(i);
    E removed = data[i];

    for (int k = i; k < size - 1; k++) {
        data[k] = data[k+1];
    }

    data[size - 1] = null;
    size--;
    return removed;
}
```

La complexité

Method	Running Time
size( )	$O(1)$
isEmpty( )	$O(1)$
get(i)	$O(1)$
set(i, e)	$O(1)$
add(i, e)	$O(n)$
remove(i)	$O(n)$