Concepts Informatiques

2018–2019

Matthieu Picantin

Tests et examens

- CC : résultat des 3 tests (ou plus) effectués en TD
- E0 : partiel (samedi 23 février ou 2 mars, à confirmer)
- E1 : examen mi-mai
- E2 : examen mi-juin

Notes finales

- Note session 1 : 20% CC + 20% E0 + 60% E1
- Note session 2 : max(E2, 20% CC + 80% E2)

Rappel

pas de note $\Longrightarrow$ pas de moyenne $\Longrightarrow$ pas de semestre

`moodlesupd.script.univ-paris-diderot.fr`

```
int res=1,cpt=2,arg=7;
while(cpt<=arg) res*=cpt++;
return res;
```

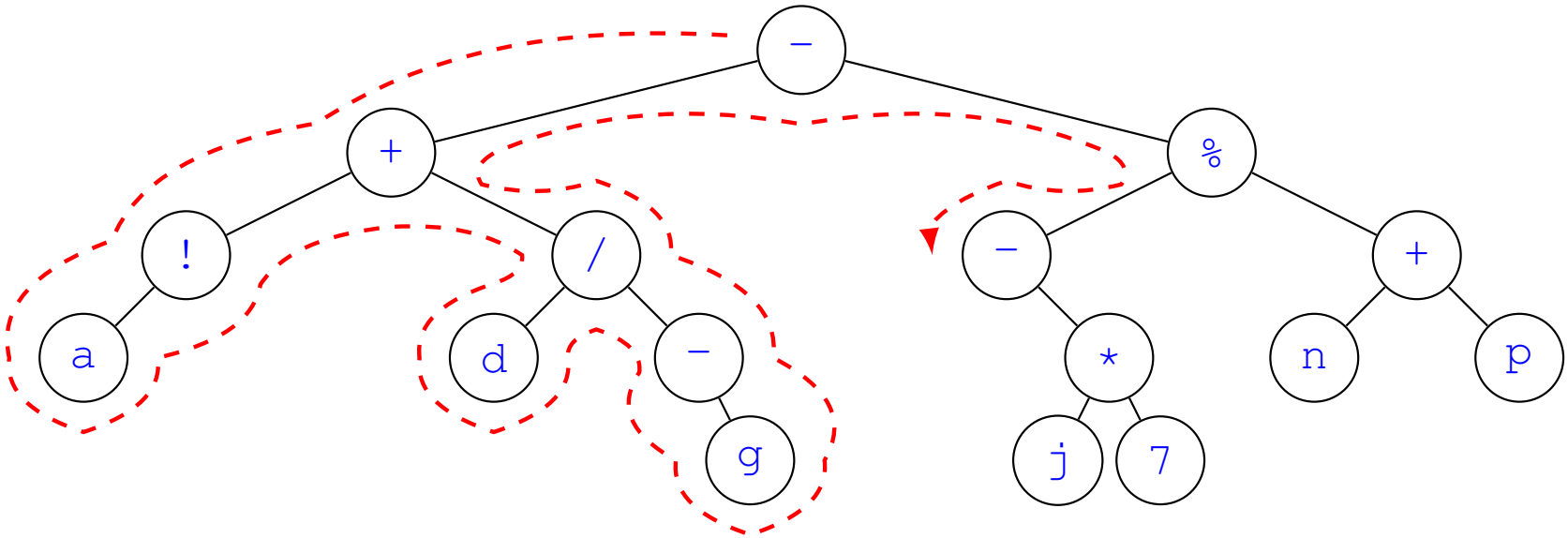
pensée

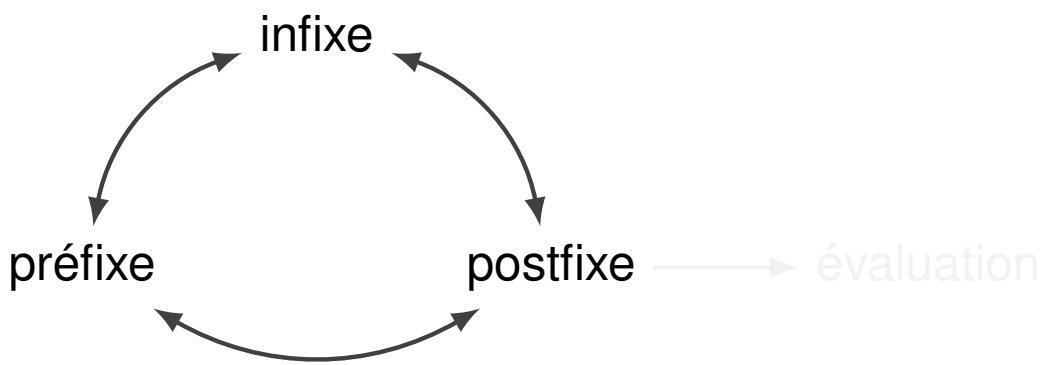
calcul
récursion
fonction
objet
⋮

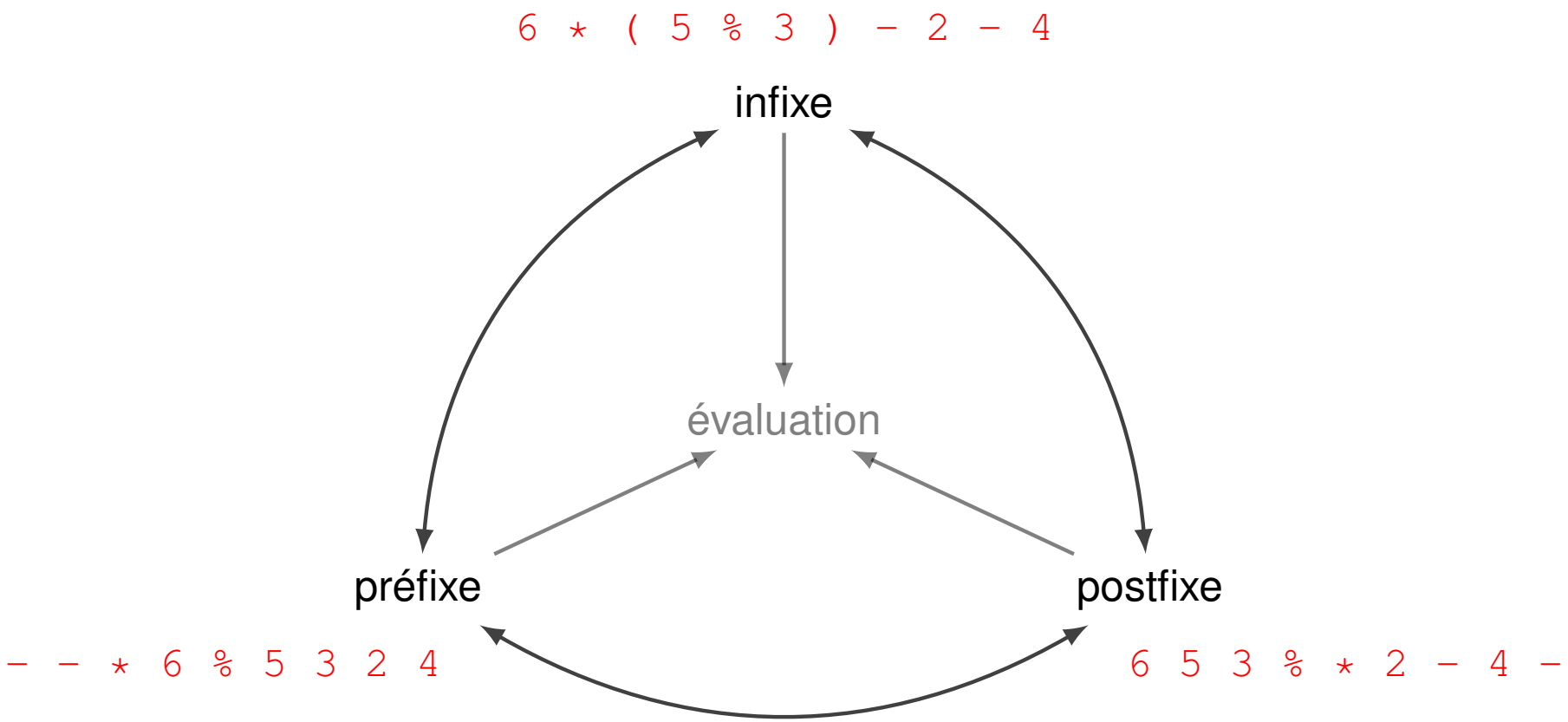
machine

circuit
pile
registre
mémoire
⋮

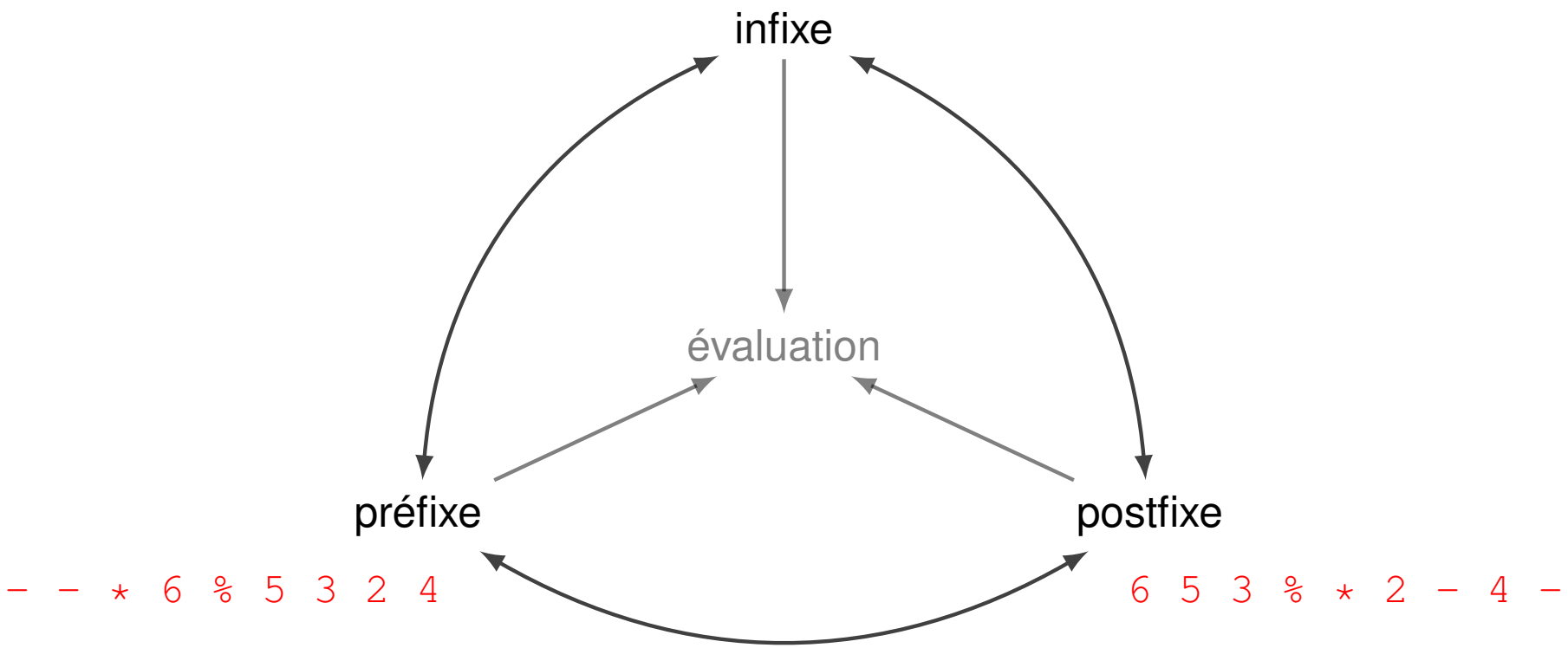
```
10111000 00000001 00000000
00000000 00000000 10111010
00000010 00000000 00000000
00000000 00111001 11011010
01111111 00000110 00001111
10101111 11000010 01000010
11101011 11110110 11000011
```


-
+
%
!
/
-
+
a
d
-
*
n
p
g
j
7


infixe
préfixe
postfixe
évaluation


6 * (5 % 3) - 2 - 4
infixe
évaluation
préfixe
postfixe
- - * 6 % 5 3 2 4
6 5 3 % * 2 - 4 -

