Langage C

TD 2: Expressions et opérateurs

Julien SEINTURIER
Associate Professor

Université de Toulon
julien.seinturier@univ-tln.fr
http://www.seinturier.fr/teaching

1. Expressions arithmétiques

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+		

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-		

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	Binaire

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	Binaire

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	Binaire
*		

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	Binaire
*	Multiplication de deux nombres	

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	Binaire
*	Multiplication de deux nombres	Binaire

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	Binaire
*	Multiplication de deux nombres	Binaire

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	Binaire
*	Multiplication de deux nombres	Binaire
/		Binaire

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	Binaire
*	Multiplication de deux nombres	Binaire
/	Division de deux nombres	

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	Binaire
*	Multiplication de deux nombres	Binaire
/	Division de deux nombres	Binaire

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	Binaire
*	Multiplication de deux nombres	Binaire
/	Division de deux nombres	Binaire

Attention: La division de **deux nombres entiers** est la division **entière** et retourne un entier: `13 / 2 = 6`

Si l'une des deux opérandes est un nombre réel, la division retourne un **nombre réel**: `13.0 / 2 = 6.5`

Exercice 1.1. Citer les cinq opérateurs arithmétiques du langage C.
Préciser leur sémantique et leur nature (unaire, binaire, ...).

Les cinq opérateurs arithmétiques du Langage C sont:

Opérateur	Sémantique	Nature
+	Addition de deux nombres	Binaire
-	Soustraction de deux nombres	Binaire
*	Multiplication de deux nombres	Binaire
/	Division de deux nombres	Binaire
%	Modulo de deux nombres (reste de division)	Binaire

Exercice 1.2. Ecrire de 4 façons différentes une expression qui affecte à une variable x sa valeur plus 1 (incrémentation de 1).

Exercice 1.2. Ecrire de 4 façons différentes une expression qui affecte à une variable x sa valeur plus 1 (incrémentation de 1).

Voici 1 façon d'incrémenter de 1 la valeur d'une variable x.

```
x = x + 1
```

Exercice 1.2. Ecrire de 4 façons différentes une expression qui affecte à une variable x sa valeur plus 1 (incrémentation de 1).

Voici 2 façons d'incrémenter de 1 la valeur d'une variable x.

```
x = x + 1
```

```
x += 1
```

Exercice 1.2. Ecrire de 4 façons différentes une expression qui affecte à une variable x sa valeur plus 1 (incrémentation de 1).

Voici 3 façons d'incrémenter de 1 la valeur d'une variable x.

```
x = x + 1
```

```
x += 1
```

```
x++
```

Exercice 1.2. Ecrire de 4 façons différentes une expression qui affecte à une variable x sa valeur plus 1 (incrémentation de 1).

Voici 4 façons d'incrémenter de 1 la valeur d'une variable x.

```
x = x + 1
```

```
x += 1
```

```
x++
```

```
++x
```

Exercice 1.3.

Soit une variable x de valeur 1. Que retournent les expressions `x--`, `--x` et `x -= 1` ?
Quelle sera la valeur de x après chaque expression (évaluées indépendamment) ?

Exercice 1.3.

Soit une variable x de valeur 1. Que retournent les expressions `x--`, `--x` et `x -= 1` ?
Quelle sera la valeur de x après chaque expression (évaluées indépendamment) ?

`x--` retourne 1

Exercice 1.3.

Soit une variable x de valeur 1. Que retournent les expressions `x--`, `--x` et `x -= 1` ? Quelle sera la valeur de x après chaque expression (évaluées indépendamment) ?

`x--` retourne 1

`--x` retourne 0

Exercice 1.3.

Soit une variable x de valeur 1. Que retournent les expressions `x--`, `--x` et `x -= 1` ? Quelle sera la valeur de x après chaque expression (évaluées indépendamment) ?

`x--` retourne 1

`--x` retourne 0

`x -= 1` retourne 0

Exercice 1.3.

Soit une variable x de valeur 1. Que retournent les expressions `x--`, `--x` et `x -= 1` ? Quelle sera la valeur de x après chaque expression (évaluées indépendamment) ?

`x--` retourne 1

`--x` retourne 0

`x -= 1` retourne 0

Les **opérateurs unaires préfixés** `++x` et `--x` retournent la valeur de x **après** modification.

Exercice 1.3.

Soit une variable x de valeur 1. Que retournent les expressions `x--`, `--x` et `x -= 1` ? Quelle sera la valeur de x après chaque expression (évaluées indépendamment) ?

`x--` retourne 1

`--x` retourne 0

`x -= 1` retourne 0

Les **opérateurs unaires préfixés** `++x` et `--x` retournent la valeur de x **après** modification.

Les **opérateurs unaires postfixés** `x++` et `x--` retournent la valeur de x **avant** modification.

Exercice 1.3.
Soit une variable x de valeur 1. Que retournent les expressions `x--`, `--x` et `x -= 1` ?
Quelle sera la valeur de x après chaque expression (évaluées indépendamment) ?

`x--` retourne 1

`--x` retourne 0

`x -= 1` retourne 0

Les **opérateurs unaires préfixés** ++x et `--x` retournent la valeur de x **après** modification.

Les **opérateurs unaires postfixés** x++ et `x--` retournent la valeur de x **avant** modification.

Dans **chaque cas**, x vaut 0 après chacune des expressions.

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

o	char
char	

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

o	char
char	int

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short
char	int	

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int
char	int	int	

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float
char	int	int	int	

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short					

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int				

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int			

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int		

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int					

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int				

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int			

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int		

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float					

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float				

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float	float			

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float	float	float		

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float	float	float	float	

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être +, -, *, / ou %) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float	float	float	float	double

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float	float	float	float	double
double					

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être +, -, *, / ou %) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float	float	float	float	double
double	double				

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float	float	float	float	double
double	double	double			

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float	float	float	float	double
double	double	double	double		

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être +, -, *, / ou %) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float	float	float	float	double
double	double	double	double	double	

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être `+`, `-`, `*`, `/` ou `%`) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float	float	float	float	double
double	double	double	double	double	double

Exercice 1.4. Soit un opérateur arithmétique o (pouvant être +, -, *, / ou %) et soit les types `char`, `short`, `int`, `float` et `double`. De quel type est l'opérateur arithmétique o pour chaque couple de type possible ? En déduire une règle de conversion de type pour les opérateurs arithmétiques.

O	char	short	int	float	double
char	int	int	int	float	double
short	int	int	int	float	double
int	int	int	int	float	double
float	float	float	float	float	double
double	double	double	double	double	double

Un opérateur arithmétique **retourne** toujours le **type le plus petit** pouvant **contenir** son résultat, **sauf pour `char` et `short`**. Ces deux types étant changés en int avant l'évaluation de l'opérateur.

2. Expressions logiques

Exercice 2.1. Quel sont les équivalents numériques des valeurs booléennes VRAI et FAUX en langage C ?

En langage C:

- Il **n'existe pas** de type booléen.

Exercice 2.1. Quel sont les équivalents numériques des valeurs booléennes VRAI et FAUX en langage C ?

En langage C:

- Il **n'existe pas** de type booléen.

- Toute valeur numérique **différente de 0** est équivalente au **VRAI**.

Exercice 2.1. Quel sont les équivalents numériques des valeurs booléennes VRAI et FAUX en langage C ?

En langage C:

- Il **n'existe pas** de type booléen.
- Toute valeur numérique **différente de 0** est équivalente au **VRAI**.
- Toute valeur numérique **égale à 0** est équivalente au **FAUX**.

Exercice 2.2. Citer les opérateurs logiques du langage C et expliciter leur sémantique.

Exercice 2.2. Citer les opérateurs logiques du langage C et expliciter leur sémantique.

Le langage C possède **3 opérateurs logiques** :

- ! qui représente la **négation logique** (non)

Exercice 2.2. Citer les opérateurs logiques du langage C et expliciter leur sémantique.

Le langage C possède **3 opérateurs logiques** :

- ! qui représente la **négation logique** (non)
- && qui représente la **conjonction logique** (et)

Exercice 2.2. Citer les opérateurs logiques du langage C et expliciter leur sémantique.

Le langage C possède **3 opérateurs logiques** :

- ! qui représente la **négation logique** (non)
- && qui représente la **conjonction logique** (et)
- || qui représente la **disjonction logique** (ou)

Exercice 2.2. Citer les opérateurs logiques du langage C et expliciter leur sémantique.

Le langage C possède **3 opérateurs logiques** :

- ! qui représente la **négation logique** (non)
- && qui représente la **conjonction logique** (et)
- || qui représente la **disjonction logique** (ou)

A retenir: étant donné que le vrai et le faux sont représentés arithmétiquement (par des nombres), les **opérateurs logiques** sont également **des opérateurs arithmétiques** (ils retournent un nombre)

```
a + b && c / d
```

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

&&	VRAI	FAUX
VRAI		
FAUX		

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

&&	VRAI	FAUX
VRAI	1	
FAUX		

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

&&	VRAI	FAUX
VRAI	1	0
FAUX		

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

&&	VRAI	FAUX
VRAI	1	0
FAUX	0	

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

&&	VRAI	FAUX
VRAI	1	0
FAUX	0	0

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

&&	VRAI	FAUX
VRAI	1	0
FAUX	0	0

||	VRAI	FAUX
VRAI		
FAUX		

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

&&	VRAI	FAUX
VRAI	1	0
FAUX	0	0

||	VRAI	FAUX
VRAI	1	
FAUX		

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

&&	VRAI	FAUX
VRAI	1	0
FAUX	0	0

||	VRAI	FAUX
VRAI	1	1
FAUX		

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

&&	VRAI	FAUX
VRAI	1	0
FAUX	0	0

||	VRAI	FAUX
VRAI	1	1
FAUX	1	

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

&&	VRAI	FAUX
VRAI	1	0
FAUX	0	0

||	VRAI	FAUX
VRAI	1	1
FAUX	1	0

Exercice 2.3. Donner les tables de valeur des opérateurs logiques du Langage C.

&&	VRAI	FAUX
VRAI	1	0
FAUX	0	0

||	VRAI	FAUX
VRAI	1	1
FAUX	1	0

!VRAI	!FAUX
0	1

Exercice 2.4. Soit a et b deux variables. Ecrire en langage C l'expression logique « *non (a et b) ou non b et non a* ». Quel problème peut se poser lors de l'évaluation de l'expression ?

Exercice 2.4. Soit a et b deux variables. Ecrire en langage C l'expression logique « *non (a et b) ou non b et non a* ». Quel problème peut se poser lors de l'évaluation de l'expression ?

```
!(a && b) || !b && !a
```

Exercice 2.4. Soit a et b deux variables. Ecrire en langage C l'expression logique « *non (a et b) ou non b et non a* ». Quel problème peut se poser lors de l'évaluation de l'expression ?

```
!(a && b) || !b && !a
```

Le problème pouvant se poser est l'ordre des opérateurs à évaluer. Plusieurs interprétations sont possibles :

!(a && b) || (!b && !a)

(!(a && b) || !b) && !a

!(a && b) || !(b && !a)

Exercice 2.4. Soit a et b deux variables. Ecrire en langage C l'expression logique « *non (a et b) ou non b et non a* ». Quel problème peut se poser lors de l'évaluation de l'expression ?

```
!(a && b) || !b && !a
```

Le problème pouvant se poser est l'ordre des opérateurs à évaluer. Plusieurs interprétations sont possibles :

!(a && b) || (!b && !a)

(!(a && b) || !b) && !a

!(a && b) || !(b && !a)

Non déterminisme de l'interprétation

3. Expressions relationnelles

Exercice 3.1. Citer les six opérateurs relationnels du langage C et expliciter leur sémantique et leurs valeurs de retour.

Exercice 3.1. Citer les six opérateurs relationnels du langage C et expliciter leur sémantique et leurs valeurs de retour.

Opérateur	Sémantique	Valeur de retour
a == b		
a > b		
a < b		
a != b		
a >= b		
a <=b		

Exercice 3.1. Citer les six opérateurs relationnels du langage C et expliciter leur sémantique et leurs valeurs de retour.

Opérateur	Sémantique	Valeur de retour
a == b	VRAI si la valeur de a est **égale** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a > b		
a < b		
a != b		
a >= b		
a <=b		

Exercice 3.1. Citer les six opérateurs relationnels du langage C et expliciter leur sémantique et leurs valeurs de retour.

Opérateur	Sémantique	Valeur de retour
a == b	VRAI si la valeur de a est **égale** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a > b	VRAI si la valeur de a est **strictement supérieure** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a < b		
a != b		
a >= b		
a <=b		

Exercice 3.1. Citer les six opérateurs relationnels du langage C et expliciter leur sémantique et leurs valeurs de retour.

Opérateur	Sémantique	Valeur de retour
a == b	VRAI si la valeur de a est **égale** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a > b	VRAI si la valeur de a est **strictement supérieure** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a < b	VRAI si la valeur de a est **strictement inférieure** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a != b		
a >= b		
a <=b		

Exercice 3.1. Citer les six opérateurs relationnels du langage C et expliciter leur sémantique et leurs valeurs de retour.

Opérateur	Sémantique	Valeur de retour
a == b	VRAI si la valeur de a est **égale** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a > b	VRAI si la valeur de a est **strictement supérieure** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a < b	VRAI si la valeur de a est **strictement inférieure** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a != b	VRAI si la valeur de a est **différente** de la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a >= b		
a <=b		

Exercice 3.1. Citer les six opérateurs relationnels du langage C et expliciter leur sémantique et leurs valeurs de retour.

Opérateur	Sémantique	Valeur de retour
a == b	VRAI si la valeur de a est **égale** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a > b	VRAI si la valeur de a est **strictement supérieure** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a < b	VRAI si la valeur de a est **strictement inférieure** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a != b	VRAI si la valeur de a est **différente** de la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a >= b	VRAI si la valeur de a est **supérieure ou égale** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a <=b		

Exercice 3.1. Citer les six opérateurs relationnels du langage C et expliciter leur sémantique et leurs valeurs de retour.

Opérateur	Sémantique	Valeur de retour
a == b	VRAI si la valeur de a est **égale** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a > b	VRAI si la valeur de a est **strictement supérieure** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a < b	VRAI si la valeur de a est **strictement inférieure** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a != b	VRAI si la valeur de a est **différente** de la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a >= b	VRAI si la valeur de a est **supérieure ou égale** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX
a <=b	VRAI si la valeur de a est **inférieure ou égale** à la valeur de b, FAUX sinon	1 si VRAI, 0 si FAUX

Exercice 3.2. Soit trois variables numériques a, b et c. Ecrire l'expression « *a est différent de b et b strictement supérieur à c ou a inférieur ou égal à c* » en langage C. Quel problème peut se poser lors de l'évaluation de l'expression ?

Exercice 3.2. Soit trois variables numériques a, b et c. Ecrire l'expression « *a est différent de b et b strictement supérieur à c ou a inférieur ou égal à c* » en langage C. Quel problème peut se poser lors de l'évaluation de l'expression ?

L'expression en langage C est :

```
a != b && b > c || a <= c
```

Exercice 3.2. Soit trois variables numériques a, b et c. Ecrire l'expression « *a est différent de b et b strictement supérieur à c ou a inférieur ou égal à c* » en langage C. Quel problème peut se poser lors de l'évaluation de l'expression ?

L'expression en langage C est :

```
a != b && b > c || a <= c
```

Le problème pouvant se poser est l'ordre des opérateurs à évaluer. Plusieurs interprétations sont possibles :

a != (b && b > c || a <= c)

a != b && (b > c || a <= c)

Exercice 3.2. Soit trois variables numériques a, b et c. Ecrire l'expression « *a est différent de b et b strictement supérieur à c ou a inférieur ou égal à c* » en langage C. Quel problème peut se poser lors de l'évaluation de l'expression ?

L'expression en langage C est :

```
a != b && b > c || a <= c
```

Le problème pouvant se poser est l'ordre des opérateurs à évaluer. Plusieurs interprétations sont possibles :

a != (b && b > c || a <= c)

a != b && (b > c || a <= c)

Non déterminisme de l'interprétation

4. Priorité des opérateurs

Exercice 4.1. Comment le langage C rend-il déterministe l'évaluation d'expression contenant plusieurs opérateurs ?

Déterminisme = 1 seule interprétation possible pour une expression.

Exercice 4.1. Comment le langage C rend-il déterministe l'évaluation d'expression contenant plusieurs opérateurs ?

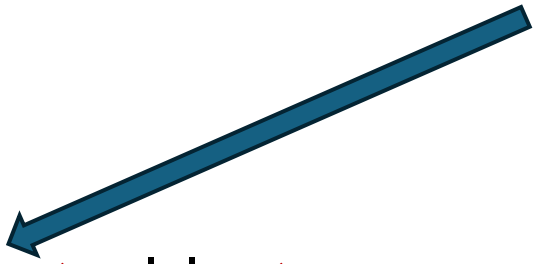
Déterminisme = 1 seule interprétation possible pour une expression.

```
a && b || !b && !a
```

Exercice 4.1. Comment le langage C rend-il déterministe l'évaluation d'expression contenant plusieurs opérateurs ?

Déterminisme = 1 seule interprétation possible pour une expression.

```
a && b || !b && !a
```

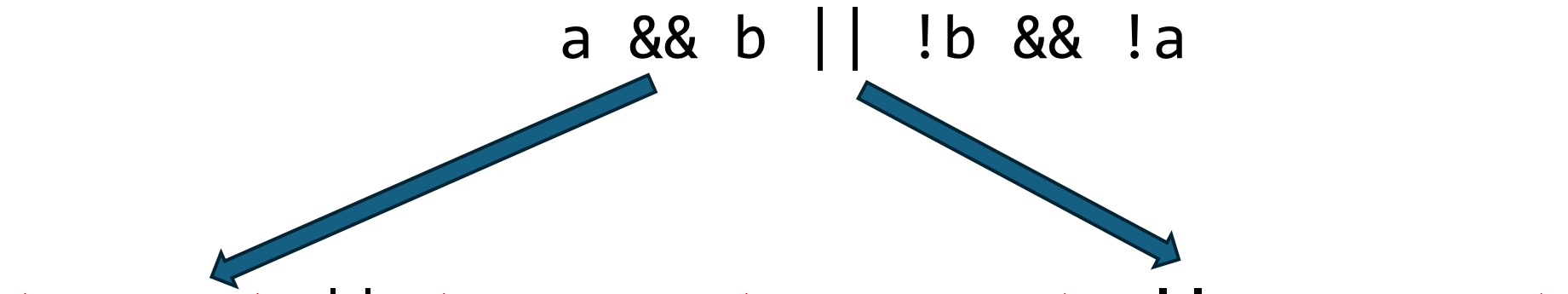


```
(a && b) || (!b && !a)
```

&& prioritaire

Exercice 4.1. Comment le langage C rend-il déterministe l'évaluation d'expression contenant plusieurs opérateurs ?

Déterminisme = 1 seule interprétation possible pour une expression.



