# Algorithmique avancée

## Piles et Files

Frédéric Guyomarch

Université de Lille1
IUT-A de Lille

2020/2021 - Semestre 3

# Introduction

- Structures de données différentes
- Tableaux, listes : stockage + accès
- Piles/files : structures conceptuelles
- Structures abstraites et accès restreint

# Plan

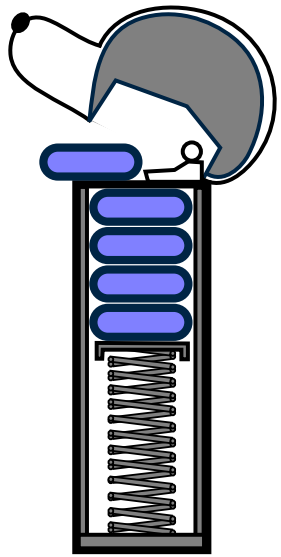
# Des Piles, pourquoi ?

Quelques exemples d'utilisation :

- Recherche de chemins (labyrinthe)
- Gestion d'un historique de modifications (fonctions undo/redo)
- Gestion de la récursivité dans un langage (pile d'appels)

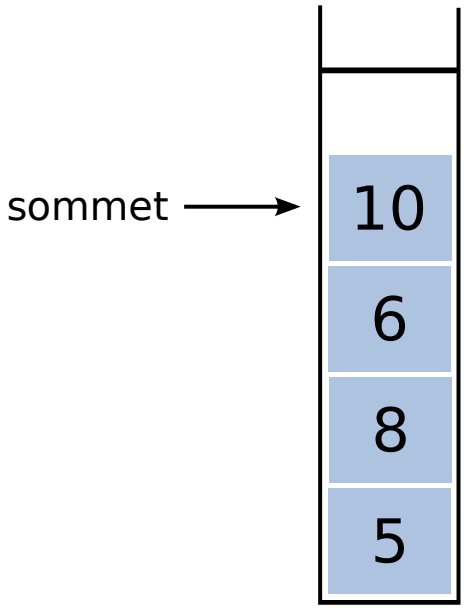
# Pile

- Une pile est un conteneur d'éléments qui respecte le principe du *Last In First Out* (LIFO) : le premier élément accessible est le dernier ajouté.
- On ne peut accéder qu'à l'élément placé au sommet de la pile.
- Un distributeur de PEZ® en est un bon exemple :



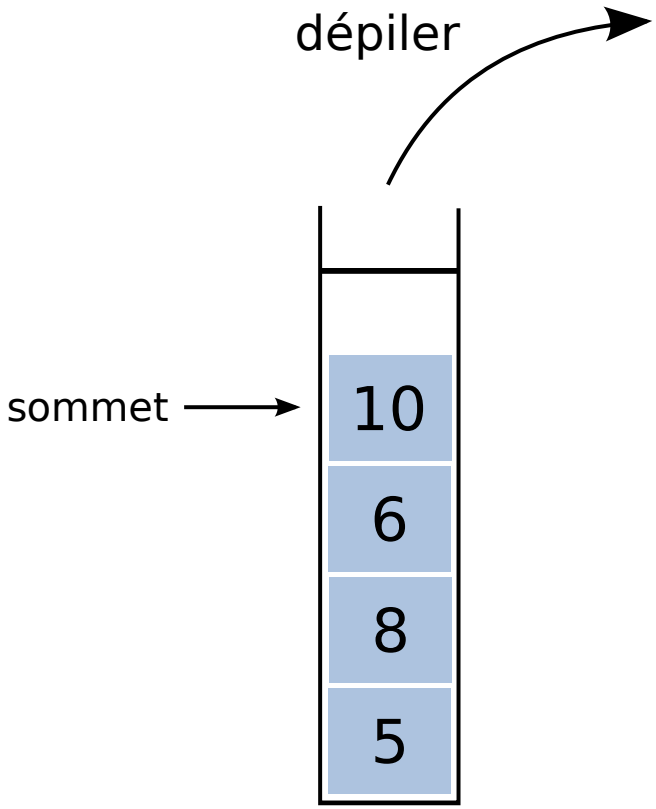
# Pile : LIFO

Pile à l'origine, 10 est la valeur au sommet :



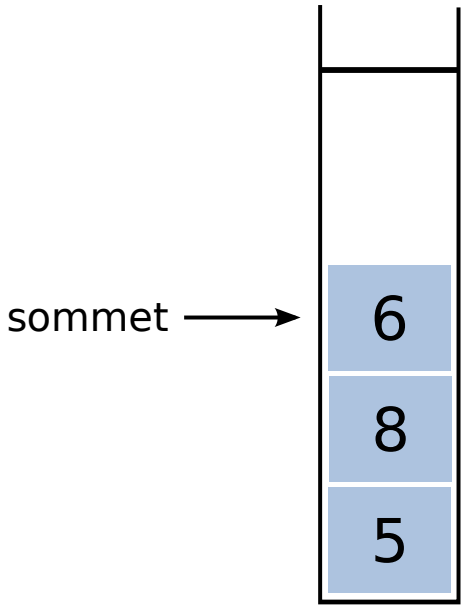
# Pile : LIFO

On dépile 10 (retire la valeur 10 au sommet) :



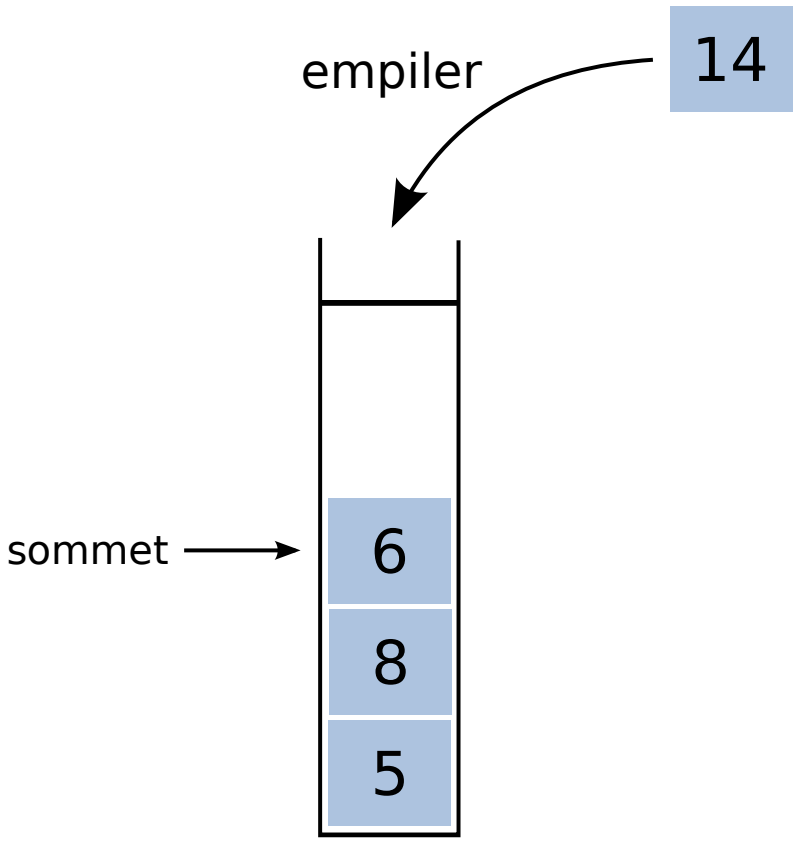
# Pile : LIFO

6 devient la valeur au sommet de la pile :



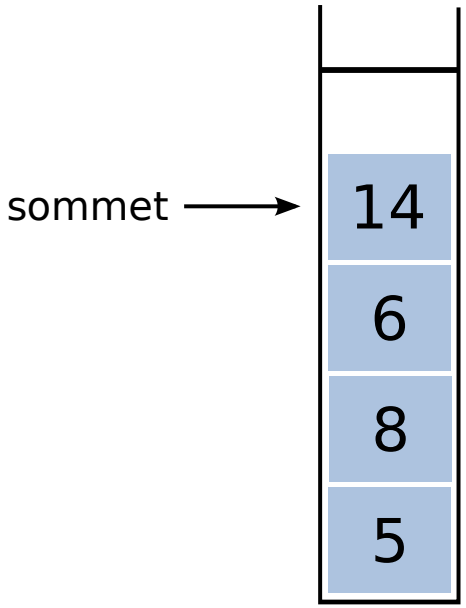
# Pile : LIFO

On empile 14 (ajoute la valeur 14 au sommet) :



# Pile : LIFO

Le sommet de la pile est 14 :



