[CI2] Concepts informatiques

Daniela Petrişan
Université Paris Cité, IRIF

Organisation du cours

Organisation

- La page Moodle du cours:
  `https://moodle.u-paris.fr/course/view.php?id=1625`
  contient les transparents du cours, les feuilles de TD, les dates d'examen, les consignes et les informations sur le QCM et les autres contrôles.
- Les TD commenceront
  - **cette semaine** pour les groupes INFO1, INFO2, INFO3, INFO4, INFO5, MI1
  - le 23 Janvier pour les groupes MI2 et INFO6

Modalités de contrôle des connaissances

- Une note de TD, obtenue à partir de trois CC
  - Un contrôle organisé comme un QCM court pendant l'Amphi 4
  - Deux autres contrôles dans votre groupe de TD
- Contrôle partiel
- Contrôle terminal

Note de la première session

30%CC + 20%Partiel + 50%ET

Note de la session seconde chance

max(SC, 30%CC + 20%Partiel + 50%SC)

Contenu du cours

- Partie 1:
  - compréhension des mécanismes sous-jacents aux exécutions des programmes
  - traductions de programmes écrits en JAVA vers un langage cible plus proche du langage-machine
  - la manipulation de la mémoire: variables, adresses, références, fonctions, paramètres, variables locales…
- Partie 2:
  - récursion et ses différentes applications
  - types récursifs, fonction récursives
  - élimination de la récursion
  - backtracking

Variables de type primitif

Variables de type primitif

En Java, avant toute manipulation, une variable doit être déclarée et initialisée.

Variables de type primitif

En Java, avant toute manipulation, une variable doit être déclarée et initialisée.

Exemple

```
int n = 14;
```

Variables de type primitif

En Java, avant toute manipulation, une variable doit être déclarée et initialisée.

Exemple

```
int n = 14;
```

```
int n;
n = 14;
```

Variables de type primitif

En Java, avant toute manipulation, une variable doit être déclarée et initialisée.

Exemple

```
int n = 14;
```

```
int n;
n = 14;
```

La déclaration permet de définir dans le programme
un identificateur (dans cet exemple `n`) associé à un espace de stockage.

Variables de type primitif

En Java, avant toute manipulation, une variable doit être déclarée et initialisée.

Exemple

```
int n = 14;
```

```
int n;
n = 14;
```

La déclaration permet de définir dans le programme
un identificateur (dans cet exemple `n`) associé à un espace de stockage.

Le type (dans cet exemple `int`) permet au compilateur de

- de déterminer la taille de l'espace de stockage
- d'interpréter le contenu de cet espace. Par exemple, sur 32 bits, nous pouvons stocker un nombre entier ou un nombre flottant
- de déterminer les opérations qui sont autorisées pour manipuler la variable associée

Variables de type primitif

Exemple

```
int n = 14;
```

Le type `int` est un type primitif en java, l'espace de stockage est de 32 bits et la valeur par défaut est 0.

Variables de type primitif

Exemple

```
int n = 14;
```

Le type `int` est un type primitif en java, l'espace de stockage est de 32 bits et la valeur par défaut est 0.

Données primitives: entiers, nombres flottants, caractères, valeurs de vérité.
Ce sont les données dont le type est l'un des types primitifs de Java :

byte	float	char	boolean
short	double		
int			
long			

Variables de type primitif

Exemple

```
int n = 14;
n = n / 2;
```

Pendant l'exécution du programme, l'identificateur `n` peut être utilisé de deux façons :

- comme une lecture du contenu de l'espace de stockage : `n = n / 2;`
- comme une affectation du contenu : `n = n / 2;`

Variables et tableaux : références

Tableau uni-dimensionnel

Exemple

```
int [] t ;
```

La variable `t` est une référence vers un tableau d'entiers.

Tableau uni-dimensionnel

Exemple

```
int [] t ;
```

La variable `t` est une référence vers un tableau d'entiers.

La variable `t` est déclarée, mais aucun tableau d'entiers n'est créé !