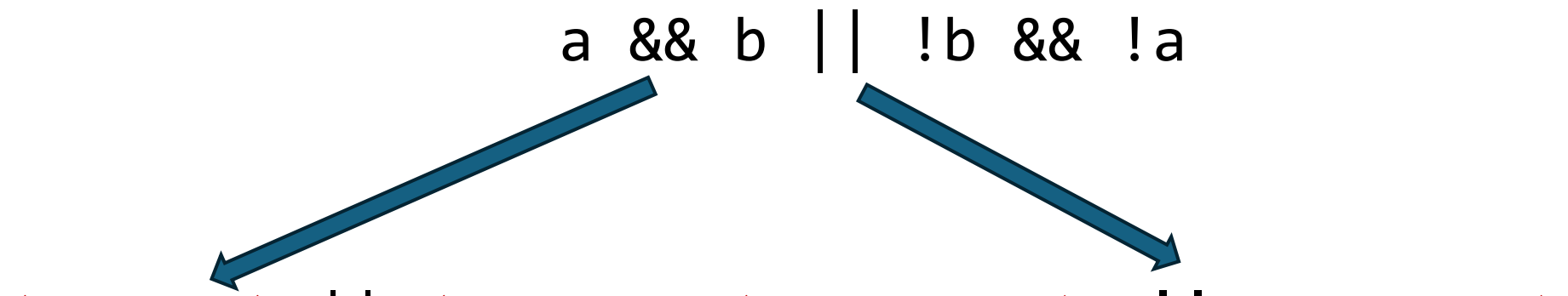
&& prioritaire

|| prioritaire

Dépends de la priorité

Exercice 4.1. Comment le langage C rend-il déterministe l'évaluation d'expression contenant plusieurs opérateurs ?

Déterminisme = 1 seule interprétation possible pour une expression.

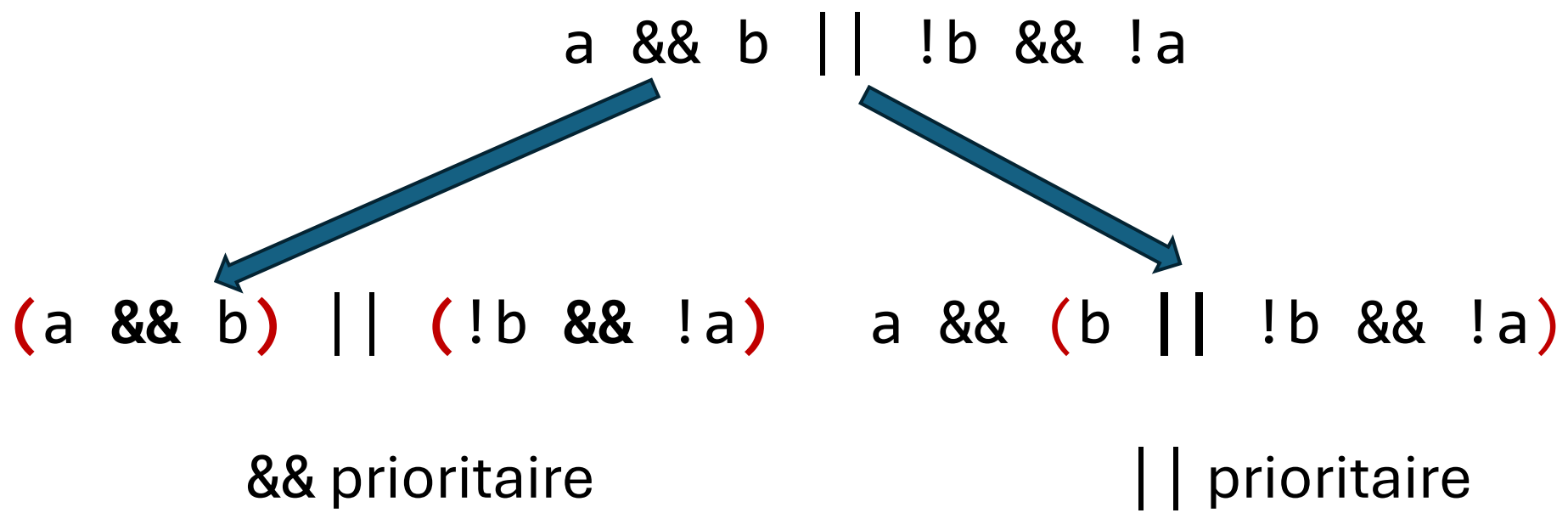


&& prioritaire

|| prioritaire

Exercice 4.1. Comment le langage C rend-il déterministe l'évaluation d'expression contenant plusieurs opérateurs ?

Déterminisme = 1 seule interprétation possible pour une expression.



Le langage C rend déterministe l'évaluation d'une expression contenant plusieurs opérateurs en **donnant des priorités** de sorte qu'il n'existe qu'une lecture possible de l'expression.

Exercice 4.1. Comment le langage C rend-il déterministe l'évaluation d'expression contenant plusieurs opérateurs ?

Opérateurs de même priorité:

```
a && b && c && d
```

Exercice 4.1. Comment le langage C rend-il déterministe l'évaluation d'expression contenant plusieurs opérateurs ?

Opérateurs de même priorité:

```
a && b && c && d
```

Evaluation de gauche à droite:

```
((a && b) && c) && d
     1
```

Exercice 4.1. Comment le langage C rend-il déterministe l'évaluation d'expression contenant plusieurs opérateurs ?

Opérateurs de même priorité:

```
a && b && c && d
```

Evaluation de gauche à droite:

```
((a && b) && c) && d
     1
```

```
((a && b) && c) && d
          2
```

Exercice 4.1. Comment le langage C rend-il déterministe l'évaluation d'expression contenant plusieurs opérateurs ?

Opérateurs de même priorité:

```
a && b && c && d
```

Evaluation de gauche à droite:

```
((a && b) && c) && d
   1
```

```
((a && b) && c) && d
        2
```

```
((a && b) && c) && d
             3
```

Exercice 4.1. Comment le langage C rend-il déterministe l'évaluation d'expression contenant plusieurs opérateurs ?

Le langage C rend déterministe l'évaluation d'une expression contenant plusieurs opérateurs en **donnant des priorités** de sorte qu'il n'existe qu'une lecture possible de l'expression.

Si plusieurs opérateurs de **même priorité** se suivent, ils sont **appliqués de gauche à droite**.

Exercice 4.2. Donner la table des priorités des opérateurs arithmétiques, logiques et relationnels du langage C.

Exercice 4.2. Donner la table des priorités des opérateurs arithmétiques, logiques et relationnels du langage C.

Opérateur	Associativité
`(), ++ (suffixe), -- (suffixe)`	De gauche à droite
`++ (préfixe), -- (préfixe)`	De droite à gauche
`!`	De droite à gauche
`*, /, %`	De gauche à droite
`+, -`	De gauche à droite
`>, <, >=, <=`	De gauche à droite
`==, !=`	De gauche à droite
`&&`	De gauche à droite
`||`	De gauche à droite
`=, +=, -=, *=, /=, %=`	De droite à gauche

Exercice 4.3. Soit l'expression suivante : ++a || (++b > ++c && (++d*++e))
Si a = 1, b = 1, c = 1, d = 1 et e = 1, quelles sont les valeurs des variables après l'exécution de cette instruction ? Que s'est-il passé durant l'exécution ?

Exercice 4.3. Soit l'expression suivante : ++a || (++b > ++c && (++d*++e))
Si a = 1, b = 1, c = 1, d = 1 et e = 1, quelles sont les valeurs des variables après l'exécution de cette instruction ? Que s'est-il passé durant l'exécution ?

```
++a || (++b > ++c && (++d*++e))
```

D'après les priorités, la première valeur évaluée est ++a qui retourne alors 2:

```
2 || (++b > ++c && (++d*++e))
```

Exercice 4.3. Soit l'expression suivante : ++a || (++b > ++c && (++d*++e))
Si a = 1, b = 1, c = 1, d = 1 et e = 1, quelles sont les valeurs des variables après l'exécution de cette instruction ? Que s'est-il passé durant l'exécution ?

```
++a || (++b > ++c && (++d*++e))
```

D'après les priorités, la première valeur évaluée est ++a qui retourne alors 2:

```
2 || (++b > ++c && (++d*++e))
```

Une valeur de 2 représentant VRAI, d'après la table de vérité du ||, l'expression || globale ne peut que valoir VRAI.

||	VRAI	FAUX
VRAI	1	1
FAUX	1	0

Exercice 4.3. Soit l'expression suivante : ++a || (++b > ++c && (++d*++e))
Si a = 1, b = 1, c = 1, d = 1 et e = 1, quelles sont les valeurs des variables après l'exécution de cette instruction ? Que s'est-il passé durant l'exécution ?

```
++a || (++b > ++c && (++d*++e))
```

D'après les priorités, la première valeur évaluée est ++a qui retourne alors 2:

Une valeur de 2 représentant VRAI, d'après la table de vérité du ||, l'expression || globale ne peut que valoir VRAI.

||	VRAI	FAUX
VRAI	1	1
FAUX	1	0

Il est inutile de calculer (++b > ++c && (++d*++e)) pour évaluer l'expression globale et ++b, ++c, ++d et ++e ne sont pas exécutés.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont :

- x = 1, y = 1 et z = 1
- x = -1, y = 1 et z = 3
- x = -1, y = 1 et z = 0

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.

`++x || (++y > z && (y*++z))`

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.

`++x || (++y > z && (y*++z))`

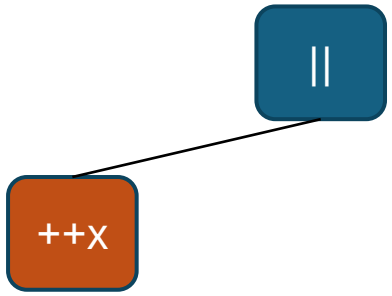
On cherche l'opérateur le moins prioritaire

Opérateur
(), ++x, --x
++x, --x
!
*, /, %
+, -
>, <, >=, <=
==, !=
&&
||
=, +=, -=, *=, /=, %=

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.



`++x || (++y > z && (y*++z))`

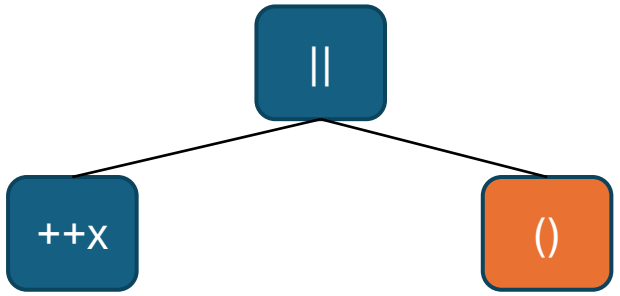
On place la sous-expression de gauche

Opérateur
(), ++x, --x
++x, --x
!
*, /, %
+, -
>, <, >=, <=
==, !=
&&
||
=, +=, -=, *=, /=, %=

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.



`++x || (++y > z && (y*++z))`

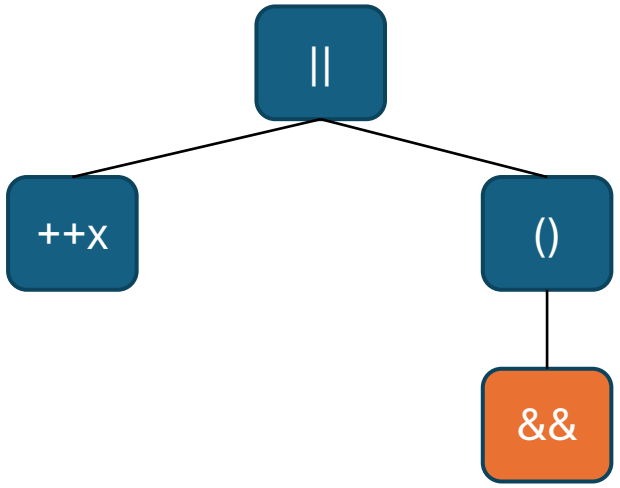
On place la sous-expression de droite

Opérateur
(), ++x, --x
++x, --x
!
*, /, %
+, -
>, <, >=, <=
==, !=
&&
||
=, +=, -=, *=, /=, %=

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.



`++x || (++y > z && (y*++z))`

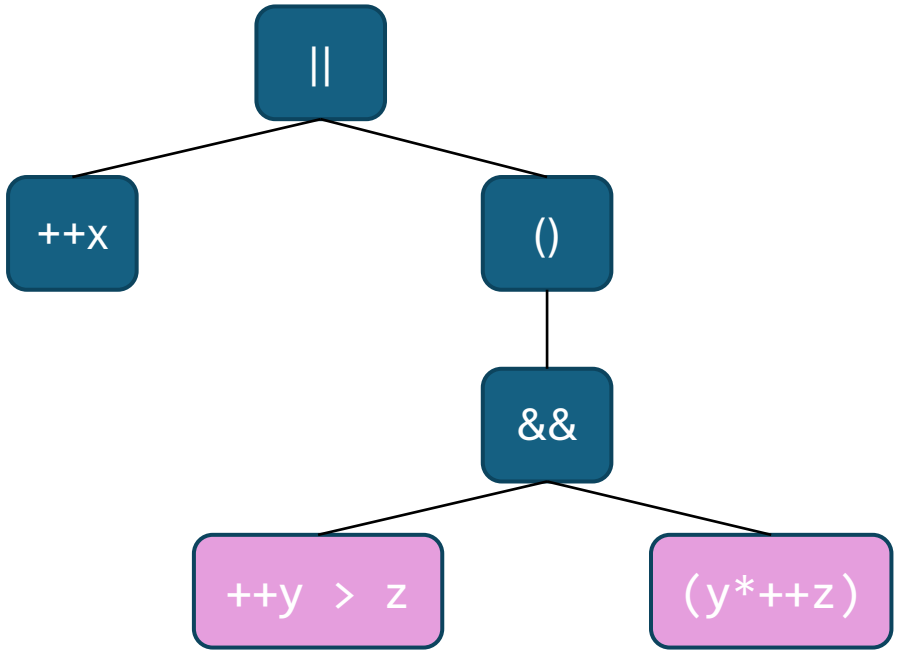
On cherche l'opérateur le moins prioritaire

Opérateur
(), ++x, --x
++x, --x
!
*, /, %
+, -
>, <, >=, <=
==, !=
&&
||
=, +=, -=, *=, /=, %=

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.



`++x || (++y > z && (y*++z))`

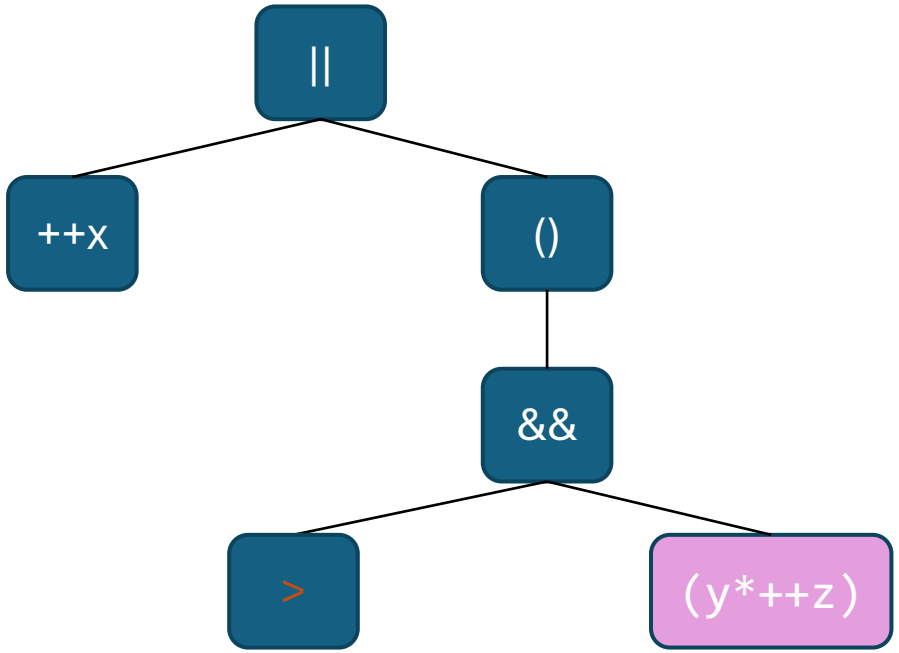
Opérateur
(), ++x, --x
++x, --x
!
*, /, %
+, -
>, <, >=, <=
==, !=
&&
||
=, +=, -=, *=, /=, %=

On découpe l'expression suivant l'opérateur choisit.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.



`++x || (++y > z && (y*++z))`

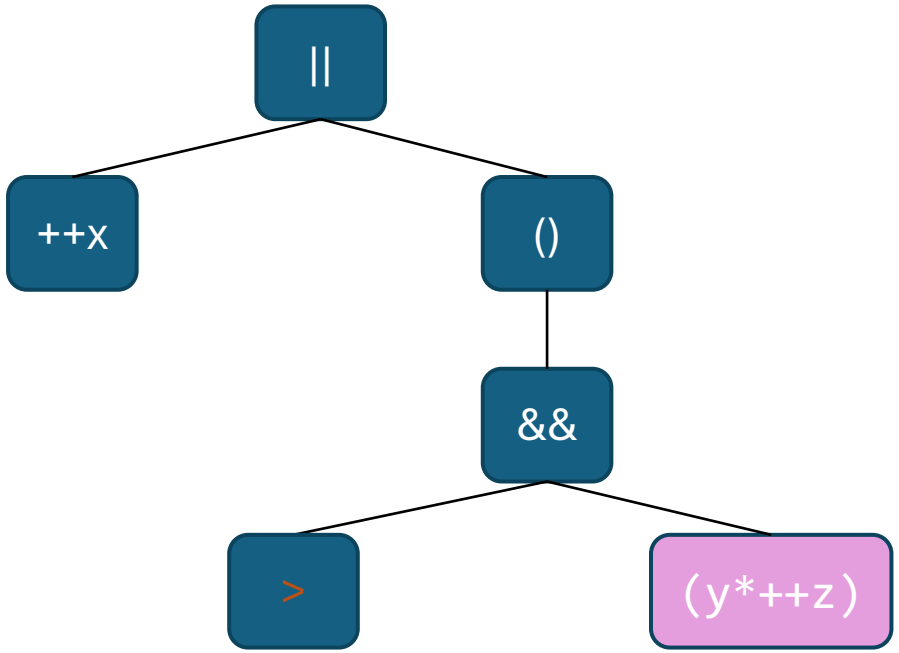
Dans la sous-expression gauche, on choisit l'opérateur le moins prioritaire.

Opérateur
(), ++x, --x
++x, --x
!
*, /, %
+, -
>, <, >=, <=
==, !=
&&
||
=, +=, -=, *=, /=, %=

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.



