Algorithmique et structures de données en C

Piles et Files

Enseignant: P. Bertin-Johannet

Une pile

- Une pile est une collection d'éléments ainsi que deux fonctions
- Une fonction empiler pour ajouter un élément.
- Une fonction dépiler pour retirer un élément.
- Les éléments sont retirés dans l'ordre inverse de celui dans lequel ils ont été enpilés.

```
Pile* p = creer_creer_pile();
empiler(p, 5);
empiler(p, 6);
printf("%d\n", depiler(p)); // affiche 6
```

LIFO et pile

- On parle aussi de LIFO (Last In First Out) car l'élément inséré en dernier sera retiré en premier.

```
Pile* p = creer_creer_pile();
empiler(p, 1);
empiler(p, 2);
empiler(p, 3);
empiler(p, 4);
printf("%d\n", depiler(p)); // affiche 4
printf("%d\n", depiler(p)); // affiche 3
```

Pile - Implémentation par tableau

- On peut implémenter une pile en utilisant un tableau (ou un vecteur si on souhaite une taille modifiable).
- Pour empiler, on ajoute un élément à la fin du tableau.
- Pour dépiler on retire l'élément à la fin du tableau.
- Pour cela, il faut enregistrer la position du dernier element et la mettre à jour à chaque opération.

```
struct Pile {
  int* elements;
  int fin_tableau;
};
```

Pile - Implémentation par Liste chainée

- On peut aussi implémenter une pile en utilisant une liste chainée.
- Pour empiler, on ajoute un élément au début de la liste.
- Pour dépiler on retire l'élément au début de la liste.
- Toutes ces opérations s'executent alors en temps constant.

Une File

- Une file est une collection d'éléments ainsi que deux fonctions.
- Une fonction enfiler pour ajouter un élément.
- Une fonction défiler pour retirer un élément.
- Les éléments sont retirés dans l'ordre dans lequel ils ont été enpilés.

```
File* p = creer_file();
enfiler(p, 5);
enfiler(p, 6);
printf("%d\n", defiler(p)); // affiche 5
```

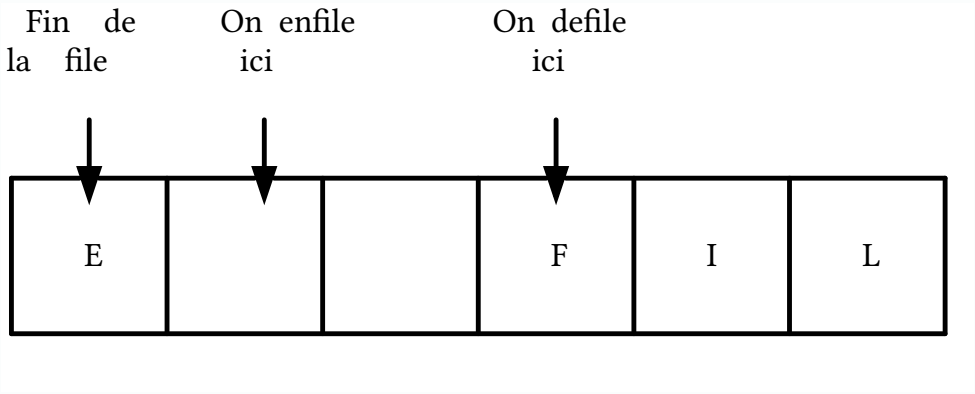
FIFO et file

- On parle aussi de FIFO (First In First Out) car l'élément inséré en premier sera retiré en premier.

```
File* p = creer_creer_pile();
enfiler(p, 1);
enfiler(p, 2);
enfiler(p, 3);
enfiler(p, 4);
printf("%d\n", defiler(p)); // affiche 1
printf("%d\n", defiler(p)); // affiche 2
```

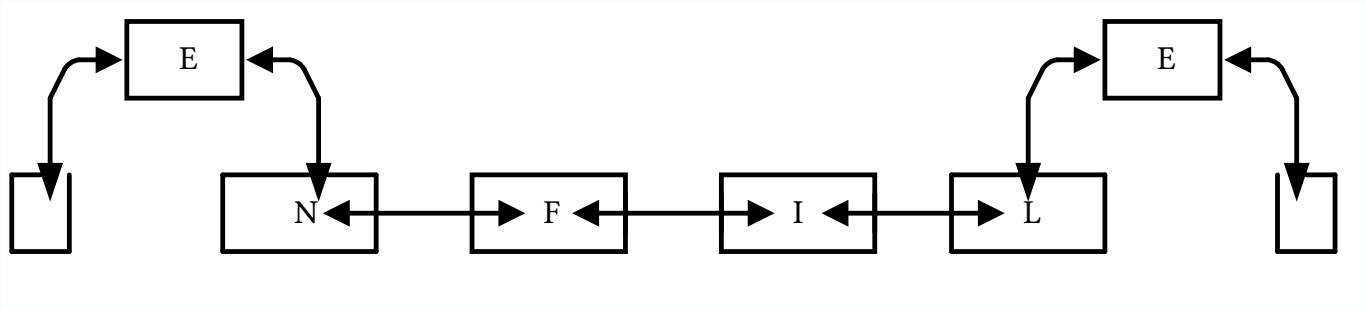
File - Implémentation par tableau circulaire

- On peut implémenter une file en utilisant un tableau circulaire.
- Il faudra enregistrer le début du tableau pour défiler.
- Il faudra retenir la fin du tableau pour enfiler.
- La file aura alors une taille limitée.



File - Implémentation par Liste doublement chainée

- On peut aussi implémenter une file en utilisant une liste doublement chainée.
- Pour enfiler, on ajoute un élément au début de la liste.
- Pour défiler on retire l'élément à la fin de la liste.
- Toutes ces opérations s'executent en temps constant grace aux noeuds sentinelles.



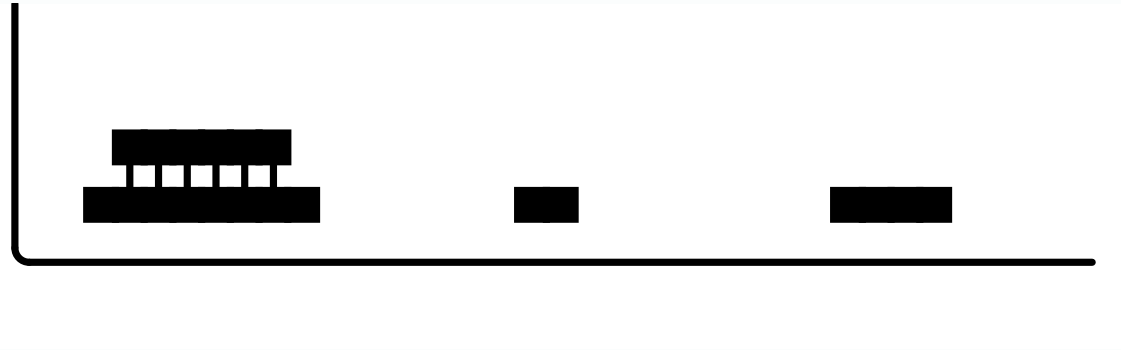
Utilisations de file et pile

- Au delà des usages évidents (pile de fonction, file de requêtes à traiter) les piles et files apparraissent dans de nombreux algorithmes
- Ce cours décrit quelques exemples d'usages de piles et de files.

Recursion

Les tours de hanoi

- On commence avec quatres disques de tailles différentes empilés les uns sur les autres.
- On dispose de trois tours.
- L'objectif est de déplacer tous les disques d'une tour vers une autre.
- Pour cela on peut déplacer un seul disque à la fois en le déposant sur un disque plus grand que lui.
- Par exemple dans le cas ci-dessous, on ne peut pas déplacer les disques de gauche car ils sont plus grands que les disques du milieu et de droite.



Les tours de hanoi - Solution

La résolution des tours de hanoi consiste à déplacer les plus petites pièces sur une tour intermédiaires avant de les redéplacer

- Si cherche à déplacer une tour de taille 1 vers la fin.

Les tours de hanoi - Solution

La résolution des tours de hanoi consiste à déplacer les plus petites pièces sur une tour intermédiaires avant de les redéplacer

- Je déplace le disque directement.

Les tours de hanoi - Solution

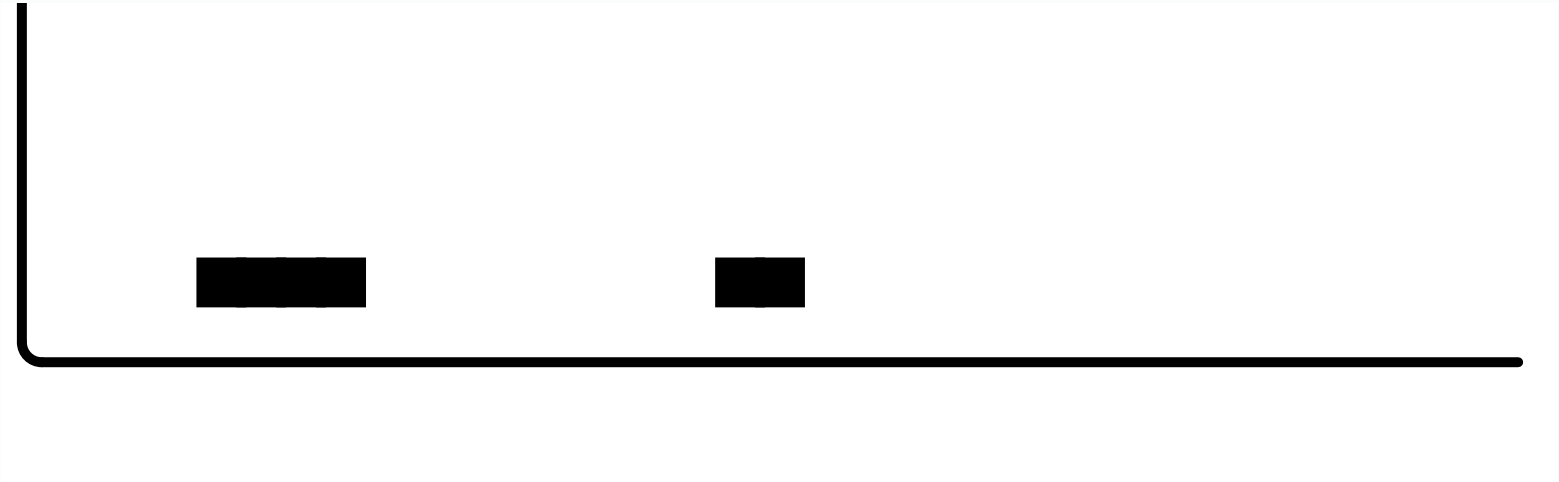
La résolution des tours de hanoi consiste à déplacer les plus petites pièces sur une tour intermédiaires avant de les redéplacer

- Si cherche à déplacer une tour de taille 2 vers la fin.

