

UNIVERSITÉ DE LIÈGE

INFO0946

INTRODUCTION À LA PROGRAMMATION

Un exercice dont vous êtes le Héros · l'Héroïne

Evaluation d'Expressions

Benoit DONNET

Simon LIÉNARDY

Tasnim SAFADI

5 octobre 2020



Préambule

Exercices

Dans ce « TP dont vous êtes le héros », nous vous proposons de suivre pas à pas la résolution d'un exercice sur l'évaluation des expressions.

Il est dangereux d'y aller seul ¹ !

Partagez vos commentaires, questions, solutions alternatives sur le forum [eCampus](#). N'hésitez jamais !

1. Référence vidéoludique bien connue des Héros.

1.1 Rappel sur les Expressions

Si vous avez déjà lu ce rappel ou si vous êtes suffisamment à l'aise avec les expressions et leurs évaluations, vous pouvez directement atteindre l'énoncé de l'exercice.

1.1.1 Expression

Une *expression* (cfr. Chapitre 1, Slide 17) est la description du calcul d'une valeur, le résultat du calcul ayant un certain type.

Plus précisément, une expression est :

1. une constante (ou valeur littérale – e.g., 'a'). L'évaluation retourne le littéral lui-même ;
2. une variable, dénotée par son identificateur (e.g., `x`). Dans ce cas, l'évaluation retourne la valeur courante de la variable ;
3. obtenue par l'application d'opérateurs à d'autres expressions.

1.1.2 Opérateurs

La définition générale d'une expression (voir point 3 de la définition d'une *expression*) indique qu'on peut appliquer un ou plusieurs opérateur(s) afin de composer une expression plus complexe.

Un *opérateur* permet d'évaluer une expression bien définie sur des valeurs (*opérandes*) en produisant un résultat (*valeur* de l'expression).

La forme générale d'une expression avec un opérateur est la suivante :

$$x \alpha x$$

où x est l'opérande et α l'opérateur.

Un opérateur peut être :

- *unaire*. Il requiert une seule opérande. Exemple : -5 .
- *binaire*. Il requiert deux opérandes. Exemple : $4 + x$.
- *ternaire*. Il requiert trois opérandes. Ce type d'opérateur ne sera pas abordé dans le cadre du cours.

Le tableau 1 donne une liste des opérateurs "simples" en C.

Alerte : Le contexte est ultra important

Attention, la signification exacte d'un opérateur peut dépendre du contexte dans lequel il est utilisé. Par exemple : $5/2$ correspond à la division entière et donne pour résultat 2. Par contre, $5/2.0$ correspond à la division réelle et donne pour résultat 2.5. Pour la division, c'est donc le type des opérandes qui détermine la signification exacte de l'opération.

Il en va de même pour l'opérateur $*$. La liste des opérateurs montre que l'opérateur $*$ a deux significations : la multiplication et le déréférencement. A nouveau, c'est le contexte dans lequel l'opérateur va être utilisé qui déterminera sa sémantique. Ainsi, par exemple, $3 * x$ correspond clairement à une multiplication car on se trouve dans le cas d'un opérateur binaire. L'expression $*x$ correspondra au déréférencement de x (opérateur unaire) à la condition, bien entendu, que le type de x soit pointeur.

Opérateur	Signification	Type
Opérateurs <i>Arithmétiques</i>		
+	addition	binaire
-	soustraction	
*	multiplication	
/	division	
%	modulo	unaire
-	changement de signe	
Opérateurs de <i>Comparaison</i>		
<	plus petit que	binaire
<=	plus petit ou égal	
>	plus grand que	
>=	plus grand ou égal	
==	égal	
!=	différent	
Opérateurs <i>Booléens</i>		
&&	et “lazy”	binaire
	ou “lazy”	
!	négation	unaire
Opérateurs sur les <i>Pointeurs</i>		
*	Déréférencement	unaire
&	Référencement	

TABLE 1 – Liste (non exhaustive) des opérateurs usuels en C.