6 * (5 % 3) - 2 - 4

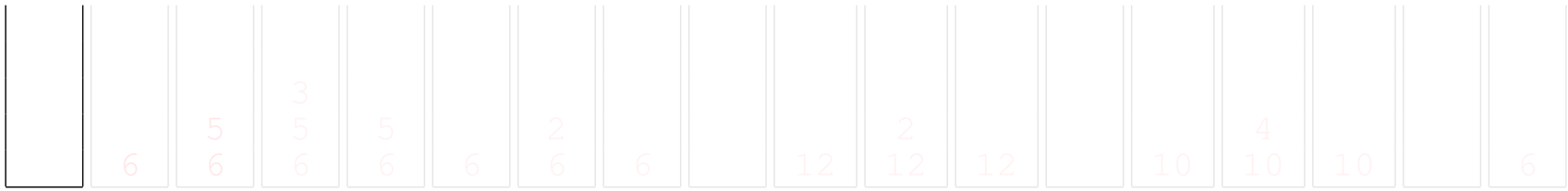


évaluation

postfixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

6 5 3 % * 2 - 4 -

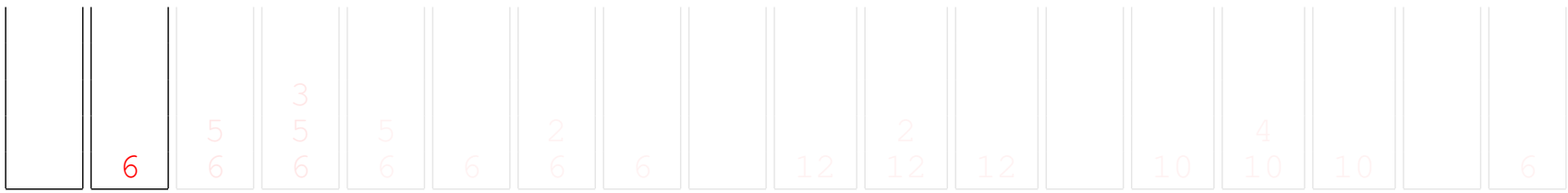


évaluation

postfixe

6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

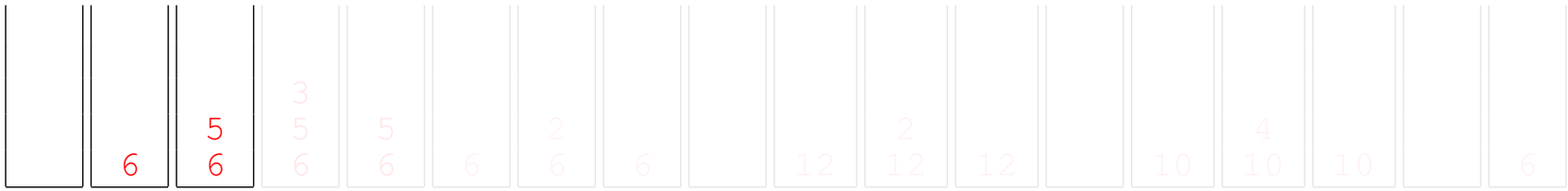


évaluation

postfixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

6 5 3 % * 2 - 4 -

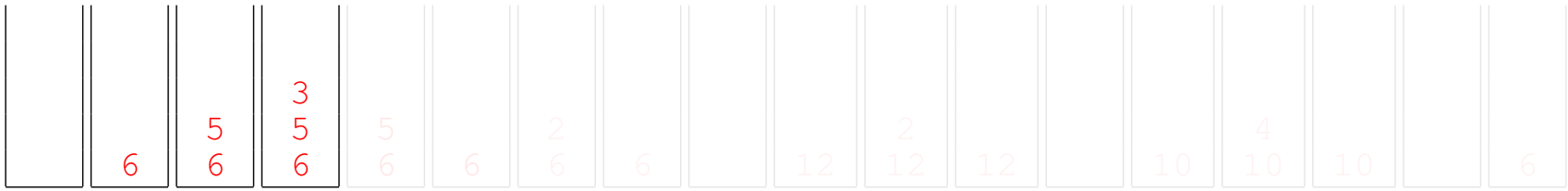


évaluation

postfixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

6 5 3 % * 2 - 4 -

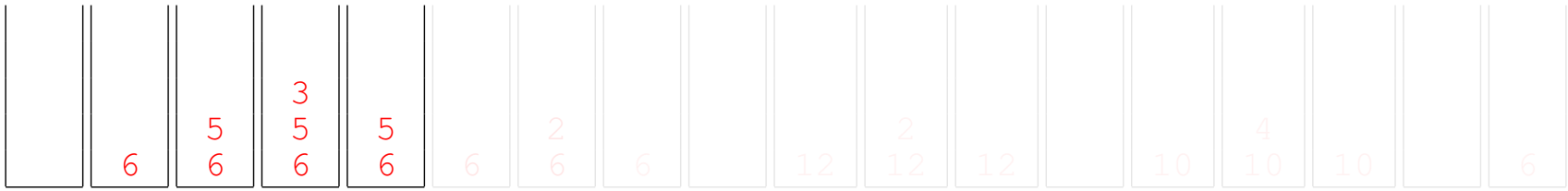


évaluation

postfixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

6 5 3 % * 2 - 4 -

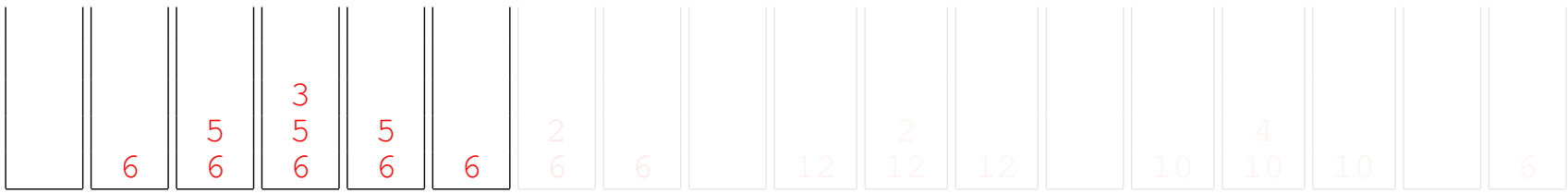


évaluation

postfixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

6 5 3 % * 2 - 4 -

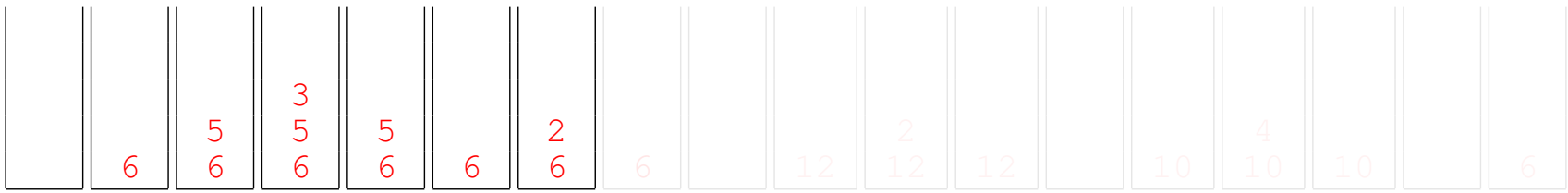


évaluation

postfixe

6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

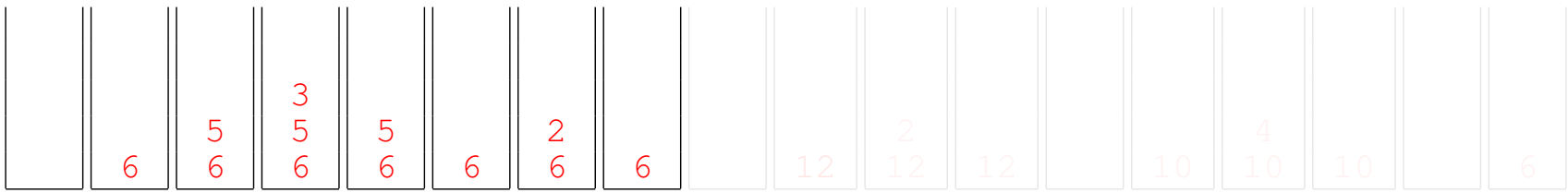


évaluation

postfixe

6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

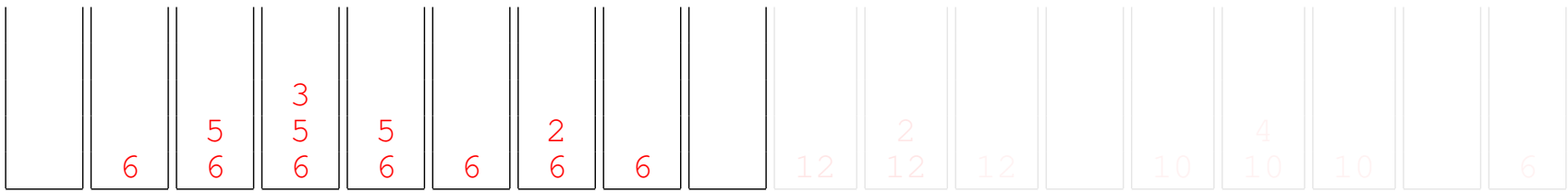


évaluation

postfixe

6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

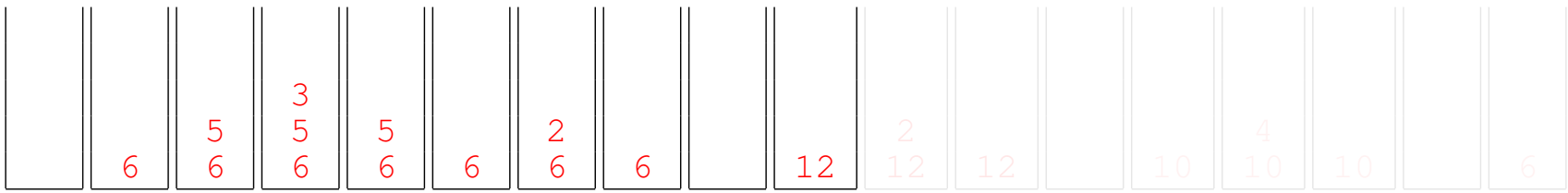


évaluation

postfixe

6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

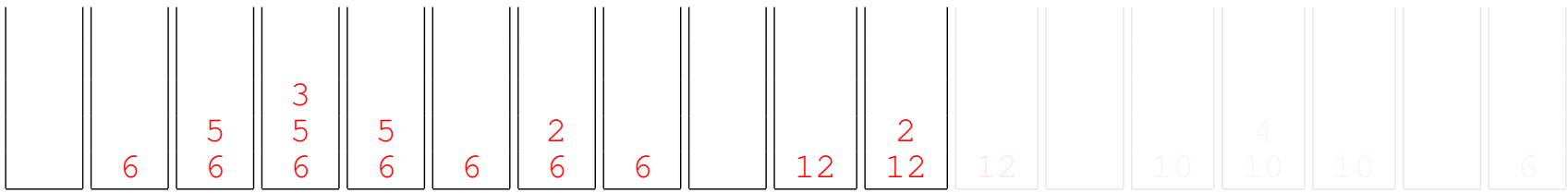


évaluation

postfixe

6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

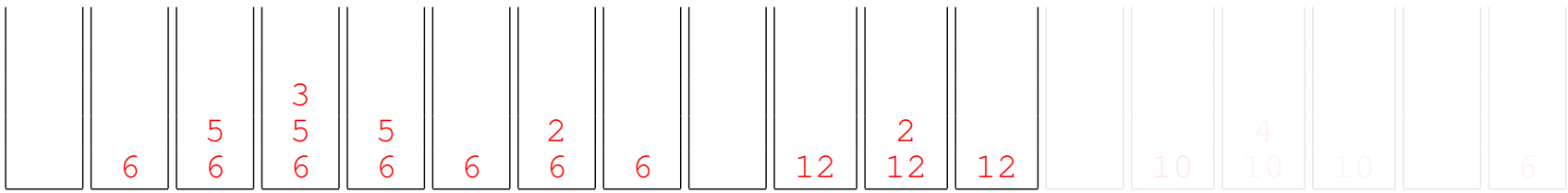


évaluation

postfixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

6 5 3 % * 2 - 4 -

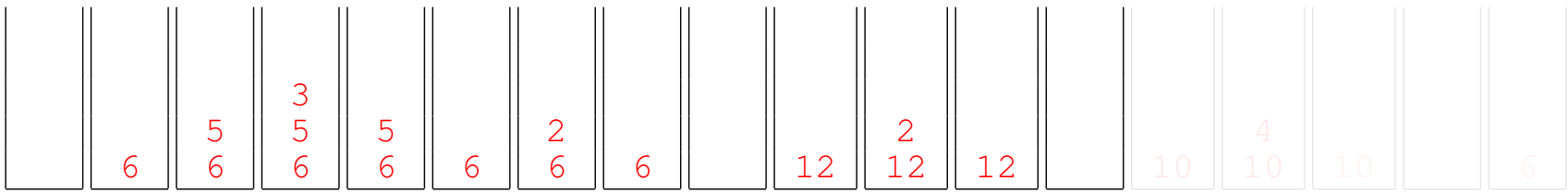


évaluation

postfixe

```
6 5 3 % * 2 - 4 -
          ▲
```

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