# TDA Pile

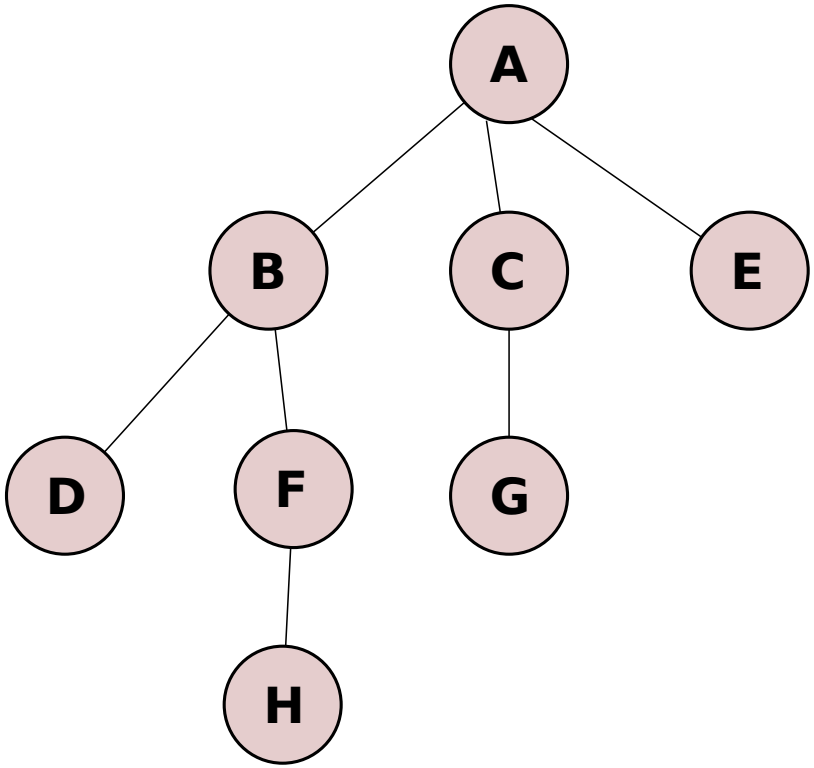
## Opérations

Le type de données abstrait Pile spécifie plusieurs opérations principales :

- `empiler(T val)` : Empile une valeur de type T
- `T dépiler()` : Dépile une valeur
- `T sommet()` : Retourne (sans la retirer) la valeur au sommet de la pile
- `boolean estVide()` : teste si la pile est vide