Ce qui est créé est plutôt une référence qui peut servir à stocker l'adresse mémoire d'un tableau.

Tableau uni-dimensionnel: quiz

```
public class Test {
  public static void main(String [] args) {
    int [] t;
    System.out.println("Valeur de t " + t);
  }
}
```

Quiz: Ce programme compile-t-il ?

Tableau uni-dimensionnel: quiz

```
public class Test {
  public static void main(String [] args) {
    int [] t;
    System.out.println("Valeur de t " + t);
  }
}
```

Quiz: Ce programme compile-t-il ?

Non, la variable `t` n'est pas initialisée! Nous obtenons l'erreur suivante

```
error: variable t might not have been initialized
    System.out.println("Valeur de t " + t);
```

Initialisation de la référence `t`

```
int [] t ;
t = new int [5] ;
```

- l'opérateur `new` retourne une adresse mémoire dans l'espace de stockage permettant de stocker 5 entiers
- cette adresse est affectée au contenu de la variable à gauche de `=` , c'est-à-dire à `t`

Initialisation de la référence `t`: Quiz

```
public class Test {
  public static void main(String [] args) {
    int [] t;
    t = new int[4];
    System.out.println("Valeur de t " + t);
  }
}
```

Quiz: Que se passe-t-il maintenant lors de l'exécution du programme ?

Initialisation de la référence `t`: Quiz

```
public class Test {
  public static void main(String [] args) {
    int [] t;
    t = new int[4];
    System.out.println("Valeur de t " + t);
  }
}
```

Quiz: Que se passe-t-il maintenant lors de l'exécution du programme ?

```
$ java Test
Valeur de t [I@4517d9a3
```

Initialisation de la référence `t`: Quiz

```
public class Test {
  public static void main(String [] args) {
    int [] t;
    t = new int[4];
    System.out.println("Valeur de t " + t);
  }
}
```

Quiz: Que se passe-t-il maintenant lors de l'exécution du programme ?

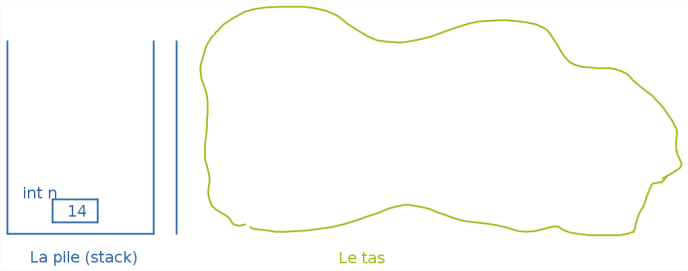
```
$ java Test
Valeur de t [I@4517d9a3
```

`[` indique qu'il s'agit de l'adresse d'un tableau, `I` qu'il contient des entiers, `@` est un séparateur `4517d9a3` est l'"adresse" du premier octet des 16 qui sont réservés pour stocker 4 entiers, (en fait le hashCode en hexadécimal).

Initialisation de la référence `t`

```java
public class Test {
  public static void main(String [] args) {
    int n = 14;
    int [] t;
    t = new int[4];
    System.out.println("Valeur de t " + t);
  }
}
```

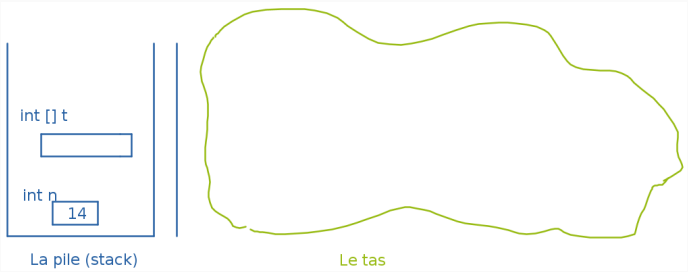
```
$ java Test
Valeur de t [I@4517d9a3
```



Initialisation de la référence `t`

```java
public class Test {
  public static void main(String [] args) {
    int n = 14;
    int [] t;
    t = new int[4];
    System.out.println("Valeur de t " + t);
  }
}
```