Les tours de hanoi - Solution

La résolution des tours de hanoi consiste à déplacer les plus petites pièces sur une tour intermédiaires avant de les redéplacer

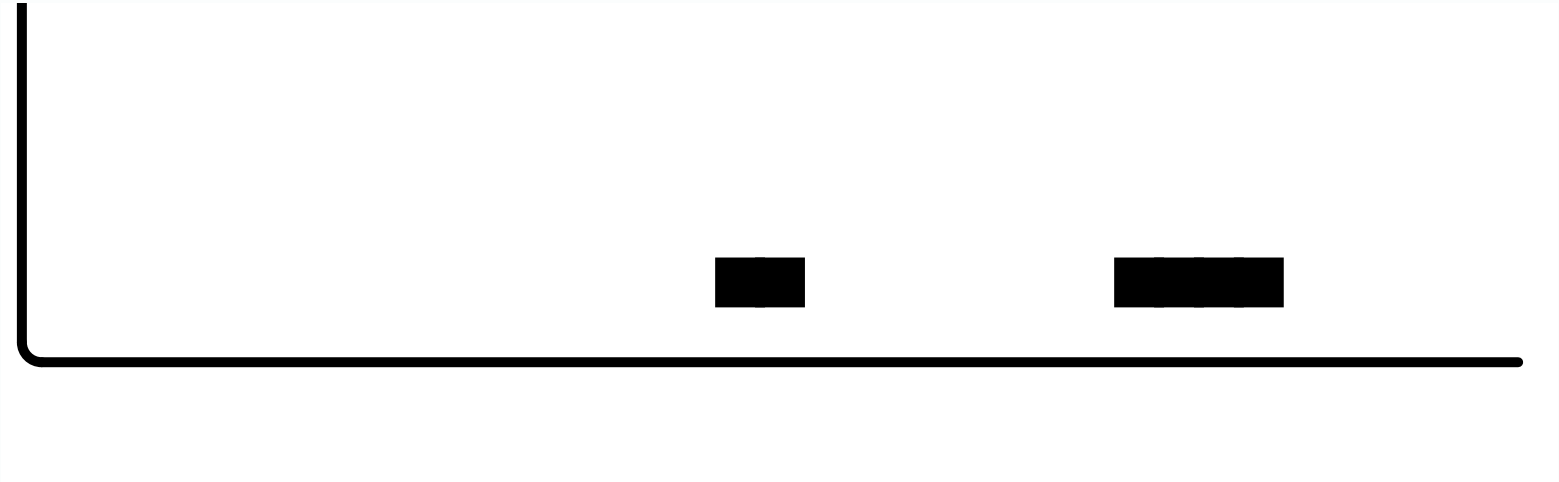
- Je déplace le disque du haut au milieu.



Les tours de hanoi - Solution

La résolution des tours de hanoi consiste à déplacer les plus petites pièces sur une tour intermédiaires avant de les redéplacer

- Puis je déplace le disque du bas à la fin.



Les tours de hanoi - Solution

La résolution des tours de hanoi consiste à déplacer les plus petites pièces sur une tour intermédiaires avant de les redéplacer

- Si cherche à déplacer une tour de taille 2 vers la fin.
- Enfin, je déplace le petit disque à la fin.

Les tours de hanoi - Solution générale

- On peut généraliser cette technique :
- Si on sait déplacer n-1 disques d'une tour vers une autre, on peut en déplacer n d'une tour A à une tour B ainsi :
  - On déplace n-1 disques de la tour A vers une tour C en utilisant cet algorithme
  - On déplace le disque qui reste de la tour A vers la tour B
  - On déplace les n-1 disques de la tour C vers la tour B
- Cette solution s'applique donc à toute taille de tour

Les tours de hanoi - Solution récursive

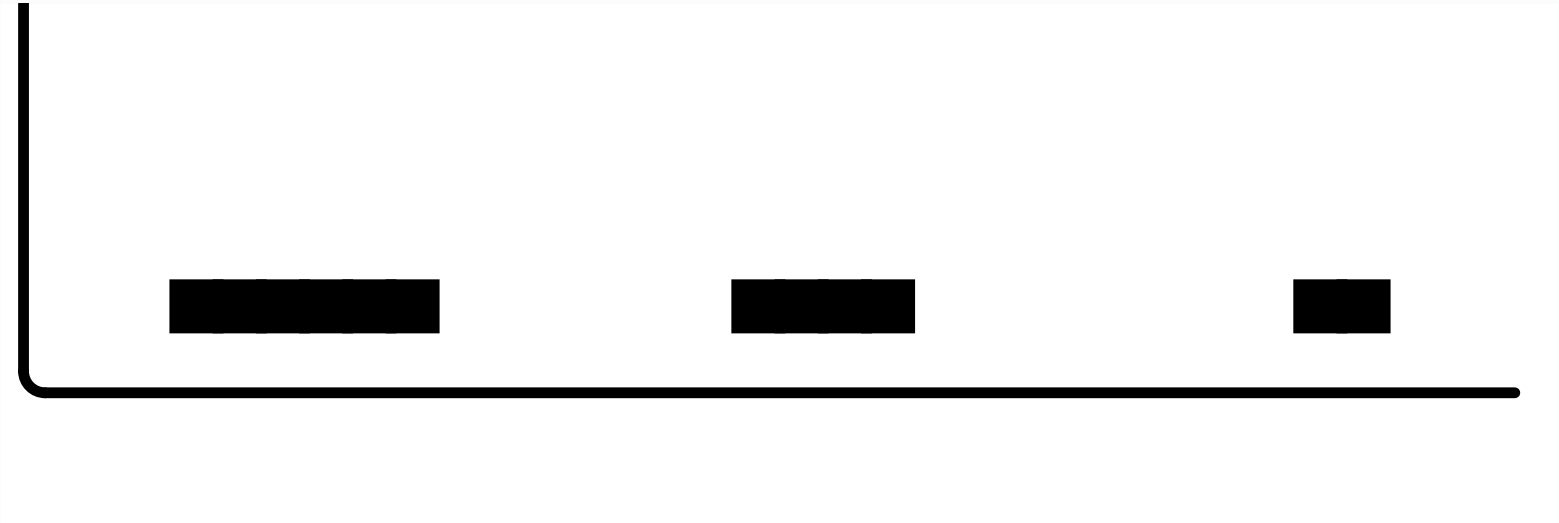
```
// fonction qui déplace n disques de depart vers fin en utilisant milieu
void tours_de_hanoi(int n, char depart, char fin, char milieu) {
  if (n == 1) { // si il n'y a qu'un disque on le déplace directement
      move_from_to(depart, fin);
  } else { // sinon on effectue la stratégie décrite plus haut
    // on déplace n-1 disques du départ vers le milieu
    tours_de_hanoi(n - 1, depart, milieu, fin);
    // on déplace un disque du départ vers la fin
    tours_de_hanoi(1, depart, fin, milieu);
    // on déplace n-1 disques du milieu vers la fin
    tours_de_hanoi(n - 1, milieu, fin, depart);
  }
}
```

Tours de hanoi - exemple à trois disques

le disque 1 va de 0 a 2

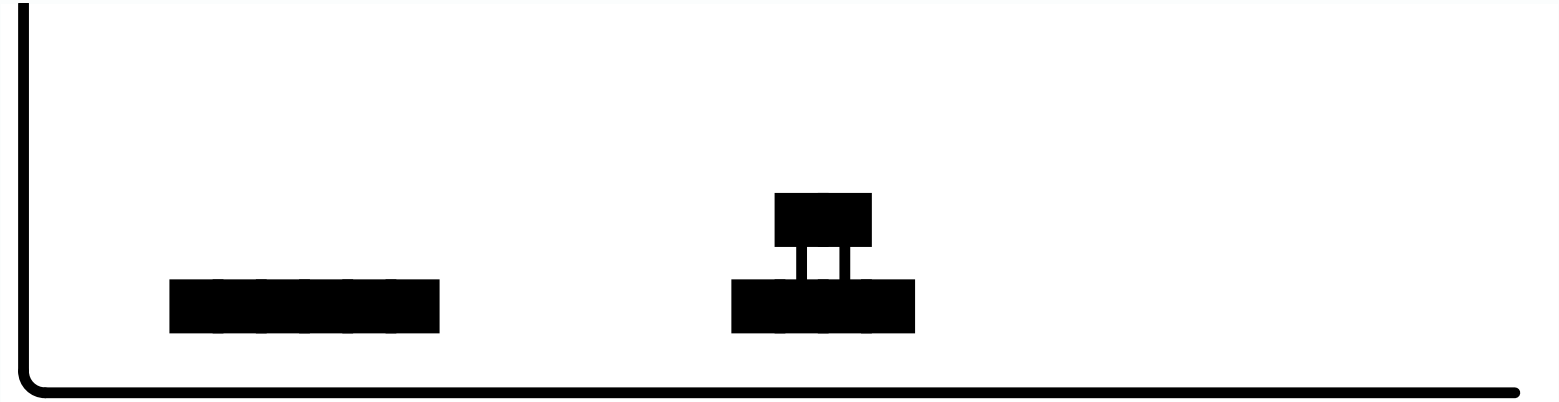
Tours de hanoi - exemple à trois disques