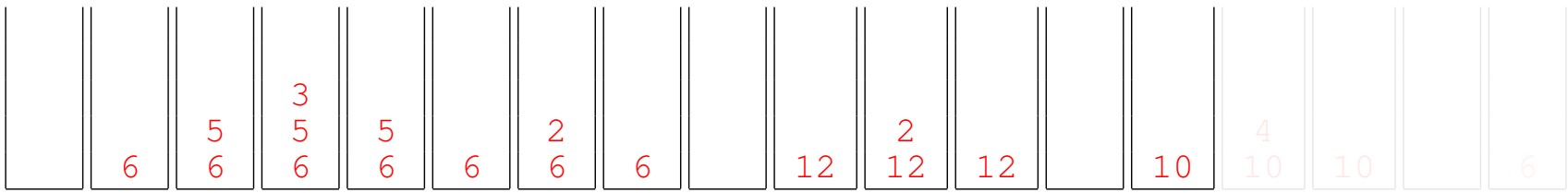


évaluation

postfixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

6 5 3 % * 2 - 4 -

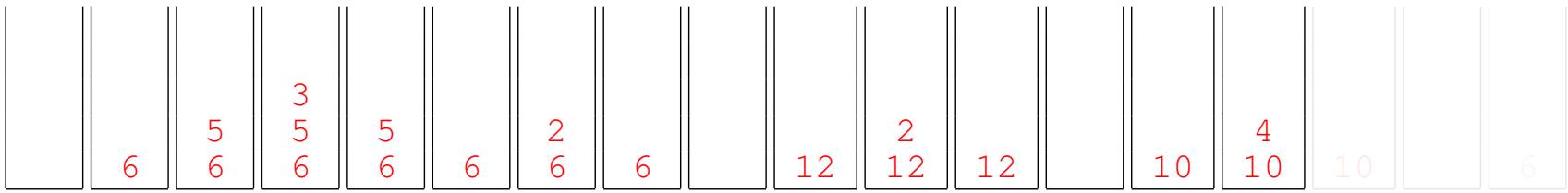


évaluation

postfixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

6 5 3 % * 2 - 4 -
▲

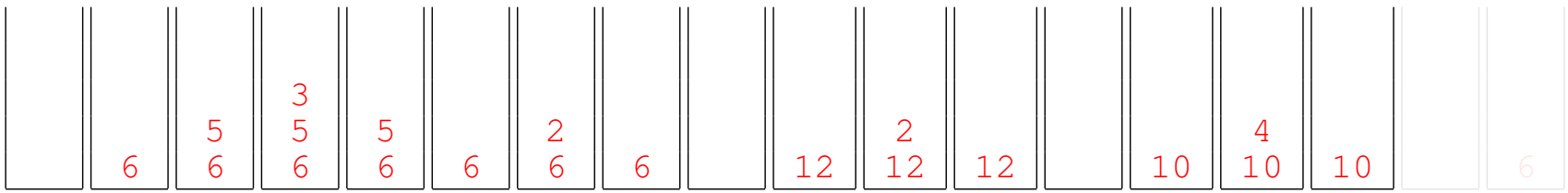


évaluation

postfixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

6 5 3 % * 2 - 4 -
▲

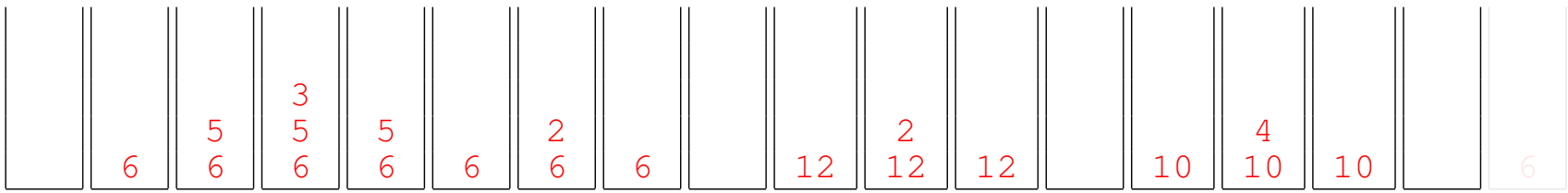


évaluation

postfixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

6 5 3 % * 2 - 4 -
▲

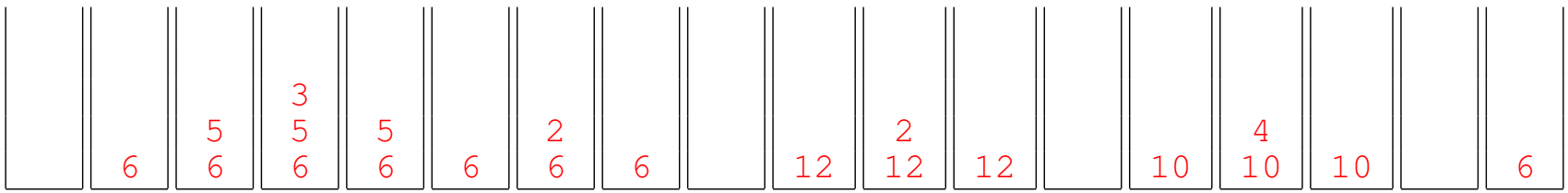


évaluation

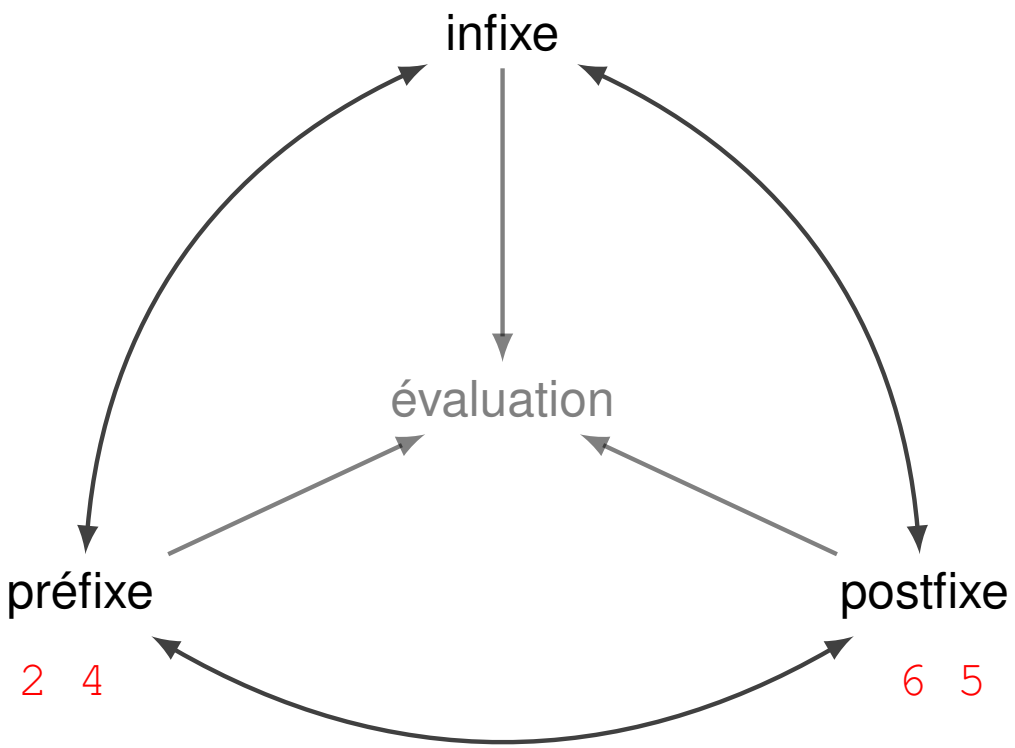
postfixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) symboles
 *   * on exécute l'opération
 *   * on empile son résultat
 */
```

6 5 3 % * 2 - 4 -
▲



6 * (5 % 3) - 2 - 4

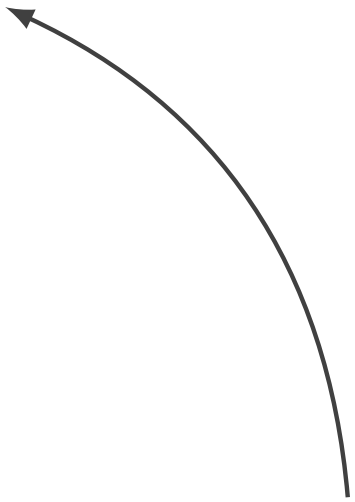


- - * 6 % 5 3 2 4

6 5 3 % * 2 - 4 -

```
6 * ( 5 % 3 ) - 2 - 4
```

infixe



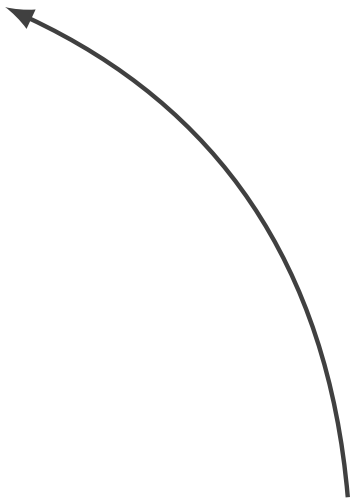
postfixe

```
6 5 3 % * 2 - 4 -
```

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```

```
6 * ( 5 % 3 ) - 2 - 4
```

infixe



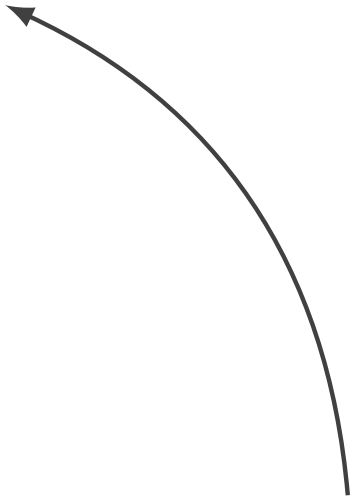
postfixe

```
6 5 3 % * 2 - 4 -
```

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```

6 * (5 % 3) - 2 - 4

infixe



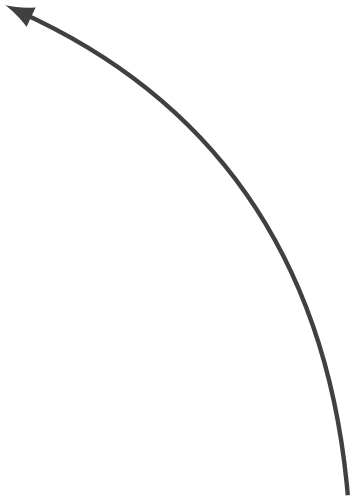
postfixe

6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```

6 * (5 % 3) - 2 - 4

infixe



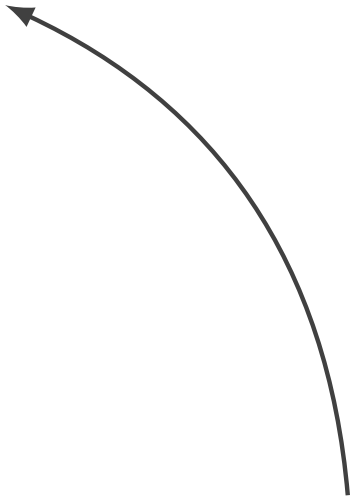
postfixe

6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```

6 * (5 % 3) - 2 - 4

infixe



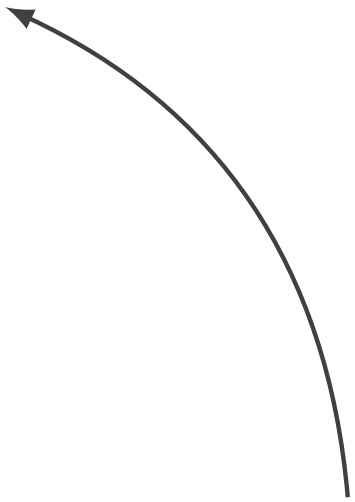
postfixe

6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```