`++x || (++y > z && (y*++z))`

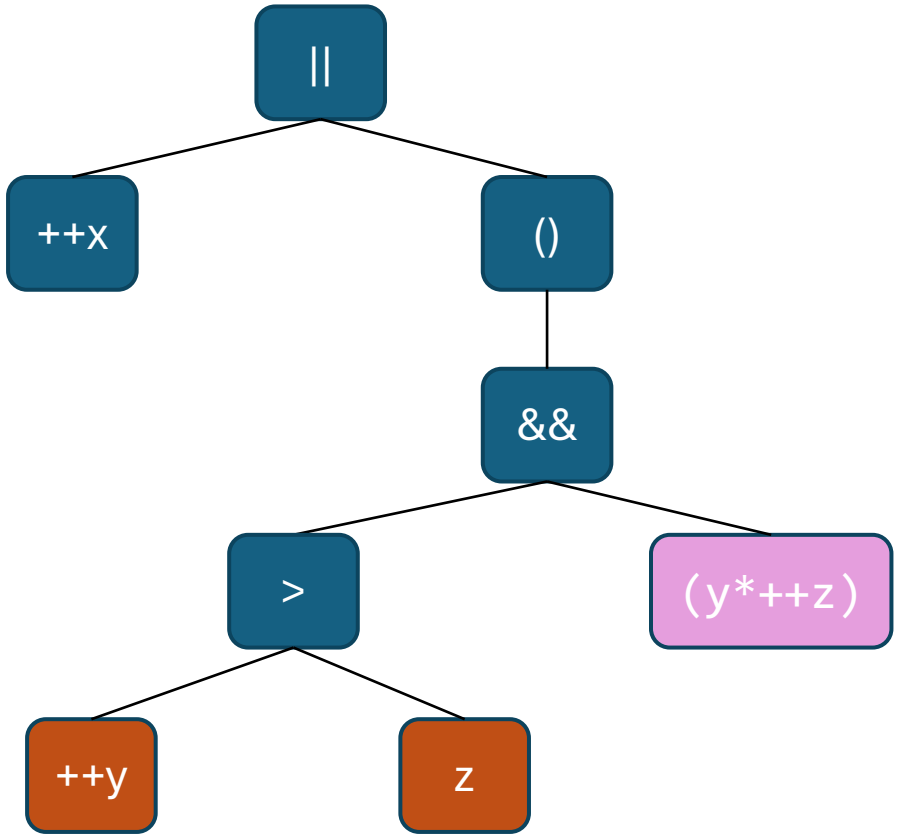
Opérateur
(), ++x, --x
++x, --x
!
*, /, %
+, -
>, <, >=, <=
==, !=
&&
||
=, +=, -=, *=, /=, %=

Dans la sous-expression gauche, on choisit l'opérateur le moins prioritaire.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.



On rattache les opérandes.

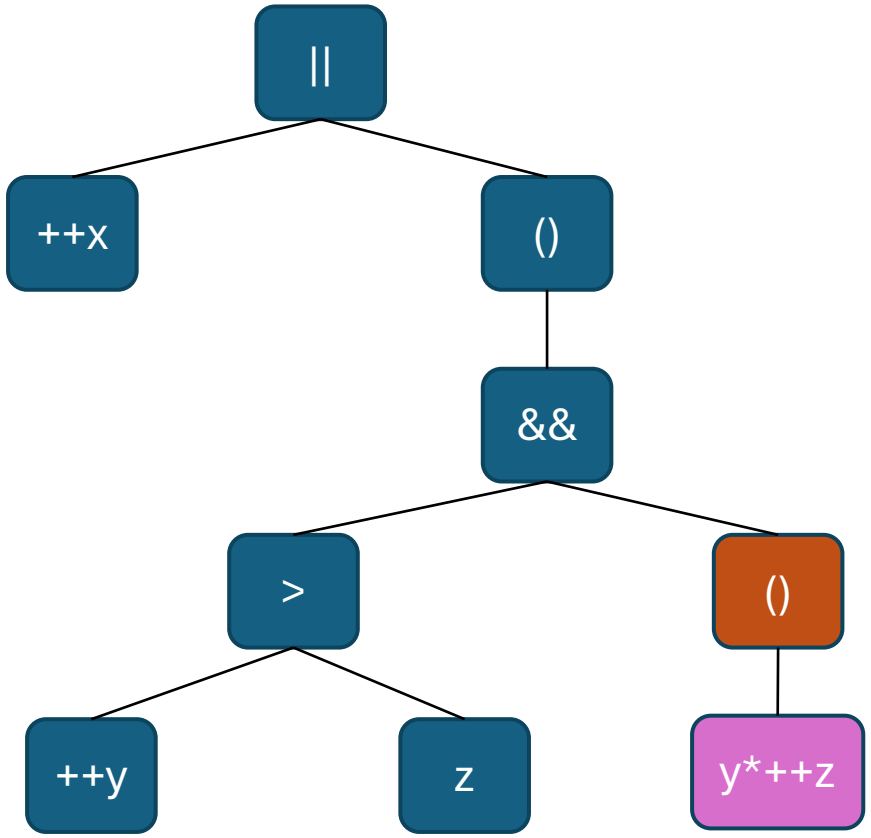
`++x || (++y > z && (y*++z))`

Opérateur
(), ++x, --x
++x, --x
!
*, /, %
+, -
>, <, >=, <=
==, !=
&&
||
=, +=, -=, *=, /=, %=

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.



`++x || (++y > z && (y*++z))`

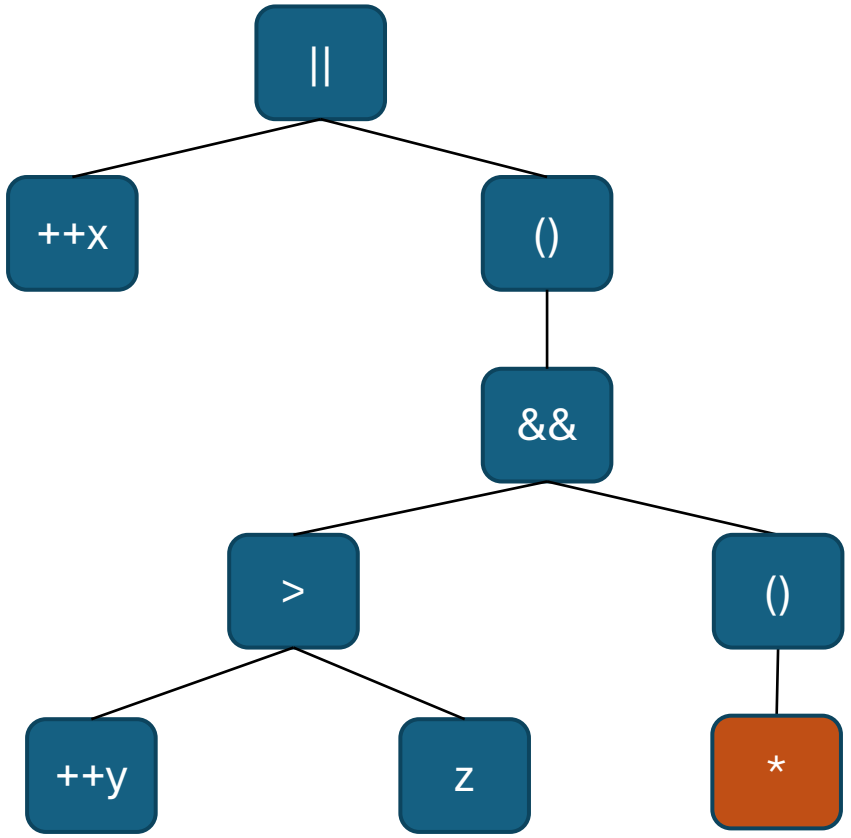
Opérateur
(), ++x, --x
++x, --x
!
*, /, %
+, -
>, <, >=, <=
==, !=
&&
||
=, +=, -=, *=, /=, %=

On cherche l'opérateur le moins prioritaire

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.



`++x || (++y > z && (y*++z))`

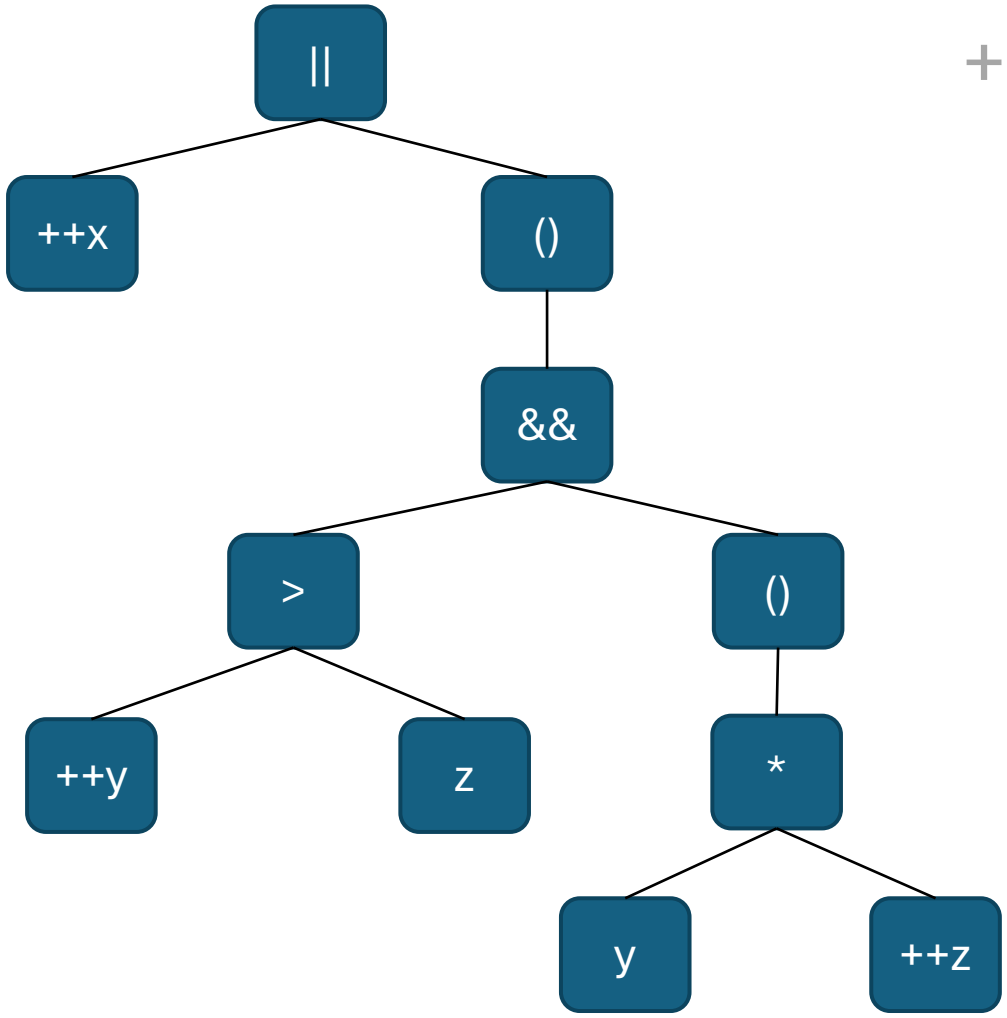
Opérateur
(), ++x, --x
++x, --x
!
*, /, %
+, -
>, <, >=, <=
==, !=
&&
||
=, +=, -=, *=, /=, %=

On cherche l'opérateur le moins prioritaire

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.



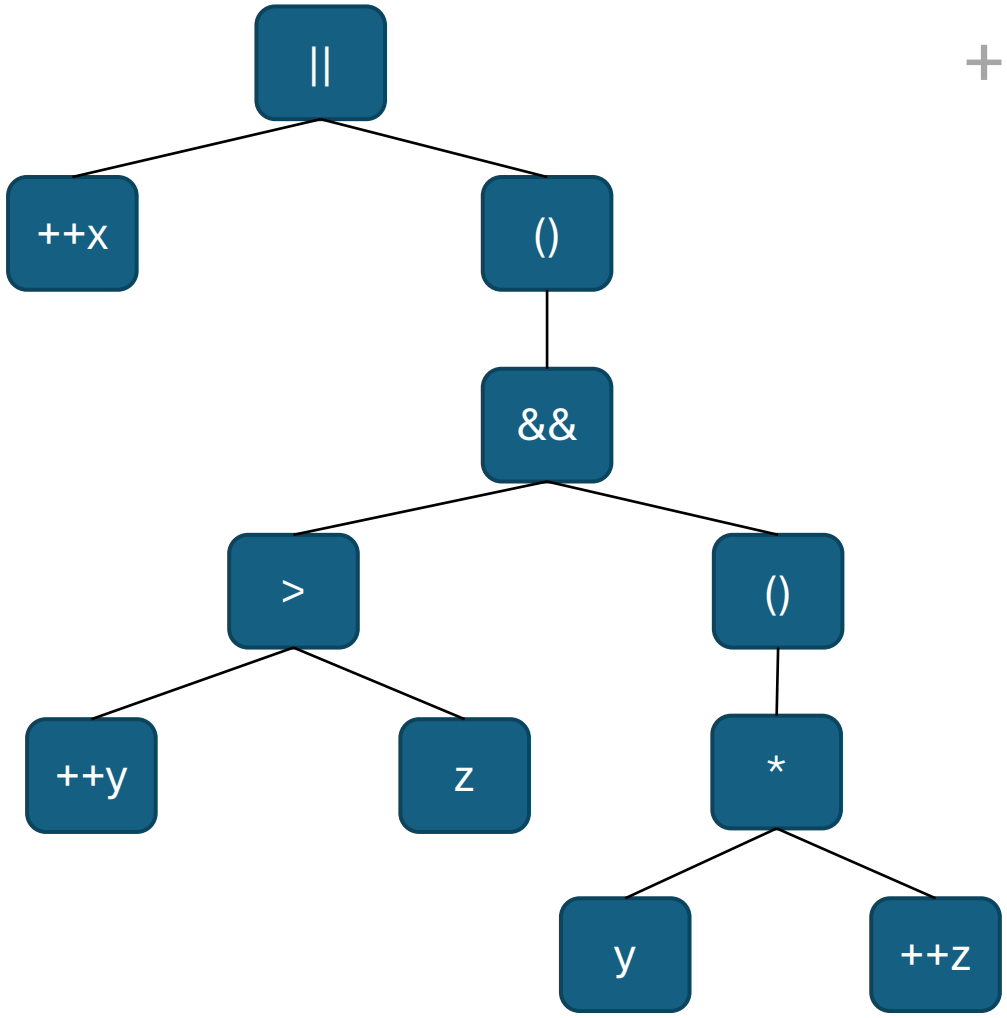
`++x || (++y > z && (y*++z))`

Opérateur
(), ++x, --x
++x, --x
!
*, /, %
+, -
>, <, >=, <=
==, !=
&&
||
=, +=, -=, *=, /=, %=

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Ecrire l'arbre d'exécution de l'expression.

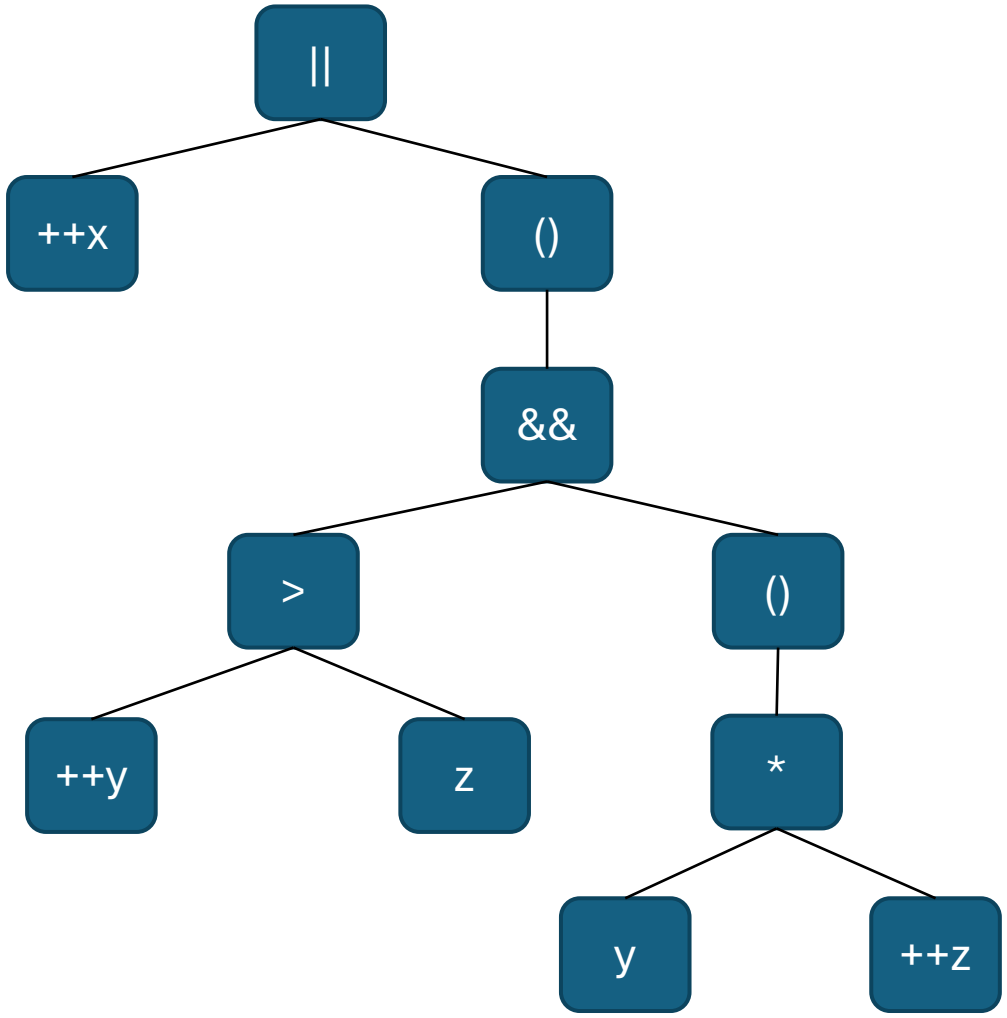


`++x || (++y > z && (y*++z))`

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

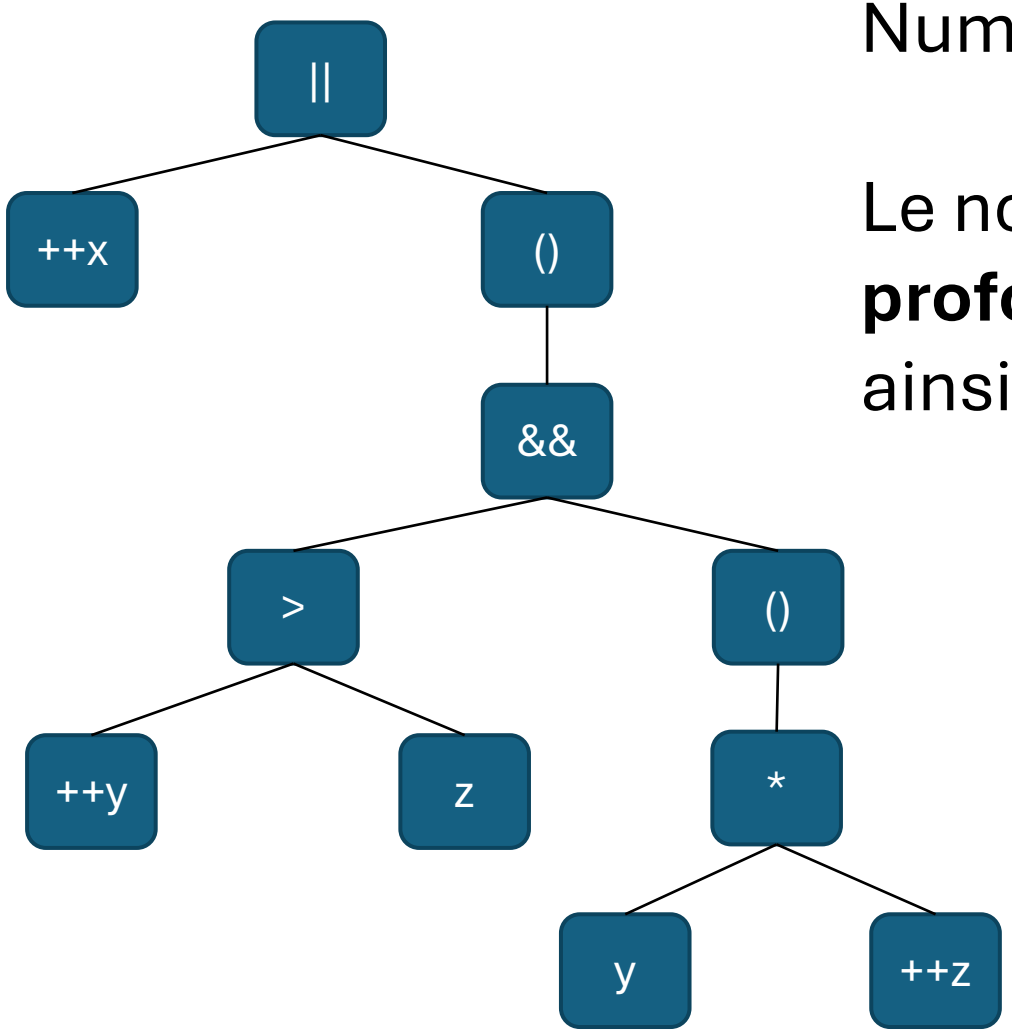
Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