1.2 Enoncé

Si `int n=10`, `int p=5`, `int q=10` et `int r=2`, complétez le tableau ci-dessous. Donnez les valeurs stockées dans les variables après l'évaluation de l'expression (les valeurs des variables sont conservées d'une expression à l'autre) :

Expressions	n	p	q	r
Initialisation	10	5	10	2
<code>r = n == (p = q)</code>				
<code>n += p += q</code>				
<code>n = p = q = 5</code>				
<code>q = (n < p) * n++ - p</code>				

1.2.1 Méthode de résolution

Pour résoudre ce problème, nous allons procéder expression par expression, en commençant par la première. Il faut être très prudent car les valeurs sont conservées d'une expression à l'autre. Dès lors, une erreur dans l'évaluation d'une expression entraînera, forcément, des erreurs en cascade dans les expressions suivantes. Voici le programme que nous allons suivre :

1. Expression `r = n == (p = q)` (Sec. 1.3);
2. Expression `n += p += q` (Sec. 1.4);
3. Expression `n = p = q = 5` (Sec. 1.5);
4. Expression `q = (n < p) * n++ - p` (Sec. 1.6);
5. Le résultat final (Sec. 1.7).

1.3 Expression $r = n == (p = q)$

Avant d'entamer la résolution de l'exercice, il est impératif que vous vous sentiez à l'aise avec la notion d'expression. Si ce n'est pas le cas, jetez un oeil au **rappel**.

Si vous voyez directement comment procéder, voyez la suite **1.3.4**

Si vous êtes un peu perdu, voyez le rappel sur la priorité des opérateurs **1.3.1**

Si l'opérateur $=$ vous semble obscur, voyez le rappel sur l'affectation **1.3.2**

Si vous avez besoin d'un indice, rendez-vous **1.3.3**

1.3.1 Rappels sur la Priorité des Opérateurs

La *priorité des opérateurs* précise l'ordre dans lequel les calculs doivent être effectués dans une expression complexe. Le Tableau 2 donne la priorité des opérateurs en C.

Priorité	Opérateurs	Sens d'Evaluation
++++	(), [], . ->	→
	!, --, ++, -, *, &	←
	*, /, %	→
	+, -	→
	«, »	→
	<, <=, >, >=	→
	==, !=	→
	&	→
	^	→
		→
	&&,	→
----	=, +=, -=, ...	←

TABLE 2 – Priorité des Opérateurs.

La deuxième ligne du [tableau](#) représente des opérateurs unaires. Dès lors, * est l'opérateur de déréférencement et & celui de référencement.

De manière générale, l'évaluation d'une expression se fait de la gauche vers la droite². Cependant, dans certains cas (opérateurs unaires et opérateurs d'affectations), l'évaluation se fait de la droite (i.e., on évalue d'abord la *valeur à droite*) vers la gauche (on place le résultat dans la *valeur à gauche*).

Suite de l'Exercice

À vous ! Évaluez [l'expression 1](#).

Si l'opérateur = vous semble obscur, passez à la Sec. [1.3.2](#) pour un rappel.

Si vous avez besoin d'un indice pour l'expression `r = n == (p = q)`, rendez-vous à la Sec. [1.3.3](#).

2. Puisque nous lisons de gauche à droite...

1.3.2 Rappels sur l'Opérateur d'Affectation

L'opérateur d'*affectation* (=) est probablement l'opérateur le plus important car il permet la sauvegarde des informations en mémoire (i.e., dans une variable). Il se présente de la façon suivante :

```
1 var = expr
```

L'opérateur d'affectation permet de stocker le résultat de l'expression `expr` (appelée aussi *valeur à droite*) dans la variable `var` (appelée aussi *valeur à gauche*). Il est évident que le type du résultat de l'évaluation de la valeur à droite doit être cohérent avec le type de la valeur à gauche. On ne stocke pas des poires dans des abricots!