le disque 2 va de 0 a 1



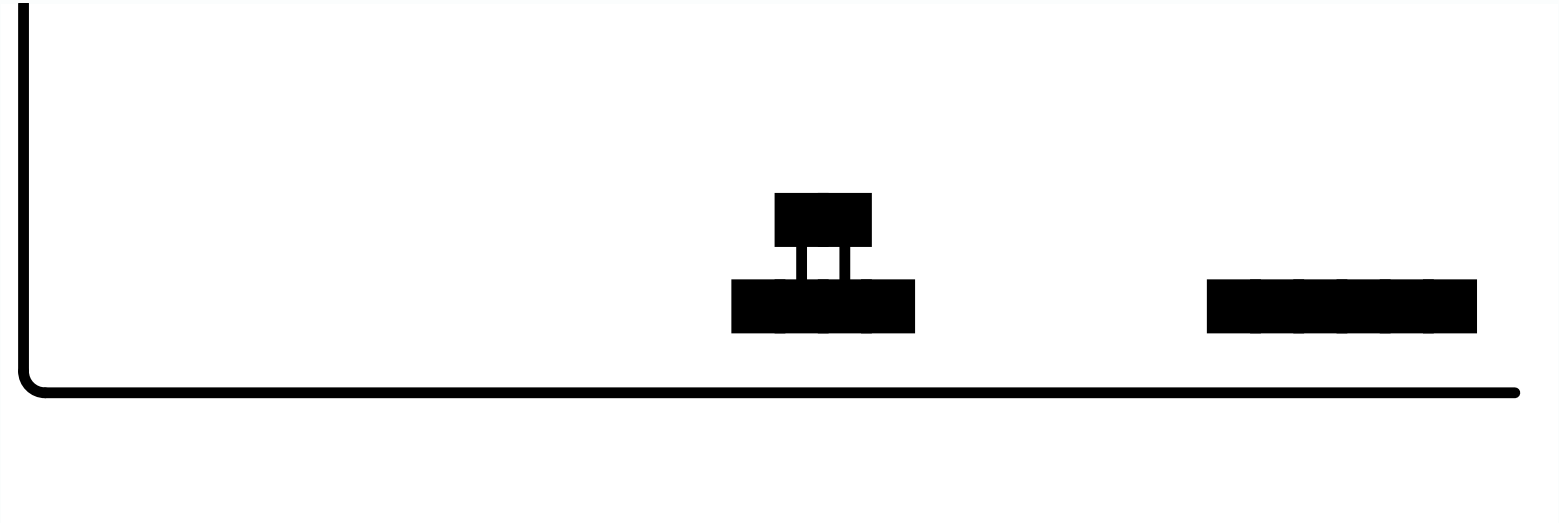
Tours de hanoi - exemple à trois disques

le disque 1 va de 2 a 1



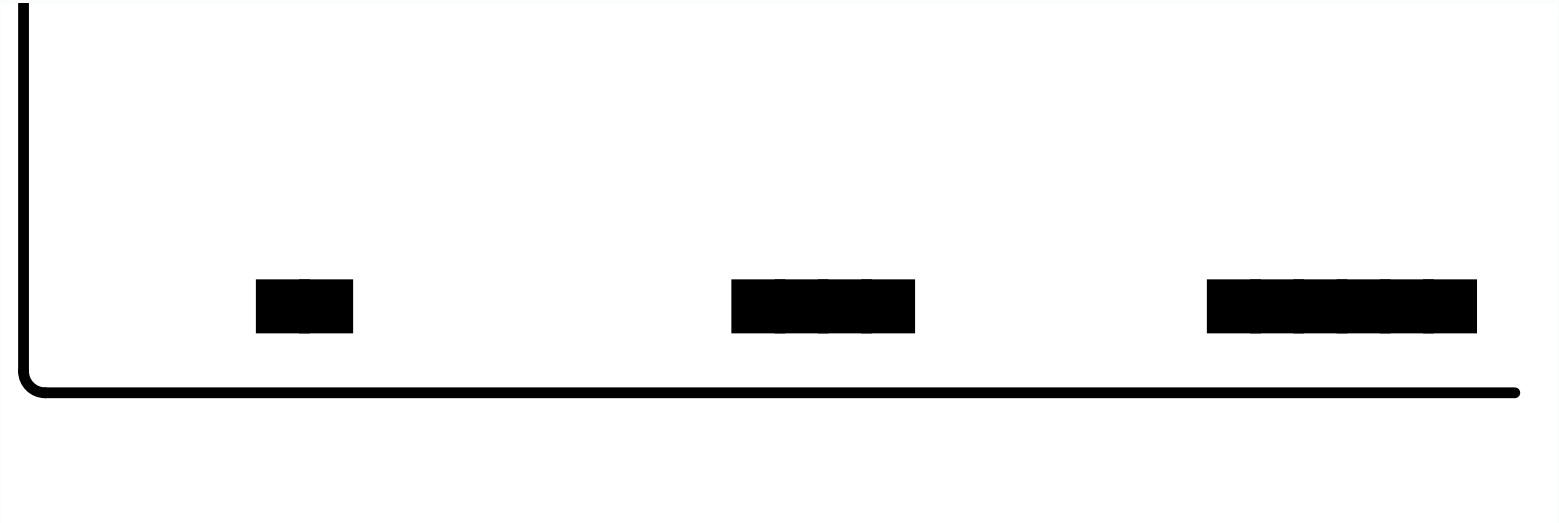
Tours de hanoi - exemple à trois disques

le disque 3 va de 0 a 2



Tours de hanoi - exemple à trois disques

le disque 1 va de 1 a 0



Tours de hanoi - exemple à trois disques

le disque 2 va de 1 a 2

Tours de hanoi - exemple à trois disques

le disque 1 va de 0 a 2

Problèmes de la récursion

- Chaque appel de fonction alloue de l'espace dans la pile pour enregistrer l'état de la mémoire et de l'execution.
- A chaque changement de fonction, l'execution du code est interrompue et les valeurs copiées.
- La mémoire des fonctions est dans la stack, qui est de taille limitée et très inferieure au tas.
- La récursion pose donc des problèmes de rapidité d'execution, de consommation mémoire mais aussi des risques d'erreur (stackoverflow si la pile est trop remplie).

Retrait de la récursion

- Il est possible d'utiliser une pile pour supprimer tout usage de la récursion.
- Au lieu d'appeller la fonction récursivement, on enregistre les paramètres dans la pile.
- On execute alors le code dans une boucle tant que la pile n'est pas vide.

Retrait de la récursion dans les tours de hanoi

```
typedef struct Objectif{int n, char depart, char fin, char milieu} Objectif;

void tours_de_hanoi(Objectif depart) {
  pile p;
  empiler(p, base);
  while (taille(p) > 0){
      Objectif courant = depiler(p);
      if (courant.n == 1) {
          move_from_to(courant.depart, courant.fin);
      } else {
          empiler(p, cree_objectif(courant.n - 1, courant.milieu, courant.fin, courant.depart));
          empiler(p, cree_objectif(1, courant.depart, courant.fin, 0));
          empiler(p, cree_objectif(courant.n - 1, courant.depart, courant.milieu, courant.fin));
      }
  }
}
```

Expressions

Calcul d'opérations

- Soit l'opération

$$d + (2 * a - 1) / c + 3 * b.$$

- Quel est le résultat de cette opération pour les valeurs suivantes ?

$$a = 5$$

$$b = 4$$

$$c = 3$$

$$d = 2$$

Calcul d'opérations

- Soit l'opération

$$d + (2 * a - 1) / c + 3 * b.$$

- Quel est le résultat de cette opération pour les valeurs suivantes ?

$$a = 5$$

$$b = 4$$

$$c = 3$$

$$d = 2$$

- $2 + \frac{2*5-1}{3} + 3 * 4 = 17$

Calcul d'opérations

- Le calcul d'opérations sous la forme standard est complexe.
- Il faut prendre en compte les parenthèses.
- Il faut prendre en compte la priorité des opérateurs.
- Cela se complexifie encore plus lorsque l'ordre d'évaluation devient important.

Exemple: Quel sera le résultat de l'opération ci-dessous ? (Pour a = 2 et tab = {1, 2, 3, 4, 5})

```
--a + tab[a += a--] += ++a .
```

Notation polonaise inversée

- Il existe une forme plus simple à executer appelée notation polonaise inversée.
- Pour cela les symboles d'opérations seront écrit **après** leurs opérandes.
- Exemple :
  - Notation standard : 1 + 2 * 4
  - Notation polonaise inversée : 1 2 4 * +

Notation polonaise inversée

- Il existe une forme plus simple à executer appelée notation polonaise inversée.
- Pour cela les symboles d'opérations seront écrit **après** leurs opérandes.
- Exemple :
  - Notation standard : 1 + 2 * 4
  - Notation polonaise inversée : 1 2 4 * +

Note: Les fonctions en C, par exemple, utilisent une notation non-inversée.

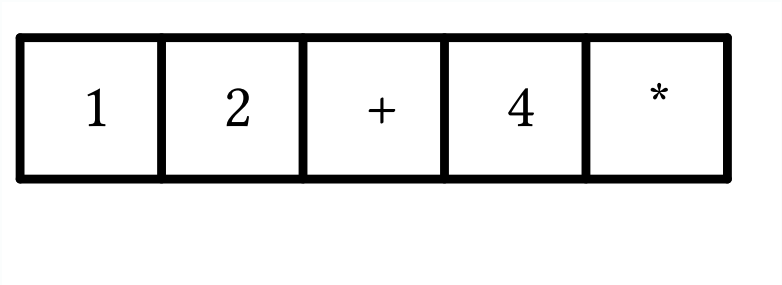
Dans l'appel `cos(pow(2, 3))`*, les opérations* `pow` *et* `cos` *arrivent avant leurs arguments.*

Notation polonaise inversée - Avantages

- Pas besoin de parenthèses, pas d'ambiguités.
- Les opérations arrivent dans l'ordre dans lequel elles doivent être executées:
  - Notation standard : 3 * (2 + 4) + 1
  - Notation polonaise inversée : 3 2 4 + * 1 +
- Il est facile d'utiliser des opérations acceptant plus de 2 arguments (c'est pourquoi on utilise par exemple la Notation Polonaise non Inversée pour les appels de fonctions).
- Une expression en NPI est souvent plus rapide à executer.