```
6 * ( 5 % 3 ) - 2 - 4
```

infixe



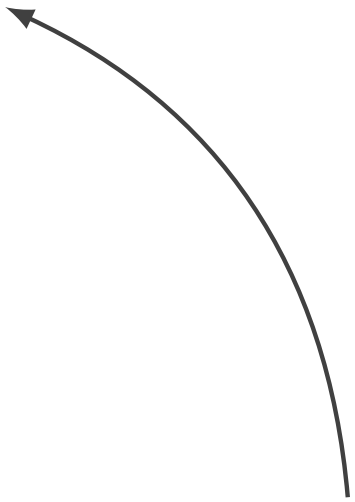
postfixe

```
6 5 3 % * 2 - 4 -
```

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```

```
6 * ( 5 % 3 ) - 2 - 4
```

infixe



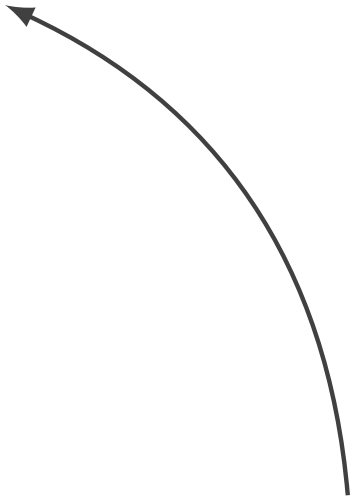
postfixe

```
6 5 3 % * 2 - 4 -
```

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```

```
6 * ( 5 % 3 ) - 2 - 4
```

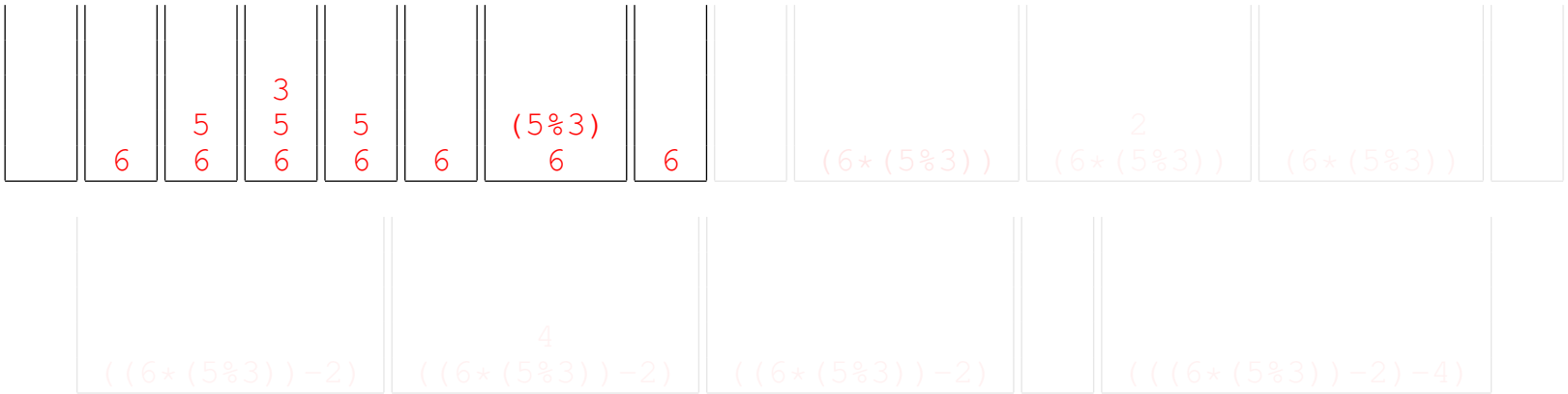
infixe



postfixe

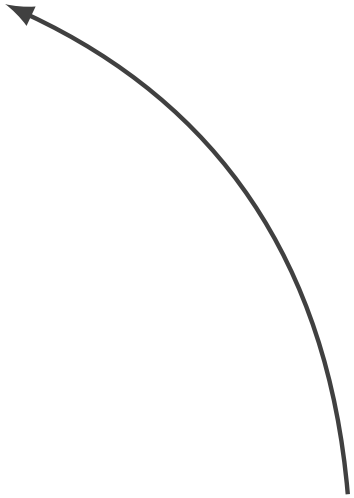
```
6 5 3 % * 2 - 4 -
```

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```



6 * (5 % 3) - 2 - 4

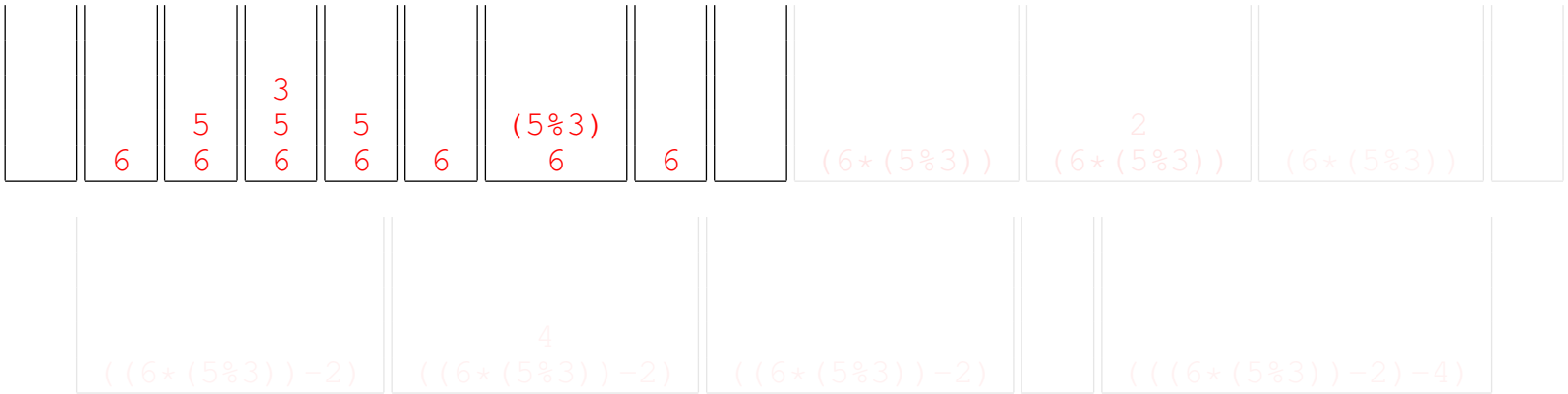
infixe



postfixe

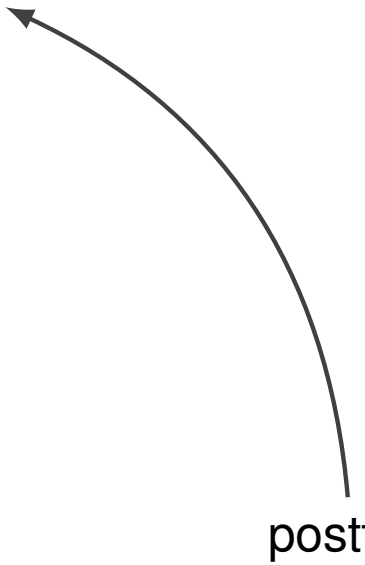
6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```



6 * (5 % 3) - 2 - 4

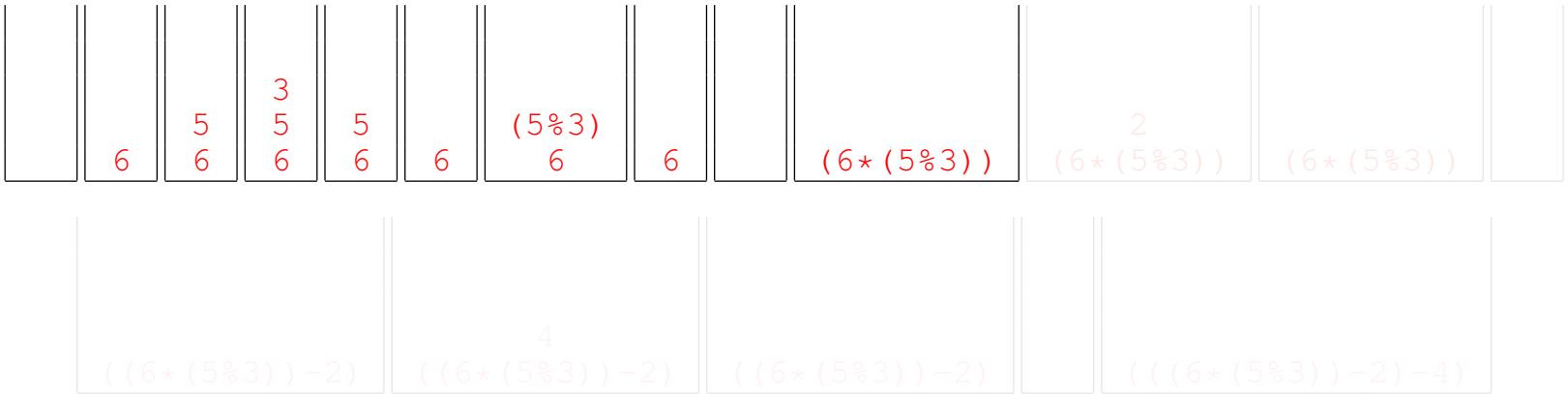
infixe



postfixe

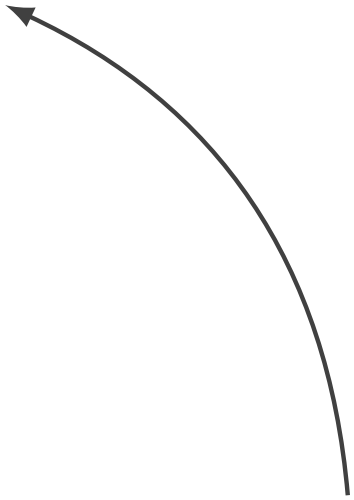
6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```



6 * (5 % 3) - 2 - 4

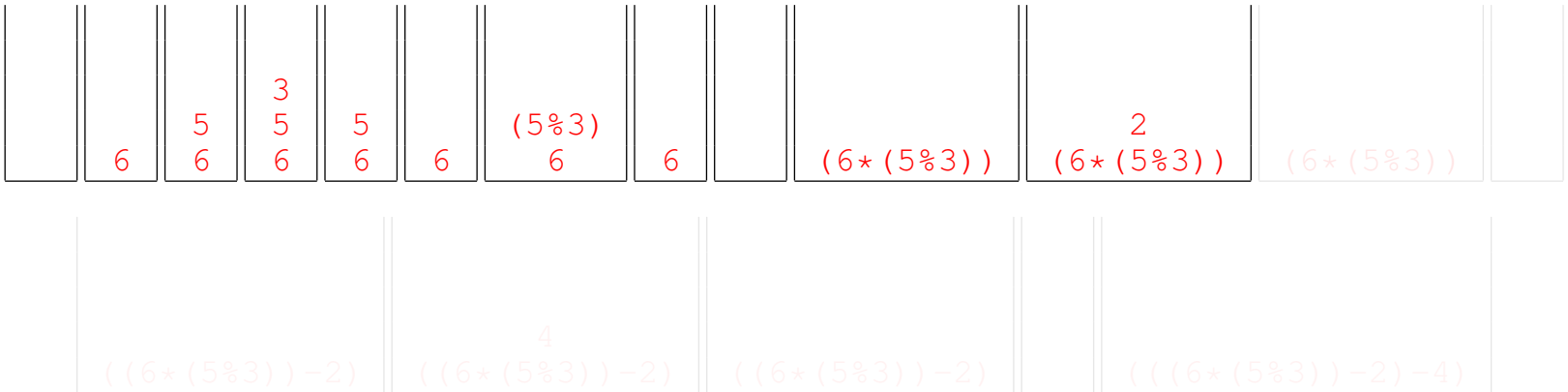
infixe



postfixe

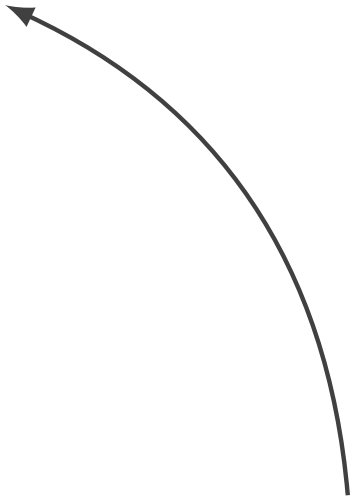
6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```



6 * (5 % 3) - 2 - 4

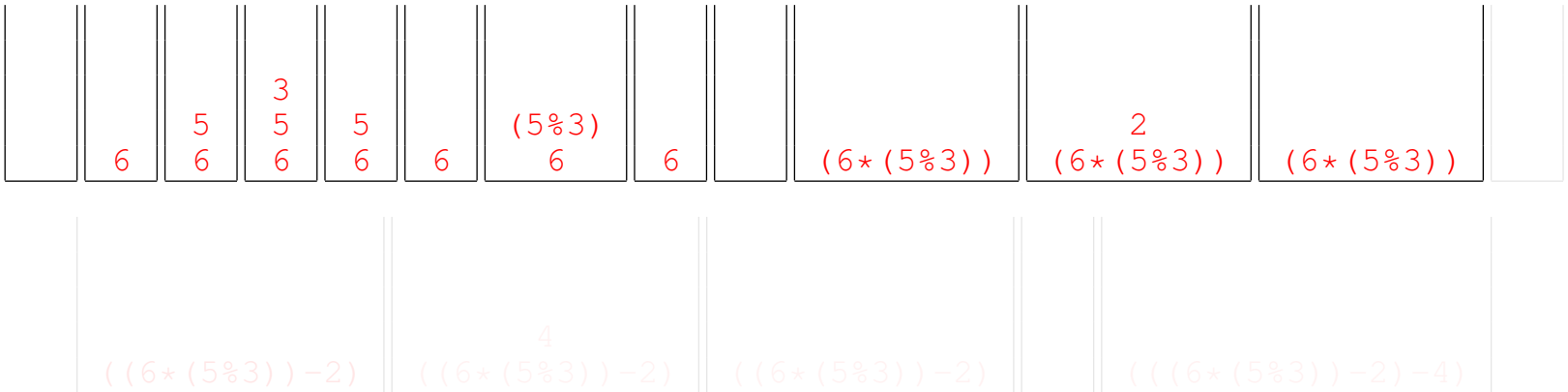
infixe



postfixe

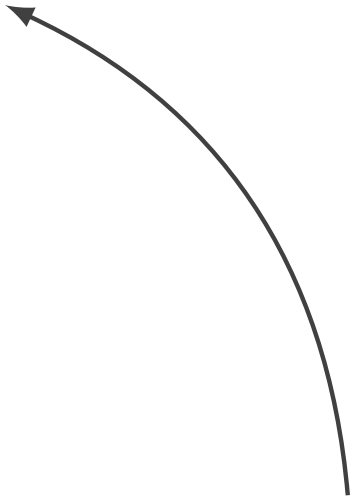
6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```



6 * (5 % 3) - 2 - 4

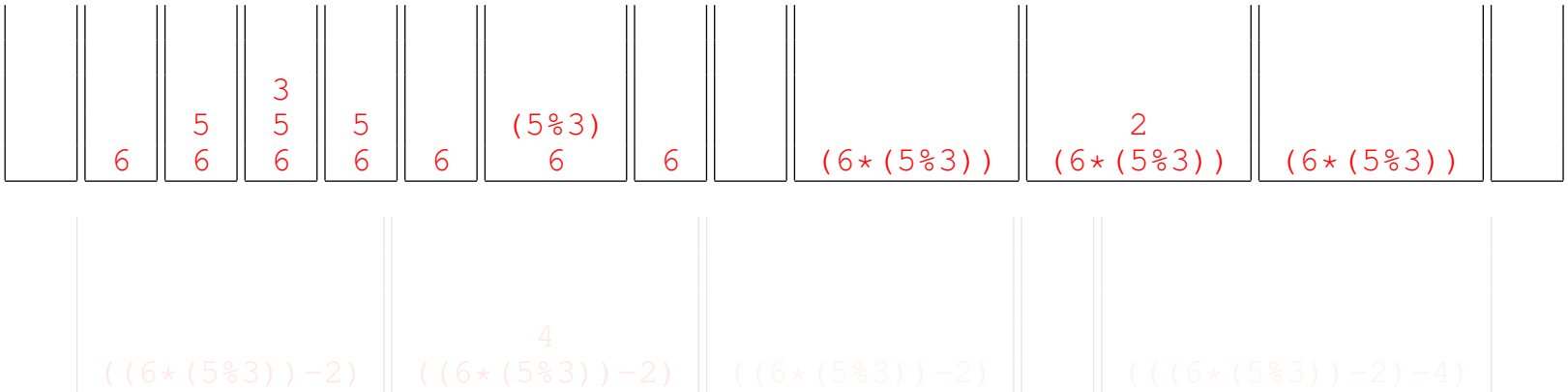
infixe



postfixe

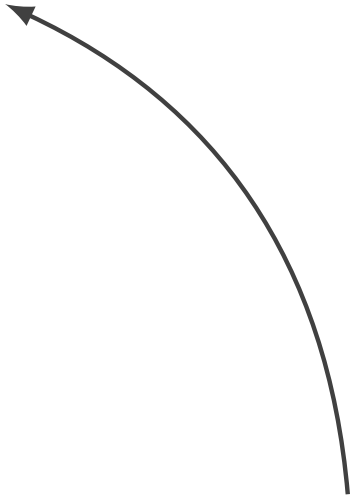
```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```

