Algorithmes utilisant des piles et files

Sébastien Jean

IUT de Valence
Département Informatique

v1.0, 12 décembre 2025

IUT Valence
Université Grenoble Alpes

Evaluation d'expression arithmétiques en notation postfixée

- *La **notation polonaise inverse** (NPI), également connue sous le nom de notation post-fixée, permet d'écrire de façon non ambiguë les formules arithmétiques sans utiliser de parenthèses* (Wikipedia)
- Par exemple, l'expression $3 * (10 + 5)$ peut s'écrire en NPI sous la forme `3 10 5+*` (l'espace sert à séparer 2 nombres sans ambigüité)
- Opérateurs : `+`, `-`, `*`, `/`, `%` et $\sim$ (opposé)
- Les nombres sont des entiers

Exercice

Enoncé du problème

On veut délimiter obtenir la prochaine unité lexicale dans une chaine représentant une NPI (à partir du début).
N.B. On dispose des fonctions :

- `est_chiffre(c : caractère)` : booléen
- `est_opérateur(c : caractère)` : booléen

Spécification du problème

- **Donnée d'entrée** : `ch`, chaîne (la NPI)
- **Donnée de sortie** : `r`, chaîne
- **Pré-condition** : `ch` débute par une unité lexicale valide ou est vide
- **Post-condition** : `r` est la première unité lexicale d'une NPI représentée par `ch` (nombre, opérateur, ou chaine vide)

Signature de la fonction

- **obtenir_lexeme** (ch : chaîne) : chaine

Exercice

```
FONCTION obtenir_lexeme(ch : chaine) : chaîne

 VARIABLE indice : entier
 VARIABLE r      : chaîne
 VARIABLE c      : caractère

 A suivre ...

  // si la chaine est vide, retourner la chaine vide
  SI est_vide(chaine) ALORS
    RETOURNER r
  FIN SI

  A suivre ...
FIN FONCTION
```

IUT Valence
Université Grenoble Alpes

Exercice

... Suite

```
// on traite le premier caractère
indice ← 1
c ← caractère_en(chaine, indice)

// si c' est un opérateur, retourner la chaine
    contenant cet opérateur
SI est_operateur(c) ALORS
   ajouter(r, c)
   RETOURNER r
FIN SI

// si c'est un espace, on l'ignore
SI caractère_en(chaine, indice) = ' ' ALORS
    indice ← indice + 1
FIN SI
```

A suivre ...

IUT Valence Université Grenoble Alpes

Exercice

... Suite

```
// sinon, on délimite le nombre
TANT QUE indice ≤ taille(chaine)
   c ← caractère_en(chaine, indice)
   SI NON est_chiffre(c) ALORS
      RETOURNER r
   FIN SI
   ajouter(r, c)
   indice ← indice + 1
FIN TANT QUE
RETOURNER r
```

Exercice

Enoncé du problème

On veut évaluer une opération arithmétique exprimée en NPI.
N.B. On dispose des fonctions :

- `obtenir_nombre(ch : chaîne)` : entier
- `calculer_binaire(c : caractère, a : entier, b : entier)` : entier