La **priorité des opérateurs** indique bien que l'évaluation de l'expression se fait de droite (la valeur à droite) à gauche (la valeur à gauche). Dès lors, après l'affectation, l'expression entière devient égale à la valeur affectée. Le fonctionnement général de l'affectation est illustré à la Fig. 1.

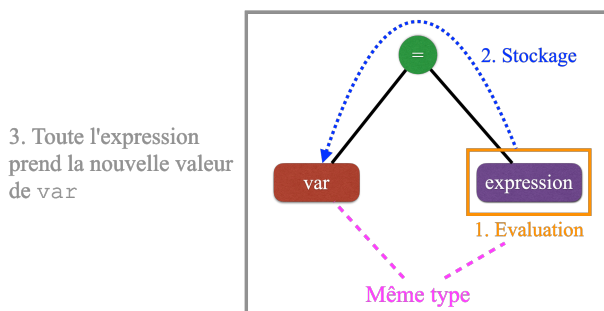


FIGURE 1 – Illustration du fonctionnement général de l'affectation.

Alerte : Risque de confusion

Il est très fréquent que les étudiants confonde l'opérateur d'*affectation* (=) et l'opérateur d'*égalité* (==). Le premier permet de stocker le résultat d'une expression dans une variable (et l'expression complète est évaluée à la valeur stockée), tandis que le deuxième permet de tester l'égalité entre deux expressions (le résultat de l'expression est alors booléen, i.e., soit *vrai*, soit *faux*). La confusion entre les deux opérateurs peut amener à des situations problématiques, en particulier dans le cadre des structures de contrôle (cfr. Chap. 2).

Une expression classique qui utilise l'opérateur d'affectation est l'incrémement (ou la décrémement) d'une variable.³ Par exemple :

```
1 x = x + 2
```

Dans ce cas, l'expression va stocker dans la variable `x` l'ancienne valeur de `x` augmentée de 2.

Le langage C fournit un raccourci (on parle de *sucre syntaxique*⁴) pour ce genre d'expression qui permet de combiner en un seul opérateur l'affectation et l'opération arithmétique (i.e., incrémement ou décrémement).

3. *Incrémenter* signifie *augmenter* une valeur. À l'inverse, *décrémenter* signifie diminuer une valeur.

4. Le sucre syntaxique est une extension de la syntaxe d'un langage de programmation afin de le rendre plus agréable à lire et à écrire, sans changer son expressivité.

Le sucre syntaxique pour l'incrémentation/décrémentation se présente de la façon suivante :

1

```
var  $\alpha$  = expr
```

où $\alpha \in \{+, -, *, /, \%\}$.

Ce raccourci est équivalent à

1

```
var = var  $\alpha$  expr
```

L'évaluation de l'expression se fait comme pour une affectation normale.

Attention, l'utilisation du sucre syntaxique ajoute, implicitement, des parenthèses autour de `expr`. Il faut donc comprendre l'expression comme suit :

1

```
var = var  $\alpha$  (expr)
```

Par exemple :

1

```
x += 2
```

est équivalent à

1

```
x = x + 2
```

Suite de l'Exercice

À vous ! Évaluez l'expression 1.

Si vous avez besoin d'un indice, rendez-vous à la Sec. 1.3.3.

1.3.3 Indice

Dans l'expression $r = n == (p = q)$, essayez d'identifier l'opérateur ayant la **priorité** la plus faible. Ensuite, construisez un arbre (comme dans les slides du cours théorique) avec cet opérateur le plus faible comme racine.

Évaluez ensuite l'expression pas à pas, de la partie la plus prioritaire (le bas de l'arbre) vers la moins prioritaire (le haut de l'arbre).

Suite de l'Exercice

À vous! Évaluez **l'expression 1**.