6 5 3 % * 2 - 4 -



6 * (5 % 3) - 2 - 4

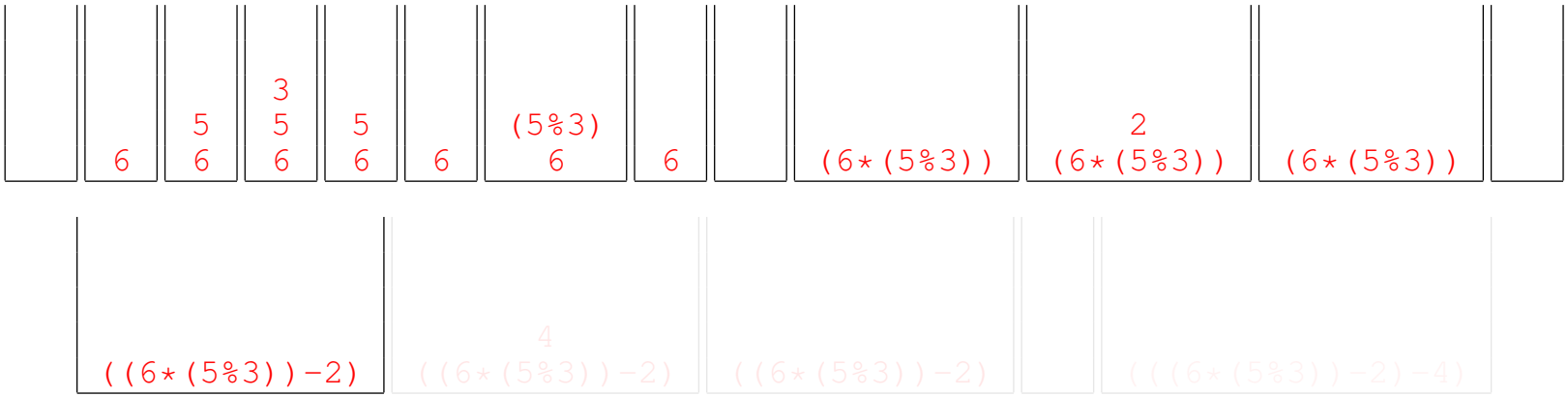
infixe



postfixe

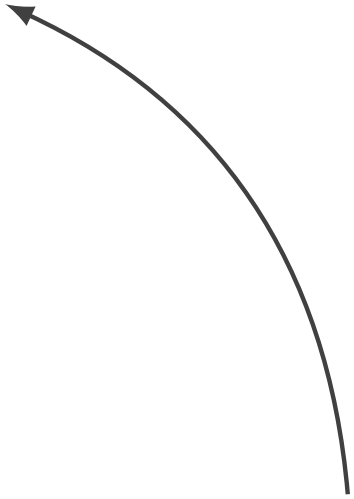
```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```

6 5 3 % * 2 - 4 -



6 * (5 % 3) - 2 - 4

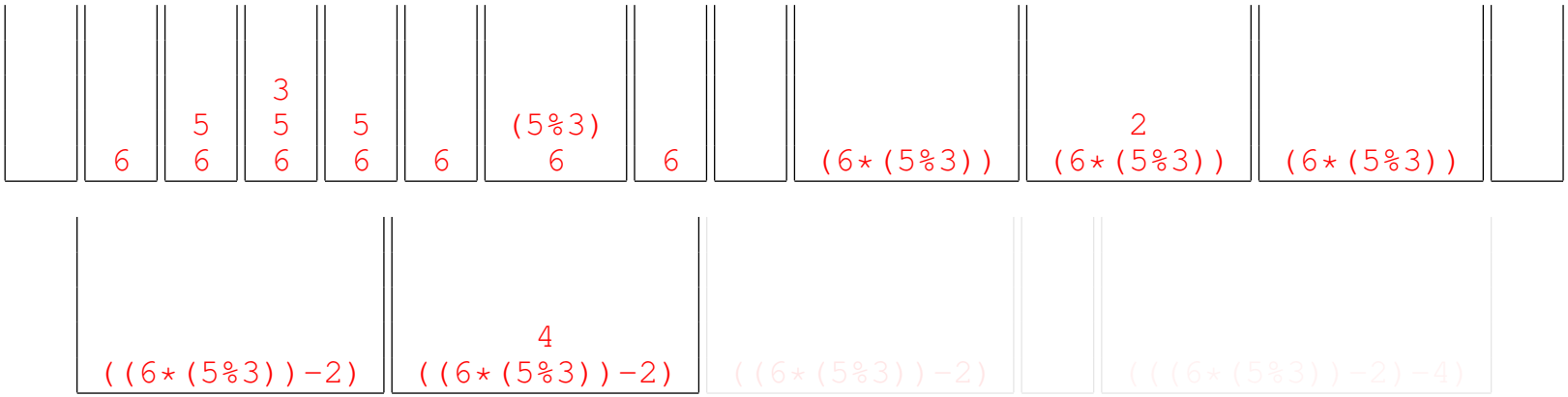
infixe



postfixe

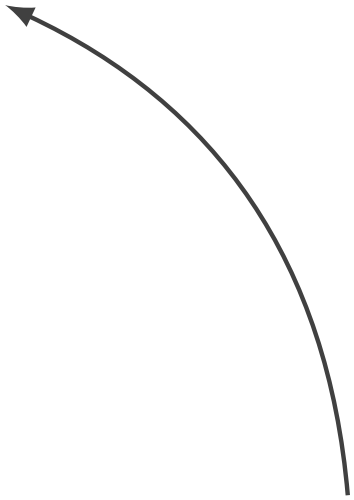
6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```