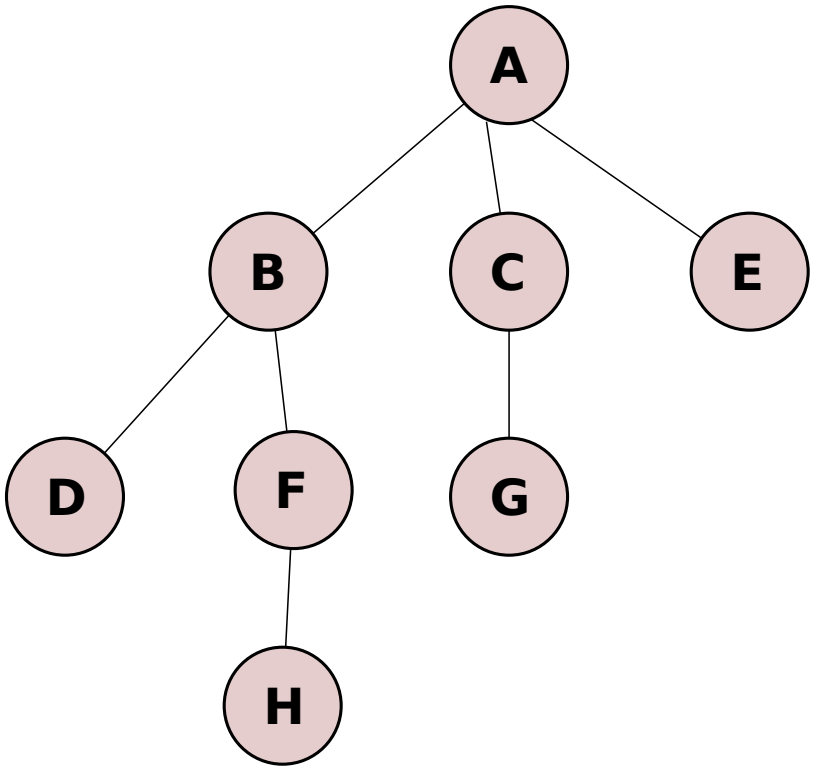
# Pile

Application : Parcours en profondeur



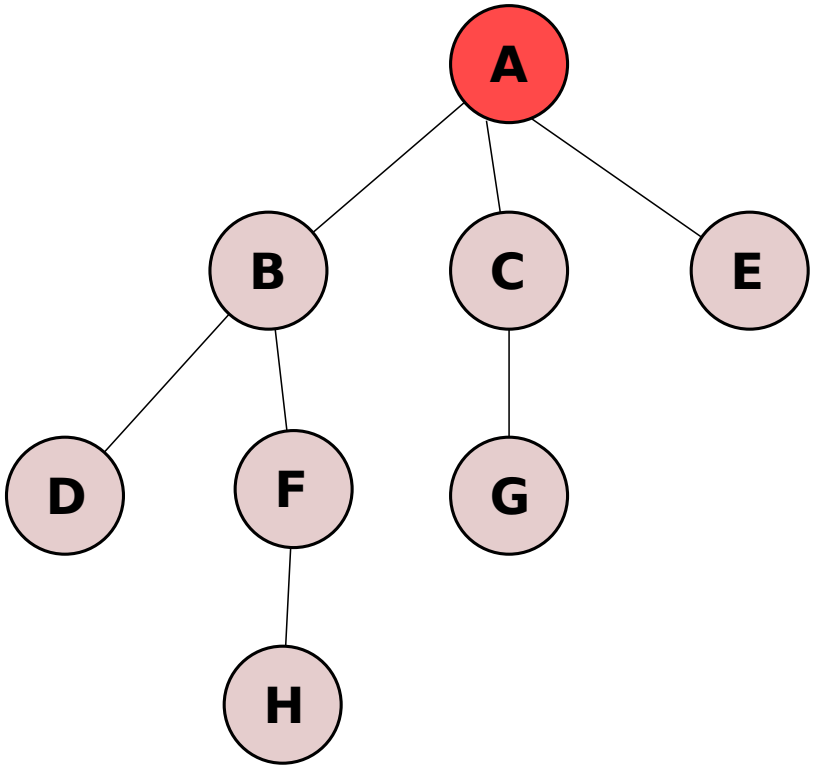
# Pile

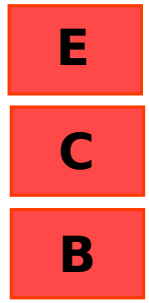
Application : Parcours en profondeur




# Pile

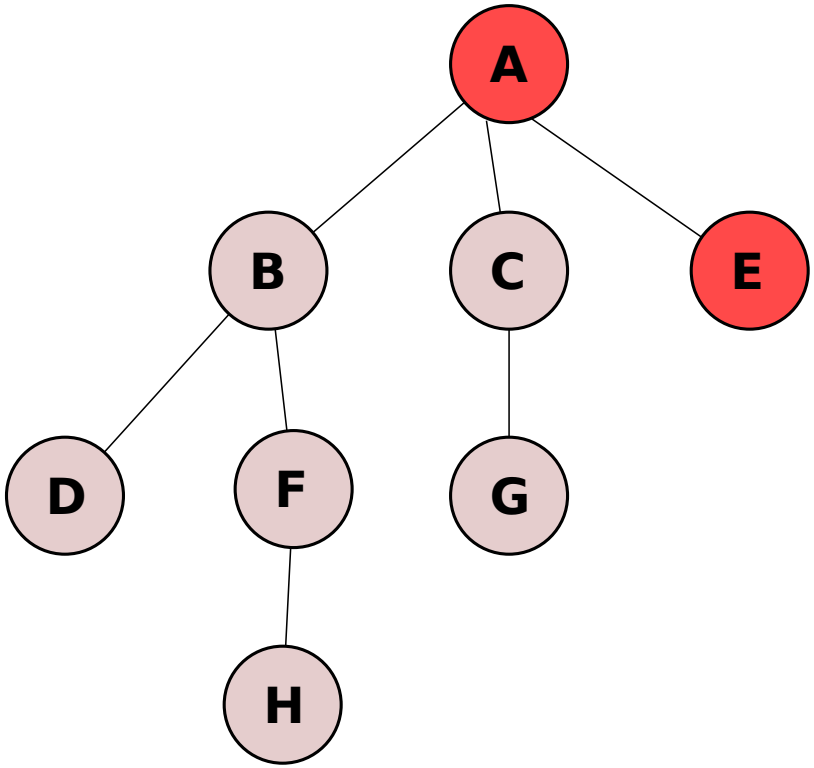
Application : Parcours en profondeur

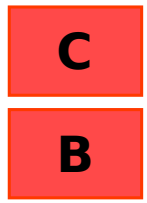


# Pile

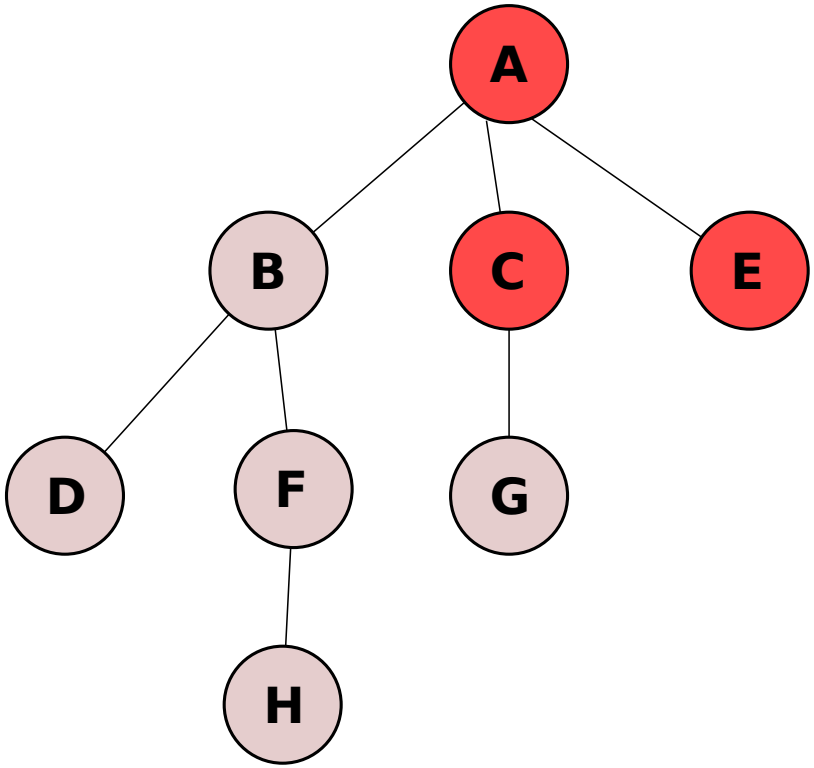
Application : Parcours en profondeur



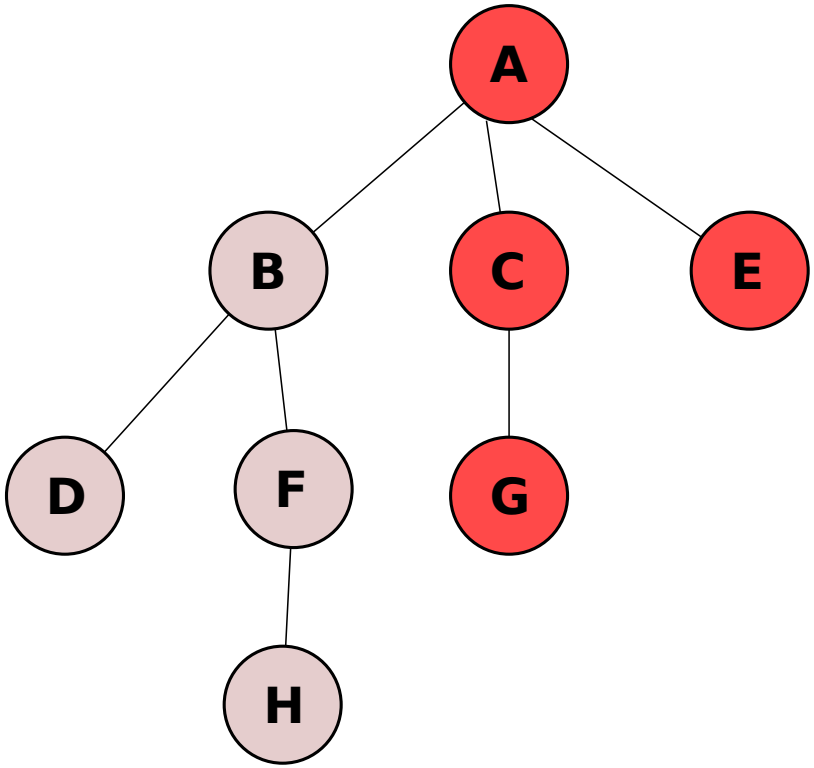
# Pile

Application : Parcours en profondeur



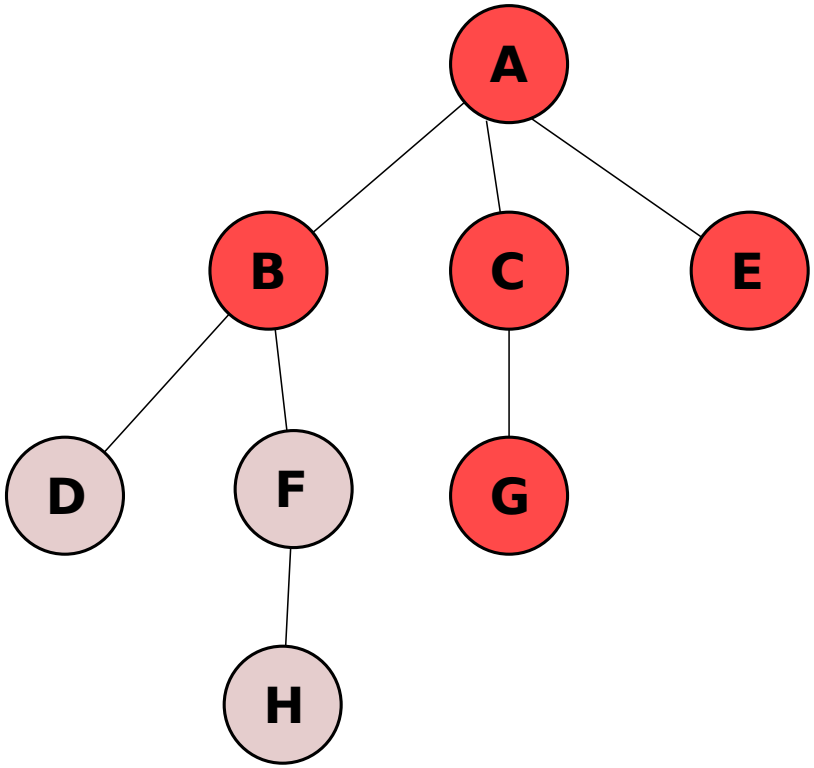

# Pile

Application : Parcours en profondeur