1.3.4 Mise en Commun de l'Evaluation de l'Expression

L'expression à évaluer est la suivante : $r = n == (p = q)$. Avant l'évaluation de cette expression, les différentes variables ont les valeurs suivantes :

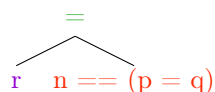
Variable	Valeur
n	10
p	10
q	10
r	2

TABLE 3 – Valeurs initiales des variables, avant évaluation de l'expression $r = n == (p = q)$.

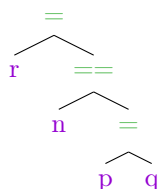
Pour commencer, identifions les différents opérateurs de l'expression et évaluons-la par ordre de **priorité**. Dans cette expression on retrouve les opérateurs suivants : = (**affectation**), == (**égalité**) et () (**parenthèses**). L'affectation est l'opération la moins prioritaire et prend la forme générale suivante :

```
1 var = expr
```

Si on adapte cette forme générale à notre expression, **var** correspond à **r** (valeur à gauche) et **expr** correspond à $n == (p = q)$ (valeur à droite). Graphiquement, cela donne :

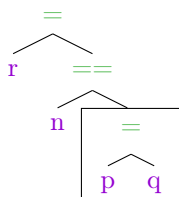


Quoiqu'il arrive, il faudra d'abord évaluer la valeur à droite (cfr. le **rappel sur l'affectation**). Celle-ci est constituée d'un == (**égalité**) et de () (**parenthèses**). C'est donc une expression booléenne à deux opérandes : **n** et **(p = q)** (les parenthèses forcent le fait que $p = q$ est la deuxième opérande). Graphiquement, on peut donc compléter la Figure ci-dessus. Cela donne :



A partir d'ici, on peut commencer à évaluer l'expression en partant du bas de l'arbre et en respectant l'ordre de **priorité** pour les éléments de même niveau :

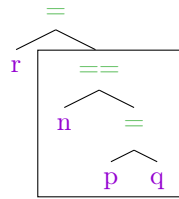
1. $(p = q)$ affecte la valeur de **q** dans **p**. La valeur de cette (sous-)expression est la nouvelle valeur de **p** (10).



Nouvelles valeurs :

Variable	Valeur
n	10
p	10
q	10
r	2

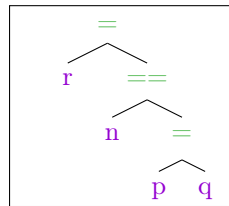
2. $n == 10$ effectue un test d'égalité entre l'expression à gauche et l'expression à droite. La valeur de n étant 10, le résultat du test d'égalité est **vrai** (1). 1 est donc la valeur de l'expression, ce qui correspond aussi à la valeur de la valeur à droite de l'affectation.



Nouvelles valeurs :

Variable	Valeur
n	10
p	10
q	10
r	2

3. $r = 1$ affecte la valeur 1 dans r .



Nouvelles valeurs :

Variable	Valeur
n	10
p	10
q	10
r	1

Finalement, les différentes variables ont les valeurs suivantes :

Variable	Valeur
n	10
p	10
q	10
r	1

TABLE 4 – Valeurs finales des variables, après évaluation de l'expression $r = n == (p = q)$.

Il faut maintenant passer à l'expression $n += p += q$. Voir la Sec. 1.4.

1.4 Expression $n += p += q$

- Si vous voyez directement comment procéder, voyez la suite [1.4.2](#)
- Si vous êtes un peu perdu, voyez le rappel sur la priorité des opérateurs [1.3.1](#)
- Si l'opérateur $=$ vous semble obscur, voyez le rappel sur l'opérateur d'affectation [1.3.2](#)
- Si vous avez besoin d'un indice, rendez-vous [1.4.1](#)

1.4.1 Indice

Essayez de commencer par réécrire l'expression $n += p += q$ en enlevant le **sucre syntaxique**. Une fois l'expression complète révélée, pensez à identifier l'opérateur de moindre **priorité** et construisez un arbre (comme dans les slides du cours théorique) avec cet opérateur comme racine.