```
6 * ( 5 % 3 ) - 2 - 4
```

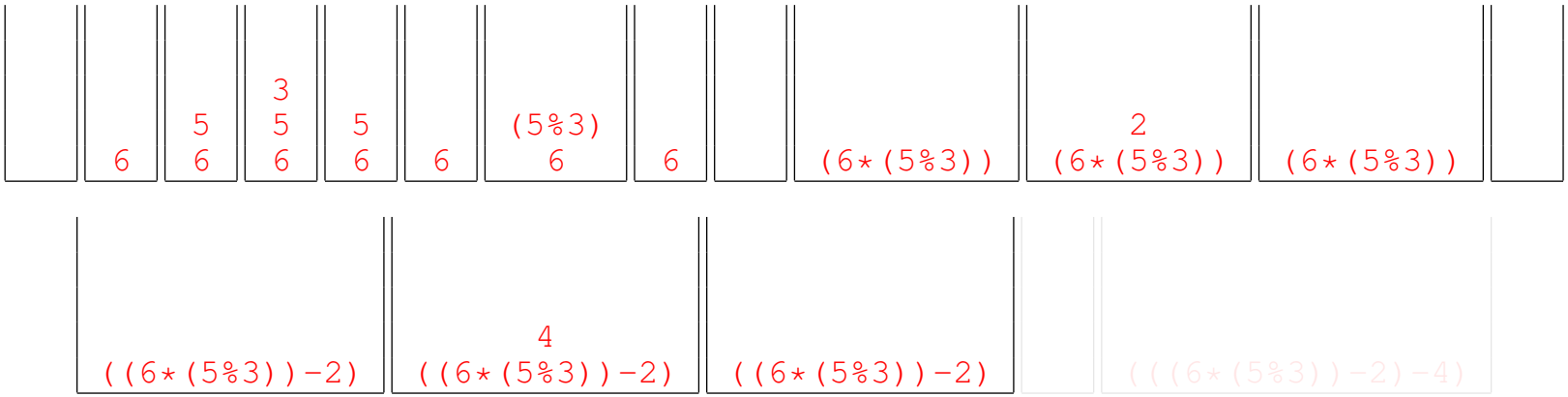
infixe



postfixe

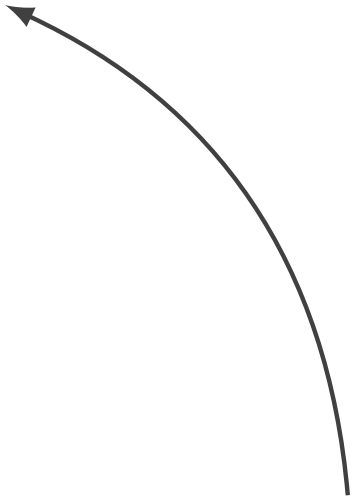
```
6 5 3 % * 2 - 4 -
```

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```



6 * (5 % 3) - 2 - 4

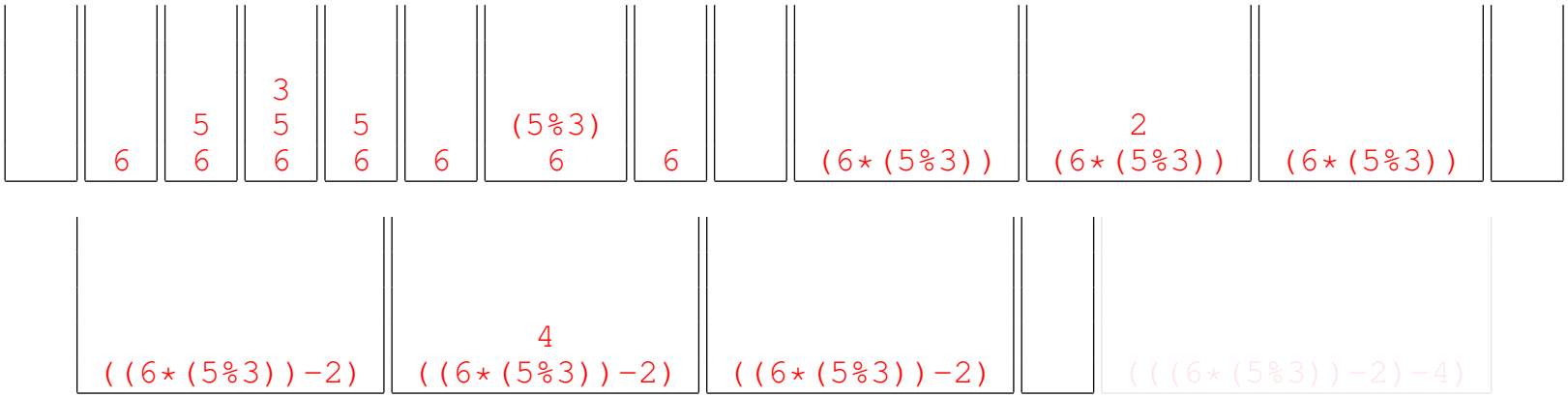
infixe



postfixe

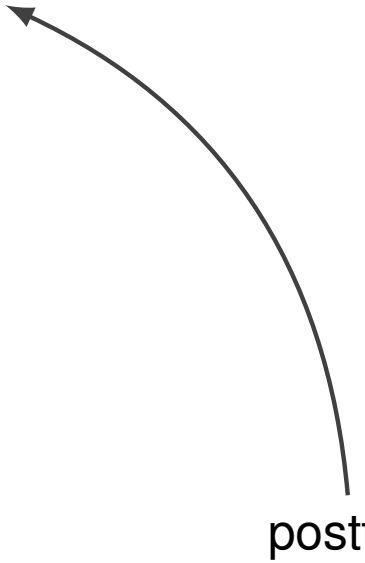
6 5 3 % * 2 - 4 -

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```



6 * (5 % 3) - 2 - 4

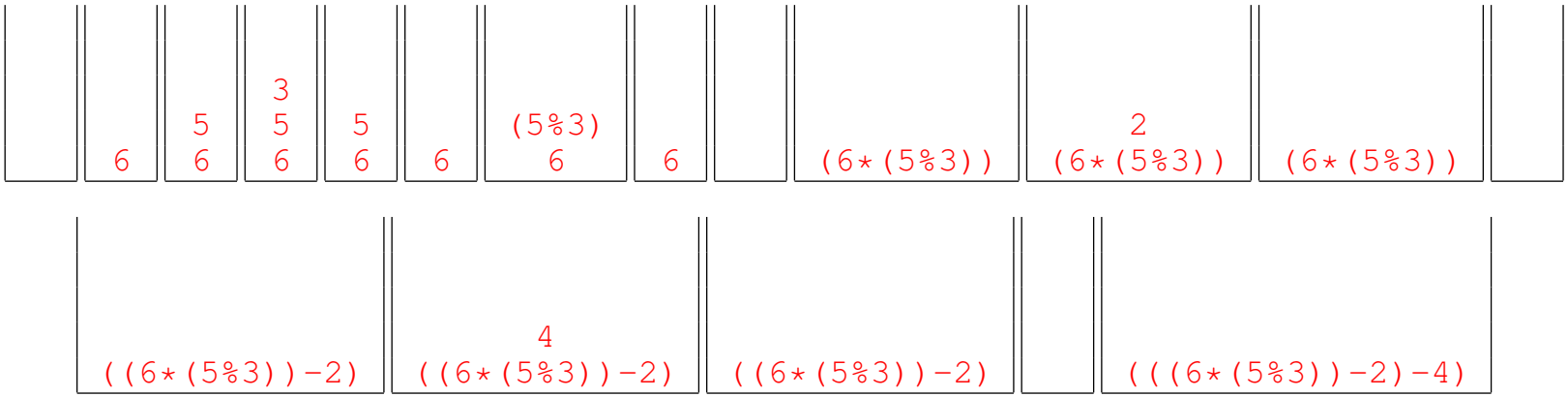
infixe



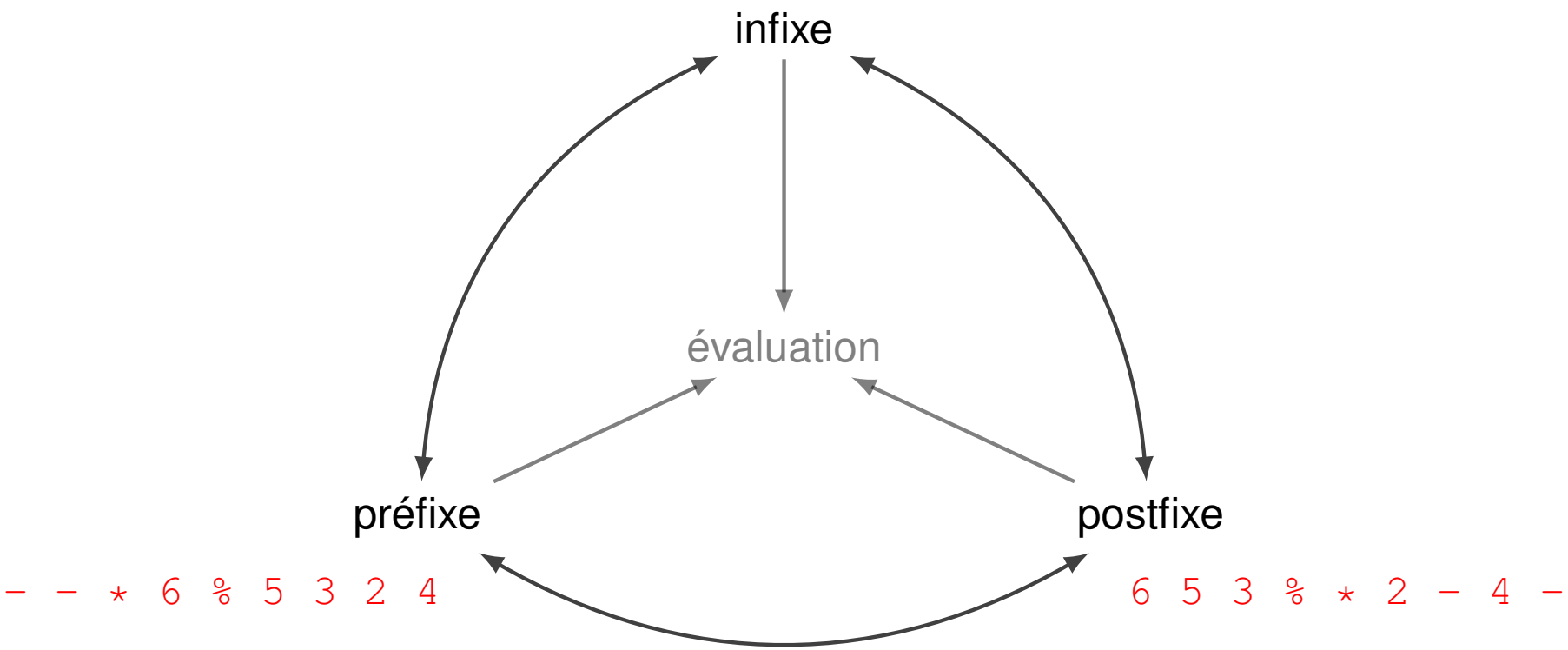
postfixe

6 5 3 % * 2 - 4 -

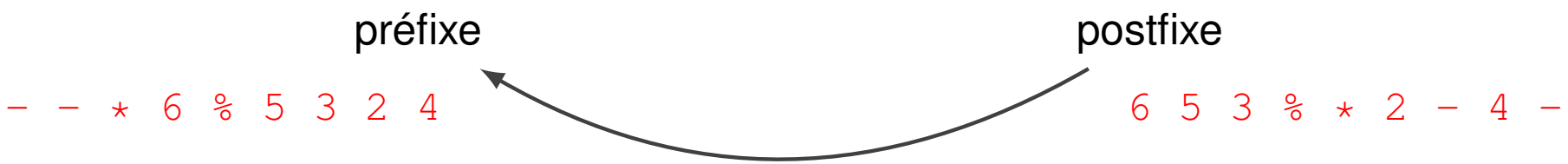
```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit l'infixe du nouvel opérande
 *     entouré de parenthèses
 *   * on l'empile
 */
```

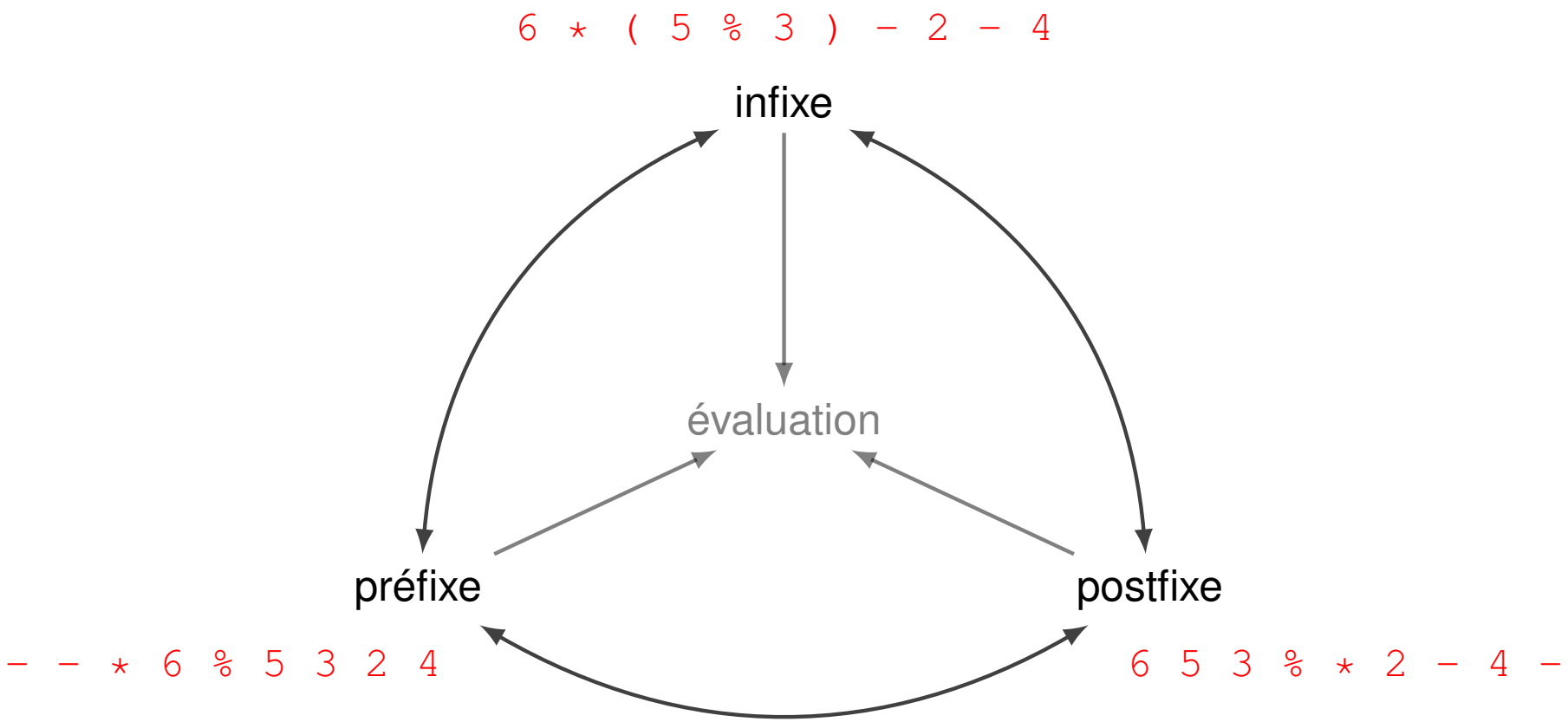


6 * (5 % 3) - 2 - 4



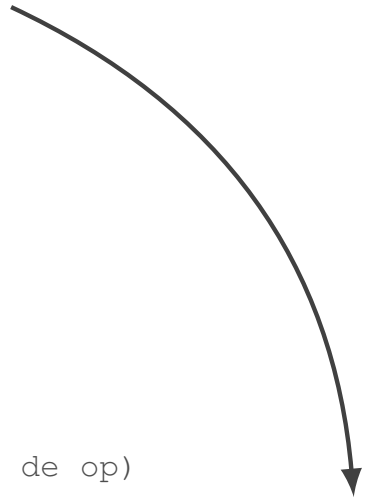
```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'empile
 * sinon (c'est un opérateur (binaire))
 *   * on dépile (deux) opérandes
 *   * on construit la préfixe du nouvel opérande
 *   * on l'empile
 */
```




6 * (5 % 3) - 2 - 4
infixe
évaluation
préfixe
postfixe
- - * 6 % 5 3 2 4
6 5 3 % * 2 - 4 -

