[CI2] Concepts informatiques
1.2 Piles

Daniela Petrişan
Université de Paris, IRIF

Pile et tas

Dans la vidéo précédente, nous avons brièvement mentionné deux espaces de stockage utilisés par la JVM :

- la pile : les espaces de stockage correspondant aux variables locales

Pile et tas

Dans la vidéo précédente, nous avons brièvement mentionné deux espaces de stockage utilisés par la JVM :

- la pile : les espaces de stockage correspondant aux variables locales
- le tas : les espaces de stockage alloués via l'opérateur `new`

Il existe une autre utilisation importante de la pile liée à la manière dont les appels de fonction sont gérés.

Pile d'appels

Un appel de fonction induit un saut incoditionnel, c'est-à-dire une rupture dans l'exécution d'une séquence d'instructions d'un programme.

Pile d'appels

Un appel de fonction induit un saut incoditionnel, c'est-à-dire une rupture dans l'exécution d'une séquence d'instructions d'un programme.

Comment savoir où revenir après la fin de l'exécution d'un tel appel de fonction ? C'est ici que la pile d'appels intervient.

Pile d'appels

Un appel de fonction induit un saut incoditionnel, c'est-à-dire une rupture dans l'exécution d'une séquence d'instructions d'un programme.

Comment savoir où revenir après la fin de l'exécution d'un tel appel de fonction ? C'est ici que la pile d'appels intervient.

Tout comme M. Jourdain, qui faisait déjà de la prose, vous avez sûrement rencontré cette pile !

```
public class Overflow {
  public static void f() {
      f();
  }
  public static void main (String [] args) {
      f();
  }
}
```

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()` 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()` qui teste si la pile `P` est vide
- `push(x)` qui ajoute un élément `x` au sommet de la pile. 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 :

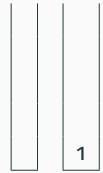
- `empty()` qui teste si la pile `P` est vide
- `push(x)` qui ajoute un élément `x` au sommet de la pile. Cette opération est également appelée `empiler`.
- `pop()` qui enlève la valeur au sommet de la pile et la renvoie. Cette opération est aussi appelée `dépiler`.

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())
}
```

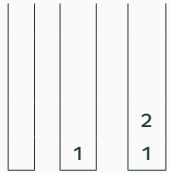
Piles: exemple

```
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())
}
```



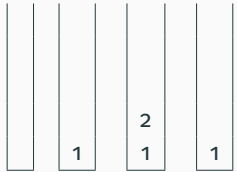
Piles: exemple

```
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

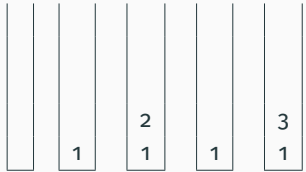
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2

Piles: exemple

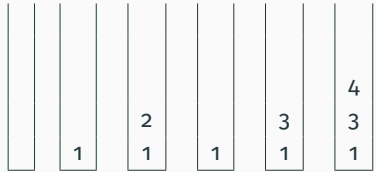
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2

Piles: exemple

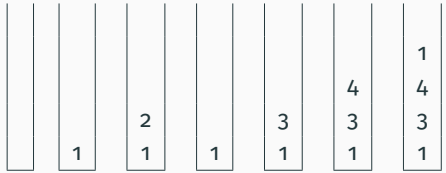
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2

Piles: exemple

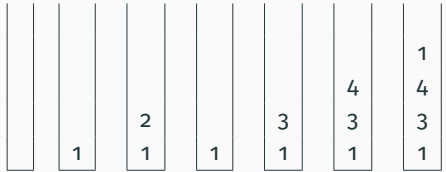
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2

Piles: exemple

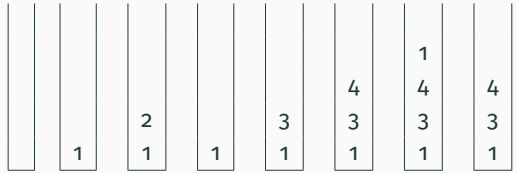
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```




```
>2
```

Piles: exemple

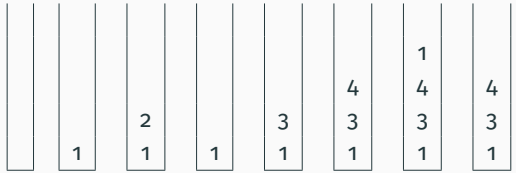
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1

Piles: exemple

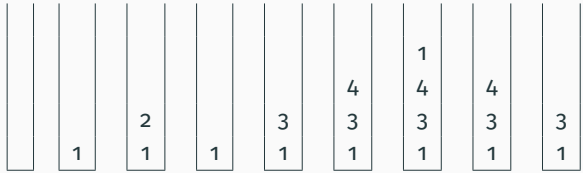
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1