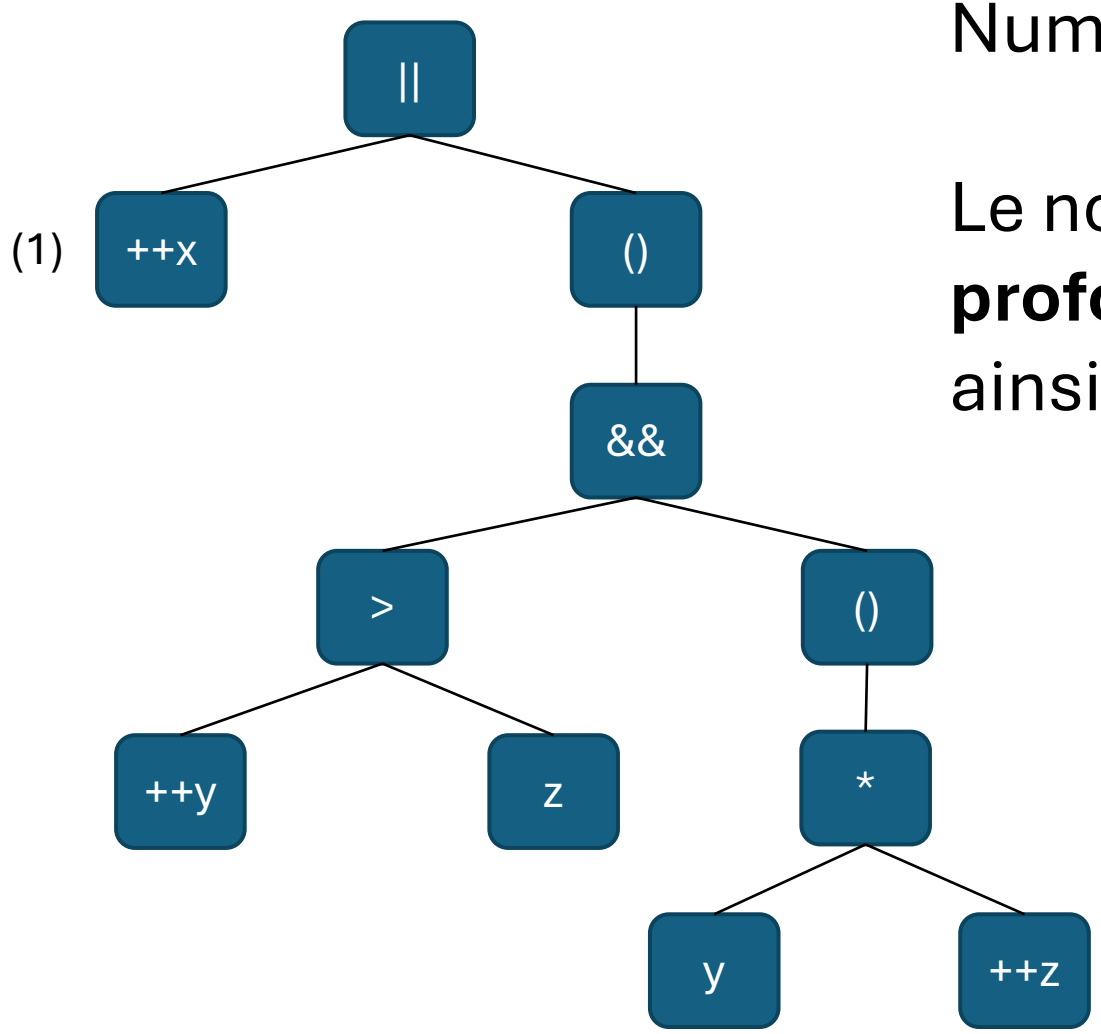
Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



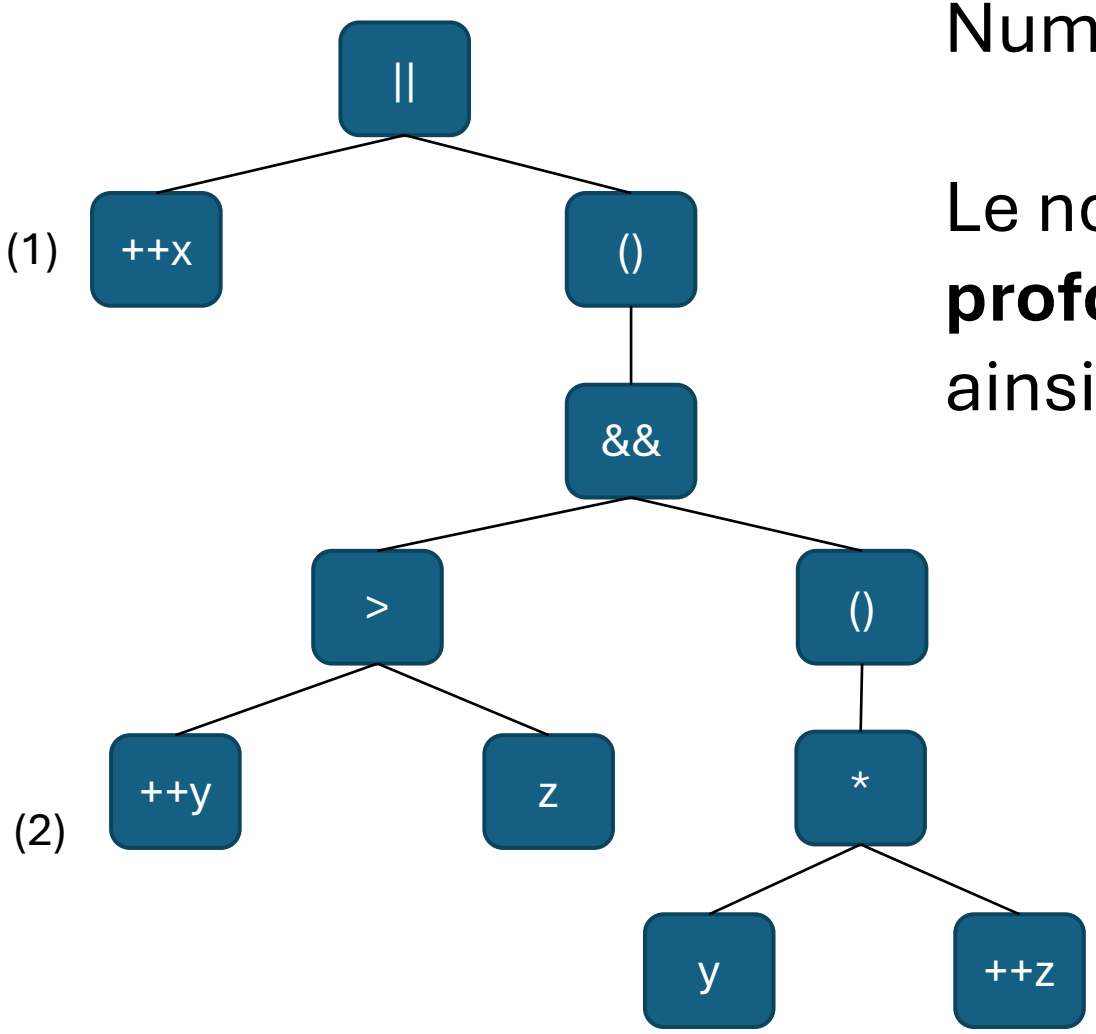
Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



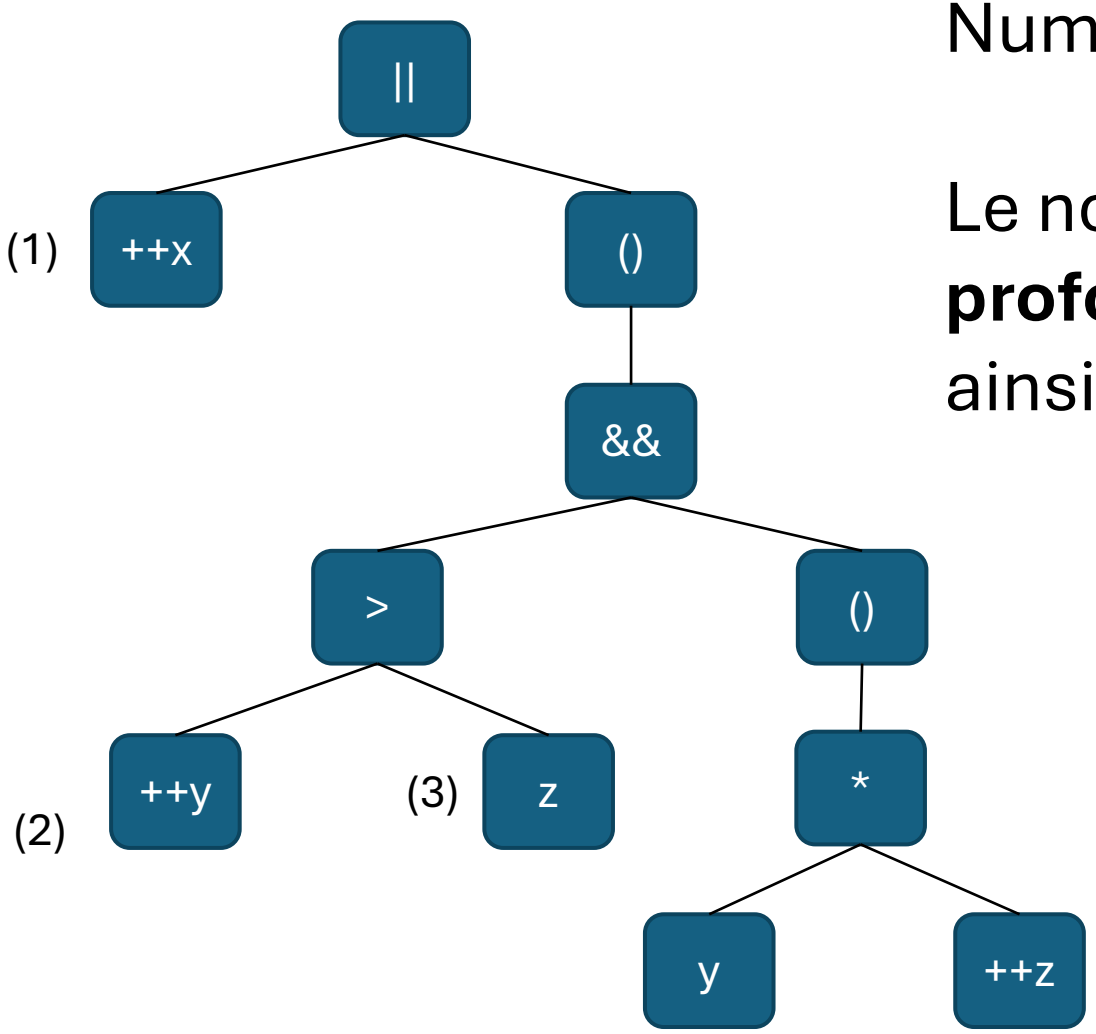
Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



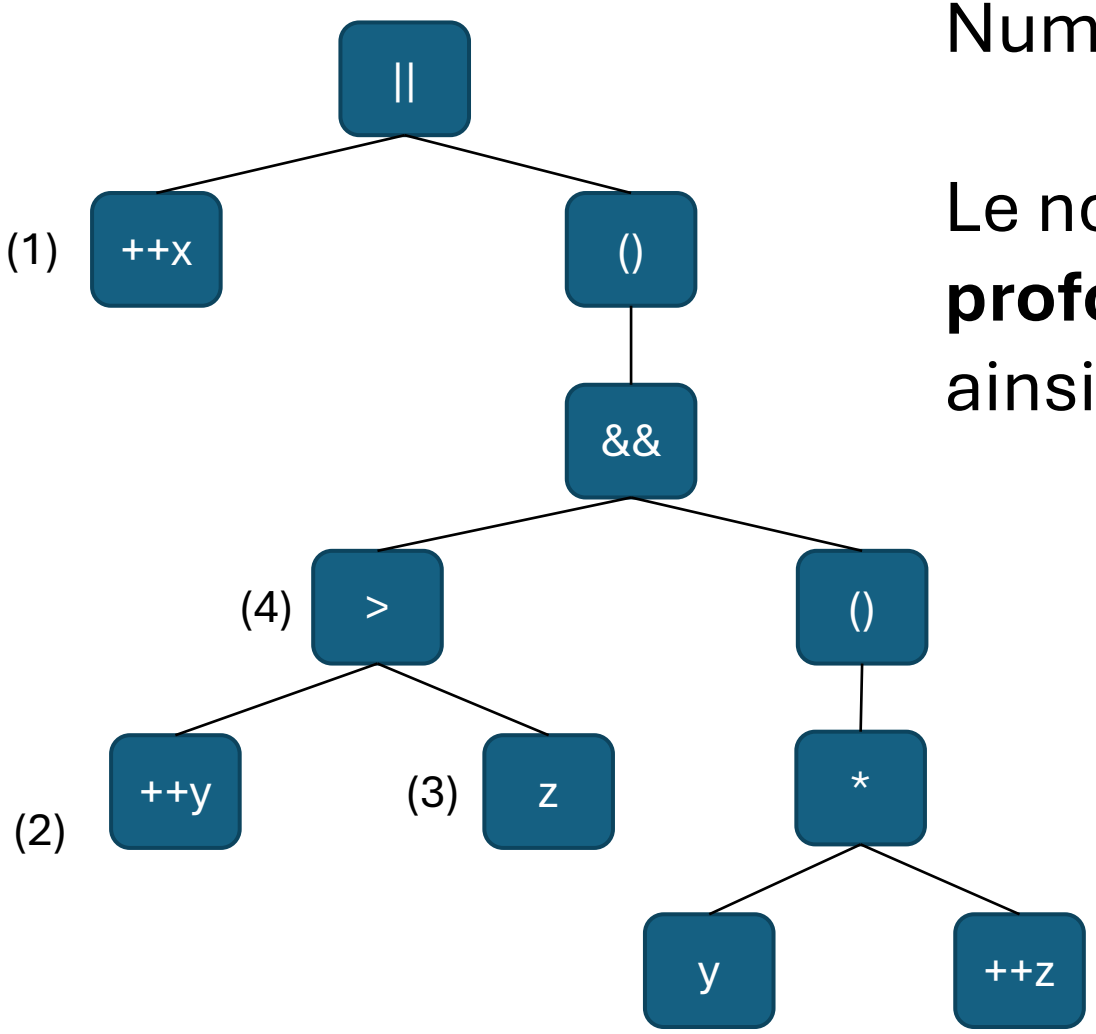
Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



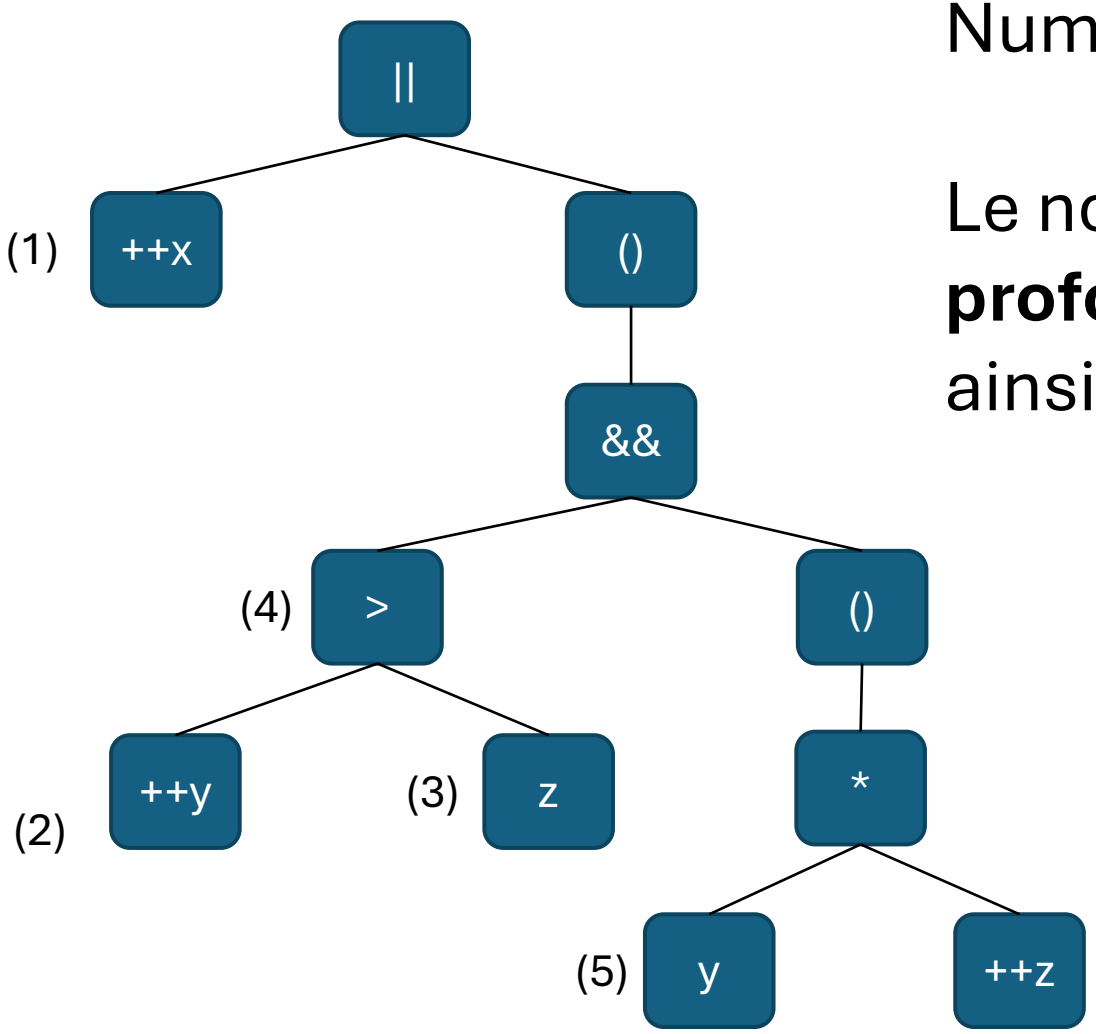
Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