Évaluez ensuite l'expression pas à pas, de la partie la plus prioritaire (le bas de l'arbre) vers la moins prioritaire (le haut de l'arbre).

Suite de l'Exercice

À vous ! Évaluez **l'expression 2**.

1.4.2 Mise en Commun de l'Evaluation de l'Expression

L'expression à évaluer est la suivante : $n += p += q$. Avant l'évaluation de cette expression, les différentes variables ont les valeurs suivantes :

Variable	Valeur
n	10
p	10
q	10
r	1

TABLE 5 – Valeurs initiales des variables, avant évaluation de l'expression $n += p += q$.

L'expression n'est composée que d'un seul opérateur : $+=$. Cet opérateur est un **sucre syntaxique** qui combine l'affectation et une opération mathématique. Sa forme générale est la suivante :

```
1  var += expr
```

ce qui est équivalent à

```
1  var = var + expr
```

Le moyen le plus simple d'évaluer l'expression $n += p += q$ est de "déplier" chacune des sous-expressions en réexprimant, dans sa forme complète, chaque sucre syntaxique. Il faut absolument commencer par l'expression la plus à droite car ce sucre syntaxique est une combinaison entre l'affectation et une opération arithmétique. Or, l'affectation s'**évalue** de droite à gauche.

Nous commençons donc par $p += q$. Dans cette expression **var** correspond à **p** et **expr** à **q**. Cela donne :

```
1  p = p + q
```

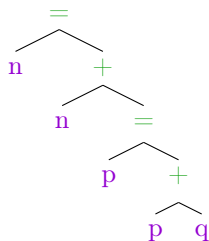
L'expression complète devient donc

```
1  n += p = p + q
```

Il reste à réécrire le dernier sucre syntaxique. Dans ce cas, **var** correspond à **n** et **expr** à $p = p + q$. Ici, pour faciliter la lecture et la compréhension, on va réécrire l'expression en ajoutant les **parenthèses implicites**. Soit

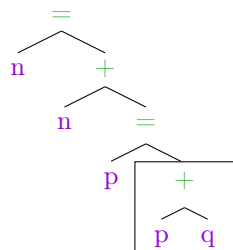
```
1  n = n + (p = p + q)
```

L'avantage d'écrire explicitement les parenthèses clarifie la priorité d'évaluation et simplifie l'écriture de la représentation graphique de l'expression. Cela donne donc



A partir d'ici, on peut commencer à évaluer l'expression en partant du bas de l'arbre et en repétant l'ordre de **priorité** pour les éléments de même niveau :

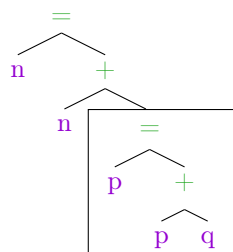
1. $p+q$ somme les valeurs de p et q .



Nouvelles valeurs :

Expression	Valeur
n	10
p	10
q	10
r	1

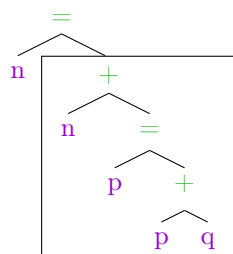
2. $p=20$ affecte le résultat obtenu précédemment (20) dans p .



Nouvelles valeurs :

Expression	Valeur
n	10
p	20
q	10
r	1

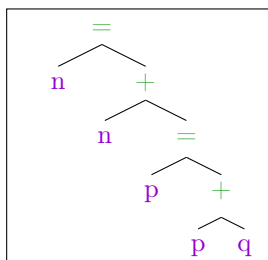
3. $n + p$ somme les valeurs de n et p .