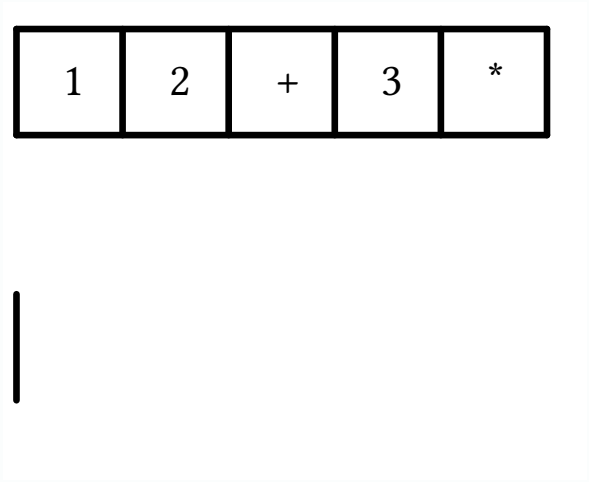
Notation polonaise inversée - calcul

- On peut executer une expression en NPI avec l'aide d'une pile.
- Pour cela on parcours l'expression :
  - lorsqu'on trouve un nombre, on l'enpile.
  - Lorsqu'on trouve une opération, on dépile assez de nombres pour le calcul et on empile le résultat.



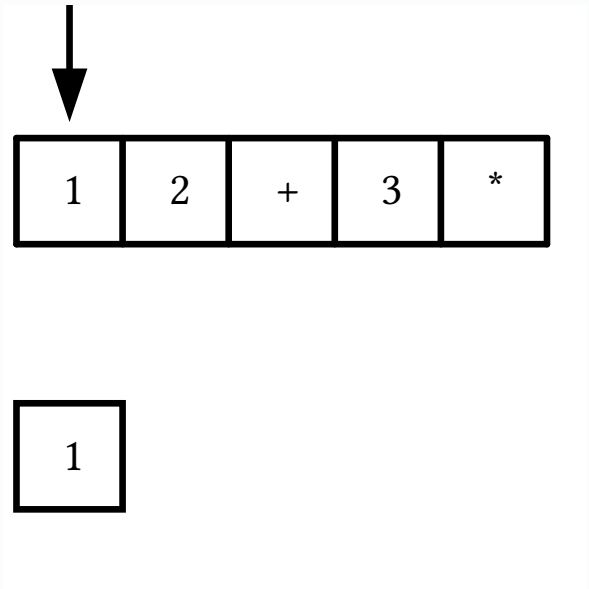
Notation polonaise inversée - exemple

on démarre avec une pile vide



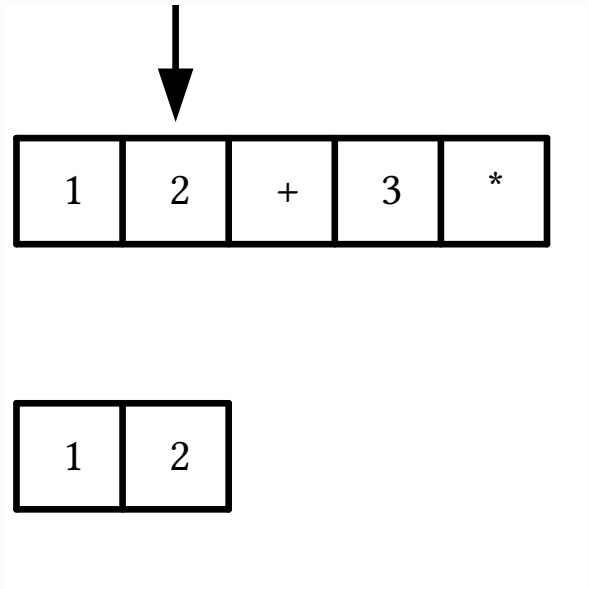
Notation polonaise inversée - exemple

1 est un nombre donc on l'empile



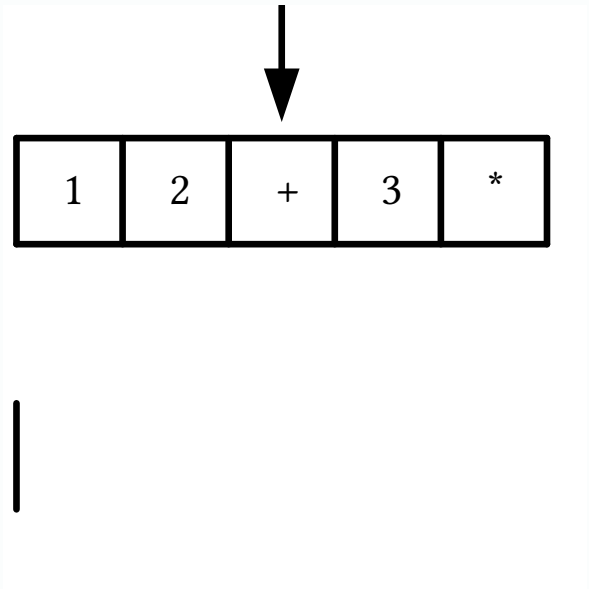
Notation polonaise inversée - exemple

2 est un nombre donc on l'empile



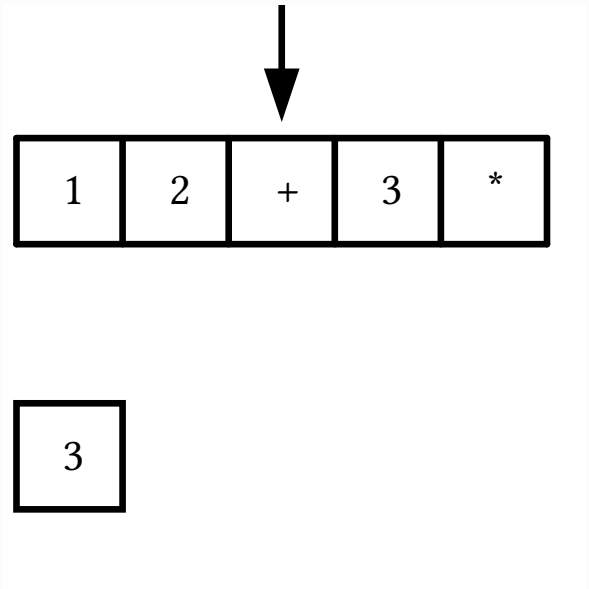
Notation polonaise inversée - exemple

+ est une opération donc on dépile 2 et 1



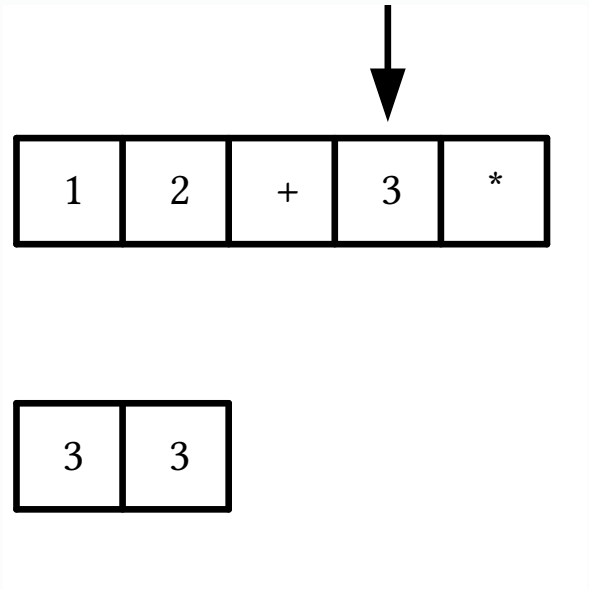
Notation polonaise inversée - exemple

on enpile le resultat de 2 + 1 (3)



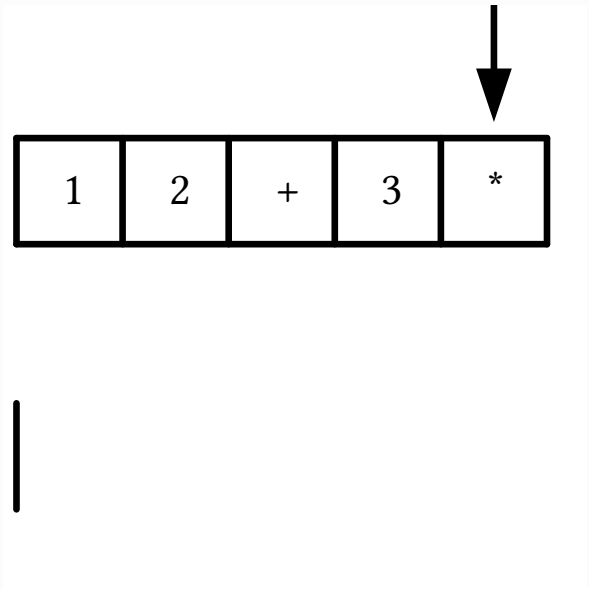
Notation polonaise inversée - exemple

3 est un nombre donc on l'empile



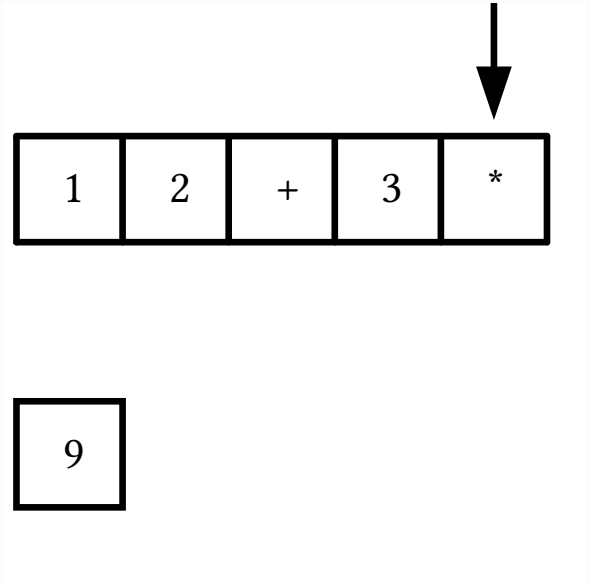
Notation polonaise inversée - exemple

* est une opération donc on dépile 3 et 3



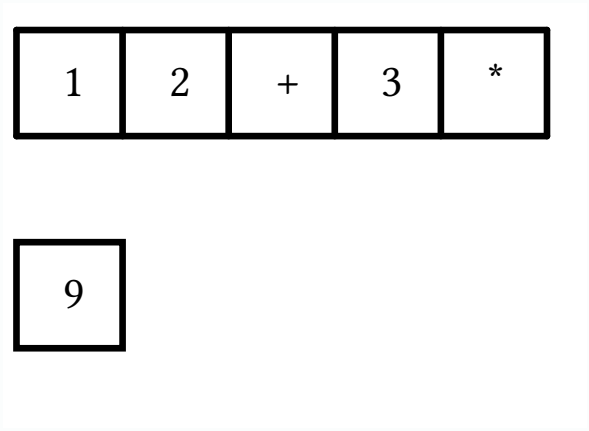
Notation polonaise inversée - exemple

on enpile le resultat de 3 * 3 (9)



Notation polonaise inversée - exemple

il reste le résultat final dans la pile



Machines à pile

- L'usage d'une pile pour executer rapidement des opérations est courant, on peut citer par exemple:
  - La machine virtuelle de CPython, executant les codes **python**.
  - La machine virtuelle java, executant les codes **java/kotlin/scala/groovy/clojure**/etc...
  - Le système d'execution virtuel, executant les codes **.NET** et **.net core**.
  - L'assembleur web, ou WebAssembly.
  - La machine virtuelle **ethereum**, executant les smart contracts.
  - La machine virtuelle de ruby, executant les codes **ruby**.
  - Ainsi que de nombreux autres.