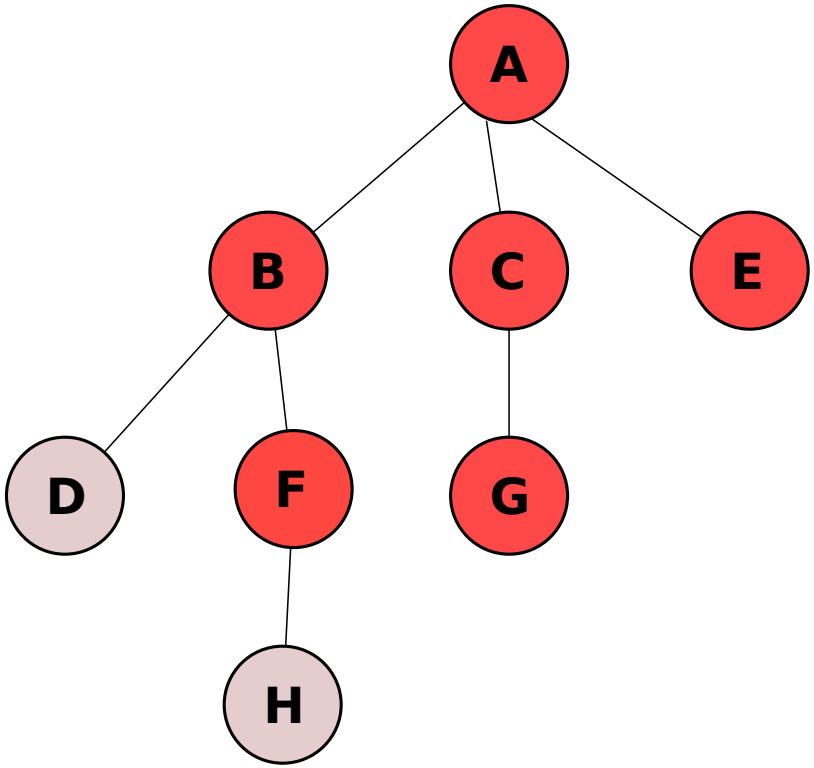

# Pile

Application : Parcours en profondeur



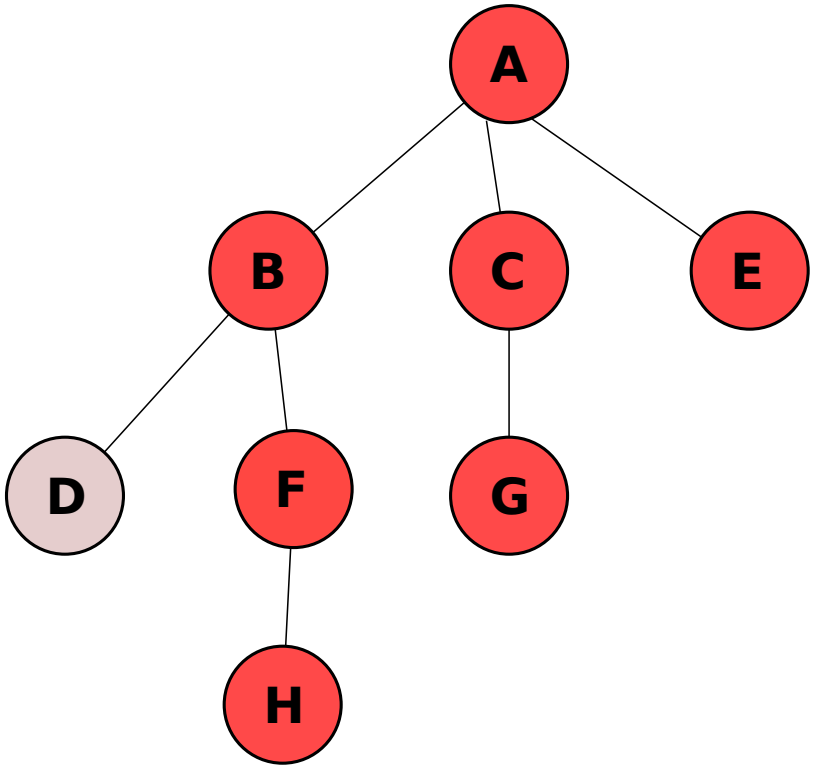

# Pile

Application : Parcours en profondeur



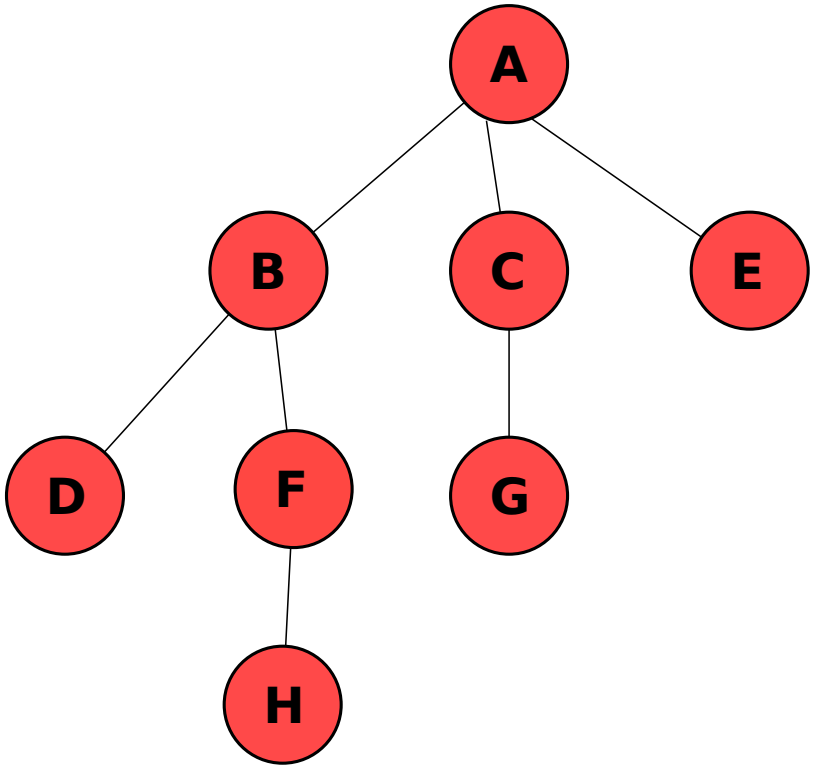

# Pile

Application : Parcours en profondeur




# Pile

Application : Parcours en profondeur



# TDA Pile

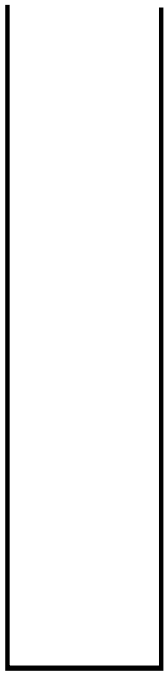
## Implémentation

- La pile est un type de donnée abstrait.
- Classiquement, elle peut être implémentée à partir d'un tableau
- Mais aussi à partir d'une liste chaînée
- En Java, on utilisera la classes `Stack` (synchronisée) ou l'interface `Deque`.