Spécification du problème

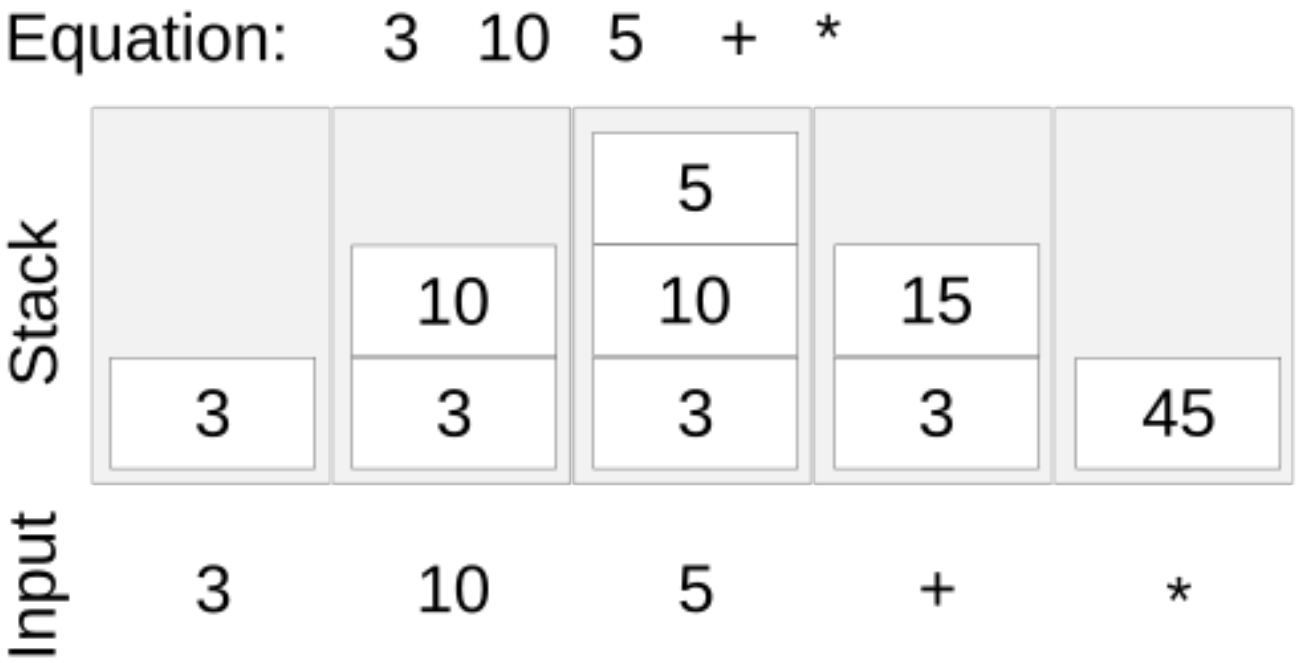
- **Donnée d'entrée** : `ch`, chaîne (la NPI)
- **Donnée de sortie** : `r`, entier (le résultat de l'opération)
- **Pré-condition** : `ch` est une NPI valide
- **Post-condition** : `r` est le résultat de l'opération arithmétique représentée par `ch`

Signature de la fonction

- **évaluer_npi** (ch : chaîne) : entier

Evaluation d'expression arithmétiques en notation postfixée

- Tant que le lexème lu est un opérande on l'**empile**
- Quand le lexème lu est un opérateur, on **dépile un ou 2 opérandes**, on **effectue le calcul** et on **empile le résultat**.
- Illustration : wikipedia



IUT Valence Université Grenoble Alpes

Exercice

```
FONCTION évaluer_npi(ch : chaine) : entier

 VARIABLE copie  : chaîne
 VARIABLE lexeme : chaîne
 VARIABLE c      : caractère
 VARIABLE op1    : entier
 VARIABLE op2    : entier
 VARIABLE p      : pile d'entiers

 copie ← ch

 TANT QUE NON est_vide(copie)
      A suivre ...
 FIN TANT QUE

 RETOURNER voir_sommet(p)

FIN FONCTION
```