Machines à pile - Exemple de CPython

- Lorsque l'on execute un code python, il est d'abord transformé dans un **bytecode**, afin d'être executé par une machine virtuelle.
- Cette machine utilise une pile afin d'executer rapidement les instructions:

```
def square(a, b, c):
  return (a + b * a) + c * b
```

Machines à pile - Exemple de CPython

- Lorsque l'on execute un code python, il est d'abord transformé dans un **bytecode**, afin d'être executé par une machine virtuelle.
- Cette machine utilise une pile afin d'executer rapidement les instructions:
- Par exemple, la fonction ci-dessous produira le bytecode suivant:

```python
def square(a, b, c):
  return (a + b * a) + c * b
```

```
LOAD_FAST_BORROW_LOAD_FAST_BORROW 1 (a, b)
LOAD_FAST_BORROW          0 (a)
BINARY_OP                 5 (*)
BINARY_OP                 0 (+)
LOAD_FAST_BORROW_LOAD_FAST_BORROW 33 (c, b)
BINARY_OP                 5 (*)
BINARY_OP                 0 (+)
RETURN_VALUE
```

Machines à pile - Exemple de Java

```
class Square {
  static int square(int a, int b, int c) {
      return (a + b * a) + c * b;
  }
}
```

```
static int square(int, int, int);
 0: iload_0 // a
 1: iload_1 // b
 2: iload_0 // a
 3: imul    // *
 4: iadd    // +
 5: iload_2 // c
 6: iload_1 // b
 7: imul    // *
 8: iadd    // +
 9: ireturn
```

Notation polonaise inversée - Traduction

- On peut rapidement traduire une expression standard en NPI en utilisant seulement une pile et en parcourant une fois seulement l'opération donnée:

Notation polonaise inversée - Traduction

- On peut rapidement traduire une expression standard en NPI en utilisant seulement une pile et en parcourant une fois seulement l'opération donnée:
- Lorsqu'on trouve un nombre, on l'ajoute au résultat.

Notation polonaise inversée - Traduction

- On peut rapidement traduire une expression standard en NPI en utilisant seulement une pile et en parcourant une fois seulement l'opération donnée:
- Lorsqu'on trouve un nombre, on l'ajoute au résultat.
- Lorsqu'on trouve une parenthèse ouvrante, on l'empile.

Notation polonaise inversée - Traduction

- On peut rapidement traduire une expression standard en NPI en utilisant seulement une pile et en parcourant une fois seulement l'opération donnée:
- Lorsqu'on trouve un nombre, on l'ajoute au résultat.
- Lorsqu'on trouve une parenthèse ouvrante, on l'empile.
- Lorsqu'on trouve un opérateur O et que le sommet de la pile n'est pas un opérateur, on empile O.

Notation polonaise inversée - Traduction

- On peut rapidement traduire une expression standard en NPI en utilisant seulement une pile et en parcourant une fois seulement l'opération donnée:
- Lorsqu'on trouve un nombre, on l'ajoute au résultat.
- Lorsqu'on trouve une parenthèse ouvrante, on l'empile.
- Lorsqu'on trouve un opérateur O et que le sommet de la pile n'est pas un opérateur, on empile O.
- Lorsqu'on trouve un opérateur O et que le sommet de la pile est un opérateur :
  - Tant que le sommet de la pile est un opérateur de plus grande priorité que O : On dépile pour ajouter au résultat.
  - Ensuite, on enpile O.

Notation polonaise inversée - Traduction

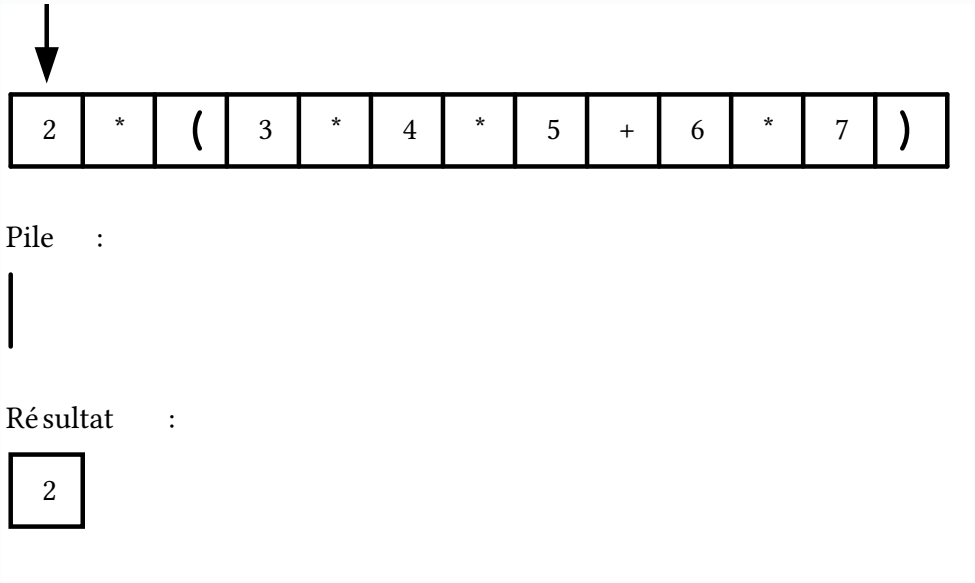
- On peut rapidement traduire une expression standard en NPI en utilisant seulement une pile et en parcourant une fois seulement l'opération donnée:
- Lorsqu'on trouve un nombre, on l'ajoute au résultat.
- Lorsqu'on trouve une parenthèse ouvrante, on l'empile.
- Lorsqu'on trouve un opérateur O et que le sommet de la pile n'est pas un opérateur, on empile O.
- Lorsqu'on trouve un opérateur O et que le sommet de la pile est un opérateur :
  - Tant que le sommet de la pile est un opérateur de plus grande priorité que O : On dépile pour ajouter au résultat.
  - Ensuite, on enpile O.
- Lorsqu'on trouve une parenthèse fermante, on dépile pour ajouter au résultat jusqu'à trouver une parenthèse ouvrante.

Notation polonaise inversée - Traduction

- On peut rapidement traduire une expression standard en NPI en utilisant seulement une pile et en parcourant une fois seulement l'opération donnée:
- Lorsqu'on trouve un nombre, on l'ajoute au résultat.
- Lorsqu'on trouve une parenthèse ouvrante, on l'empile.
- Lorsqu'on trouve un opérateur O et que le sommet de la pile n'est pas un opérateur, on empile O.
- Lorsqu'on trouve un opérateur O et que le sommet de la pile est un opérateur :
  - Tant que le sommet de la pile est un opérateur de plus grande priorité que O : On dépile pour ajouter au résultat.
  - Ensuite, on enpile O.
- Lorsqu'on trouve une parenthèse fermante, on dépile pour ajouter au résultat jusqu'à trouver une parenthèse ouvrante.
- Lorsqu'on a fini de parcourir l'expression standard, on dépile tout ce qui reste dans la pile pour l'ajouter dans la liste.

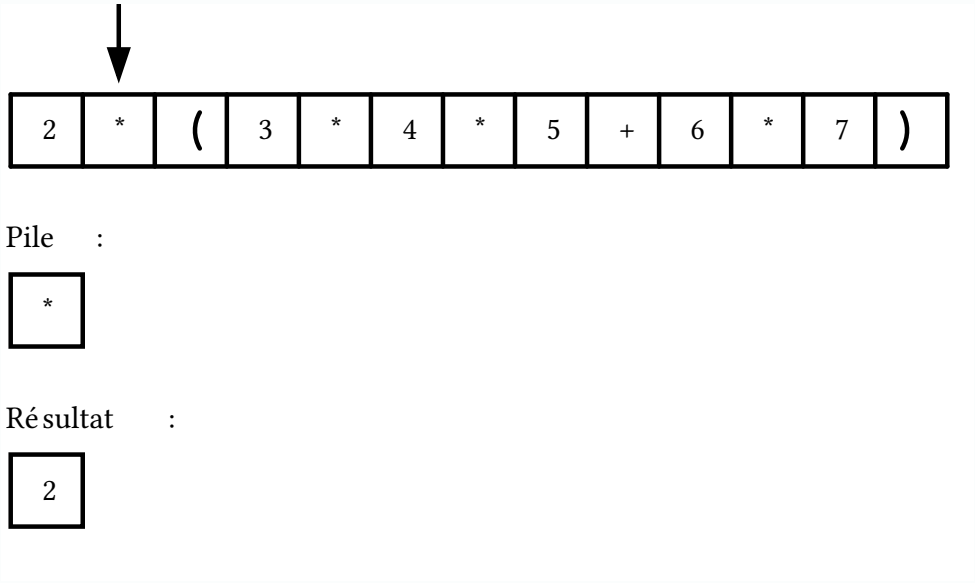
Notation polonaise inversée - Traduction