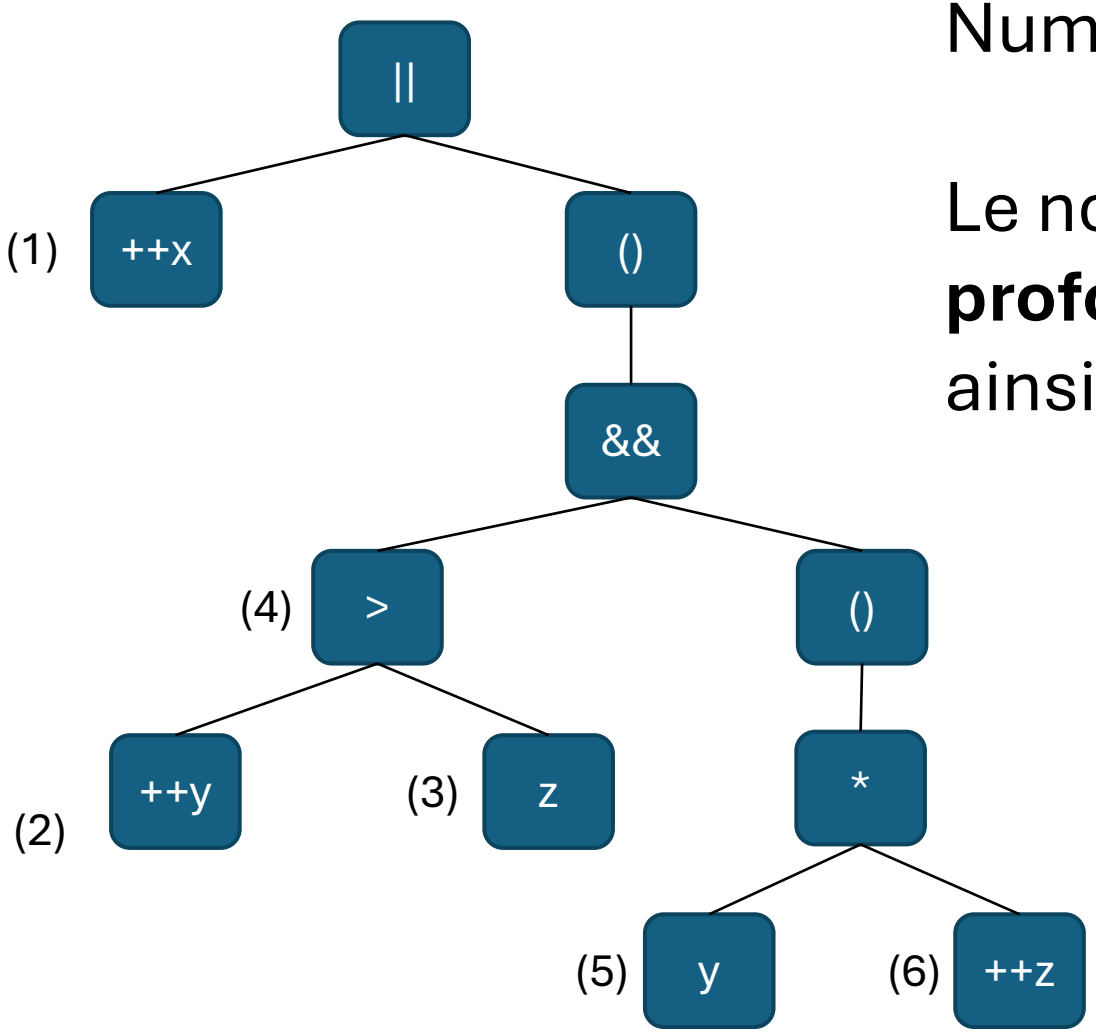
Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



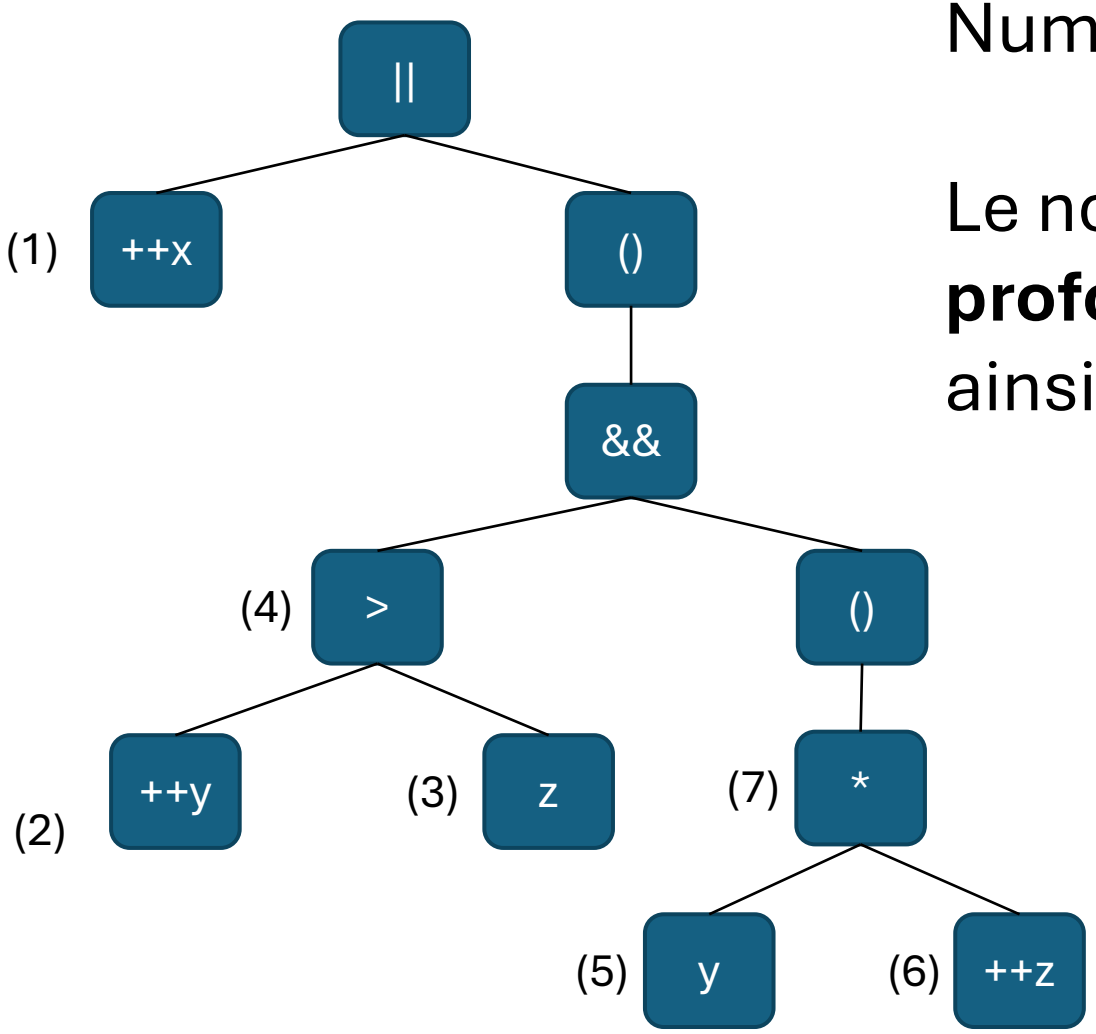
Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



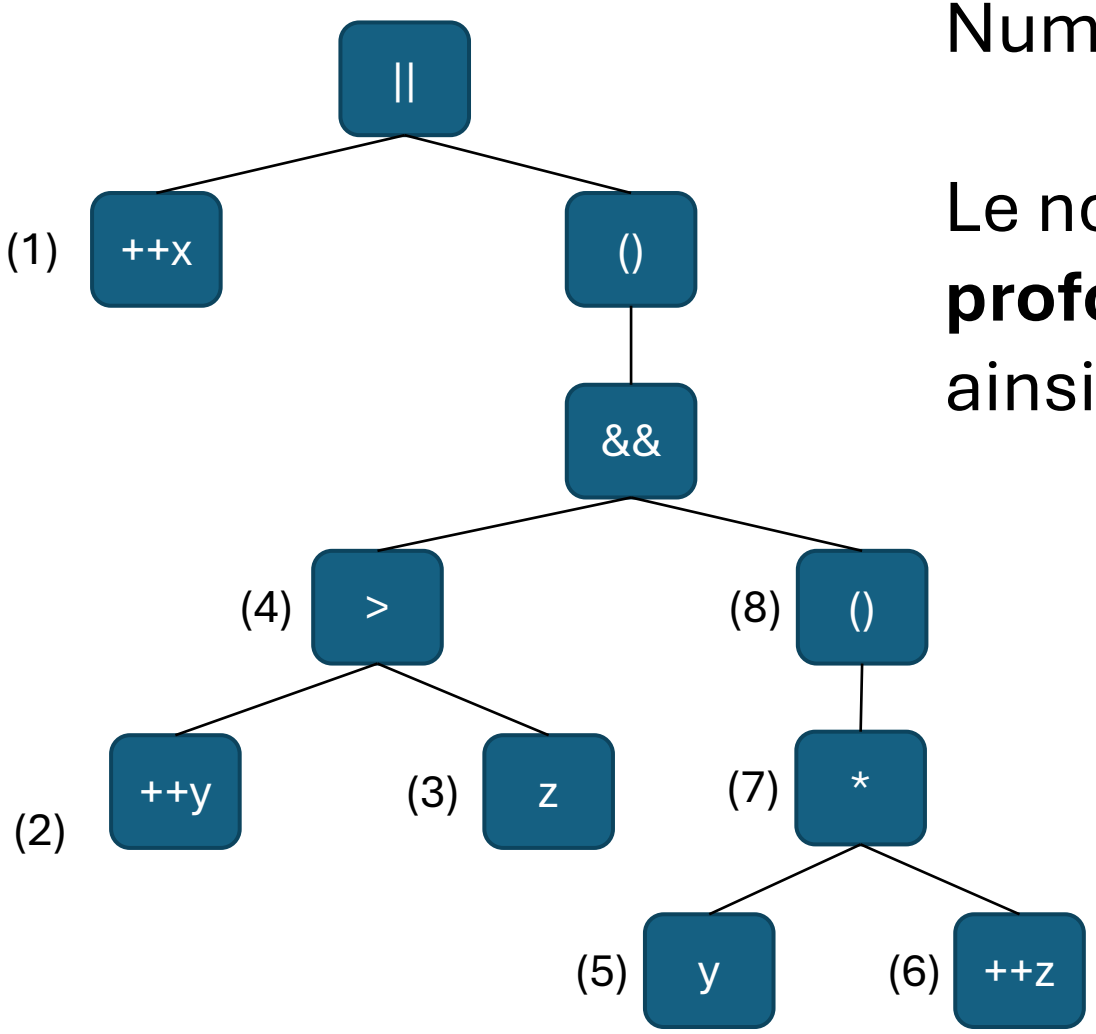
Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



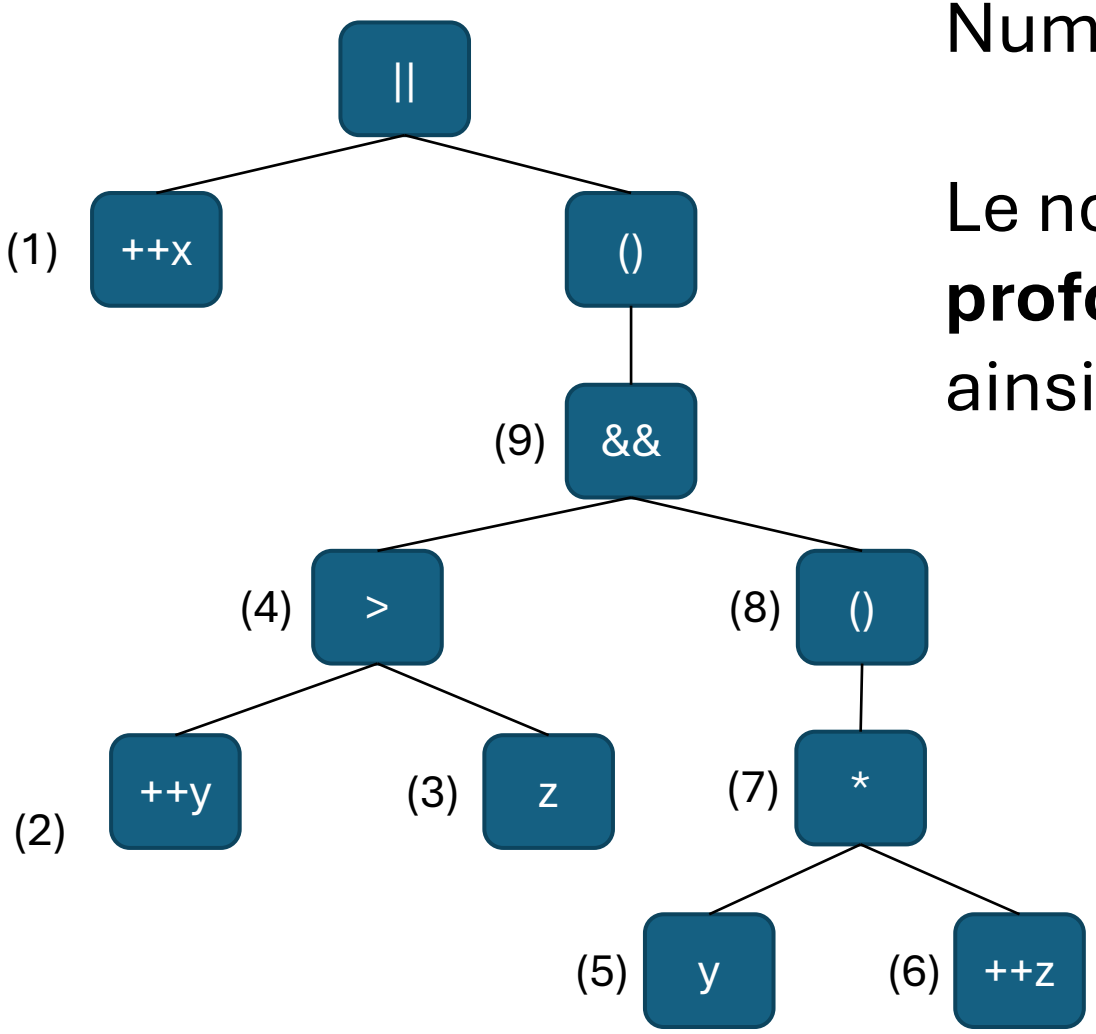
Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



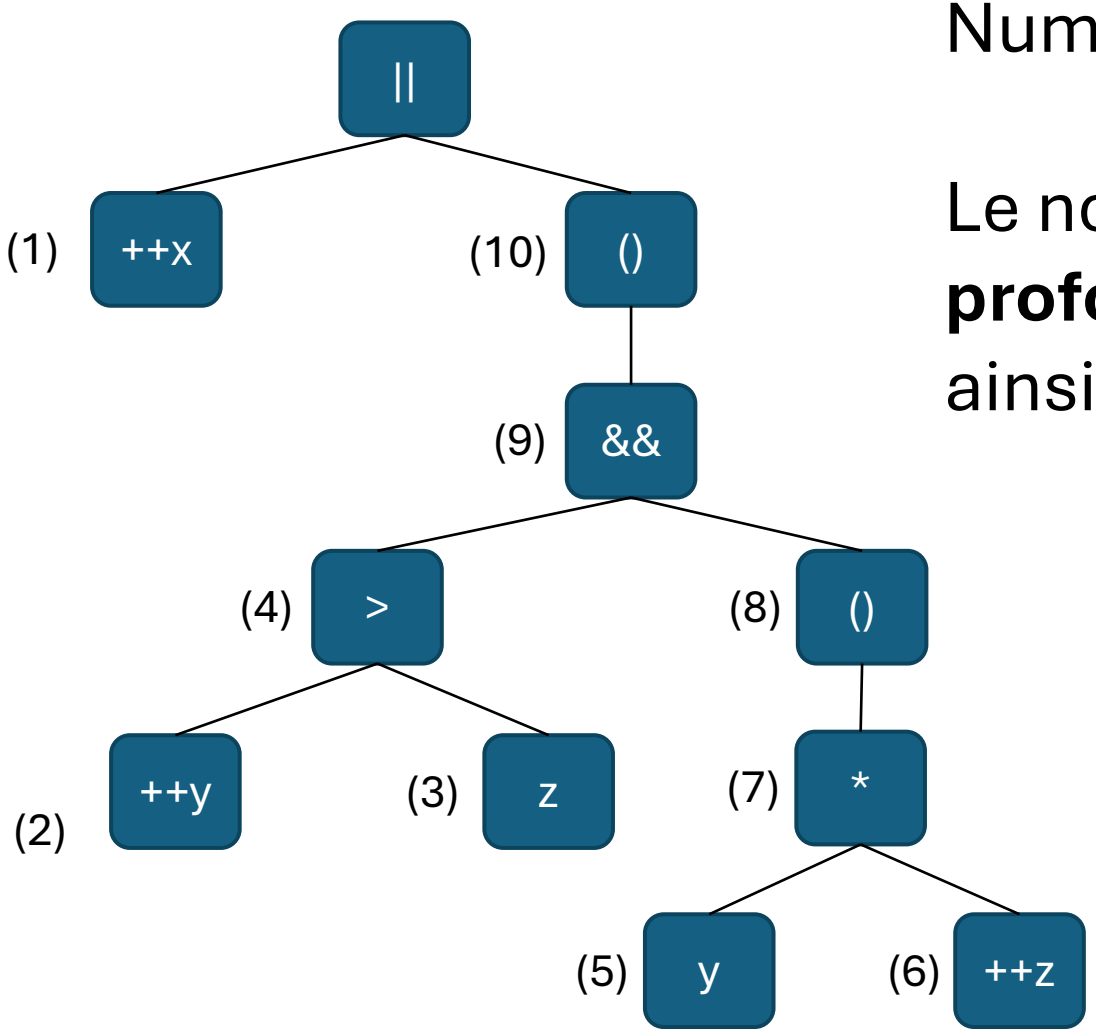
Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