# Exemple

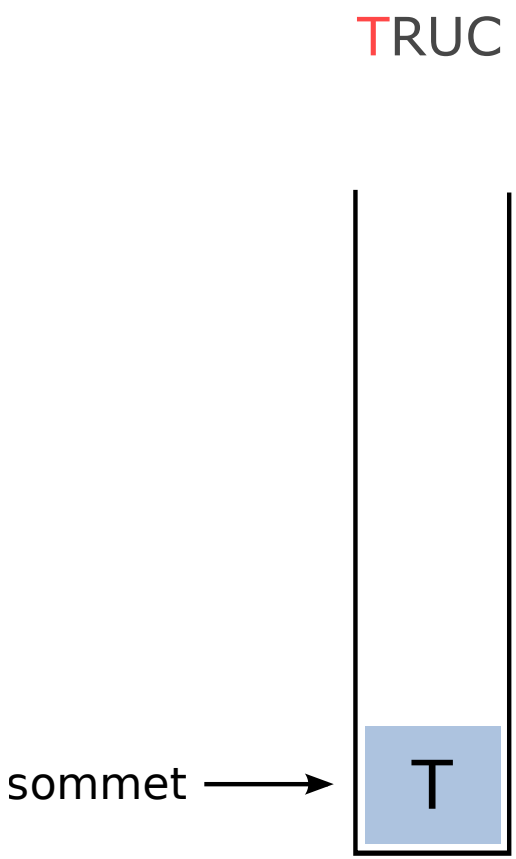
Inverser une chaîne de caractères

TRUC



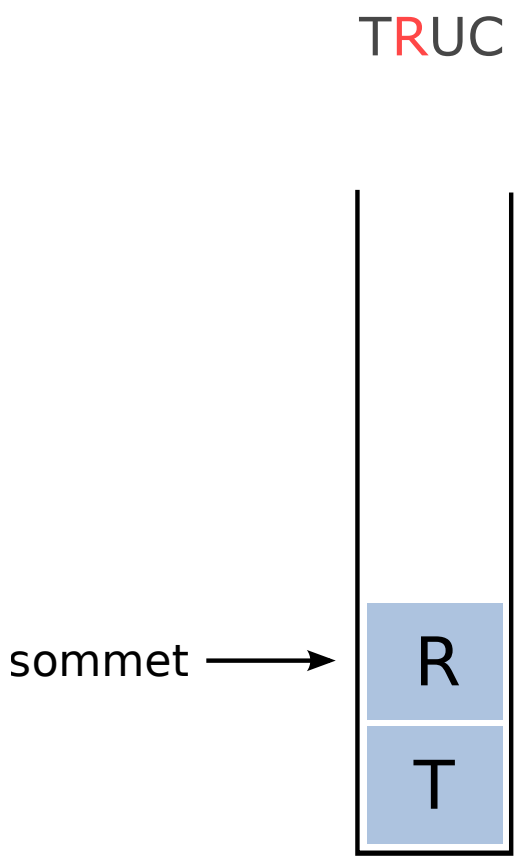
# Exemple

Inverser une chaîne de caractères



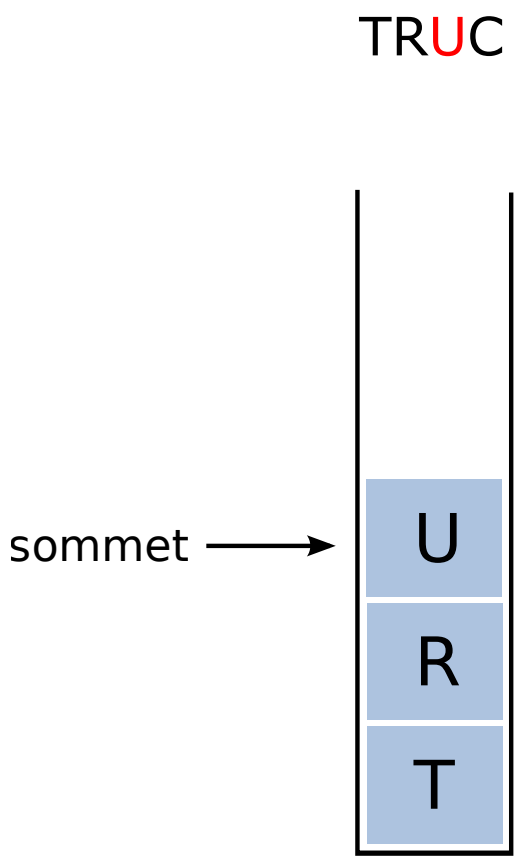
# Exemple

Inverser une chaîne de caractères



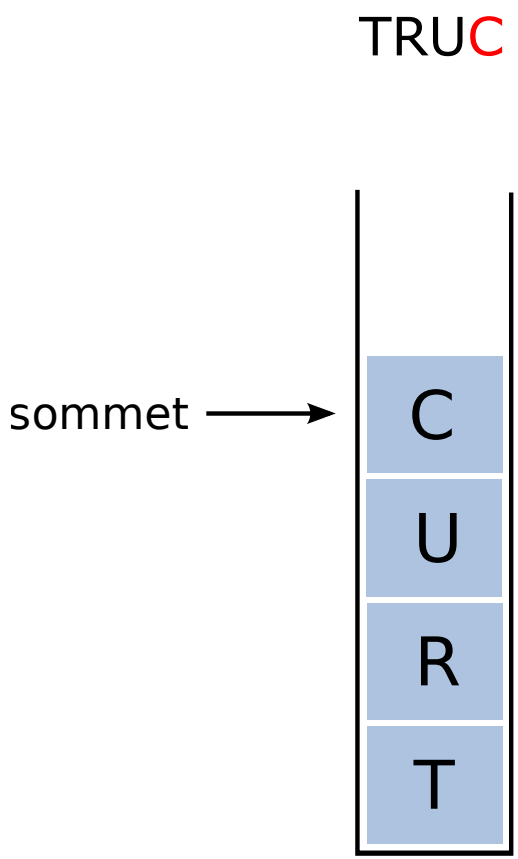
# Exemple

Inverser une chaîne de caractères



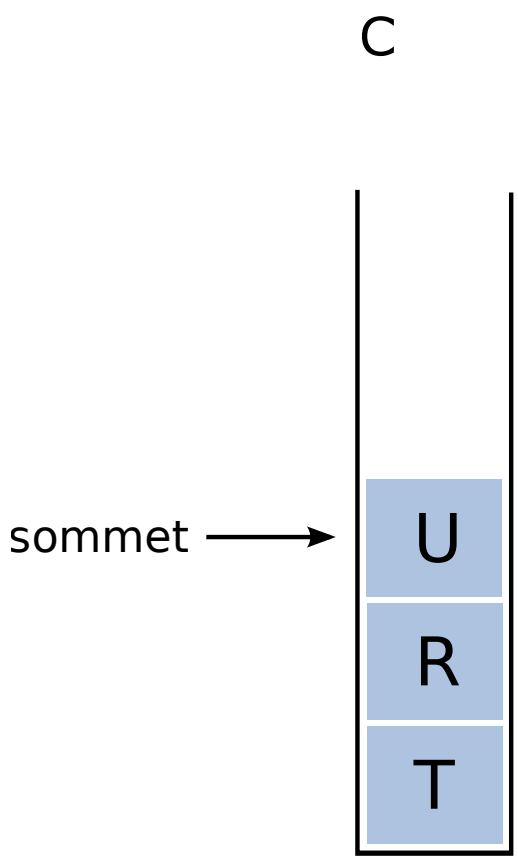
# Exemple

Inverser une chaîne de caractères



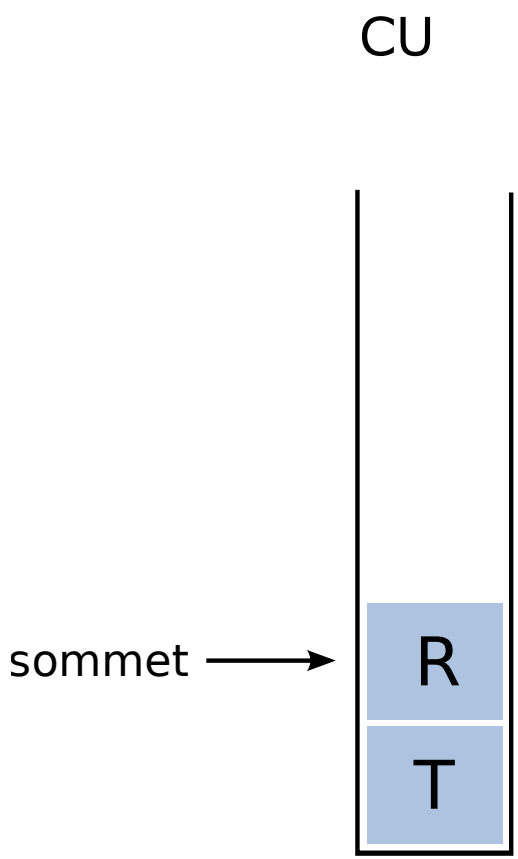
# Exemple

Inverser une chaîne de caractères



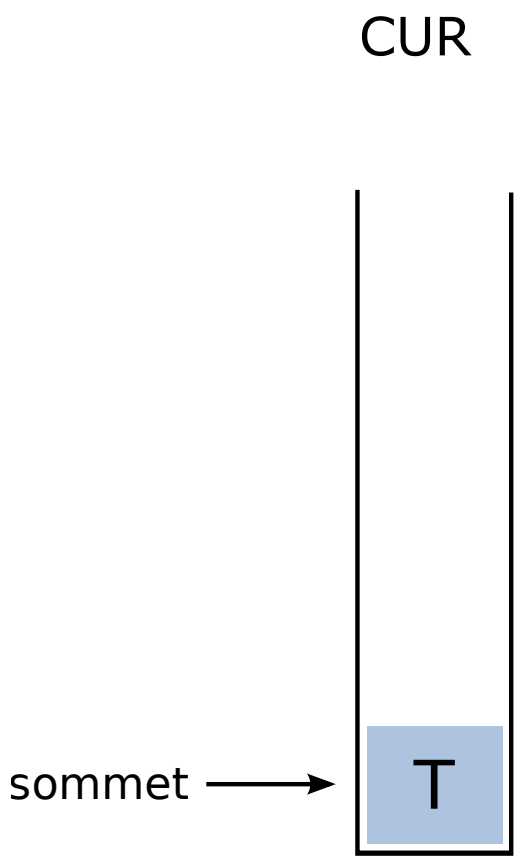
# Exemple

Inverser une chaîne de caractères



# Exemple

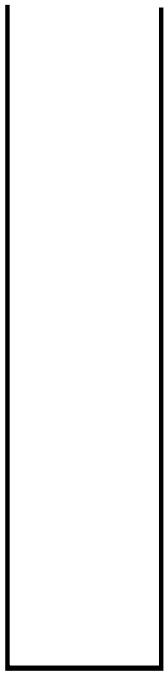
Inverser une chaîne de caractères



# Exemple

Inverser une chaîne de caractères

CURT



# Code

Inverser une chaîne de caractères

```
String input = "TRUC";
String output = "";

Stack <Character> stack = new Stack <Character>();

for(int i = 0; i < input.length(); i++){
  stack.push(input.charAt(i));
}
while(!stack.isEmpty()){
  output+=stack.pop();
}
```

# Des files, pourquoi ?

Quelques exemples d'utilisation :

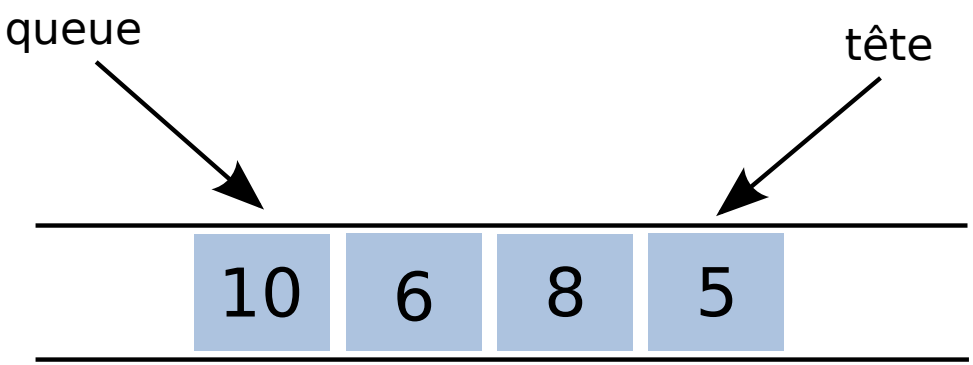
- File d'attente dans un magasin
- Gestion de stocks de denrées périssables
- Ordonnancement de processus

# File

- Une file est un conteneur d'éléments qui respecte le principe du *First In First Out* (FIFO) : le premier élément accessible est le premier ajouté.
- Le premier élément enlevé est le plus ancien de la file
- On insert (enfile) les éléments par la queue et on les retire (défile) par la tête comme dans une file d'attente :

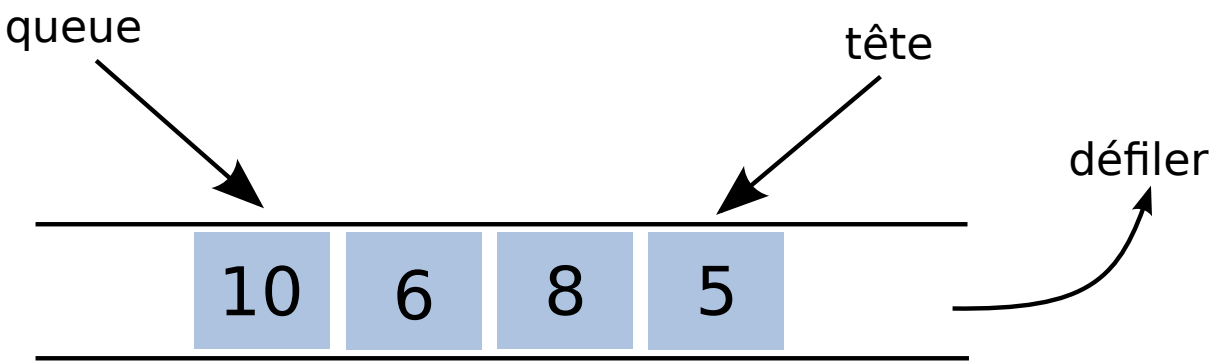
# File : FIFO

File à l'origine :



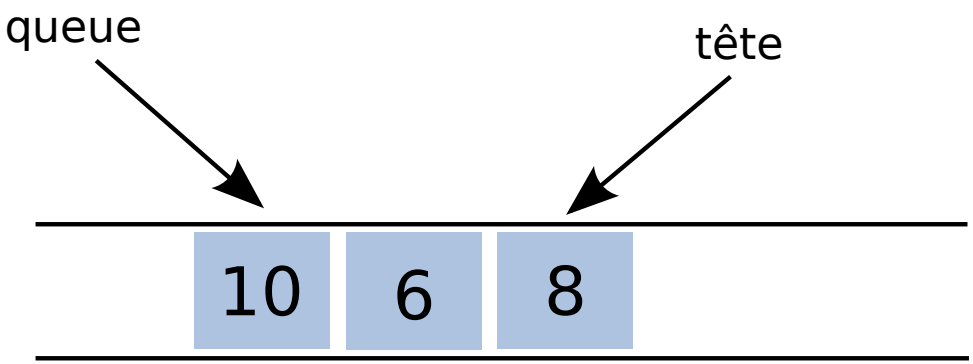
# File : FIFO

On défile la valeur 5 en tête :



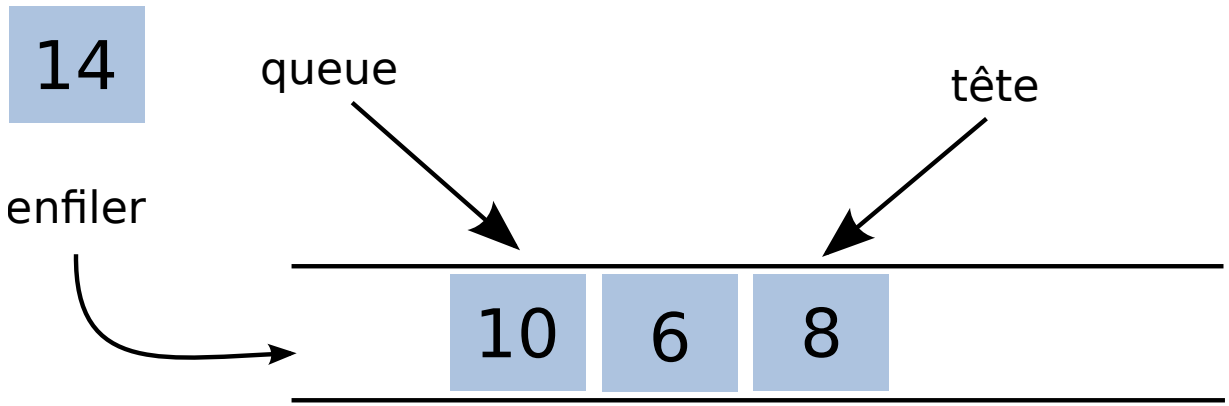
# File : FIFO

8 devient la valeur en tête de ma file :