- On ajoute au résultat car 2 est un nombre



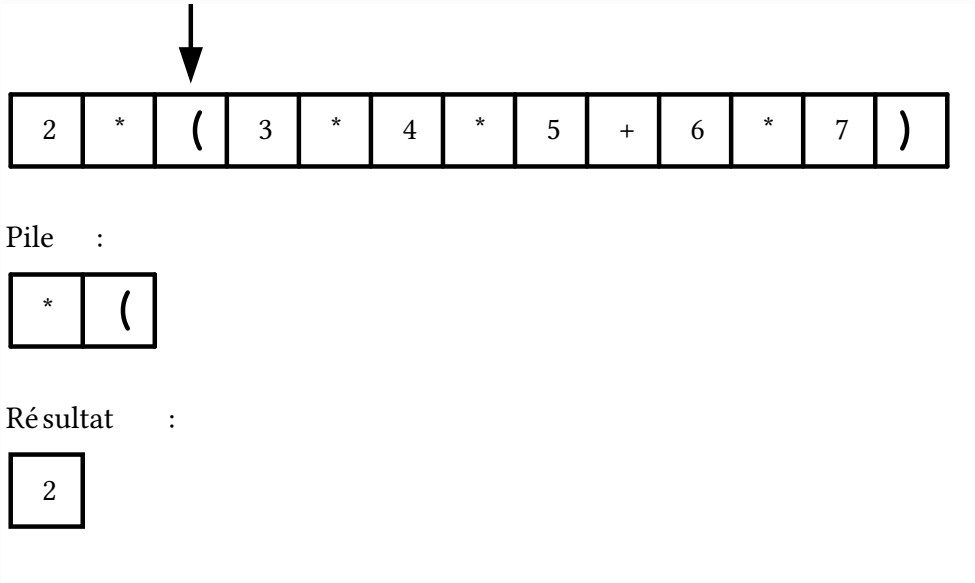
Notation polonaise inversée - Traduction

- * est un opérateur mais le sommet de la pile n'est pas un opérateur donc on l'empile



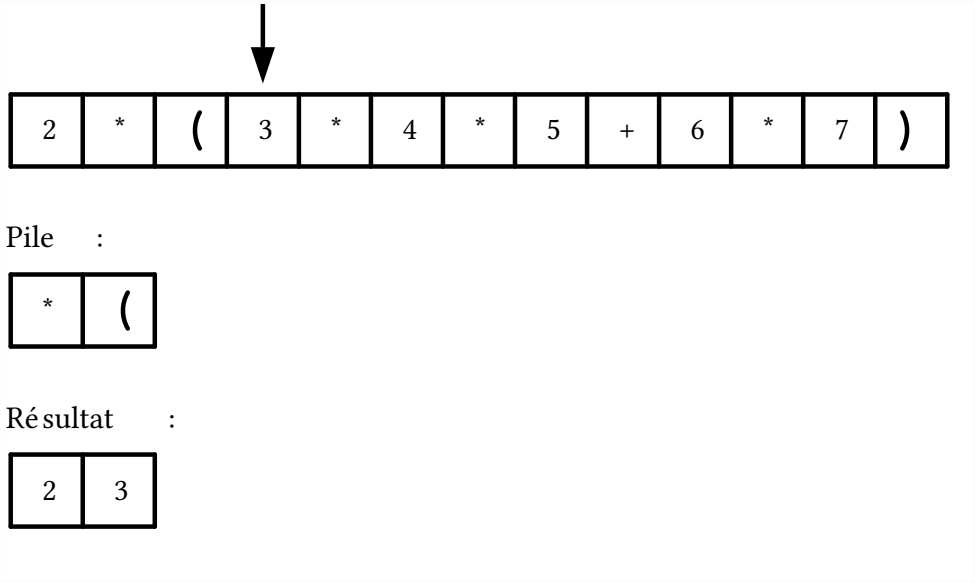
Notation polonaise inversée - Traduction

- On enpile directement une parenthèse ouvrante



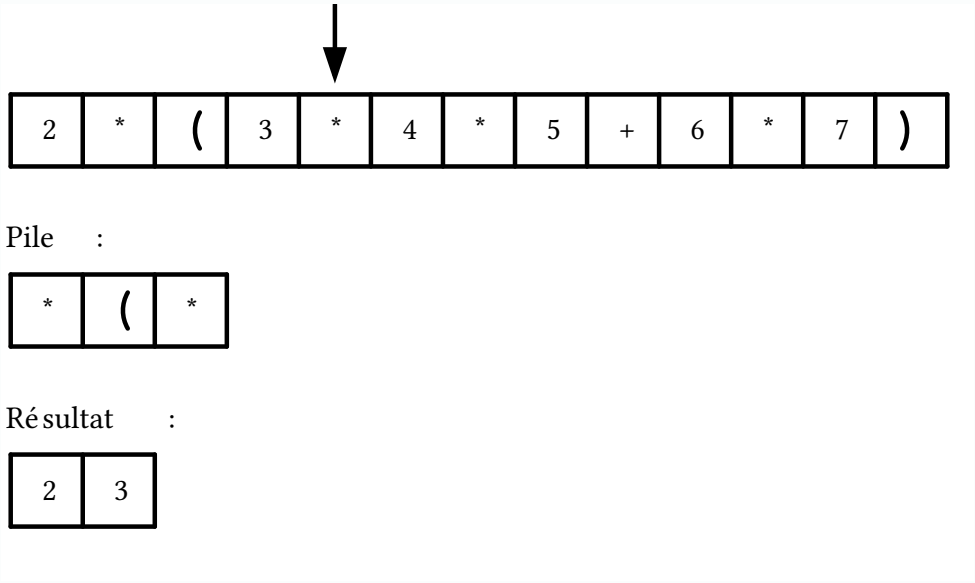
Notation polonaise inversée - Traduction

- On ajoute au résultat car 3 est un nombre



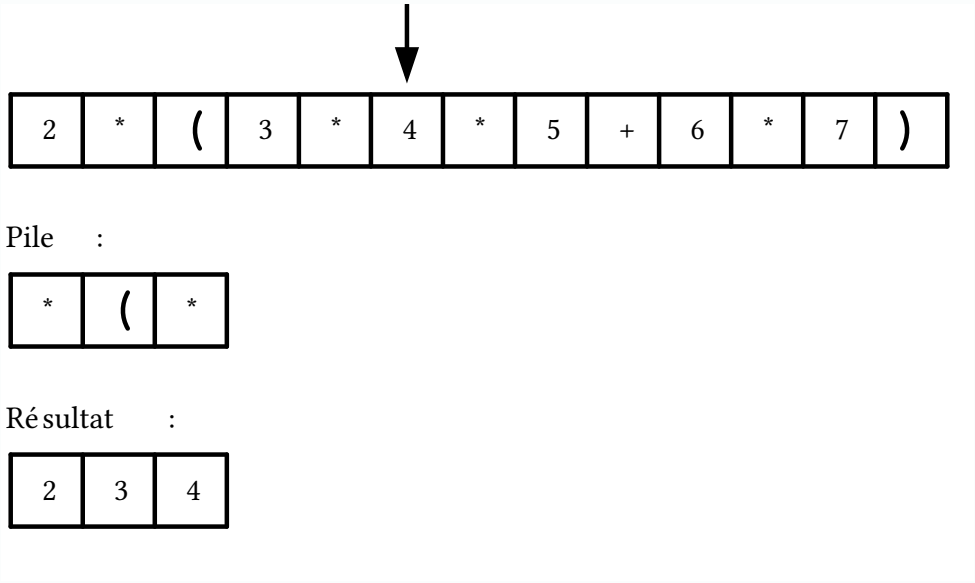
Notation polonaise inversée - Traduction

- * est un opérateur mais le sommet de la pile n'est pas un opérateur donc on l'empile



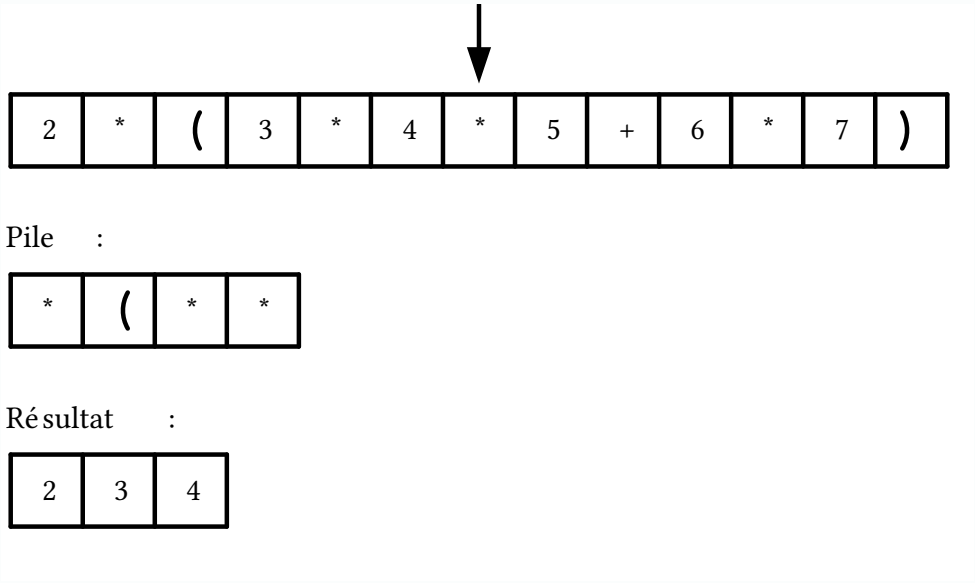
Notation polonaise inversée - Traduction

- On ajoute au résultat car 4 est un nombre



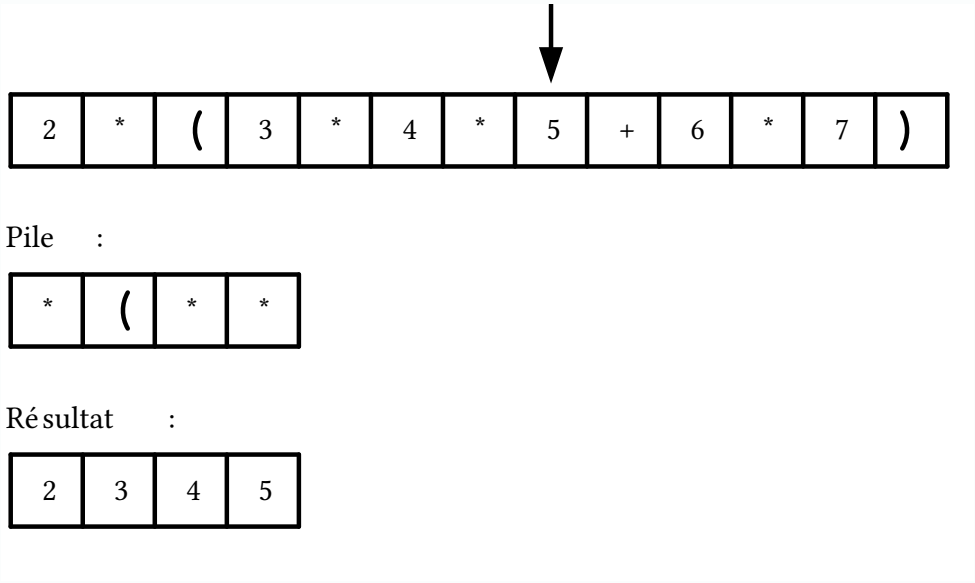
Notation polonaise inversée - Traduction