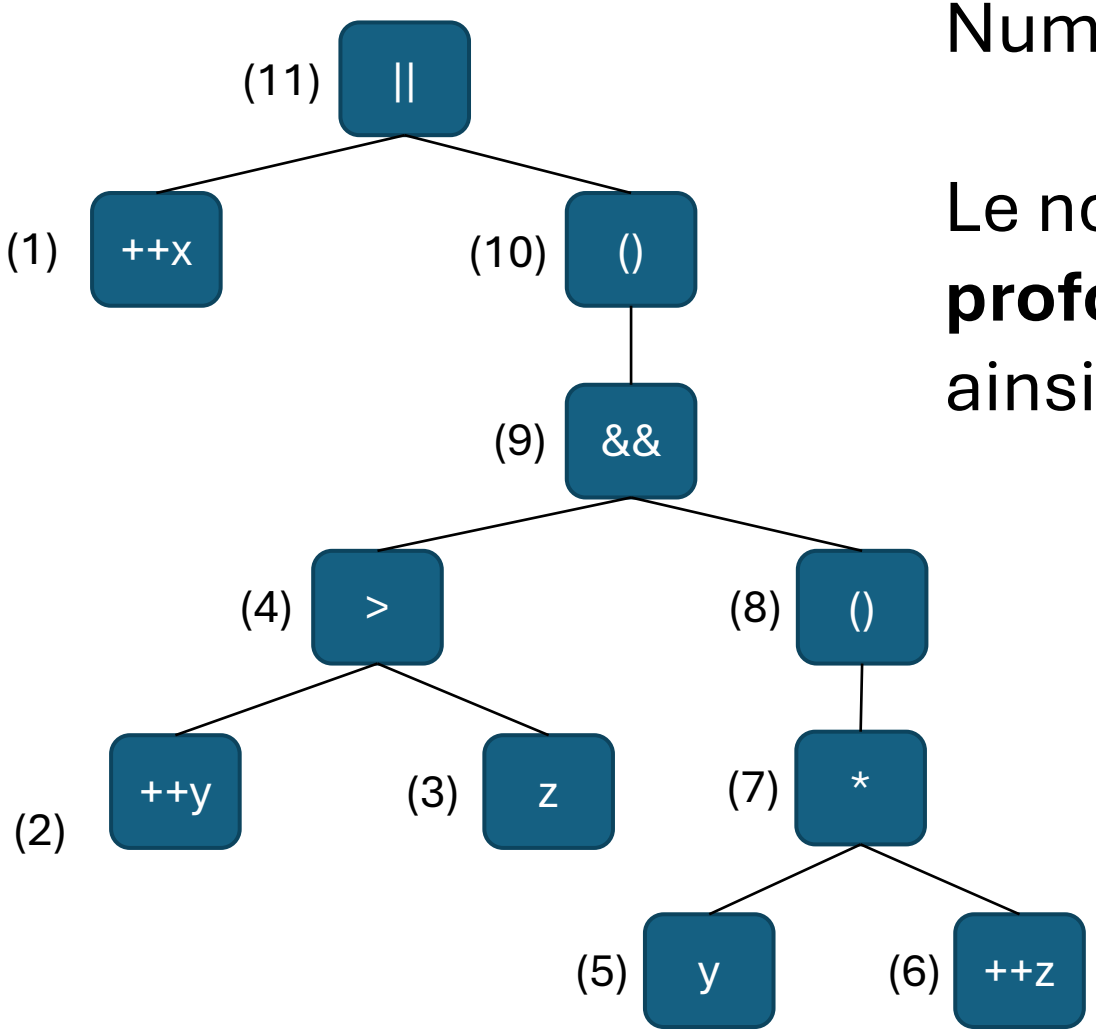
Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Numéroter les nœuds de l'arbre dans l'ordre de l'évaluation de l'instruction.



Numérotation selon un **parcours pré-fixé**:

Le nœud le **plus à gauche et le plus en profondeur** est numéroté **en premier** et ainsi de suite.

Exercice 4.4.

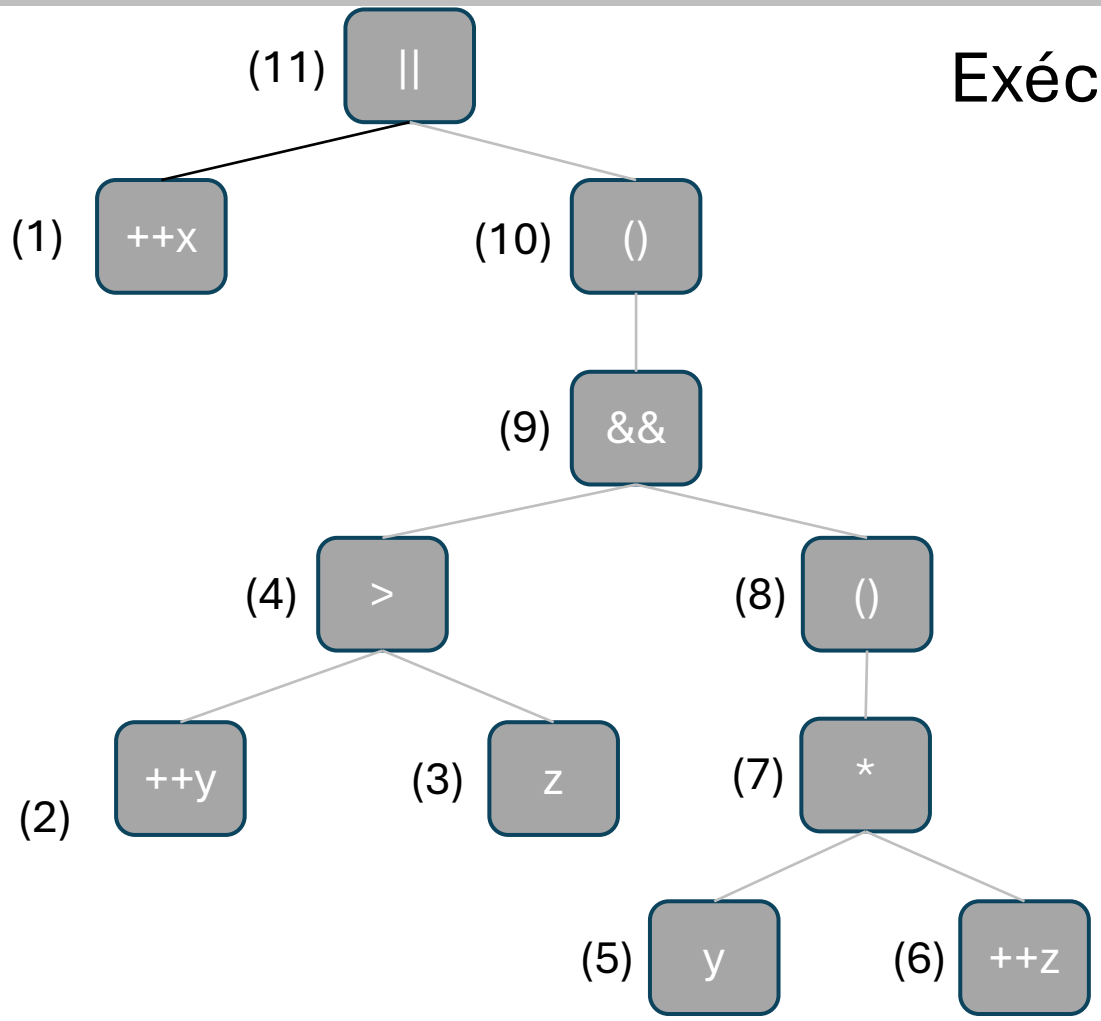
Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = 1, y = 1 et z = 1

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = 1, y = 1 et z = 1

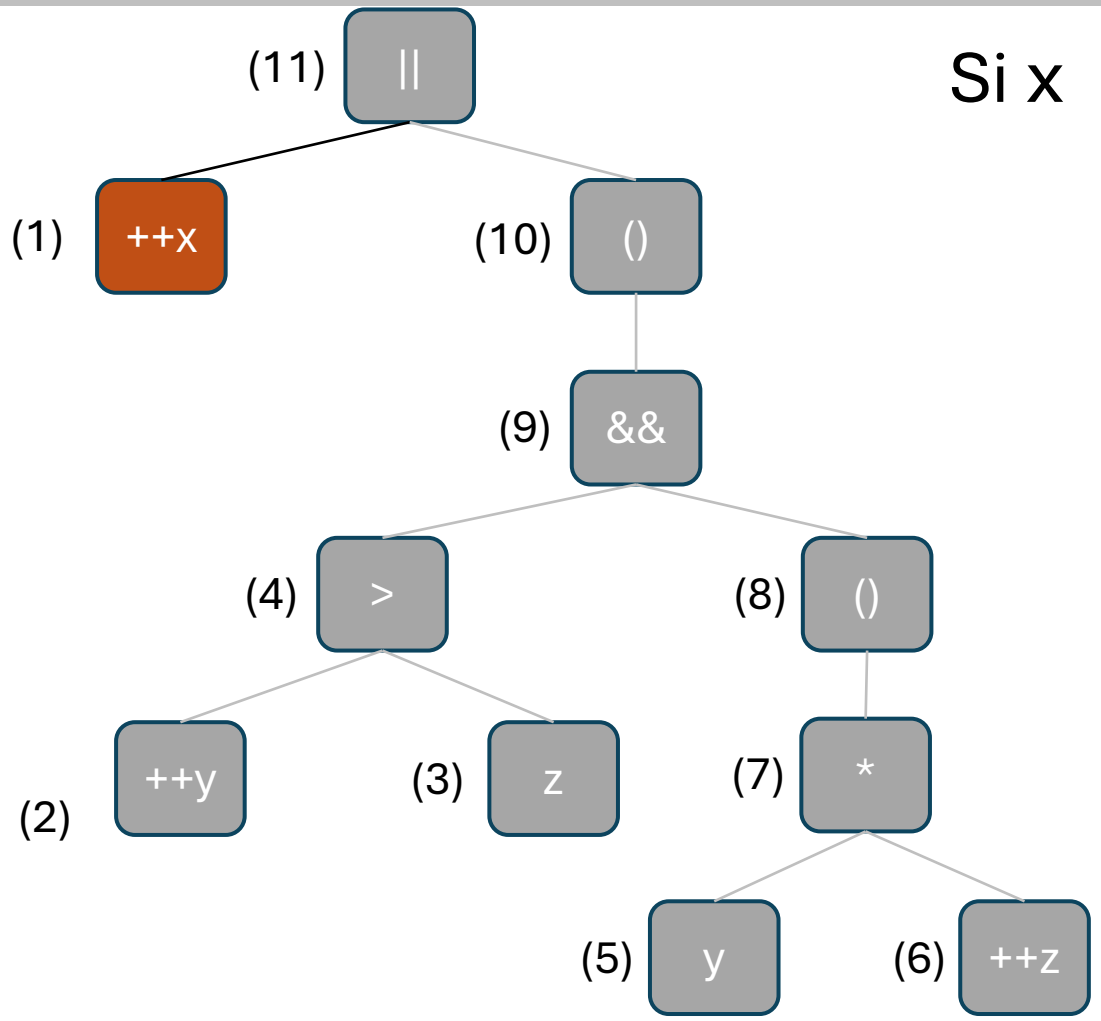


Exécution par parcours de l'arbre

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = 1, y = 1 et z = 1

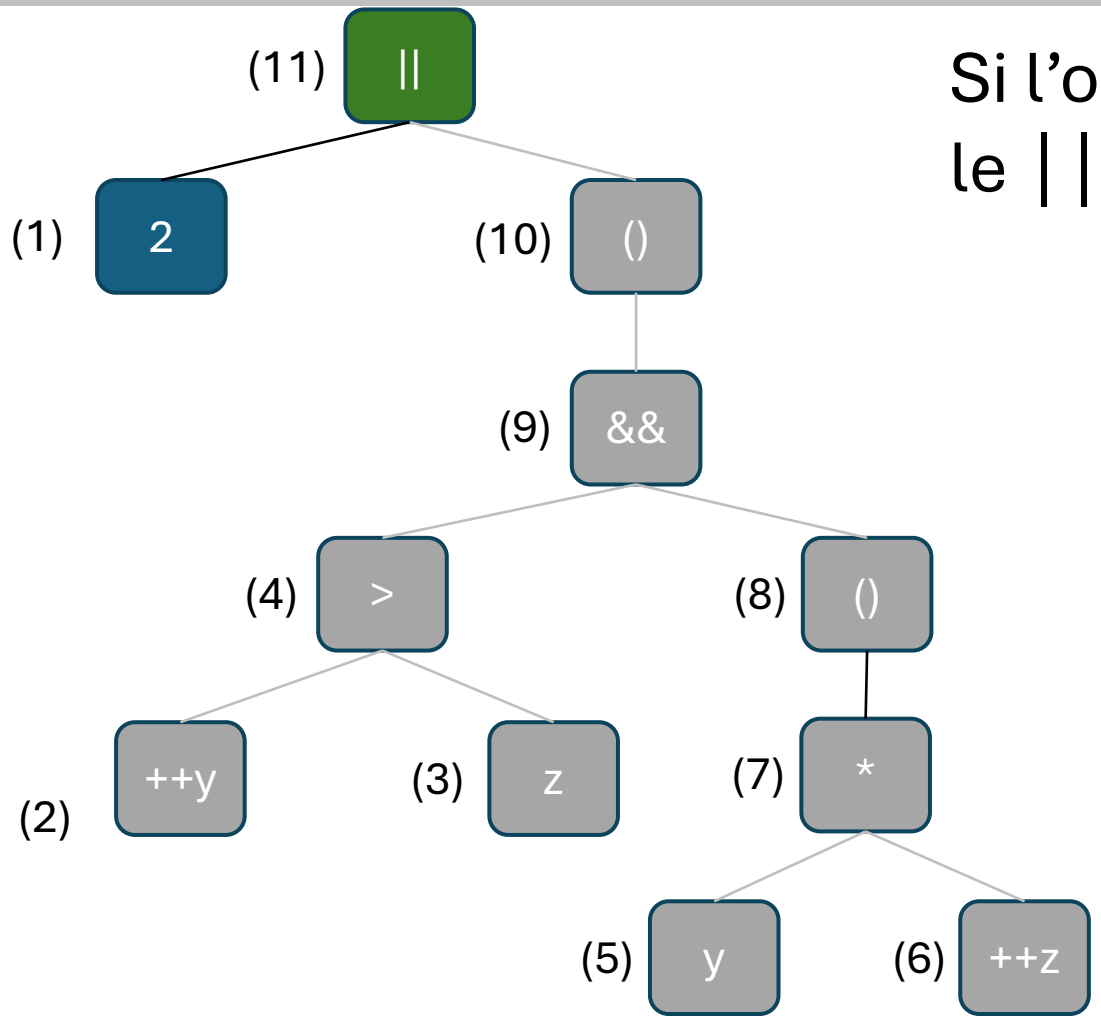


Si x = 1, ++x = 2

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = 1, y = 1 et z = 1



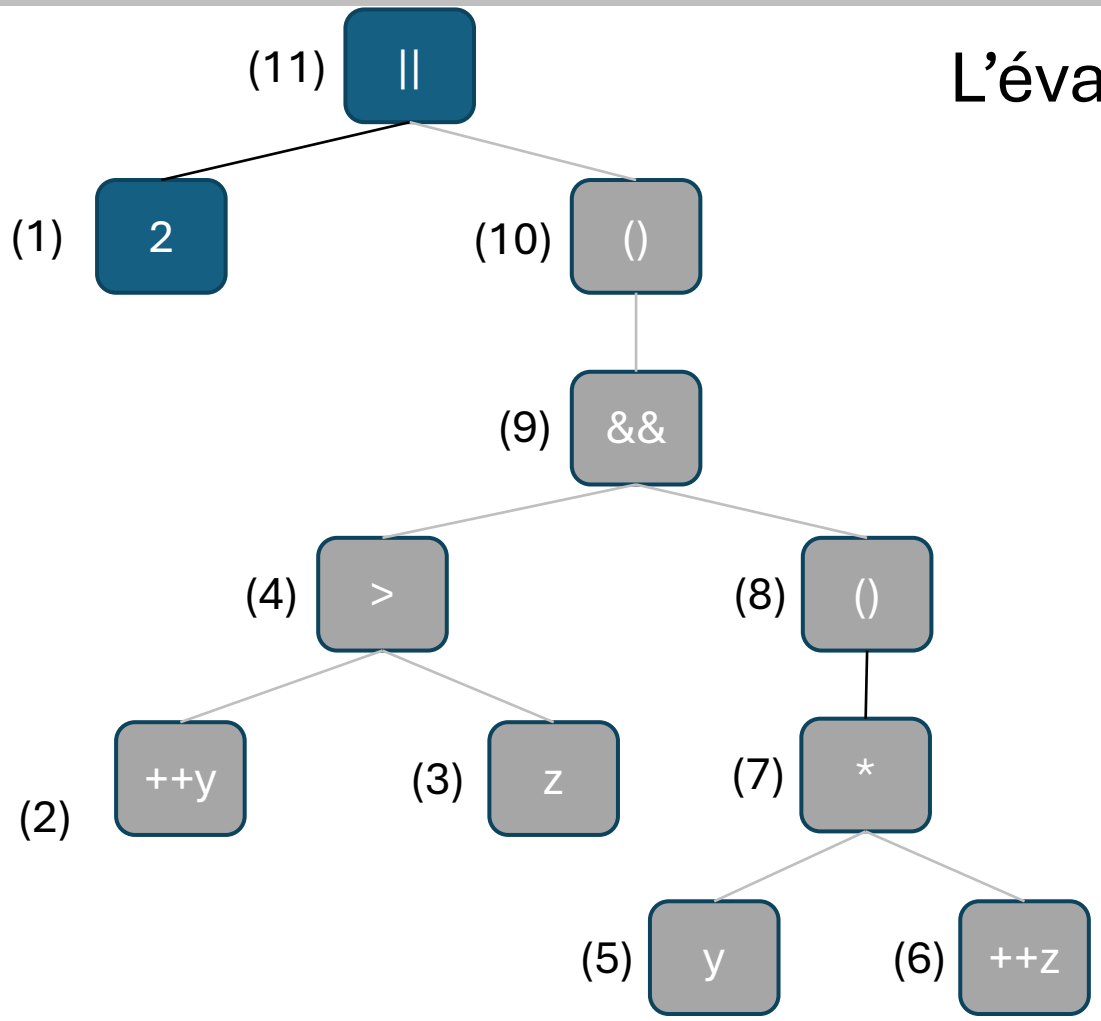
Si l'opérande gauche de || est VRAI alors le || sera toujours VRAI

||	VRAI	FAUX
VRAI	1	1
FAUX	1	0

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = 1, y = 1 et z = 1

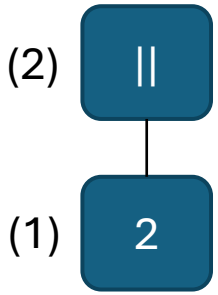


L'évaluation de l'expression est terminée

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = 1, y = 1 et z = 1



Arbre optimisé

Exercice 4.4.