```
6 * ( 5 % 3 ) - 2 - 4
```

infixe



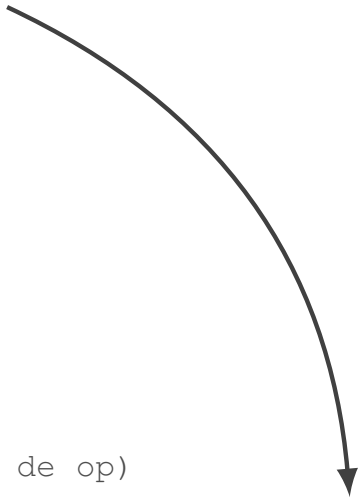
postfixe

```
6 5 3 % * 2 - 4 -
```

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```

```
6 * ( 5 % 3 ) - 2 - 4
▲
```

infixe



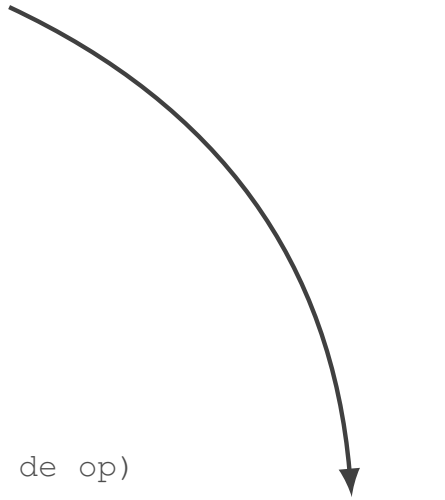
postfixe

6

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```

6 * (5 % 3) - 2 - 4

infixe



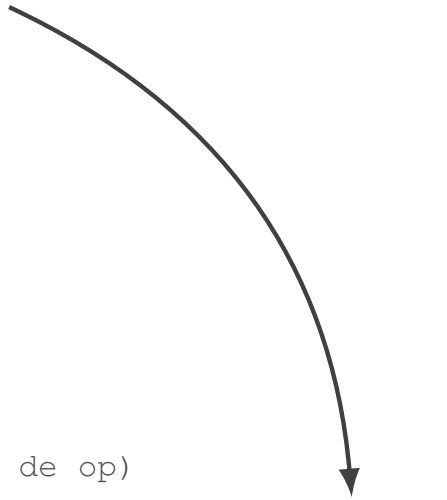
postfixe

6

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```

6 * (5 % 3) - 2 - 4

infixe



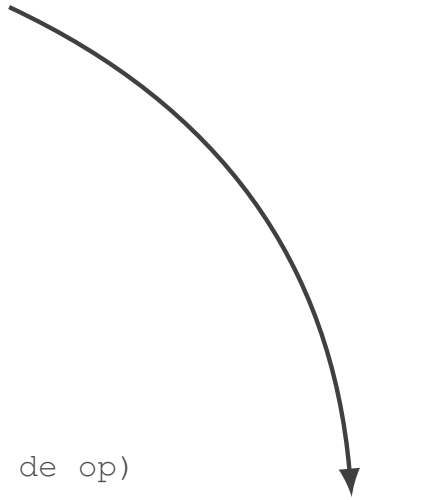
postfixe

6

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```

6 * (5 % 3) - 2 - 4

infixe



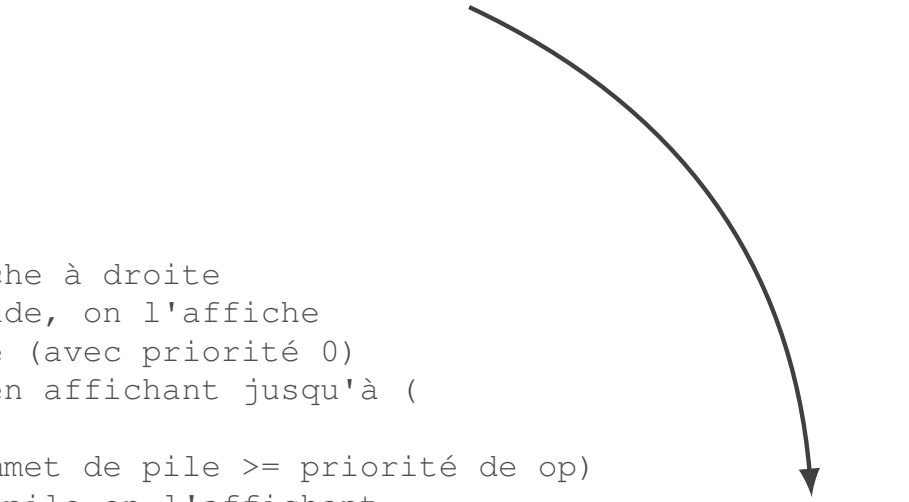
postfixe

6 5

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```

6 * (5 % 3) - 2 - 4

infixe



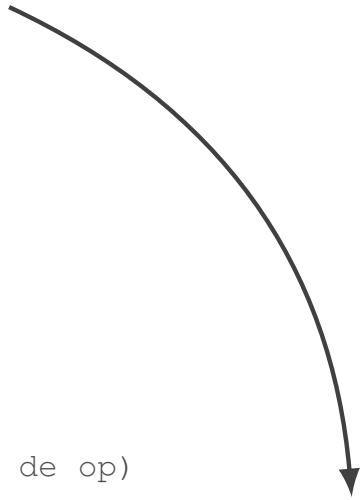
postfixe

6 5

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```

6 * (5 % 3) - 2 - 4

infixe



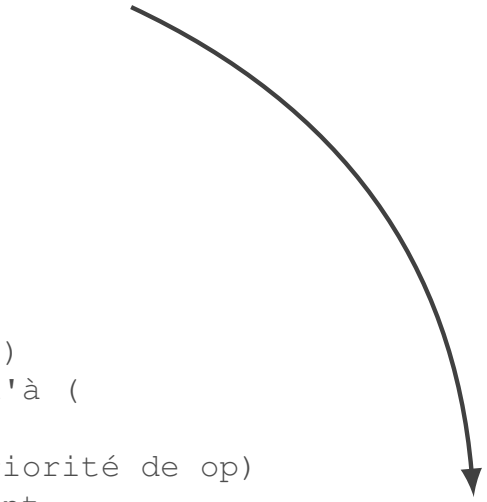
postfixe

6 5 3

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```

6 * (5 % 3) - 2 - 4

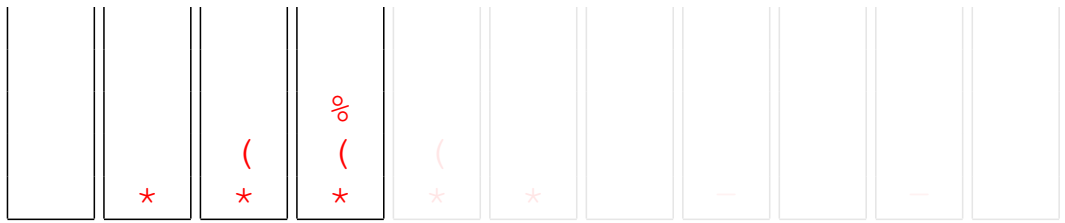
infixe



postfixe

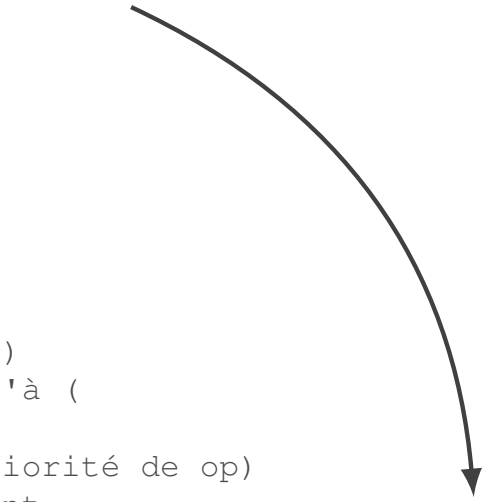
6 5 3

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```



6 * (5 % 3) - 2 - 4

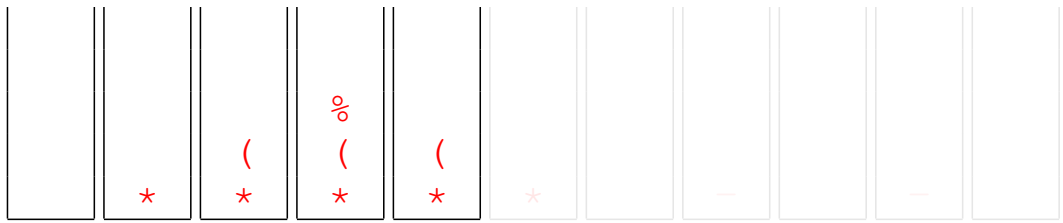
infixe



postfixe

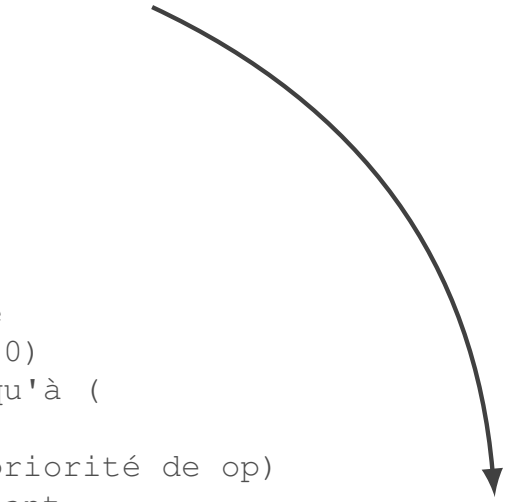
6 5 3 %

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```



6 * (5 % 3) - 2 - 4

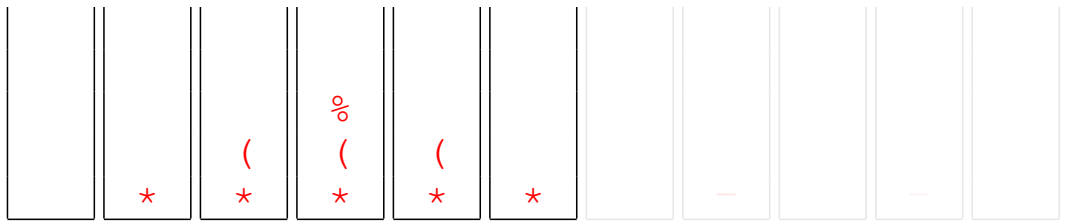
infixe



postfixe

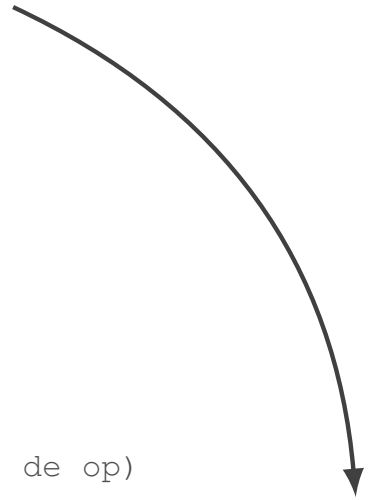
6 5 3 %

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```



6 * (5 % 3) - 2 - 4

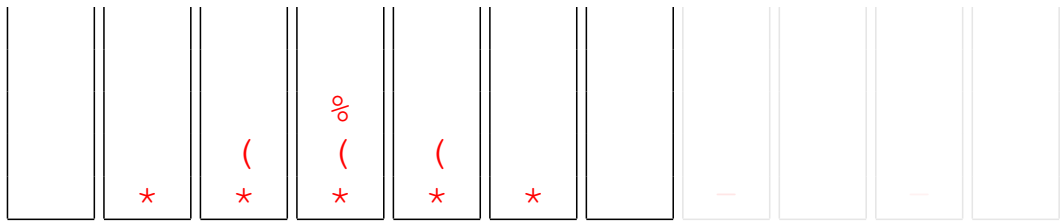
infixe



postfixe

6 5 3 %

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```



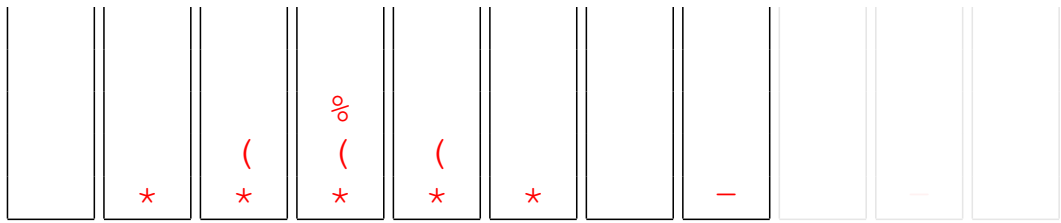
6 * (5 % 3) - 2 - 4

infixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```

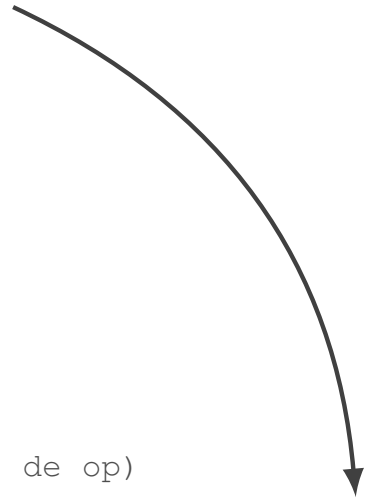
postfixe

6 5 3 % *



6 * (5 % 3) - 2 - 4

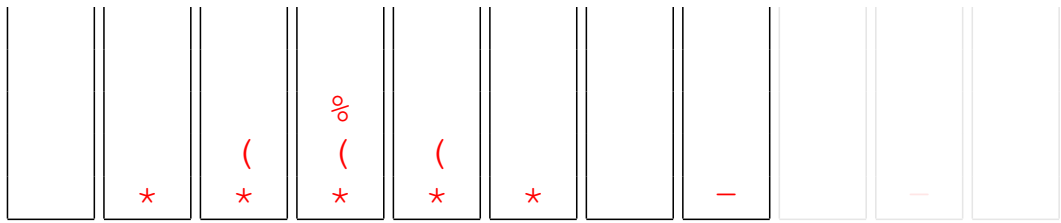
infixe



postfixe

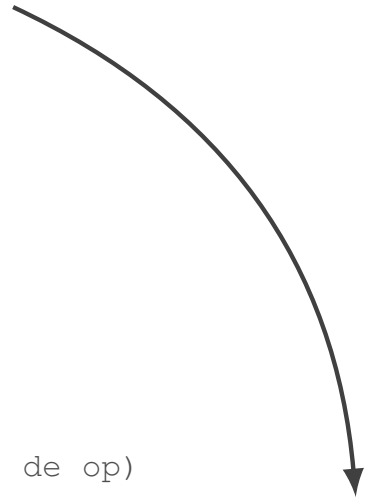
6 5 3 % * 2

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```



6 * (5 % 3) - 2 - 4

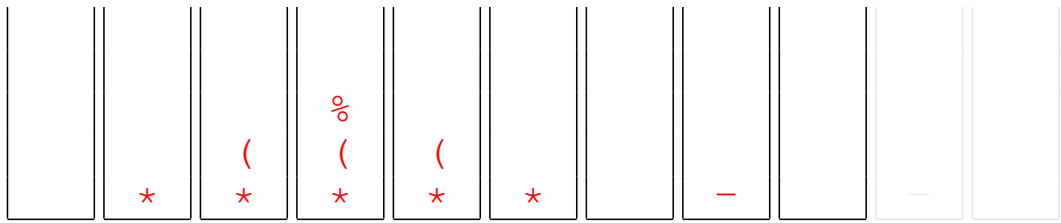
infixe



postfixe

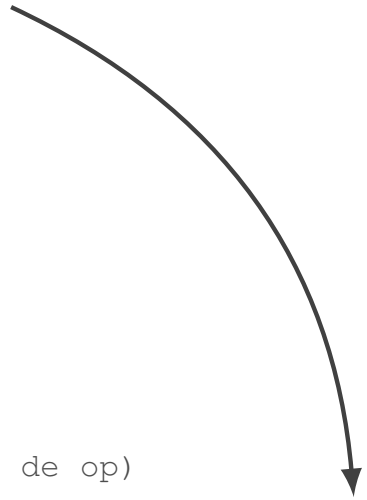
6 5 3 % * 2

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```