- Si la pile est vide ou que son sommet est de priorité inferieure ou égale à *, on enpile



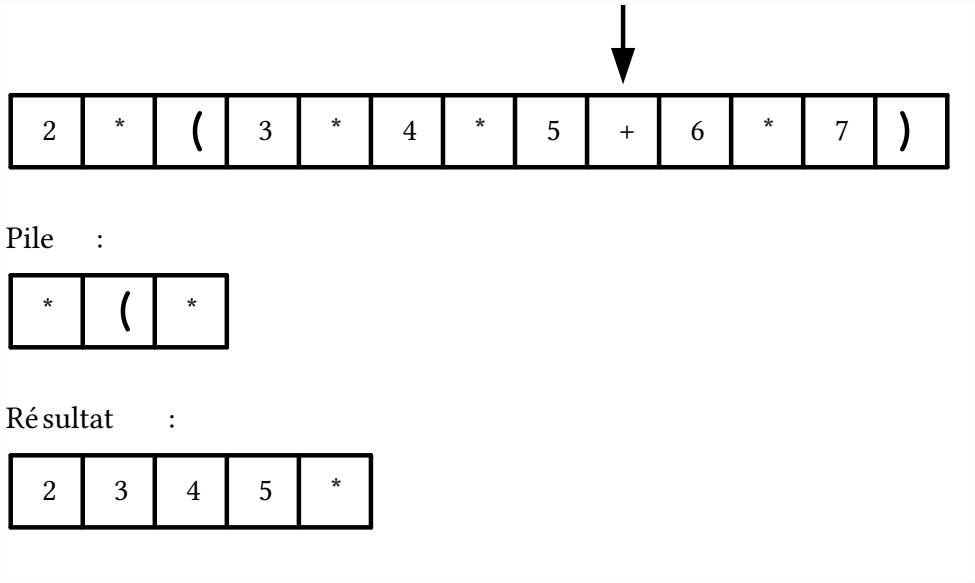
Notation polonaise inversée - Traduction

- On ajoute au résultat car 5 est un nombre



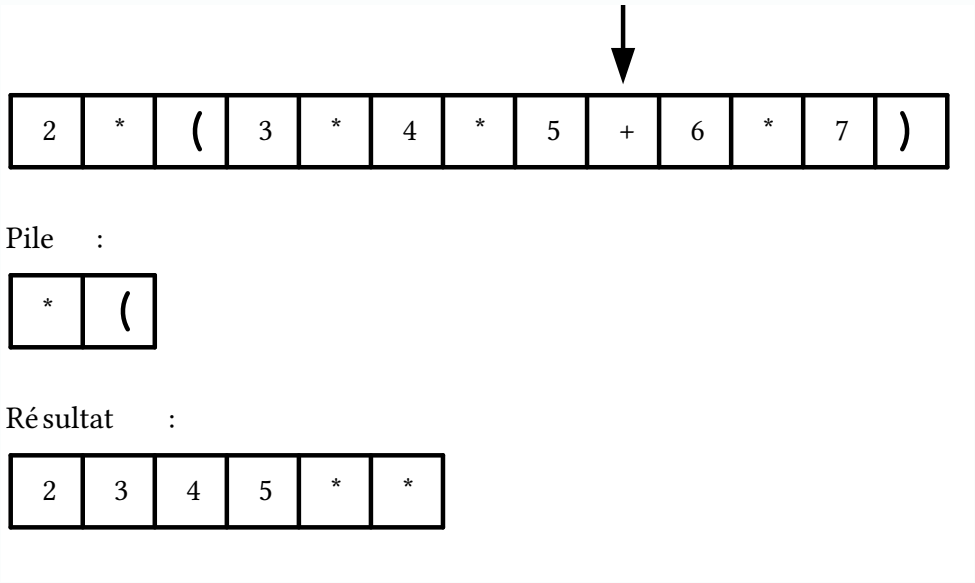
Notation polonaise inversée - Traduction

- Tant que le sommet de la pile est un opérateur de plus grande priorité que +, on le dépile pour l'ajouter au résultat



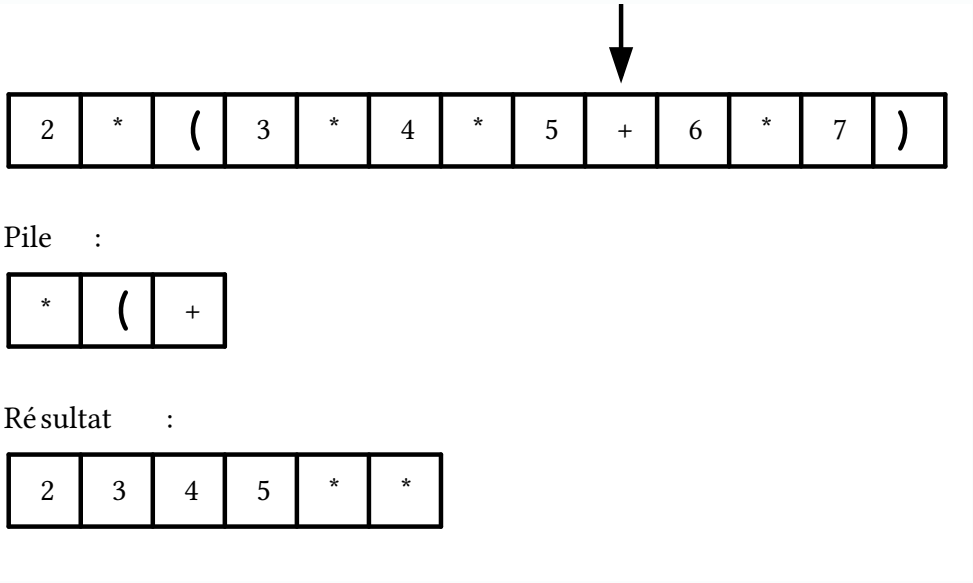
Notation polonaise inversée - Traduction

- Tant que le sommet de la pile est un opérateur de plus grande priorité que +, on le dépile pour l'ajouter au résultat



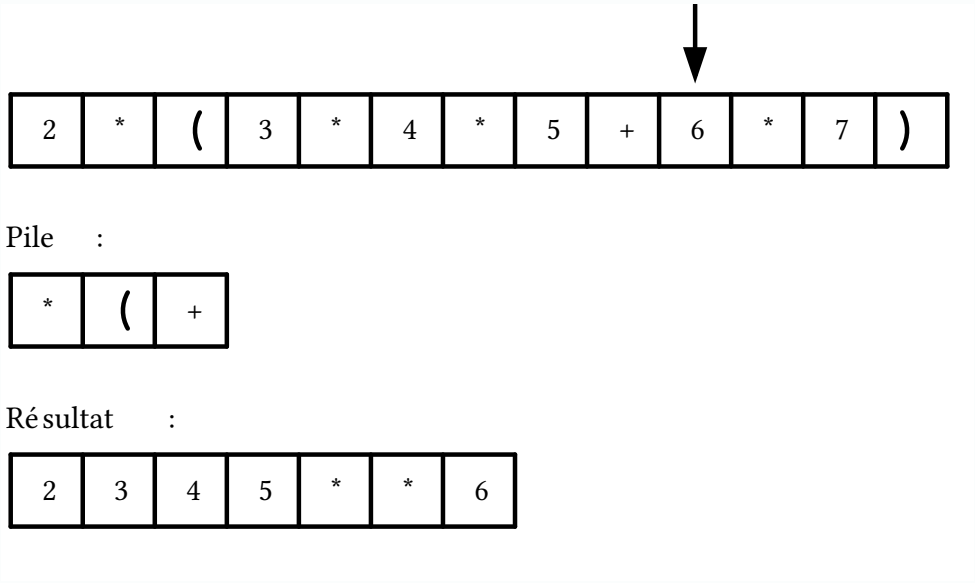
Notation polonaise inversée - Traduction

- Si la pile est vide ou que son sommet est de priorité inferieure ou égale à +, on enpile



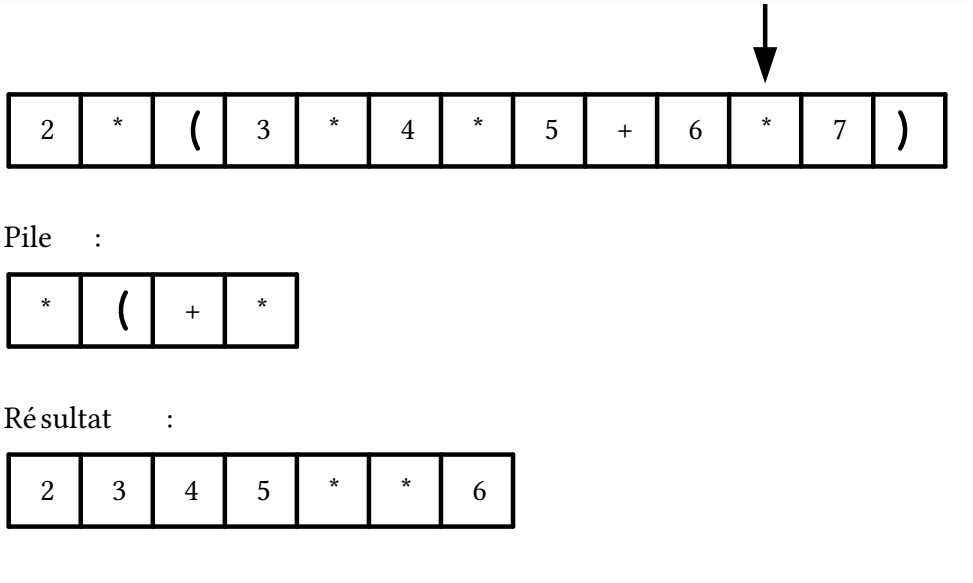
Notation polonaise inversée - Traduction