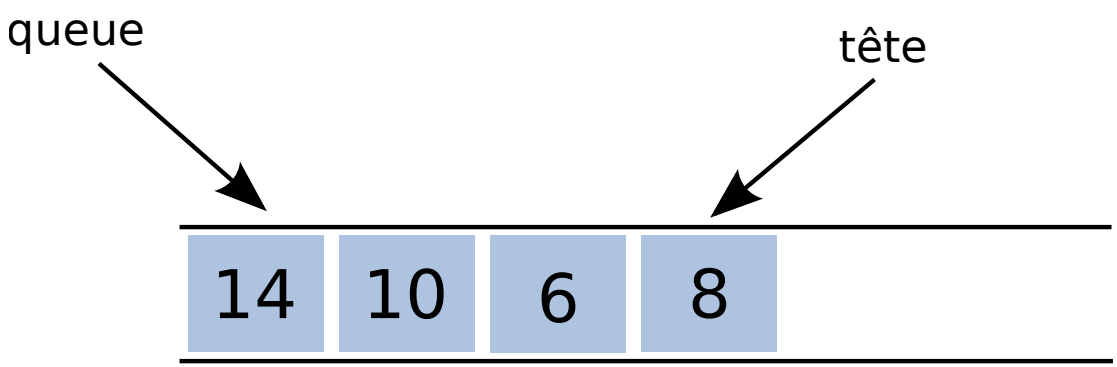
# File : FIFO

On enfile 14 :



# File : FIFO

14 devient la valeur en queue de file :



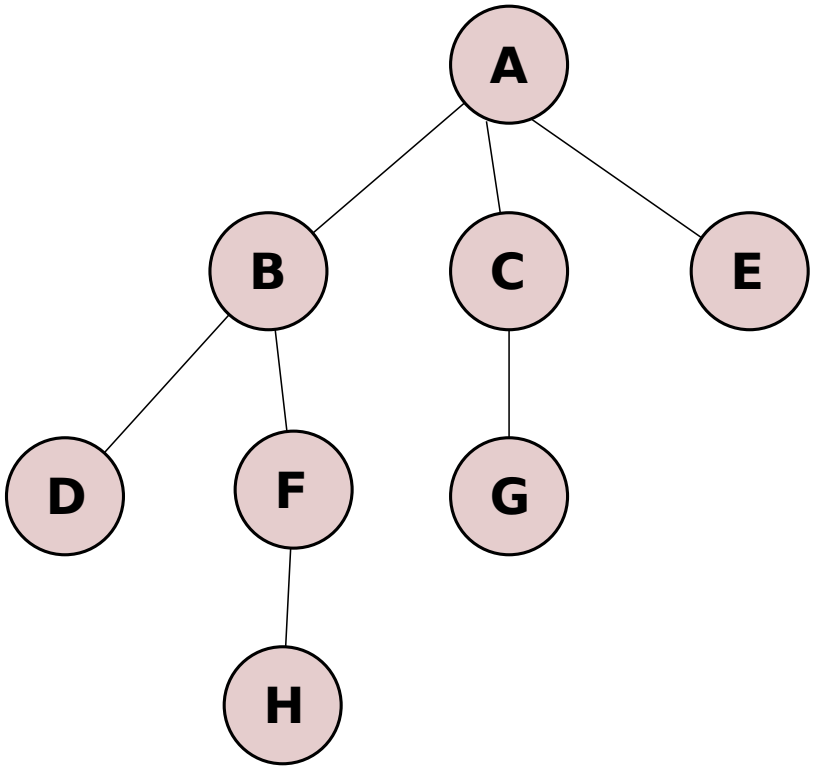
# TDA File

## Opérations

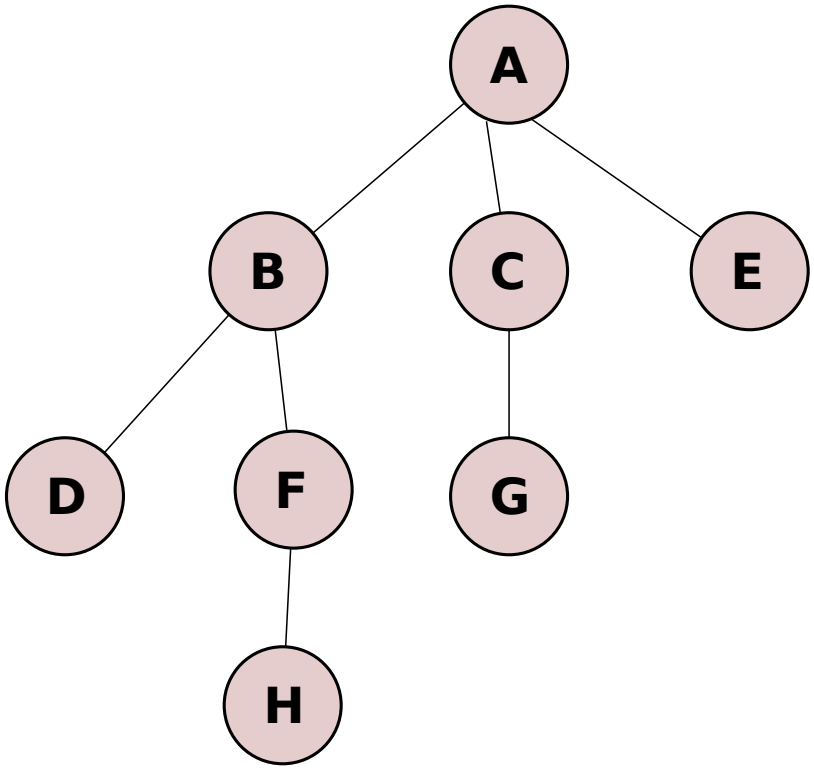
Le type de données abstrait File spécifie plusieurs opérations principales :

- `enfile(T val)` : Ajoute une valeur val de type T en queue de file
- `T défiler()` : Retire une valeur en tête
- `T tête()` : Retourne (sans la retirer) la valeur en tête de file
- `int taille()` : Retourne le nombre d'éléments
- `boolean estVide()` : teste si la file est vide