```
6 * ( 5 % 3 ) - 2 - 4
```

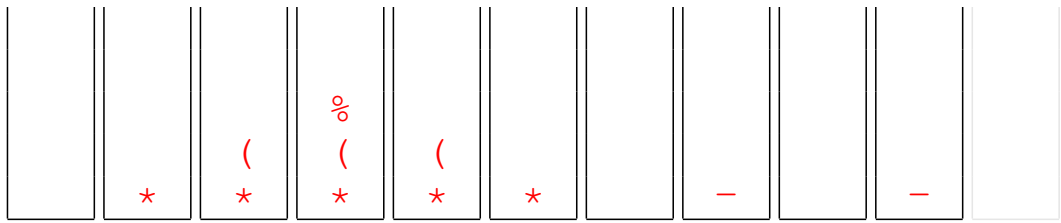
infixe



postfixe

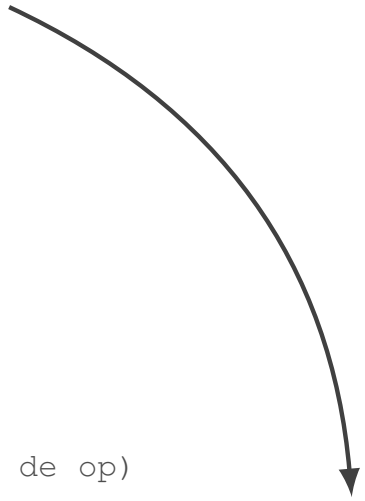
```
6 5 3 % * 2 -
```

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```



```
6 * ( 5 % 3 ) - 2 - 4
```

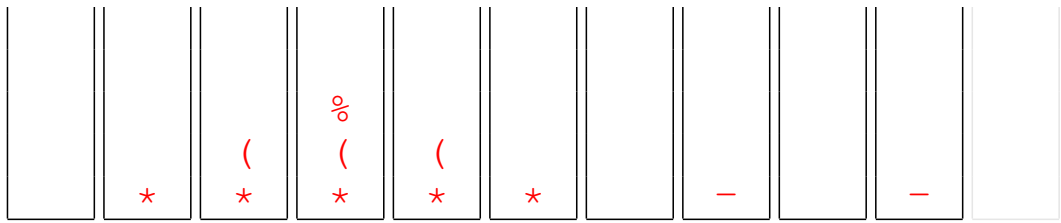
infixe



postfixe

```
6 5 3 % * 2 - 4
```

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```



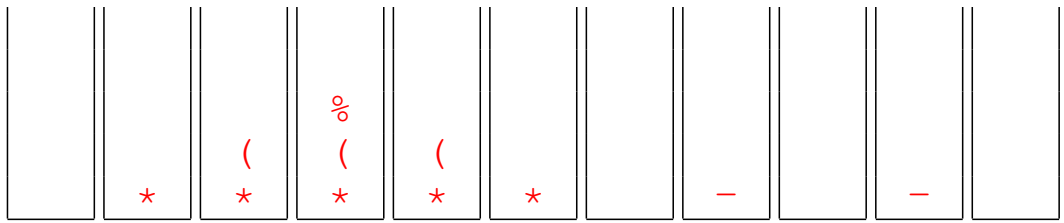
```
6 * ( 5 % 3 ) - 2 - 4
```

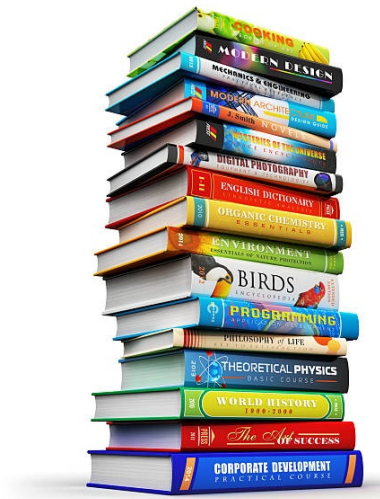
infixe

```
/* on lit l'expression de gauche à droite
 * si le symbole est un opérande, on l'affiche
 * si c'est une (, on l'empile (avec priorité 0)
 * si c'est une ), on dépile en affichant jusqu'à (
 * si c'est un opérateur op
 * * tant que (priorité du sommet de pile >= priorité de op)
 * * * on dépile le sommet de pile en l'affichant
 * * on empile op
 */
```

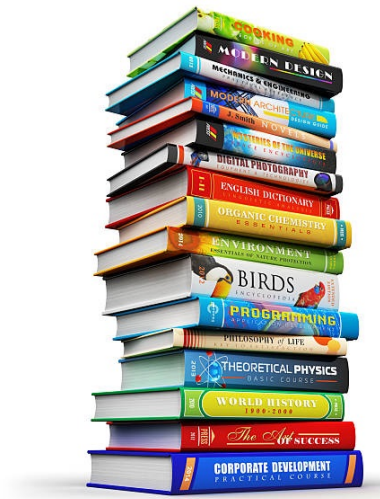
postfixe

```
6 5 3 % * 2 - 4 -
```




COOKING
MODERN DESIGN
DIGITAL PHOTOGRAPHY
ENGLISH DICTIONARY
ORGANIC CHEMISTRY
ESSENTIALS
ENVIRONMENT
BIRDS
PROGRAMMING
PHILOSOPHY of LIFE
THEORETICAL PHYSICS
BASIC COURSE
WORLD HISTORY
The Art of SUCCESS
CORPORATE DEVELOPMENT
PRACTICAL COURSE

pile
(stack)



pile
(stack)

tas
(heap)