- On ajoute au résultat car 6 est un nombre



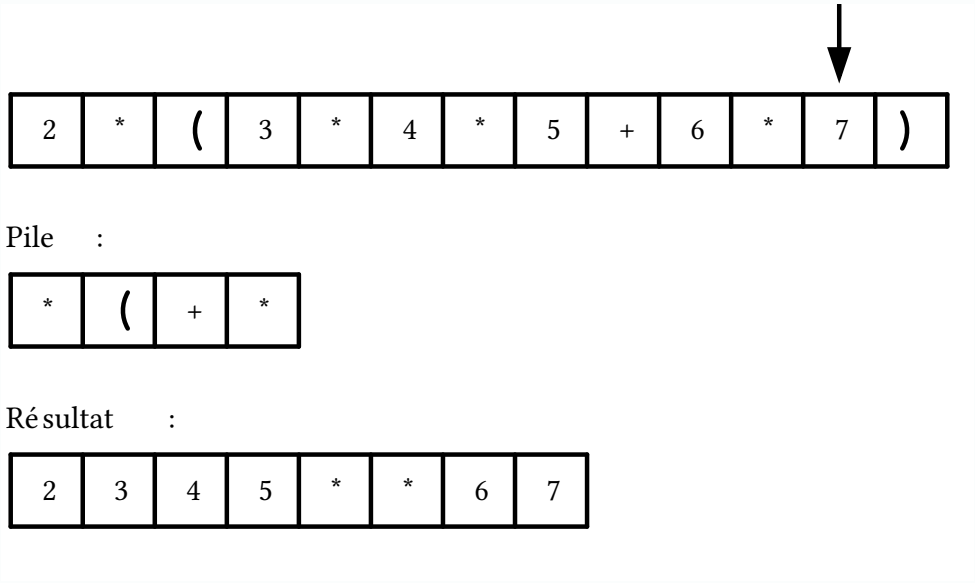
Notation polonaise inversée - Traduction

- Si la pile est vide ou que son sommet est de priorité inferieure ou égale à *, on enpile



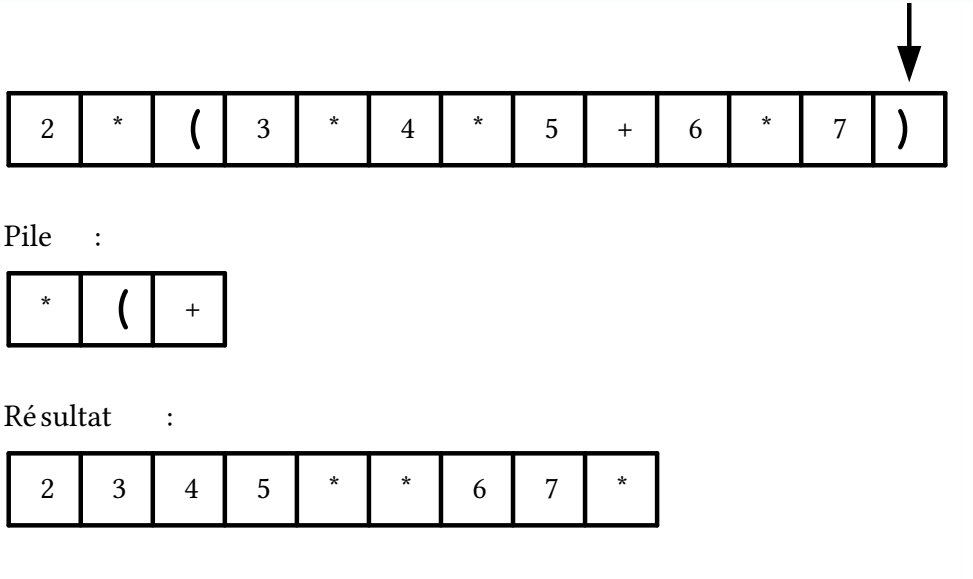
Notation polonaise inversée - Traduction

- On ajoute au résultat car 7 est un nombre



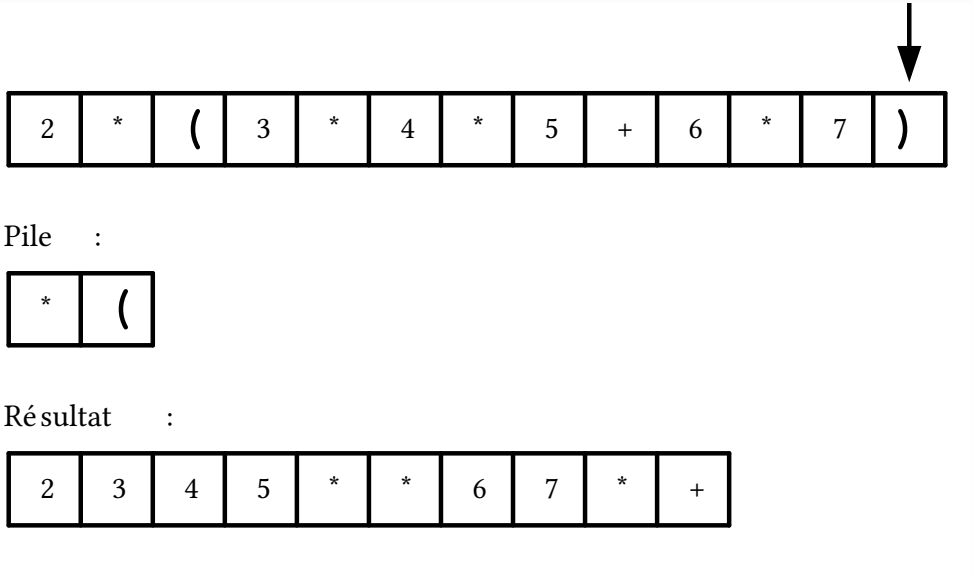
Notation polonaise inversée - Traduction

- Lorsqu'on trouve une parenthèse fermante, on dépile en ajoutant dans le résultat jusqu'a trouver la parenthèse ouvrante



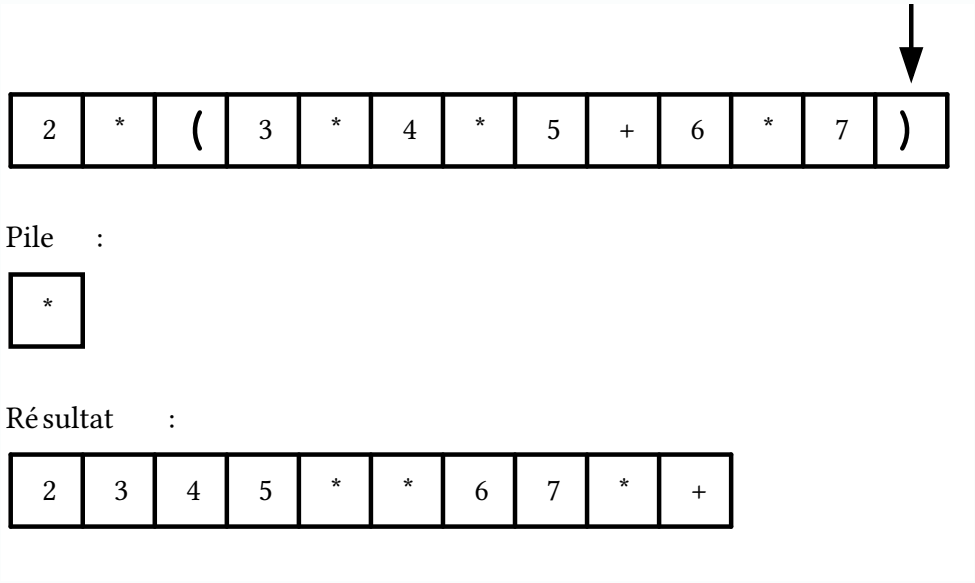
Notation polonaise inversée - Traduction

- Lorsqu'on trouve une parenthèse fermante, on dépile en ajoutant dans le résultat jusqu'a trouver la parenthèse ouvrante



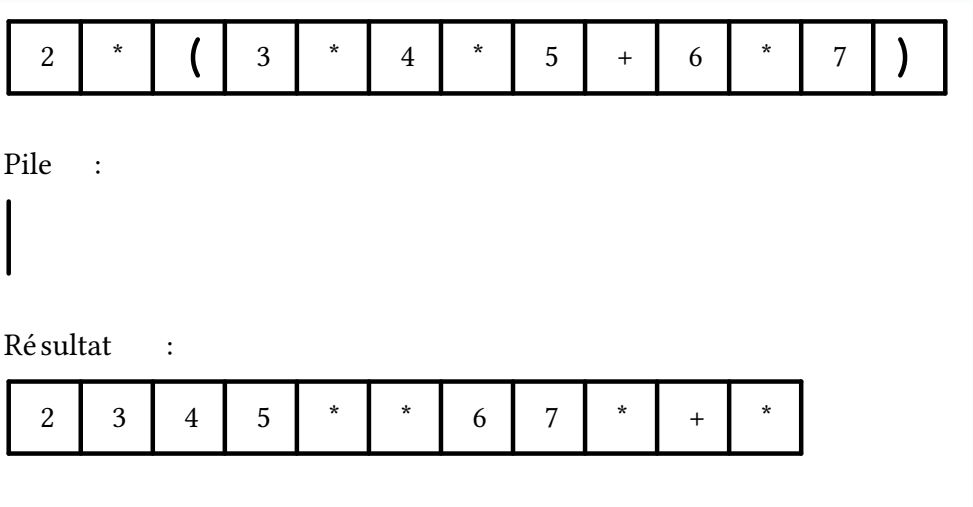
Notation polonaise inversée - Traduction

- Ensuite on enlève la parenthèse



Notation polonaise inversée - Traduction

On dépile dans le résultat tant qu'il reste des éléments dans la pile



Mise en pratique

```
printf("
          _________
        < TP Algo >
          --------
                 \   ^__^
                  \  (oo)\_______
                     (__)\       )\/\
                         ||----w |
                         ||     ||
        ")
```