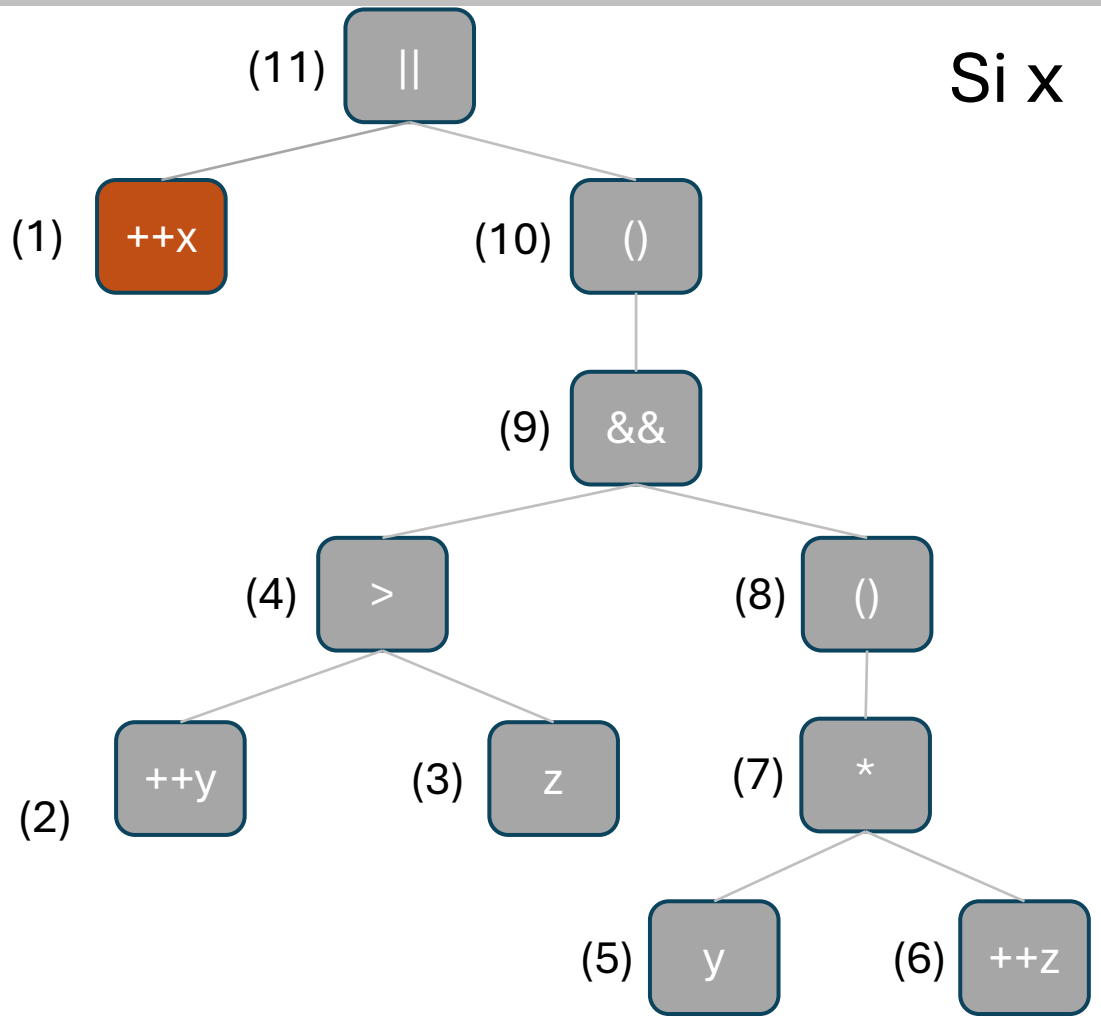
Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = -1, y = 1 et z = 3

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = -1, y = 1 et z = 3

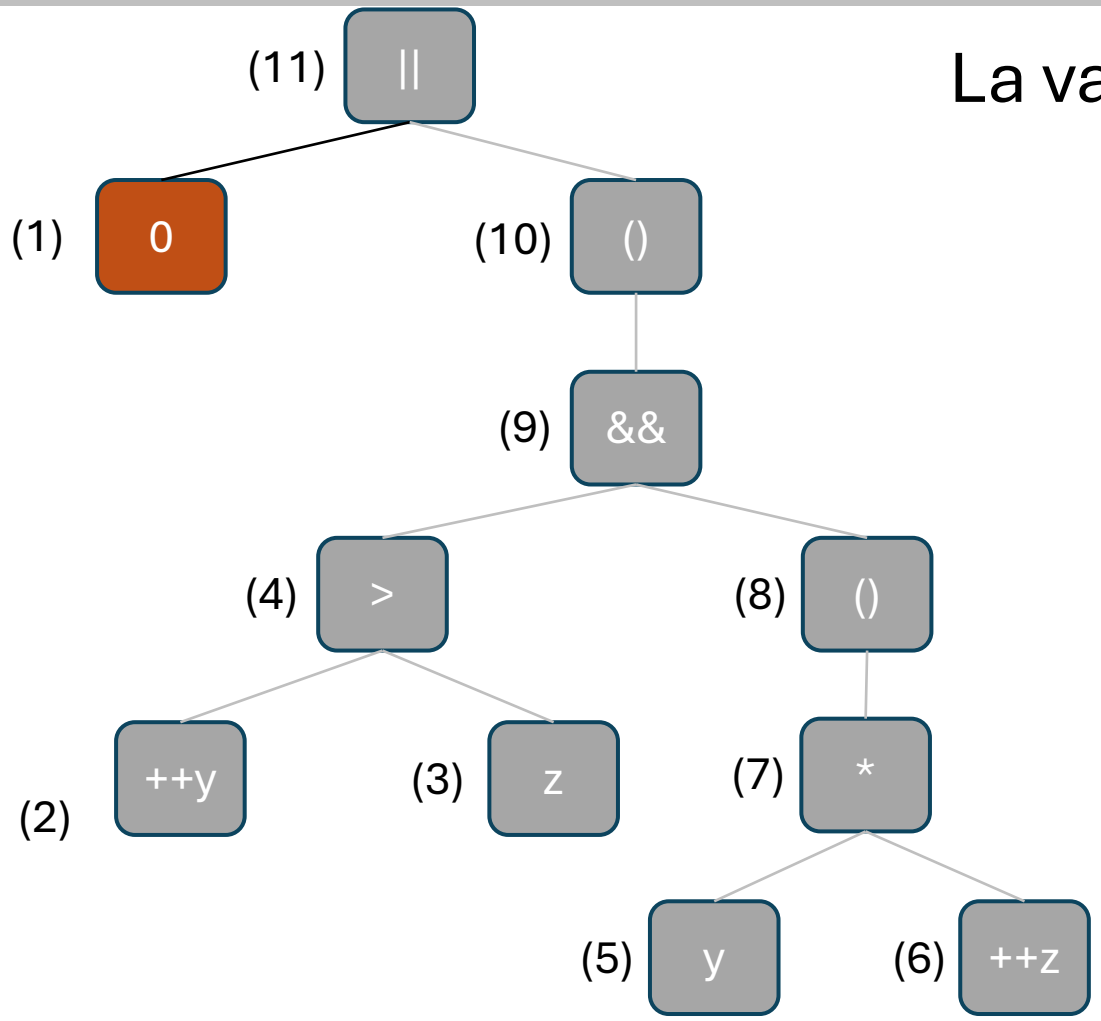


Si `x = -1, ++x = 0`

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = -1, y = 1 et z = 3



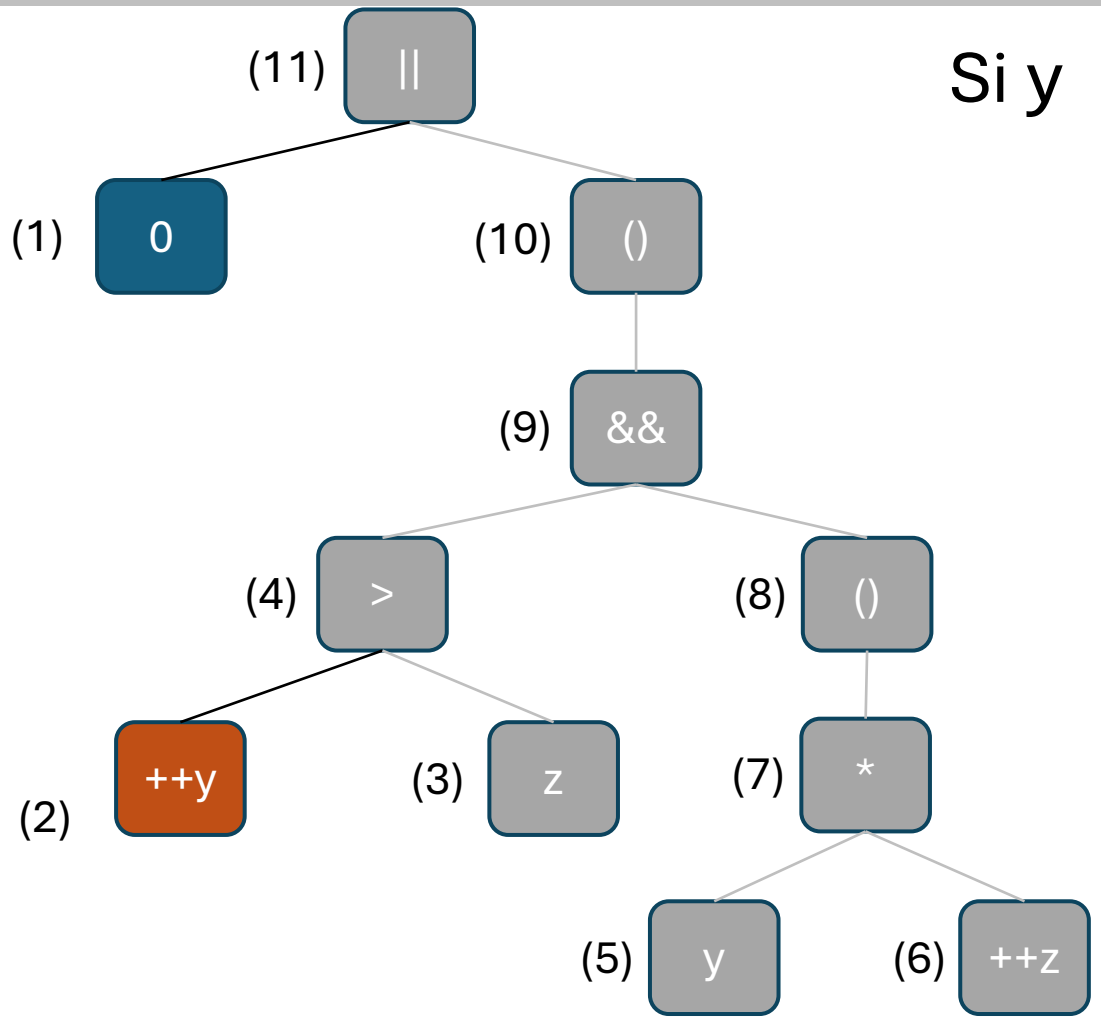
La valeur de `0||?` est inconnue

||	VRAI	FAUX
VRAI	1	1
FAUX	1	0

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = -1, y = 1 et z = 3

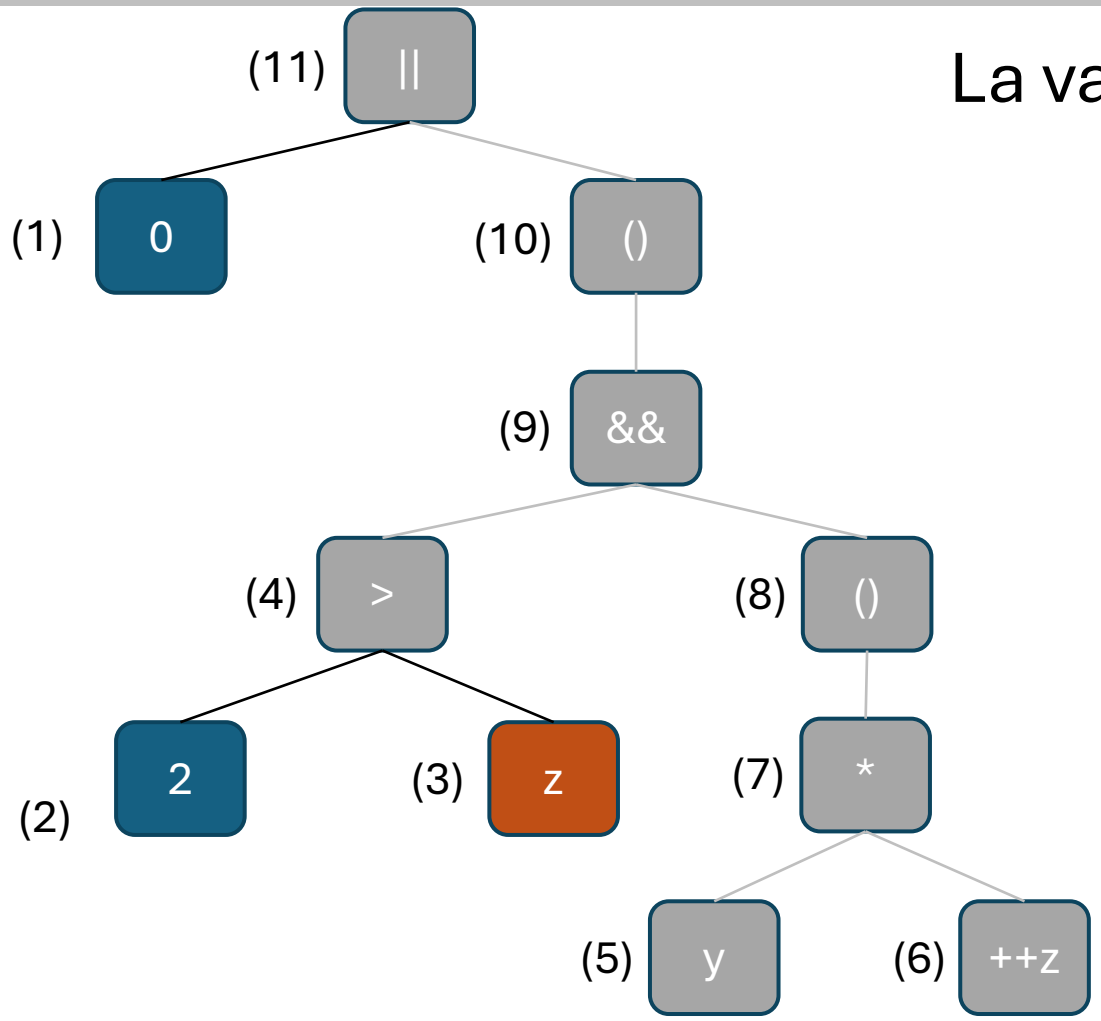


Si y = 1, ++y = 2.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = -1, y = 1 et z = 3