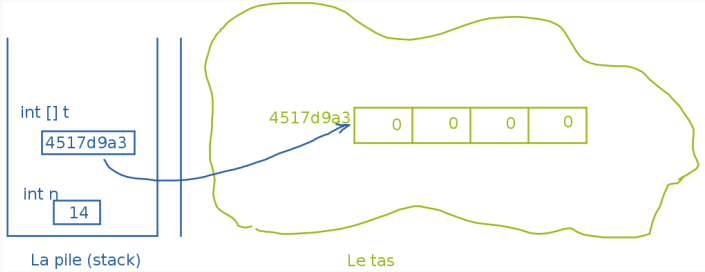
```
$ java Test
Valeur de t [I@4517d9a3
```



Initialisation de la référence `t`

```java
public class Test {
  public static void main(String [] args) {
    int n = 14;
    int [] t;
    t = new int[4];
    System.out.println("Valeur de t " + t);
  }
}
```

```
$ java Test
Valeur de t [I@4517d9a3
```



Piles

Pile et tas

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

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 :

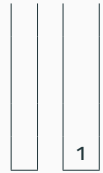
- `empty()` qui teste si la pile `P` est vide
- `push(x)` qui ajoute un élément `x` au sommet de la pile. Cette opération est également appelée `empiler`.
- `pop()` qui enlève la valeur au sommet de la pile et la renvoie. Cette opération est aussi appelée `dépiler`.

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())
}
```

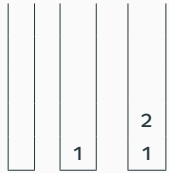
Piles: exemple

```
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())
}
```



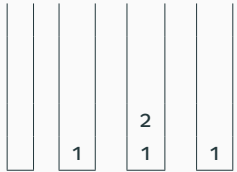
Piles: exemple

```
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

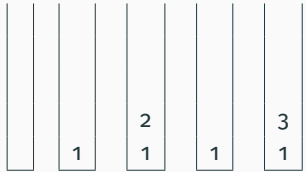
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2

Piles: exemple

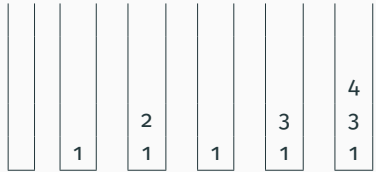
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2

Piles: exemple

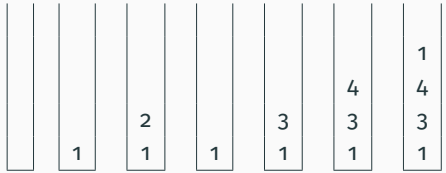
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2

Piles: exemple

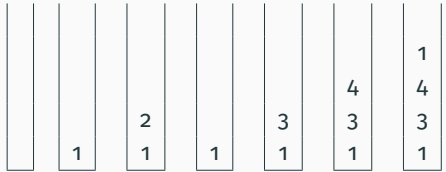
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2

Piles: exemple

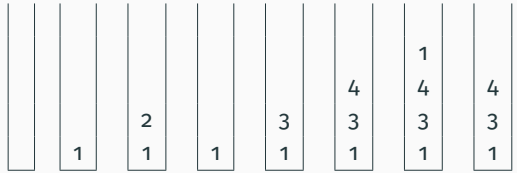
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```




```
>2
```

Piles: exemple

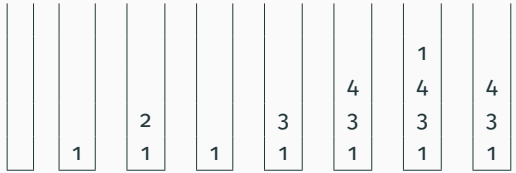
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1

Piles: exemple

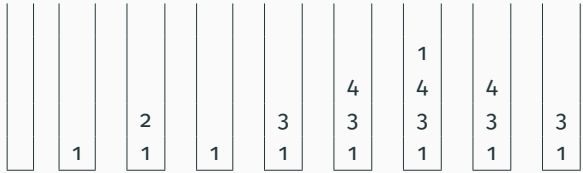
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1