IUT Valence
Université Grenoble Alpes

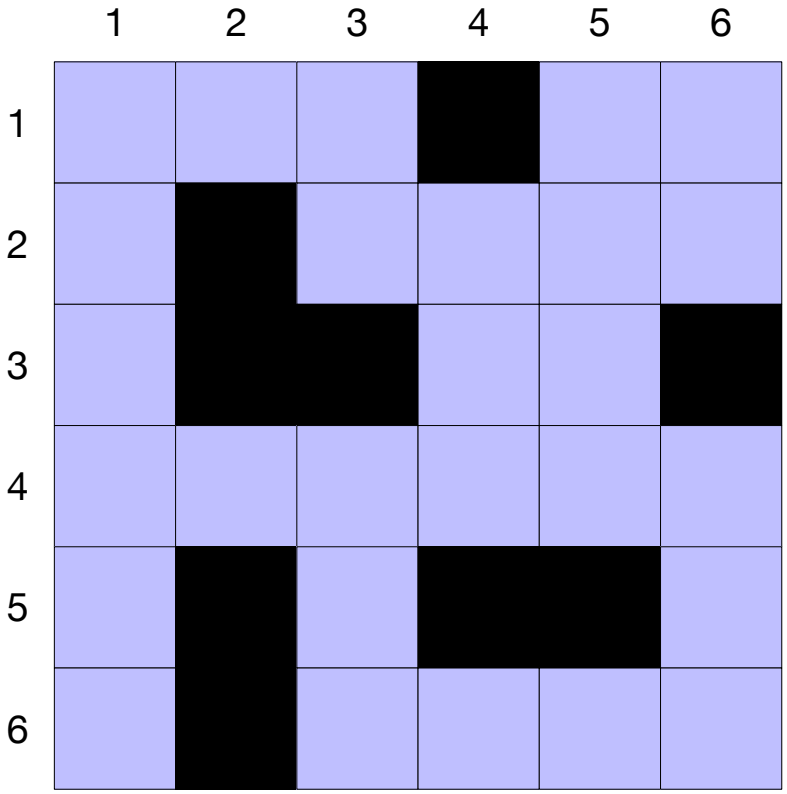
Exercice

```
lexeme ← obtenir_lexeme(copie)
c ← caractère_en(lexeme, 1)
SI est_chiffre(c) ALORS
  empiler(p, obtenir_nombre(lexeme))
SINON
  SI c = '~' ALORS
   op2 ← voir_sommet(p)
   depiler(p)
   empiler(p, -op2)
  SINON
    op2 ← voir_sommet(p)
    depiler(p)
    op1 ← voir_sommet(p)
    depiler(p)
    empiler(p, calculer_binaire(c, op1, op2))
  FIN SI
FIN SI
copie ← sous_chaine(copie,taille(lexeme)+1,taille(copie))
```

IUT Valence Université Grenoble Alpes

Calcul de plus court chemin sur une grille

- On s'intéresse à des déambulations sur des grilles où les cases sont vides ou occupées par des murs
- Les cases occupées par des murs ne peuvent être traversées



La **grille** est représentée par un **ensemble** contenant les **positions des murs**.

N.B. l'ensemble inclut le pourtour de la grille (colonnes 0 et 7, lignes 0 et 7)

IUT Valence Université Grenoble Alpes

Exercice

- On définit un enregistrement `Position` permettant de représenter une Position sur la grille

```
ENREGISTREMENT Position

   CHAMPS ligne   : entier
   CHAMPS colonne : entier

FIN ENREGISTREMENT
```

Exercice

Enoncé du problème

On souhaite trouver le plus court chemin entre 2 positions de la grille

Spécification du problème

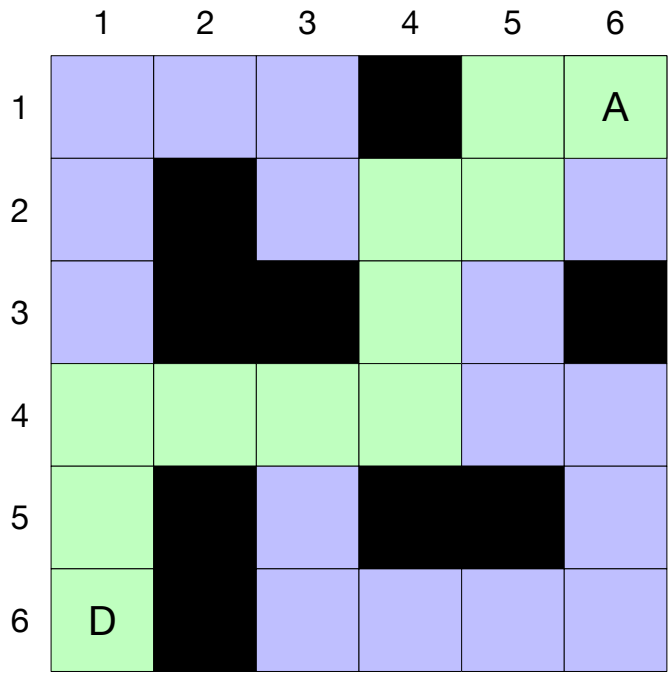
- **Donnée d'entrée** : `murs`, Ensemble de Position (les murs de la grille)
- **Donnée d'entrée** : `depart`, Position (la position de départ)
- **Donnée d'entrée** : `arrivee`, Position (la position d'arrivée)
- **Donnée de sortie** : `r`, Vecteur de Position (le plus court chemin)
- **Pré-condition** : `murs` représente une grille valide, `depart` et `arrivee` sont 2 positions libres et différentes dans la grille.
- **Post-condition** : `r` est un plus court chemin entre `depart` (1er élément) et `arrivée` (dernier élément), vecteur vide si pas de chemin