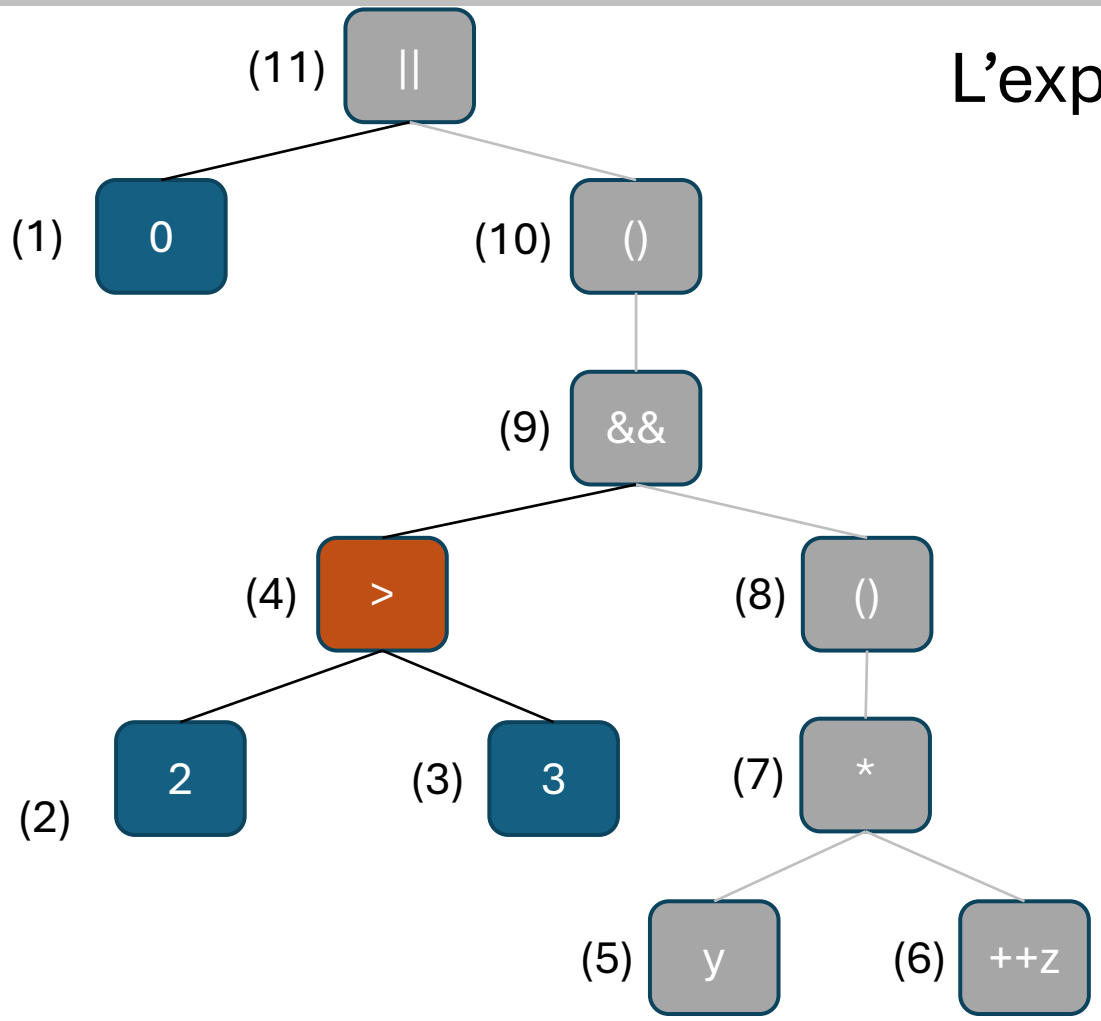


La variable z vaut 3

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = -1, y = 1 et z = 3

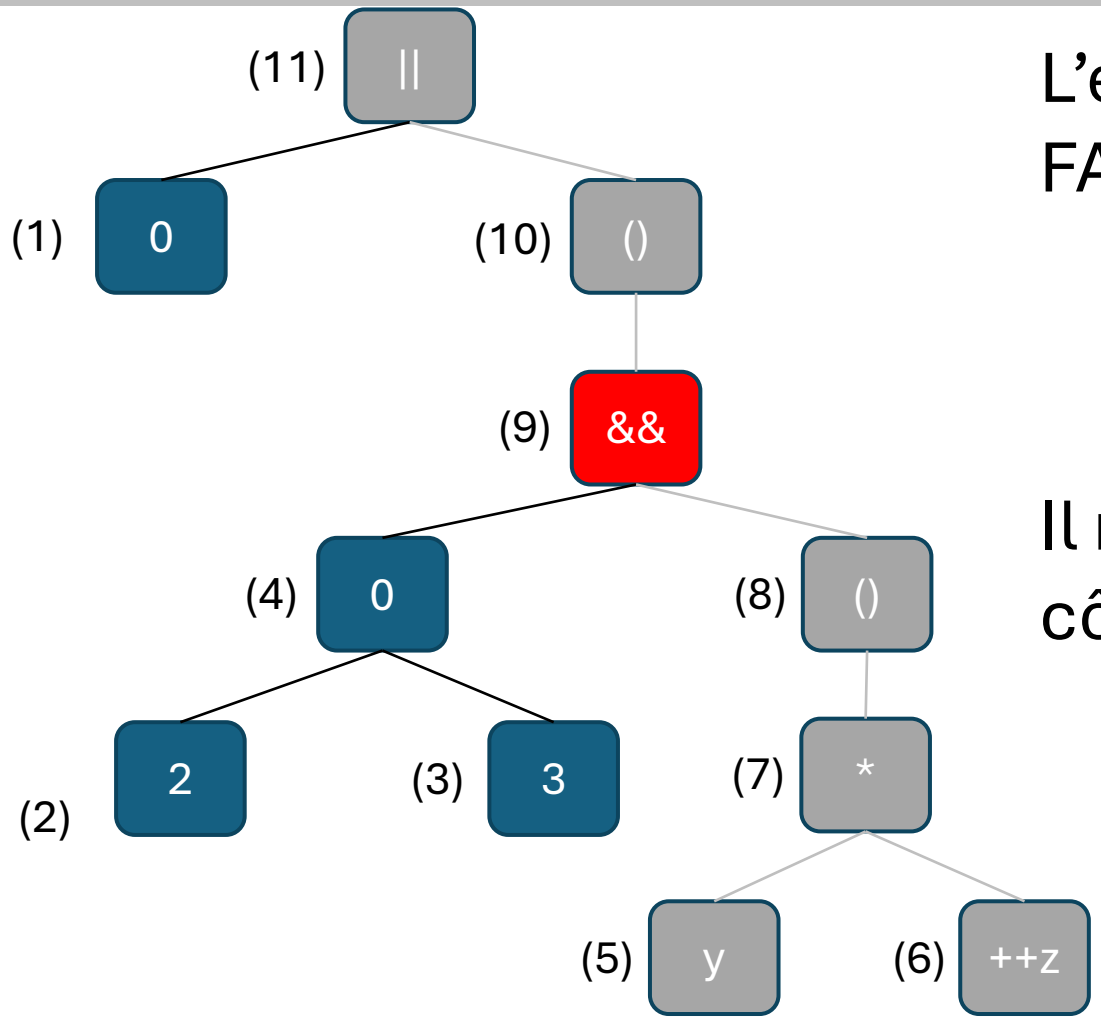


L'expression `2 > 3` vaut FAUX (`0`)

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = -1, y = 1 et z = 3



L'expression `0 && ?` vaut toujours FAUX (`0`).

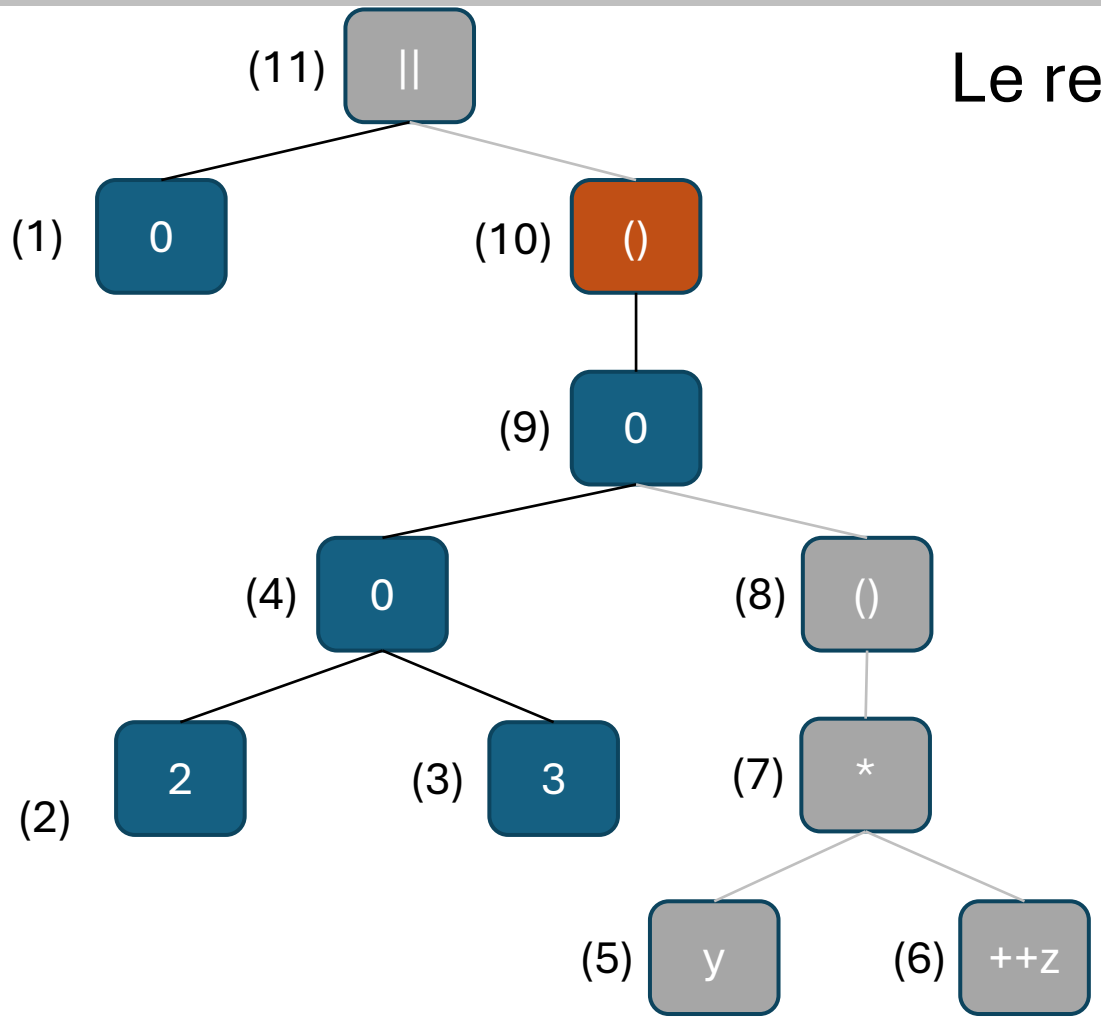
&&	VRAI	FAUX
VRAI	1	0
FAUX	0	0

Il **n'est pas nécessaire** de calculer le côté droit de l'expression.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = -1, y = 1 et z = 3

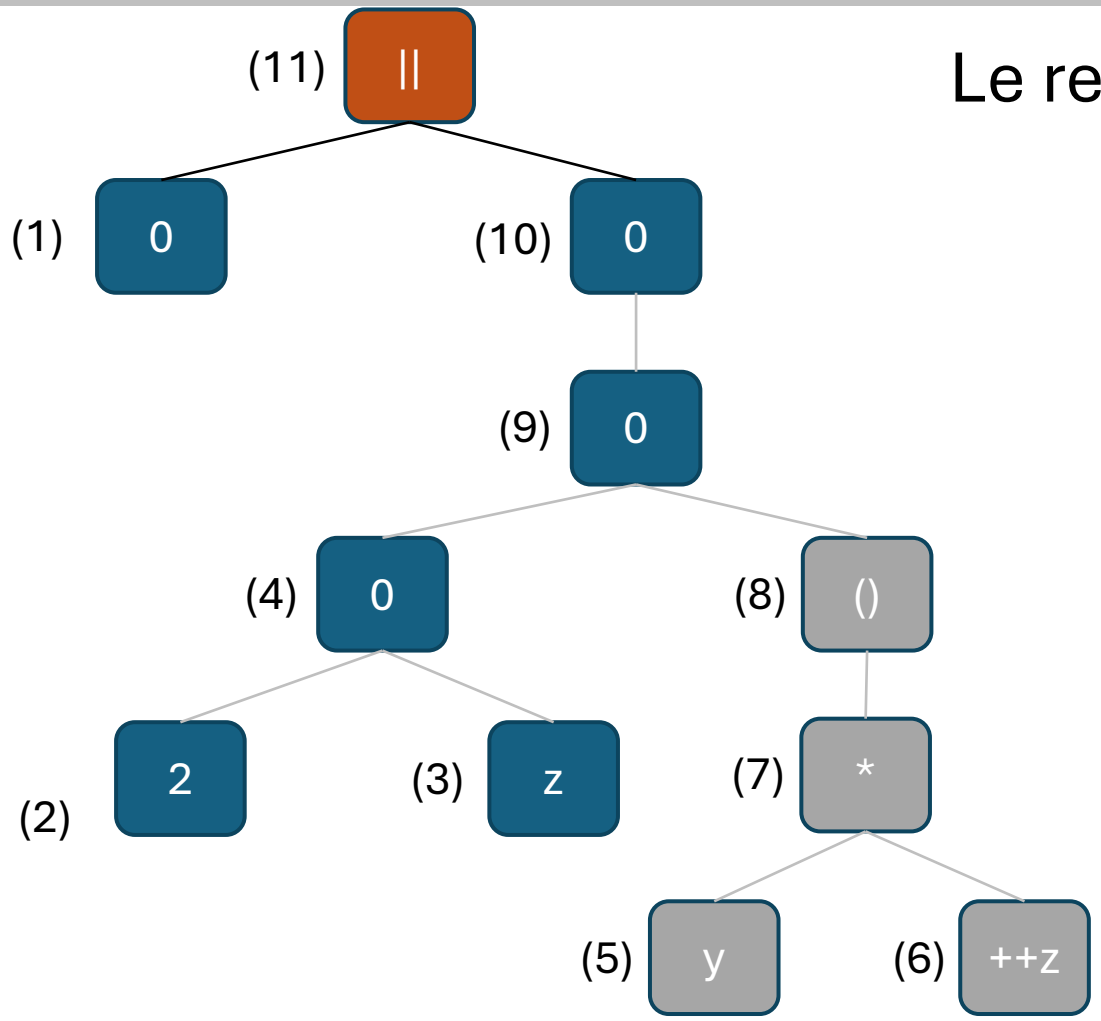


Le reste de l'expression est calculé.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = -1, y = 1 et z = 3

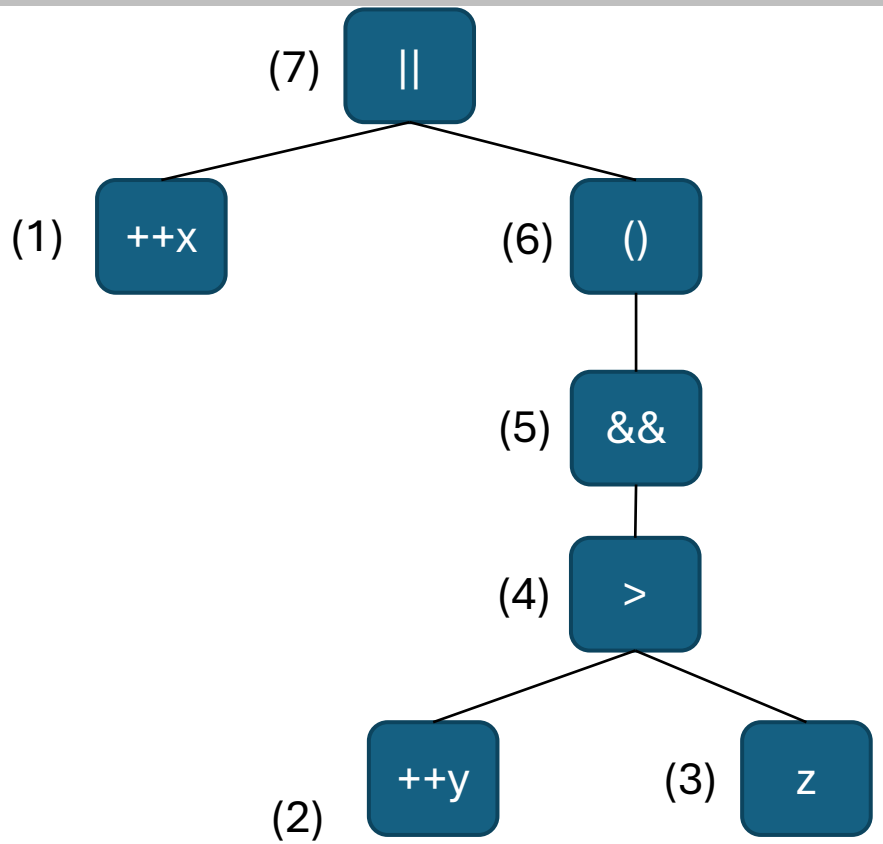


Le reste de l'expression est calculé.

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = -1, y = 1 et z = 3

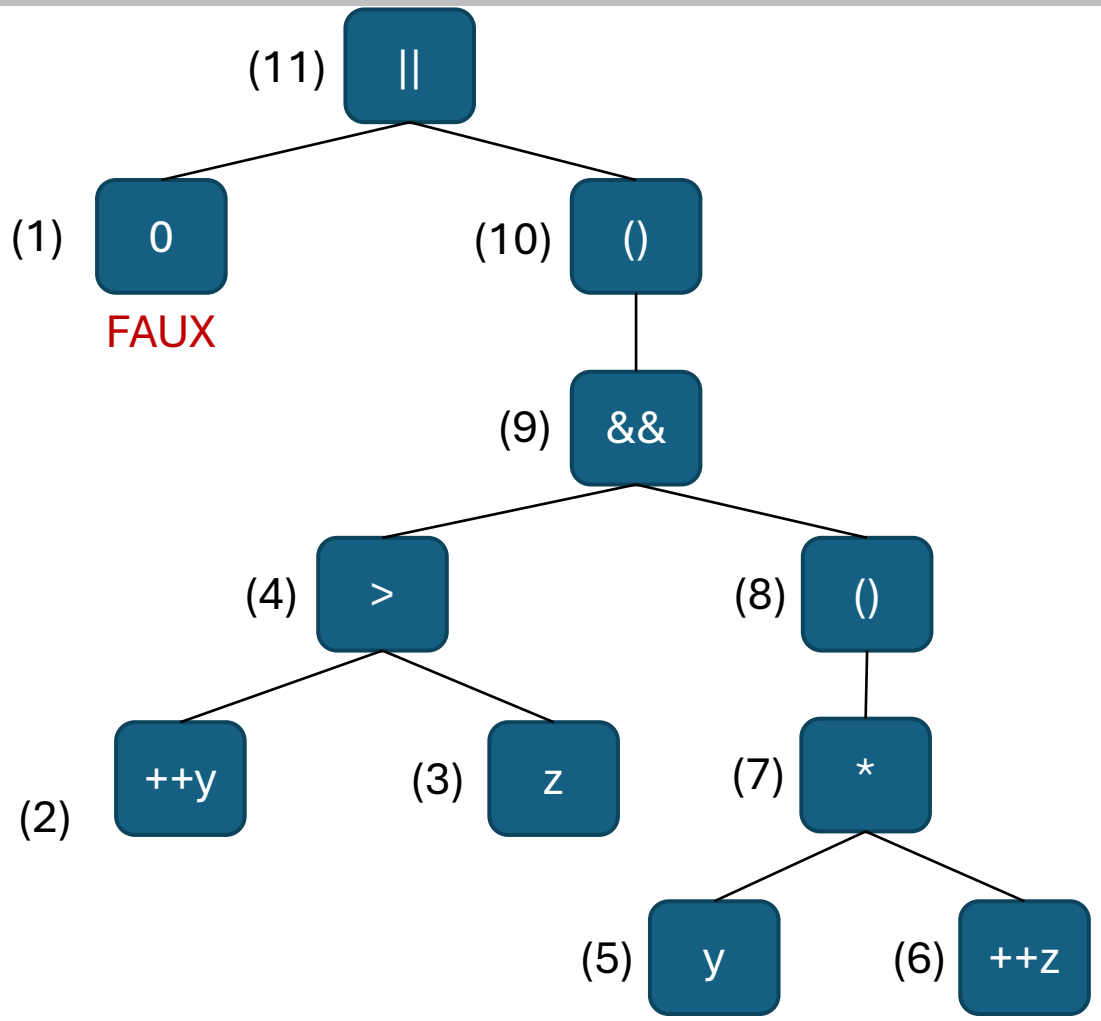


Arbre optimisé

Exercice 4.4.

Soit l'expression suivante: `++x || (++y > z && (y*++z))`

Faire apparaitre les optimisations (en retirant de l'arbre les nœuds inutilisés) si les valeurs des variables avant expression sont : x = -1, y = 1 et z = 3



	a	b	c	x	y	z
`int a = 5, b = 3, c = 2;`	5	3	2	-	-	-
`int x = 0, y = 0, z = 1;`	5	3	2	0	0	1
`a = b;`	3	3	2	0	0	1
`x || (y && z);`	3	3	2	0	0	1
`a & b;`	3	3	2	0	0	1
`a + b - c;`	3	3	2	0	0	1
`(x == y) || (x != z);`	3	3	2	0	0	1
`(a | b) & c;`						
`(a ^ b) || c;`						
`(a * c) / b;`						
`(a < b) != ( !(b >= c) );`						