Piles: exemple

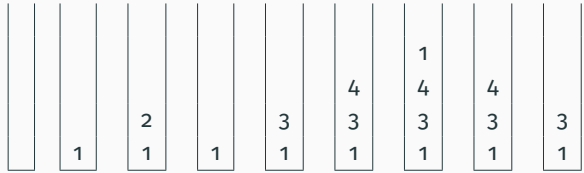
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1 4

Piles: exemple

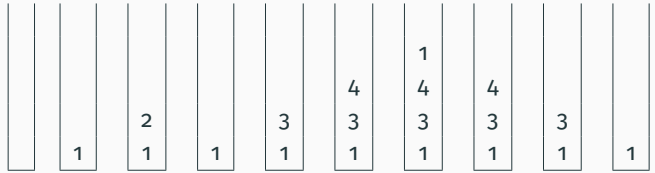
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1 4

Piles: exemple

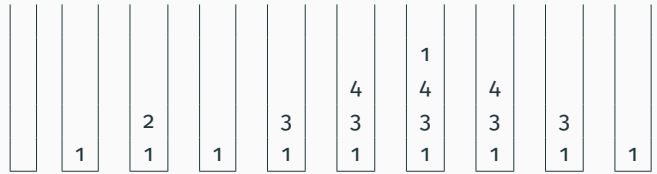
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1 4 3

Piles: exemple

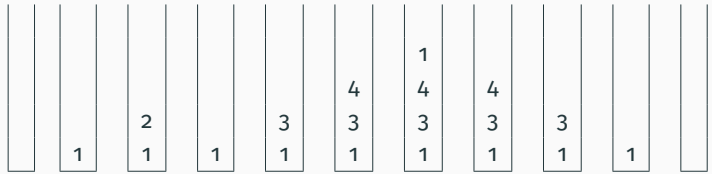
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>2 1 4 3

Piles: exemple

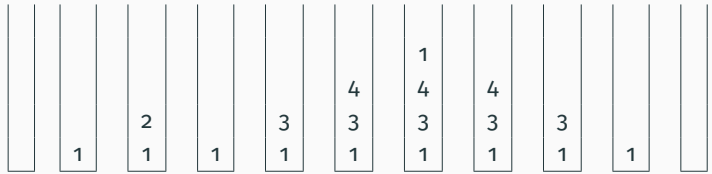
```
p:= new Pile();
p.push(1);
p.push(2);
print(p.pop());
p.push(3);
p.push(4);
p.push(1);
while(!p.empty()) {
    print(p.pop())
}
```