Piles: exemple

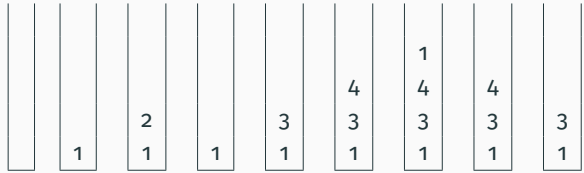
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1 4

Piles: exemple

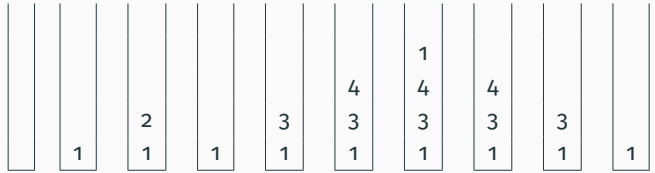
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1 4

Piles: exemple

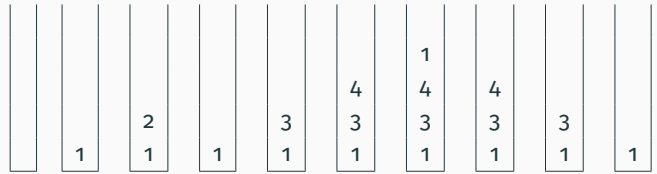
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1 4 3

Piles: exemple

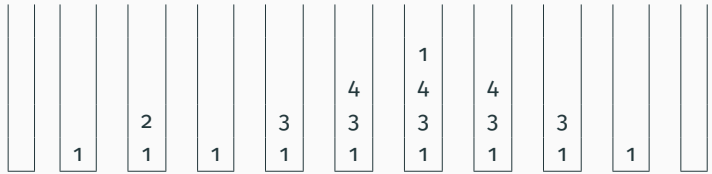
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1 4 3