# Application : Parcours en largeur

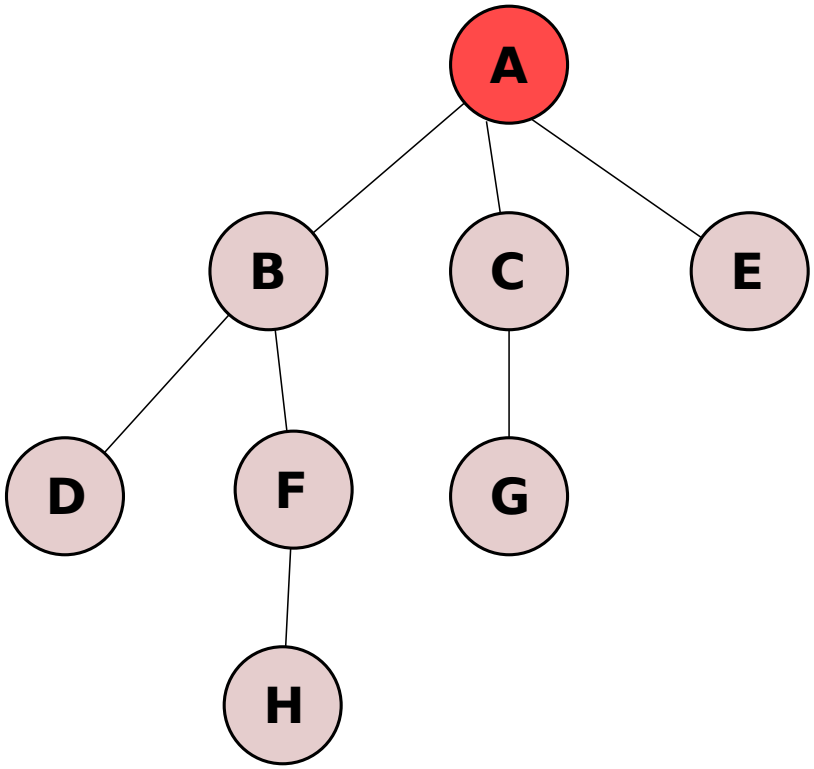


# Application : Parcours en largeur

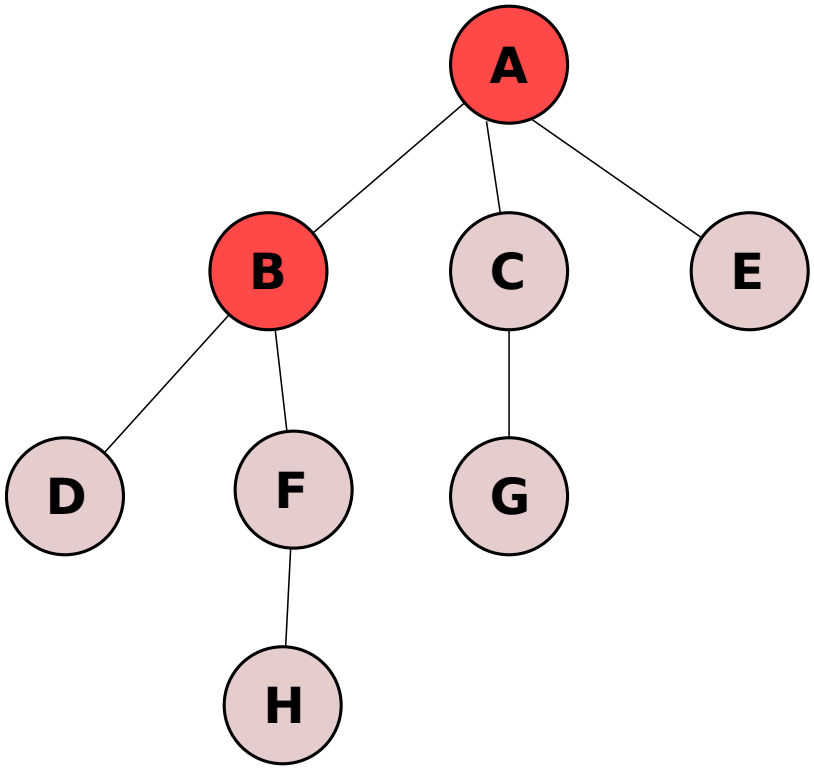



**A**

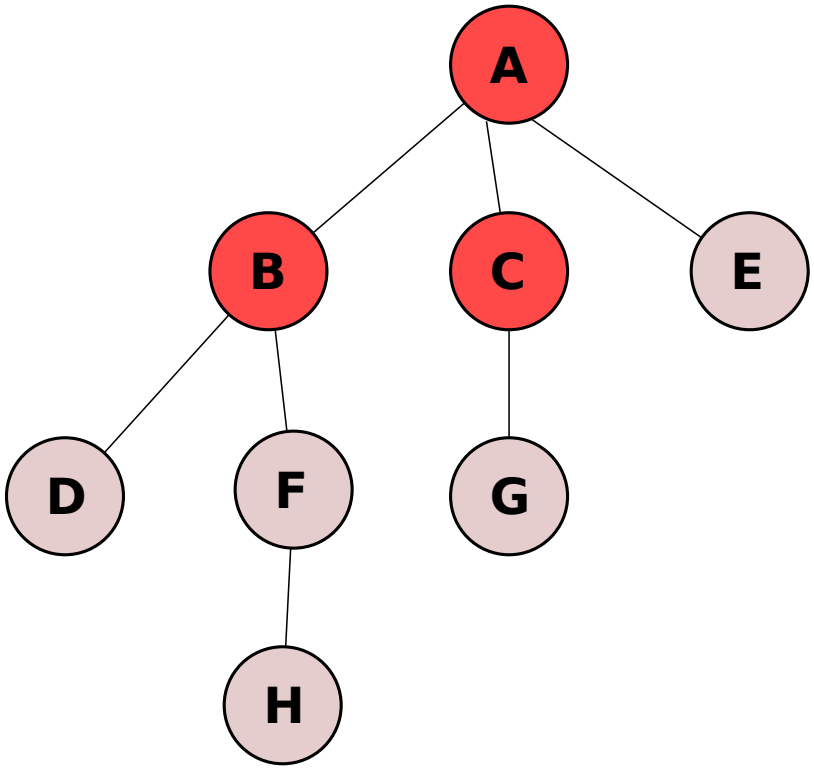
# Application : Parcours en largeur



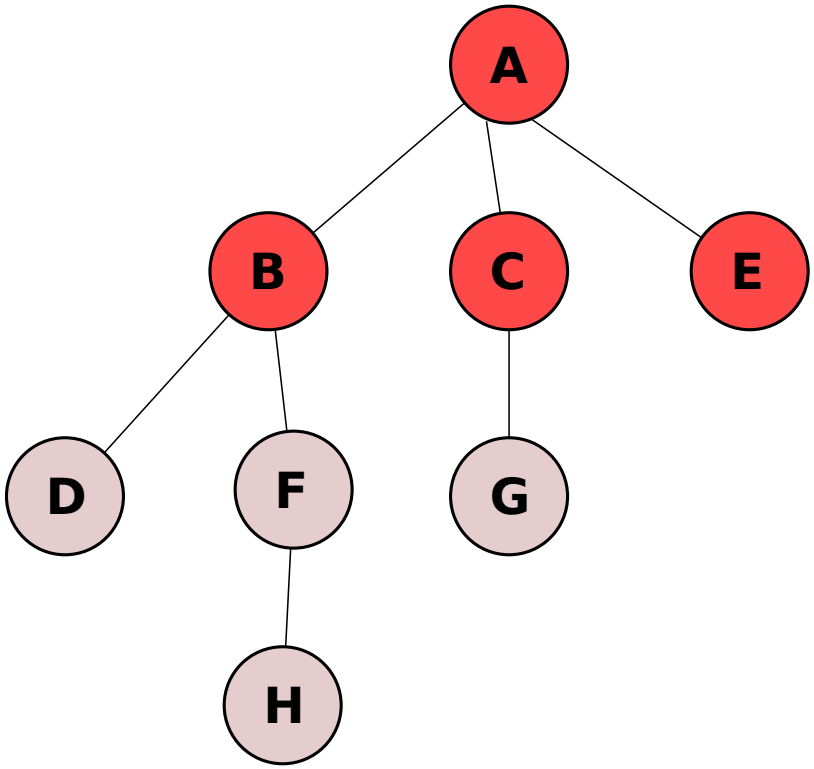

# Application : Parcours en largeur



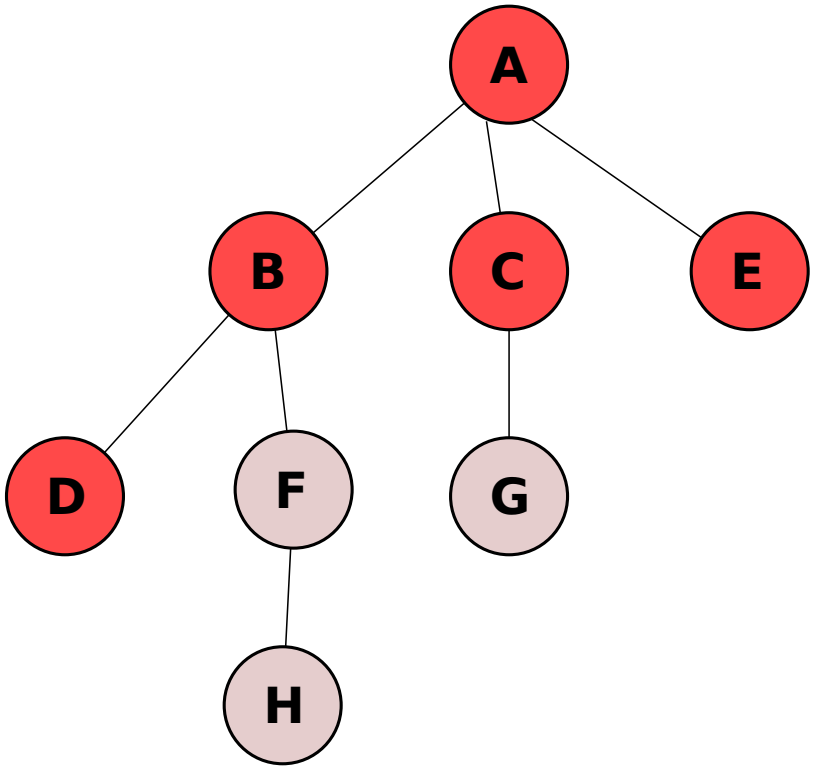

# Application : Parcours en largeur



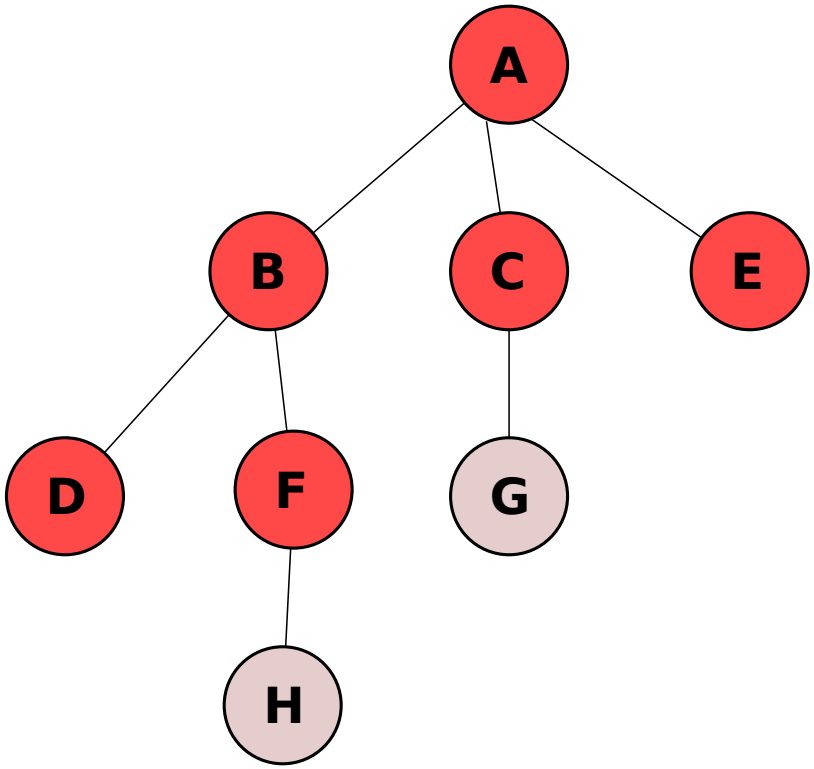

# Application : Parcours en largeur



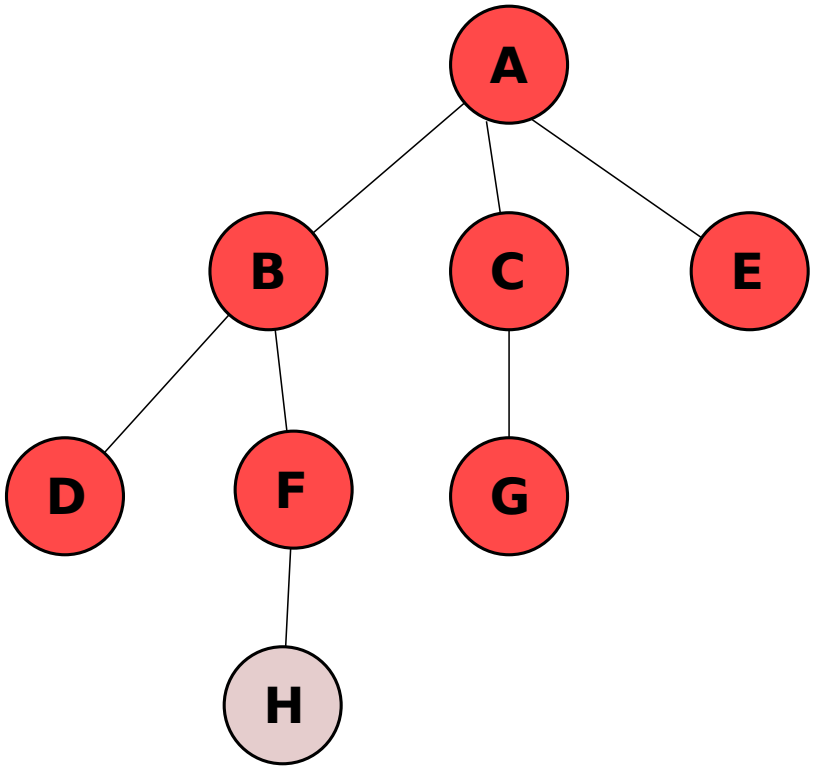

# Application : Parcours en largeur



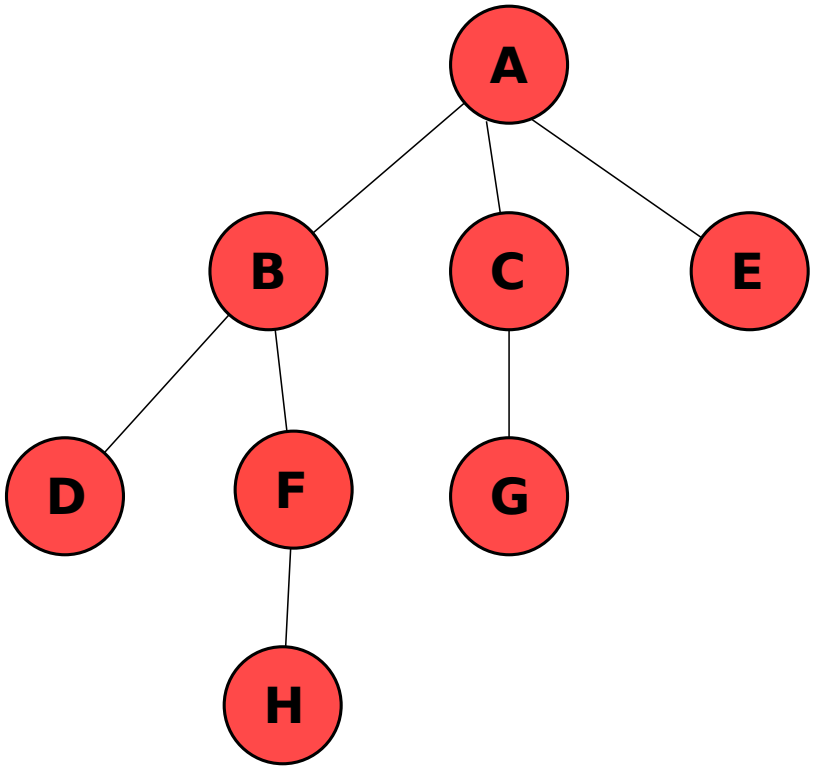

# Application : Parcours en largeur




# Application : Parcours en largeur




# Application : Parcours en largeur



# TDA File

## Implémentation

- La file est un type de donnée abstrait
- Elle pourrait être implémentée à partir d'un tableau
- Mais le plus souvent, une liste est utilisée
- En Java, on utilisera l'interface *Queue* ou *Deque*.

```
public class MyQueue <T>{
    private LinkedList<T> list;

    public MyQueue(){
        list = new LinkedList<T>();
    }
    public void enqueue( T element) {//
        list.addFirst(element);
    }
    public T dequeue() {
        return list.removeLast();
    }

        public static void main(String[] args){
        MyQueue <Integer> q = new MyQueue <Integer>();
        q.enqueue(24);
        q.enqueue(34);
        int i = q.dequeue(); // i = 24
     }
}
```

# Bilan

Dans ce cours nous avons vu :

- Deux nouveaux types de données abstraits,
- Avec un accès limité à un seul élément,
- Les piles : le dernier élément ajouté $\Rightarrow$ premier sorti,
- Les files : le premier élément ajouté $\Rightarrow$ premier sorti.

# Expressions arithmétiques

- Notation infixée : $2 + 5$
- Notation postfixée : $2\ 5\ +$
- Notation préfixée : $+\ 2\ 5$

# Expressions arithmétiques

$70 + 150 * 1.0725 = 235.95$ ou $230.875$ ?

# Expressions arithmétiques

70 + 150 * 1.0725 = 235.95 ou 230.875 ?
Notation postfixée : 70 150 + 1.0725 *

# Expressions arithmétiques

$70 + 150 * 1.0725 = 235.95$ ou $230.875$ ?
Notation postfixée : $70\ 150 + 1.0725 *$
Les parenthèses ne sont nécessaires.

# Conversion infixée vers postfixée

## Principe

- Quand un opérande est lu, on l'ajoute à la sortie
- Quand un opérateur est lu, on l'empile si la pile est vide. Sinon, on dépile tous les opérateurs de priorité égale ou supérieure que l'on ajoute à la sortie et après on empile l'opérateur courant.
- Quand une parenthèse ouvrante est lue, on l'empile
- Quand une parenthèse fermante est lue, on dépile jusqu'à dépiler l'ouvrante et on ajoute les opérateurs à la sortie.
- Quand on a fini de lire l'entrée, on dépile tous les éléments restants qu'on ajoute à la sortie.

# Conversion infixée vers postfixée

Exemple

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile :
- sortie :

# Conversion infixée vers postfixée

- entrée : **(**3+7)*(6-5)/((4-2)*(2+3))
- pile : (
- sortie :

# Conversion infixée vers postfixée

- entrée : (**3**+7)*(6-5)/((4-2)*(2+3))
