

# Elements d'Algorithmique

## CMTD7 : Piles

---

Daniela Petrişan  
Université de Paris, IRIF



INSTITUT  
DE RECHERCHE  
EN INFORMATIQUE  
FONDAMENTALE



# Piles

Les piles sont des structures de données dynamiques représentant une séquence d'éléments ordonnés selon leur ordre d'insertion et dont nous ne pouvons supprimer que l'élément qui a été inséré le plus récemment.

La pile met en œuvre le principe **dernier entré, premier sorti**, ou **LIFO** (last in, first out).



# Piles

Les piles sont des structures de données dynamiques représentant une séquence d'éléments ordonnés selon leur ordre d'insertion et dont nous ne pouvons supprimer que l'élément qui a été inséré le plus récemment.

La pile met en œuvre le principe **dernier entré, premier sorti**, ou **LIFO** (last in, first out).

Une **pile** est une structure de données **abstraite** sur laquelle sont définies trois opérations :



# Piles

Les piles sont des structures de données dynamiques représentant une séquence d'éléments ordonnés selon leur ordre d'insertion et dont nous ne pouvons supprimer que l'élément qui a été inséré le plus récemment.

La pile met en œuvre le principe **dernier entré, premier sorti**, ou **LIFO** (last in, first out).

Une **pile** est une structure de données **abstraite** sur laquelle sont définies trois opérations :

- **empty(P)** qui teste si la pile P est vide ;



# Piles

Les piles sont des structures de données dynamiques représentant une séquence d'éléments ordonnés selon leur ordre d'insertion et dont nous ne pouvons supprimer que l'élément qui a été inséré le plus récemment.

La pile met en œuvre le principe **dernier entré, premier sorti**, ou **LIFO** (last in, first out).

Une **pile** est une structure de données **abstraite** sur laquelle sont définies trois opérations :

- **empty**(P) qui teste si la pile P est vide ;
- **push**(x,P) qui ajoute un élément x au sommet de la pile P. Cette opération est également appelée **empiler** ;



# Piles

Les piles sont des structures de données dynamiques représentant une séquence d'éléments ordonnés selon leur ordre d'insertion et dont nous ne pouvons supprimer que l'élément qui a été inséré le plus récemment.

La pile met en œuvre le principe **dernier entré, premier sorti**, ou **LIFO** (last in, first out).

Une **pile** est une structure de données **abstraite** sur laquelle sont définies trois opérations :

- **empty**(P) qui teste si la pile P est vide ;
- **push**(x, P) qui ajoute un élément x au sommet de la pile P. Cette opération est également appelée **empiler** ;
- **pop**(P) qui enlève la valeur au sommet de la pile P et la renvoie. Cette opération est aussi appelée **dépiler**.



## Piles : exemple

```
p:= new Pile();
```

```
p.push(1);
```

```
p.push(2);
```

```
print(p.pop());
```

```
p.push(3);
```

```
p.push(4);
```

```
p.push(1);
```

```
while(!p.empty()) {
```

```
    print(p.pop())
```

```
}
```



>

## Piles : exemple

```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```

1

>



## Piles : exemple

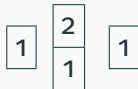
```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



>

## Piles : exemple

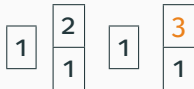
```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



>2

## Piles : exemple

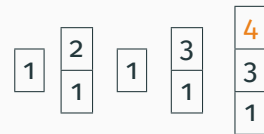
```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



>2

## Piles : exemple

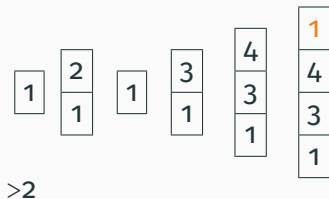
```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



>2

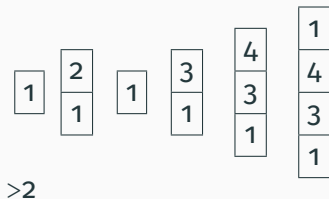
## Piles : exemple

```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



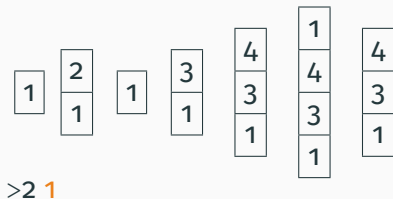
## Piles : exemple

```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



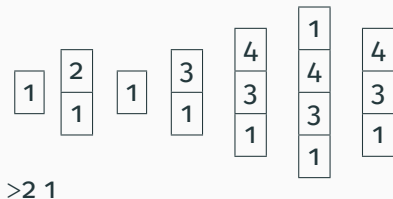
## Piles : exemple

```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



## Piles : exemple

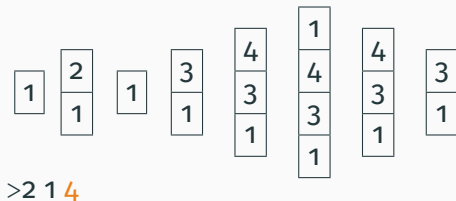
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```





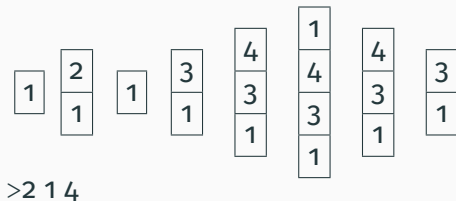
## Piles : exemple

```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



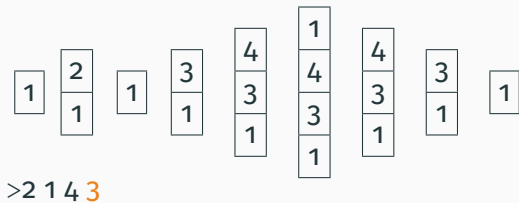
## Piles : exemple

```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



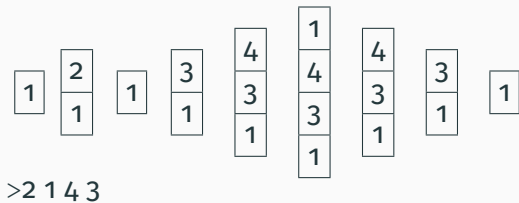
## Piles : exemple

```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



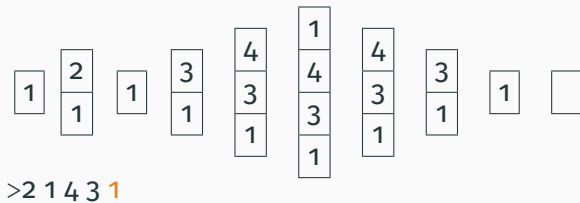
## Piles : exemple

```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



## Piles : exemple

```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```



## Piles : exemple

```
p:= new Pile();  
p.push(1);  
p.push(2);  
print(p.pop());  
p.push(3);  
p.push(4);  
p.push(1);  
while(!p.empty()) {  
    print(p.pop())  
}
```