Nouvelles valeurs :

Expression	Valeur
n	10
p	20
q	10
r	1

4. $n = 30$ affecte le résultat obtenu précédemment (30) dans n .



Nouvelles valeurs :

Expression	Valeur
n	30
p	20
q	10
r	1

Finalement, les différentes variables ont les valeurs suivantes :

Il faut maintenant passer à l'expression $n = p = q = 5$. Voir la Sec. 1.5.

Variable	Valeur
n	30
p	20
q	10
r	1

TABLE 6 – Valeurs finales des variables, après évaluation de l’expression $n += p += q$.

1.5 Expression $n = p = q = 5$

- Si vous voyez directement comment procéder, voyez la suite [1.5.2](#)
- Si vous êtes un peu perdu, voyez le rappel sur la priorité des opérateurs [1.3.1](#)
- Si l'opérateur `=` vous semble obscur, voyez le rappel sur l'opérateur d'affectation [1.3.2](#)
- Si vous avez besoin d'un indice, rendez-vous [1.5.1](#)

1.5.1 Indice

Pensez au sens de lecture/évaluation d'une expression basée sur l'affectation.

Suite de l'Exercice

À vous ! Évaluez l'expression 3.

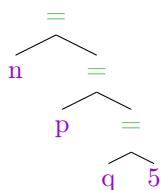
1.5.2 Mise en Commun de l'Evaluation de l'Expression

L'expression à évaluer est la suivante : $n = p = q = 5$. Avant l'évaluation de cette expression, les différentes variables ont les valeurs suivantes :

Variable	Valeur
n	30
p	20
q	10
r	1

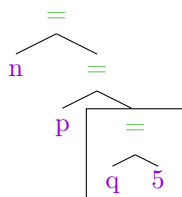
TABLE 7 – Valeurs initiales des variables, avant évaluation de l'expression $n = p = q = 5$.

Cette troisième expression enchaîne l'opérateur d'affectation (=). La représentation graphique de l'expression se fait en gardant en mémoire que l'affectation s'évalue de droite à gauche. Il vient donc



A partir d'ici, on peut commencer à évaluer l'expression en partant du bas de l'arbre et en respectant l'ordre de **priorité** pour les éléments de même niveau :

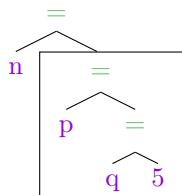
1. $q = 5$ affecte la valeur 5 à q.



Nouvelles valeurs :

Expression	Valeur
n	30
p	20
q	5
r	1

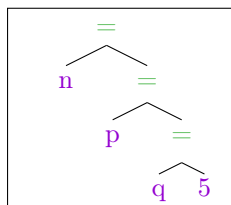
2. $p = 5$ affecte la valeur de l'expression à droite obtenue précédemment (5) dans p.



Nouvelles valeurs :

Expression	Valeur
n	30
p	5
q	5
r	1

3. $n = 5$ affecte la valeur de l'expression à droite obtenue précédemment (5) dans n.



Nouvelles valeurs :

Expression	Valeur
n	5
p	5
q	5
r	1

Finalement, les différentes variables ont les valeurs suivantes :

Variable	Valeur
n	5
p	5
q	5
r	1

TABLE 8 – Valeurs finales des variables, après évaluation de l’expression $n = p = q = 5$.

Passons à l’expression $q = (n < p) * n++ - p$. Voir la Sec. 1.6.

1.6 Expression $q = (n < p) * n++ - p$

- Si vous voyez directement comment procéder, voyez la suite [1.6.3](#)
- Si vous êtes un peu perdu, voyez le rappel sur la priorité des opérateurs [1.3.1](#)
- Si nécessaire, voyez le rappel sur les opérateurs d’incrément et de décrémentation [1.6.1](#)
- Si l’opérateur `=` vous semble obscur, voyez le rappel sur l’opérateur d’affectation [1.3.2](#)
- Si vous avez besoin d’un indice, rendez-vous [1.6.2](#)

1.6.1 Rappels sur l'Opérateur d'Incrémentation/Décrémentation

Le **rappel sur l'affectation** indique que le sucre syntaxique permet, entre autre, de simplifier l'incréméntation/décrémentation d'une variable. Il existe une version encore plus simplifiée à l'aide des opérateurs d'incréméntation et de décréméntation.