>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())
}
```

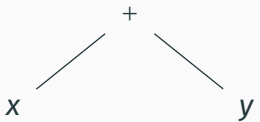


>2 1 4 3 1

Utilisation des piles : analyse syntaxique

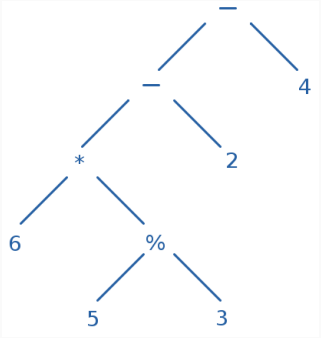
Plusieurs façons d'écrire les expressions arithmétiques :

- comme un arbre

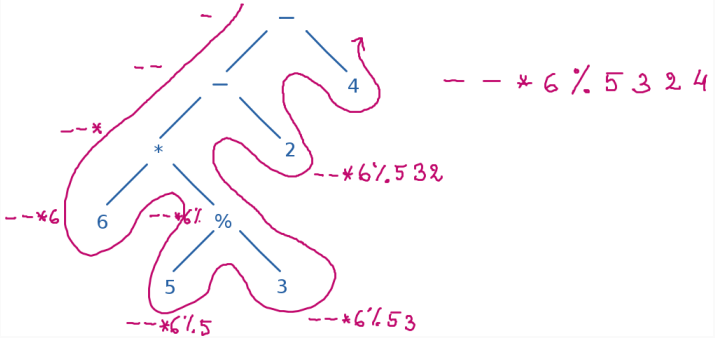


- forme infixe : $x + y$
- forme postfixe : $xy+$
- forme préfixe : $+xy$

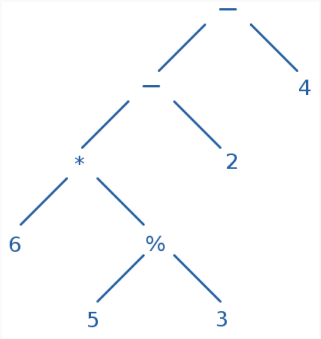
forme infixe : $6 * (5\%3) - 2 - 4$



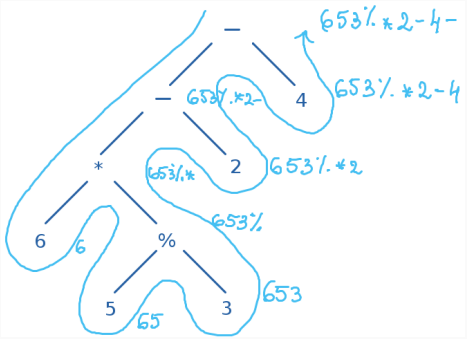
forme préfixe : $- - *6\%5324$



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

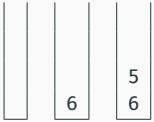
on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

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

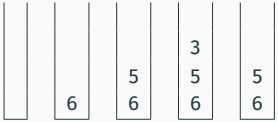
on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

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

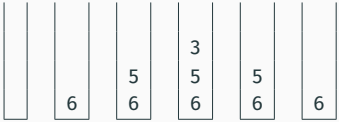
forme postfixe : $653\% * 2 - 4-$



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

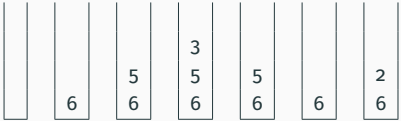
forme postfixe : $653\% * 2 - 4-$



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

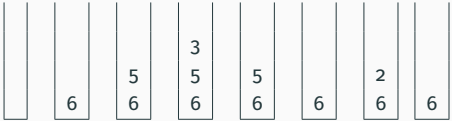
forme postfixe : $653\% * 2 - 4-$



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

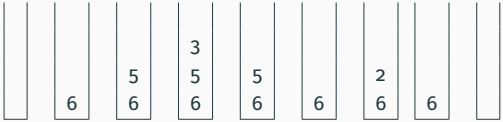
forme postfixe : $653\% * 2 - 4 -$



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

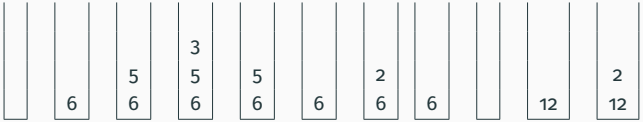
on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

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

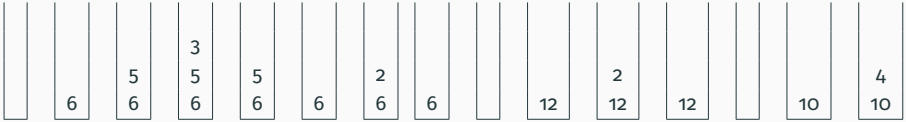
on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

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

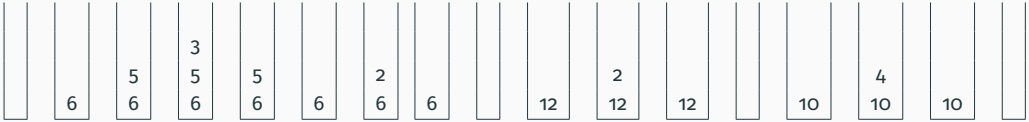
on lit l'expression de gauche à droite
si un élément est un opérande, on l'empile
sinon (c'est un opérateur (binaire))
    on dépile (deux) éléments
    on exécute l'opération
    on empile son résultat
à la fin, la pile contient l'évaluation

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";
}
```