Piles: exemple

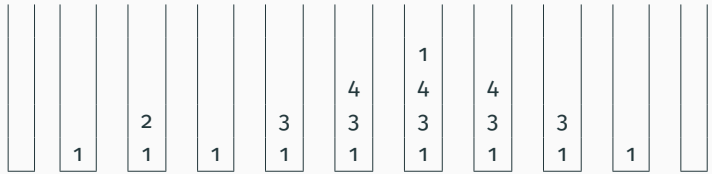
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>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())
}
```

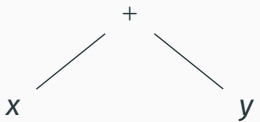


>2 1 4 3 1

Utilisation des piles : analyse syntaxique

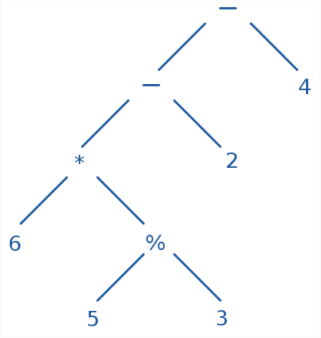
Plusieurs façons d'écrire les expressions arithmétiques :

- comme un arbre

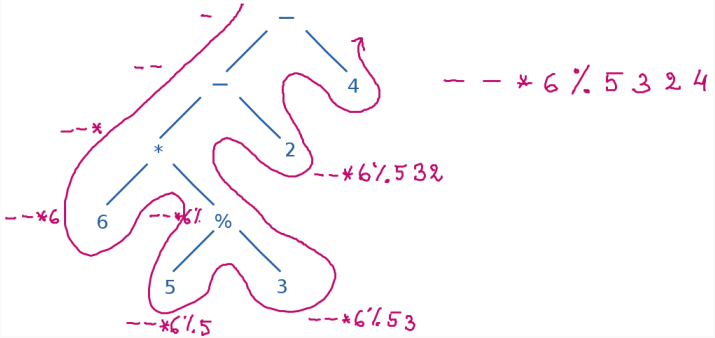


- forme infixe : $x + y$
- forme postfixe : $xy+$
- forme préfixe : $+xy$

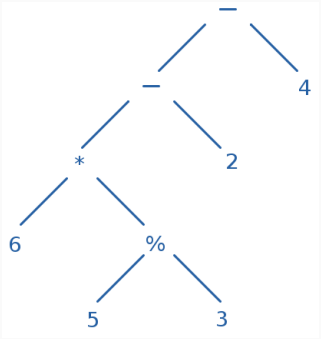
forme infixe : $6 * (5\%3) - 2 - 4$



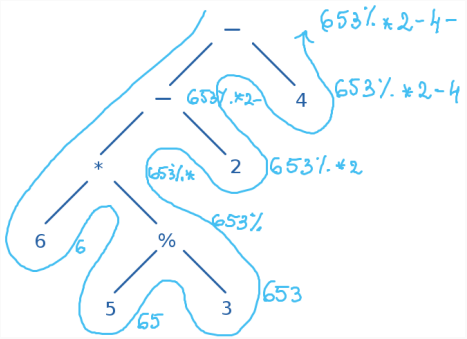
forme préfixe : $- - *6\%5324$



forme infixe : $6 * (5\%3) - 2 - 4$



forme postfixe : $653\% * 2 - 4-$



Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$

Algorithme pour l'évaluation d'une forme postfixe

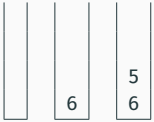
on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$

Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$



Algorithme pour l'évaluation d'une forme postfixe

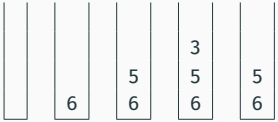
on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4 -$

Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$



Algorithme pour l'évaluation d'une forme postfixe

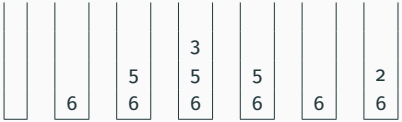
on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$

Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

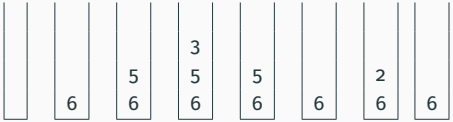
forme postfixe : $653\% * 2 - 4-$



Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

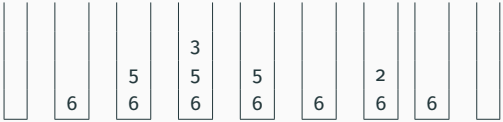
forme postfixe : $653\% * 2 - 4 -$



Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$



Algorithme pour l'évaluation d'une forme postfixe

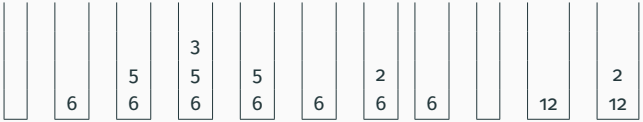
on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4 -$

Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$



Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$

Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$

Algorithme pour l'évaluation d'une forme postfixe

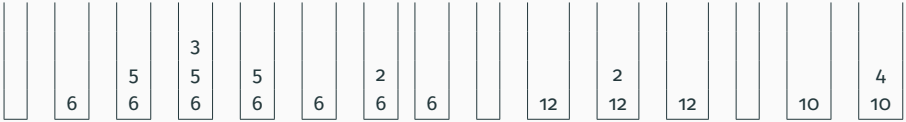
on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$

Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$



Algorithme pour l'évaluation d'une forme postfixe

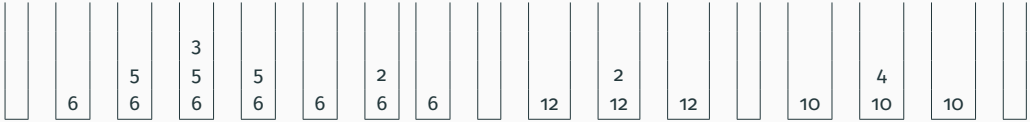
on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$

Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$



Algorithme pour l'évaluation d'une forme postfixe

on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

forme postfixe : $653\% * 2 - 4-$

En java

```
static String evaluation(String[] e){
    Stack<Integer> p=new Stack<Integer>();  //pile d'entier
    int e1,e2;
    /* rappel pour convertir un entier n en une chaîne: ""+n
       rappel pour convertir une chaîne s en un entier: Integer.valueOf(s) */
    for(int i=0; i<e.length; i++)   //parcours de l'expression
        if(e[i].equals("+") || e[i].equals("%")
           || e[i].equals("x") || e[i].equals("/") || e[i].equals("-")){
           //on cherche à dépiler deux opérandes:
            if(p.empty()) return "expression mal formée";//manque un opérande
            e1 = p.pop();
            if(p.empty()) return "expression mal formée";//manque un opérande
            e2 = p.pop();
            switch(e[i]){
                case "+": p.push(e2+e1); break;
                case "%": p.push(e2%e1); break;
                case "x": p.push(e2*e1); break;
                case "/": p.push(e2/e1); break;
                case "-": p.push(e2-e1); break;
            }}
        else
            p.push(Integer.valueOf(e[i]));
    if(p.empty()) return "expression mal formée";
    e1 = p.pop();
    if(p.empty()) return "l'évaluation est "+e1;
    return "expression mal formée";
}
```