Elements d'Algorithmique
Piles

Daniela Petrişan
Université de Paris, IRIF

Piles

Les piles sont utilisées pour représenter des ensembles dynamiques dans lesquels, s'ils ne sont pas vides, 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 utilisées pour représenter des ensembles dynamiques dans lesquels, s'ils ne sont pas vides, 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 utilisées pour représenter des ensembles dynamiques dans lesquels, s'ils ne sont pas vides, 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 utilisées pour représenter des ensembles dynamiques dans lesquels, s'ils ne sont pas vides, 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 utilisées pour représenter des ensembles dynamiques dans lesquels, s'ils ne sont pas vides, 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

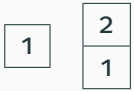
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```

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

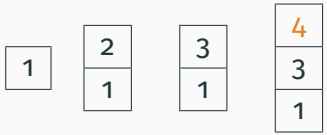
Piles: exemple

```
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())
}
```

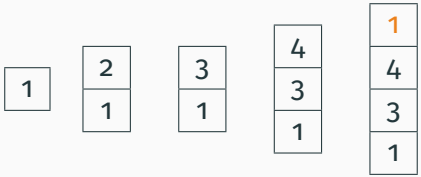


>2

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())
}
```

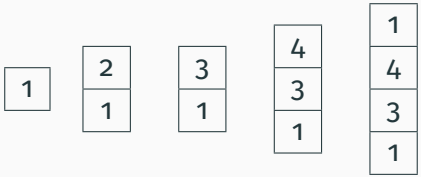
>2



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())
}
```

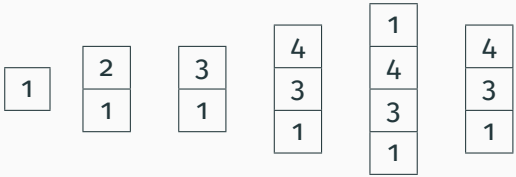
>2



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())
}
```

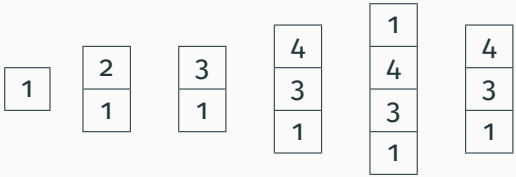
>2 1



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())
}
```

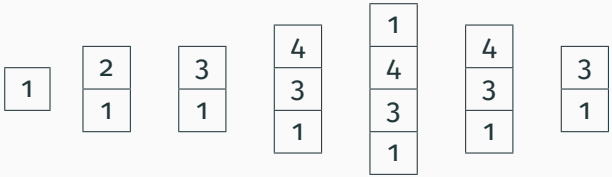
>2 1



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())
}
```

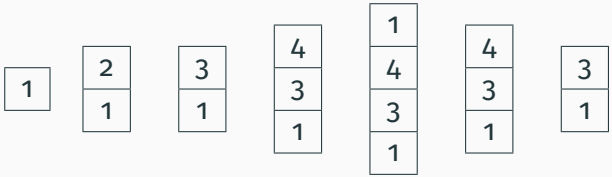
>2 1 4



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())
}
```

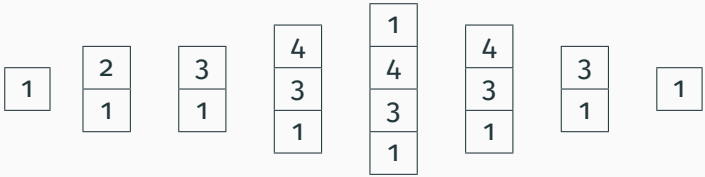
>2 1 4



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())
}
```

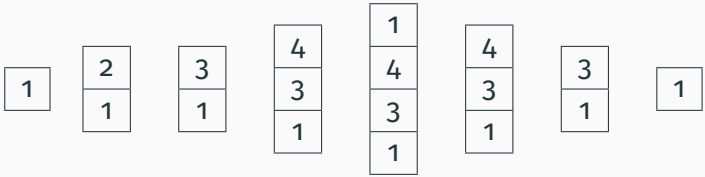
>2 1 4 3



Piles: exemple

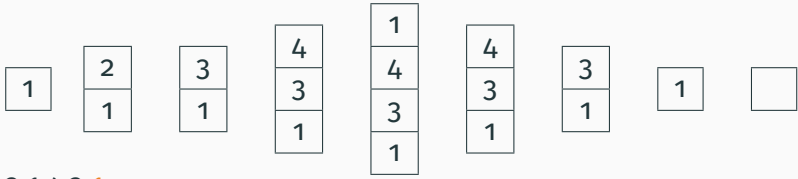
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```

>2 1 4 3



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 1 4 3 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())
}
```

>2 1 4 3 1