Signature de la fonction

- **plus_court** (murs : Ensemble de Position, depart : Position, arrivee : Position) : Vecteur de Position

Calcul de plus court chemin sur une grille

- on construit des chemins en s'éloignant de la position de départ
- les chemins s'étendent par les positions voisines (haut/bas/gauche/droite)
- on abandonne les chemins qui conduisent à des murs ou une position déjà visitée
- le premier chemin trouvé conduisant à l'arrivée est le plus court



IUT Valence
Université Grenoble Alpes

Exercice

```
FONCTION plus_court(murs : Ensemble de Position, depart :
    Position,arrivee : Position) : Vecteur de Position

  // chemins à explorer
 VARIABLE chemins : File de Vecteur de Position
  // positions visitées
 VARIABLE visite  : Ensemble de Position
  // chemin courant
 VARIABLE chemin  : Vecteur de Position
  //  chemin temporaire
 VARIABLE copie   : Vecteur de Position
 // voisins
 VARIABLE voisins : Tableau de Position[4]
 VARIABLE ligne   : entier
 VARIABLE colonne : entier
 VARIABLE position : Position
 VARIABLE indice  : entier

 A suivre ...
```

IUT Valence Université Grenoble Alpes

Exercice

...Suite

```
ajouter(visite, depart)
inserer(chemin, 1, depart)
ajouter(chemins, chemin) // chemin à explorer au début

TANT QUE est_vide(chemins) = FAUX
    A suivre ...
FIN TANT QUE

vider(copie)      // si pas de chemin trouvé,
RETOURNER copie // vecteur vide en retour
```

IUT Valence Université Grenoble Alpes

Exercice

...Suite

```
// Explorer un des chemins en cours
// (chemins les plus courts en tête de file)
chemin ← voir_prochain(chemins)
retirer(chemins)

A suivre ...

// s'intéresser à la dernière position du chemin
// énumérer ses voisins

// pour chaque voisin : poursuite possible ?
//    si oui, ajouter aux chemins le  chemin constitué
//    du chemin en cours augmenté du voisin
```

IUT Valence Université Grenoble Alpes

Exercice

```
...Suite
// s'intéresser à la dernière position du chemin
position ← lire(chemin, taille(chemin))

// énumérer ses voisins
voisins[1].ligne    ← position.ligne
voisins[1].colonne  ← position.colonne - 1
voisins[2].ligne    ← position.ligne
voisins[2].colonne  ← position.colonne + 1
voisins[3].ligne    ← position.ligne - 1
voisins[3].colonne  ← position.colonne
voisins[4].ligne    ← position.ligne + 1
voisins[4].colonne  ← position.colonne

A suivre ...
// pour chaque voisin : poursuite possible ?
//    si oui, ajouter aux chemins le  chemin constitué
//    du chemin en cours augmenté du voisin
```

IUT Valence Université Grenoble Alpes

Exercice

```
...Suite
// pour chaque voisin : poursuite possible ?
//    si oui, ajouter aux chemins le  chemin constitué
//    du chemin en cours augmenté du voisin
POUR indice de 1 A 4 PAR PAS DE 1
    position ← voisins[indice]
    copie ← chemin
    SI appartient(murs, position) = FAUX ALORS
      ajouter(copie, position)
      SI position = arrivee ALORS
        RETOURNER copie
      SINON
        SI appartient(position, visite) = FAUX ALORS
          ajouter(visite, position)
          ajouter(chemins, copie)
        FIN SI
       FIN SI
     FIN SI
FIN POUR
```

IUT Valence Université Grenoble Alpes

Fin !