- pile : (
- sortie : 3

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : ( +
- sortie : 3

# Conversion infixée vers postfixée

- entrée : (3+**7**)*(6-5)/((4-2)*(2+3))
- pile : ( +
- sortie : 3 7

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile :
- sortie : 3 7 +

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : *
- sortie : 3 7 +

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : * (
- sortie : 3 7 +

# Conversion infixée vers postfixée

- entrée : (3+7)*(**6**-5)/((4-2)*(2+3))
- pile : * (
- sortie : 3 7 + 6

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : * ( -
- sortie : 3 7 + 6

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-**5**)/((4-2)*(2+3))
- pile : * ( -
- sortie : 3 7 + 6 5

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-**5**)/((4-2)*(2+3))
- pile : * ( -
- sortie : 3 7 + 6 5

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : *
- sortie : 3 7 + 6 5 -

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile :
- sortie : 3 7 + 6 5 - *

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : /
- sortie : 3 7 + 6 5 - *

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : / (
- sortie : 3 7 + 6 5 - *

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : / ( (
- sortie : 3 7 + 6 5 - *

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : / ( (
- sortie : 3 7 + 6 5 - *

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((**4**-2)*(2+3))
- pile : / ( (
- sortie : 3 7 + 6 5 - * 4

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : / ( ( -
- sortie : 3 7 + 6 5 - * 4

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-**2**)*(2+3))
- pile : / ( ( -
- sortie : 3 7 + 6 5 - * 4 2

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : / (
- sortie : 3 7 + 6 5 - * 4 2 -

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : / ( *
- sortie : 3 7 + 6 5 - * 4 2 -

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : / ( * (
- sortie : 3 7 + 6 5 - * 4 2 -

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(**2**+3))
- pile : / ( * (
- sortie : 3 7 + 6 5 - * 4 2 - 2

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : / ( * ( +
- sortie : 3 7 + 6 5 - * 4 2 - 2

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+**3**))
- pile : / ( * ( +
- sortie : 3 7 + 6 5 - * 4 2 - 2 3

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : / ( *
- sortie : 3 7 + 6 5 - * 4 2 - 2 3 +

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile : /
- sortie : 3 7 + 6 5 - * 4 2 - 2 3 + *

# Conversion infixée vers postfixée

- entrée : (3+7)*(6-5)/((4-2)*(2+3))
- pile :
- sortie : 3 7 + 6 5 - * 4 2 - 2 3 + * /

# Évaluation postfixée

- On a vu comment convertir de l'infixé vers le postfixé
- On va de nouveau utiliser une pile

# Évaluation postfixée

## Principe

- On lit un élément (token) à la fois
- Si c'est un opérande (un entier), on l'empile
- Si c'est un opérateur binaire, on dépile deux éléments sur lesquelles on applique l'opérateur et on empile le résultat.

# Évaluation postfixée

## Exemple

5 9 3 + 4 2 * * 7 + *