L'opérateur d'incréméntation, `++`, est un opérateur unaire (i.e., il ne s'applique qu'à une seule opérande) dont l'opérande est une valeur à gauche (i.e., l'opérande est là pour stocker le résultat de l'opération – ce ne peut donc être qu'une variable). L'équivalent pour la décréméntation existe : `--`.

L'opérateur d'incréméntation (resp. de décréméntation) peut être placé :

à droite de l'opérande :

```
1 x++
2 y--
```

la variable est incréméntée (`x`) ou décréméntée (`y`) d'une unité. Mais, attention, la valeur de toute l'expression est celle de l'opérande **avant** l'incréméntation/décréméntation.

à gauche de l'opérande :

```
1 ++x
2 --y
```

la variable est incréméntée (`x`) ou décréméntée (`y`) d'une unité. Mais, attention, la valeur de toute l'expression est celle de l'opérande **après** l'incréméntation/décréméntation.

Un moyen simple (mais efficace) pour se souvenir du fonctionnement de l'opérateur, c'est de s'appuyer sur le sens de lecture en français (gauche → droite). Si l'opérateur est à droite de l'opérande, alors on lit d'abord la valeur de variable, cette valeur devenant la valeur de toute l'expression, et, ensuite, on incrémente/décrémente la variable. Si l'opérateur est à gauche, on effectue d'abord l'incréméntation/décréméntation et, ensuite, on lit la valeur de la variable qui a déjà été modifiée par l'opérateur (la valeur de l'expression est alors cette nouvelle valeur de la variable).

Alerte : Lequel faut-il utiliser ?

Est-il souhaitable de placer l'opérateur à gauche ou à droite de l'opérande ?

En théorie, placer l'opérateur à gauche est plus efficace (une opération en moins pour le CPU, comparé au placement à droite).

En pratique, avec les machines dont on dispose en 2020, la différence entre les deux est devenue totalement négligeable.

Dans les codes que vous aurez à construire durant le quadrimestre, vous avez la possibilité d'utiliser le positionnement que vous préférez pour l'opérateur d'incréméntation/décréméntation.

Suite de l'Exercice

Si vous avez besoin d'un indice pour l'expression `q = (n < p) * n++ - p`, rendez-vous à la Sec. 1.6.2. À vous ! Évaluez l'expression 4.

1.6.2 Indice

Dans l'expression $q = (n < p) * n++ - p$, essayez d'identifier l'opérateur ayant la **priorité** la plus faible. Ensuite, construisez un arbre (comme dans les slides du cours théorique) avec cet opérateur le plus faible comme racine.

Évaluez ensuite l'expression pas à pas, de la partie la plus prioritaire (le bas de l'arbre) vers la moins prioritaire (le haut de l'arbre).

Suite de l'Exercice

À vous ! Évaluez **l'expression 4**.

1.6.3 Mise en Commun de l'Evaluation de l'Expression

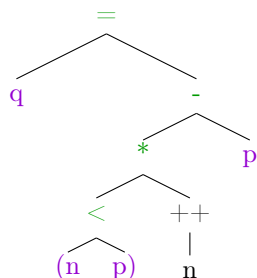
L'expression à évaluer est la suivante : $q = (n < p) * n++ - p$. Avant l'évaluation de cette expression, les différentes variables ont les valeurs suivantes :

Variable	Valeur
n	5
p	5
q	5
r	1

TABLE 9 – Valeurs initiales des variables, avant évaluation de l'expression $q = (n < p) * n++ - p$.

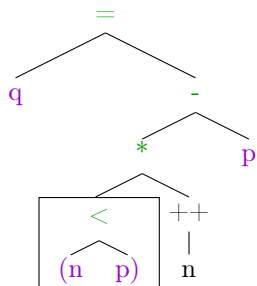
Dans cette expression, les parenthèses sont prioritaires ; $(n < p)$ sera donc la première expression à évaluer. L'opérateur d'incrément $(++)$ est le deuxième prioritaire. Comme celui-ci est à droite de l'opérande (n) , lors de l'évaluation de l'expression, la valeur de l'expression correspond à celle de n avant l'incrément. On retrouve ensuite les deux opérateurs arithmétiques $*$ et $-$ avec des priorités identiques à l'algèbre. Enfin, on a un opérateur d'affectation.

Si on représente graphiquement cette expression, cela donne



A partir d'ici, on peut commencer à évaluer l'expression en partant du bas de l'arbre et en respectant l'ordre de **priorité** pour les éléments de même niveau :

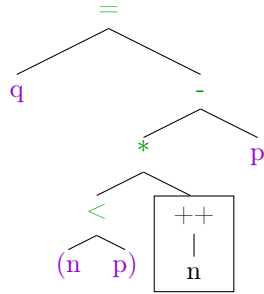
1. $(n < p)$ contient un opérateur de comparaison, l'expression prendra donc une valeur booléenne. n n'étant pas plus petit que p , l'expression sera évaluée à faux 0.



Nouvelles valeurs :

Expression	Valeur
n	5
p	5
q	5
r	1

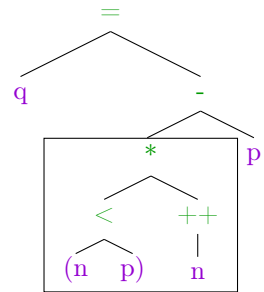
2. $n++$ incrémentera d'une unité n après l'évaluation de l'expression (qui vaudra l'ancienne valeur de n , soit 5).



Nouvelles valeurs :

Expression	Valeur
n	6
p	5
q	5
r	1

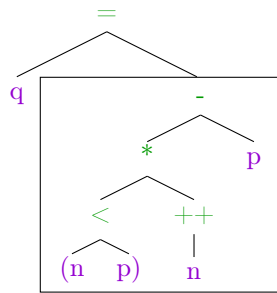
3. $0 * n$ prend la valeur du résultat de cette multiplication ; 0.



Nouvelles valeurs :

Expression	Valeur
n	6
p	5
q	5
r	1

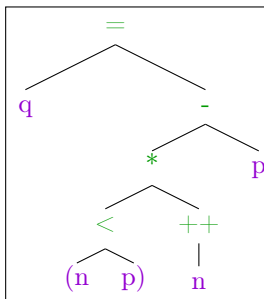
4. L'expression $0 - p$ est évaluée à -5.



Nouvelles valeurs :

Expression	Valeur
n	6
p	5
q	5
r	1

5. $q = -5$ affecte la valeur de l'expression à droite obtenue précédemment (-5) dans q.



Nouvelles valeurs :

Expression	Valeur
n	6
p	5
q	-5
r	1

Finalement, les différentes variables ont les valeurs indiquées dans le Tableau 10. Pour avoir un rappel des différentes valeurs obtenues aux étapes, voir la Sec. 1.7.

Variable	Valeur
n	6
p	5
q	-5
r	1

TABLE 10 – Valeurs finales des variables, après évaluation de l’expression $q = (n < p) * n++ - p$.

1.7 Résultat Final

Le résultat final est le suivant :

Expressions	n	p	q	r
Initialisation	10	5	10	2
$r = n == (p = q)$	10	10	10	1
$n += p += q$	30	20	10	1
$n = p = q = 5$	5	5	5	1
$q = (n < p) * n++ - p$	6	5	-5	1