CS-107 2025

Introduction à la Programmation : Mini-projet 1 en auto-évaluation

Opérateurs binaires

Rafael Pires
rafael.pires@epfl.ch

EPFL, Lausanne, 15.10.2024

Stéganographie

Challenge : Jouez les détectives

Capture de drapeau : `FLAG{…}`

Prix aux 10 premiers groupes à atteindre 100 points

But : atteindre au moins 60 points avant le 03.11.2025 (après cette date, le système d'évaluation sera fermé)

Ce projet n'est pas noté.

On peut néanmoins mettre des questions liées à ce projet dans l'examen.

Quelques annonces sur le mini-projet 1

- **N’oubliez pas de vous inscrire dans un groupe jusqu’à 15:59 aujourd’hui**
- Dès que le grader sera ouvert le vendredi 17 matin, **vous ne pourrez plus vous inscrire dans un groupe**

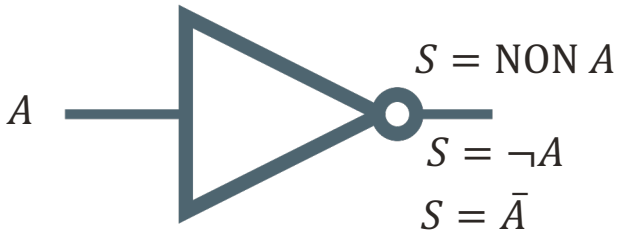
- le retour du grader peut être **lent** (dans certains cas limites atteindre **les 10 minutes**)
- les tests ont une composante aléatoire qui fait que **deux soumissions du même code** peuvent avoir des **notes légèrement différentes**.

Essayez ces exercices avant le TP de vendredi.

Objectifs du cours d’aujourd’hui

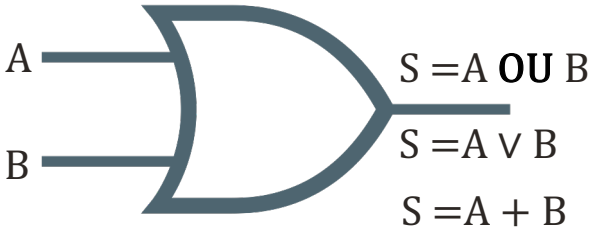
- Opérateurs binaires en Java
- Les pièges communs et conseils

Les portes logiques



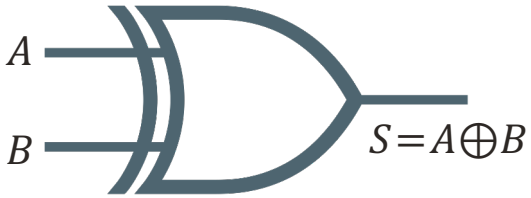
A	S = ¬ A
0	1
1	0

A	B	S = A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1



A	B	S = A ∨ B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A ⊕B
0	0	0
0	1	1
1	0	1
1	1	0



Les entiers en Java (1/3)

Tous les entiers en Java sont signés !

Nous les utilisons le plus souvent en base 10 .

- il ne s'agit que d'une représentation

Trois façons d'écrire la même chose :

`13` → $3*10^0+1*10^1$

`0b1101` → $1*2^0 +0*2^1 +1*2^2 +1*2^3$

`0xD` → $D*16^0$

(en base 16, A vaut 10, B vaut 11, ... F vaut 15)

un byte est codé sur un octet (8 bits)

- Entier signé valant entre -128 et 127

Représentation binaire des nombres entiers relatifs

complément à deux

- Avec n bits, on utilise la convention suivante pour représenter les nombres entiers relatifs (positifs et négatifs) :

$$N = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

- Exemple avec 8 bits : N = -43 est représenté par 11010101, car

$$\begin{aligned} -43 &= -128 + 64 + 0 + 16 + 0 + 4 + 0 + 1 \\ &= \mathbf{-1} \cdot 2^7 + \mathbf{1} \cdot 2^6 + \mathbf{0} \cdot 2^5 + \mathbf{1} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{1} \cdot 2^0 \end{aligned}$$

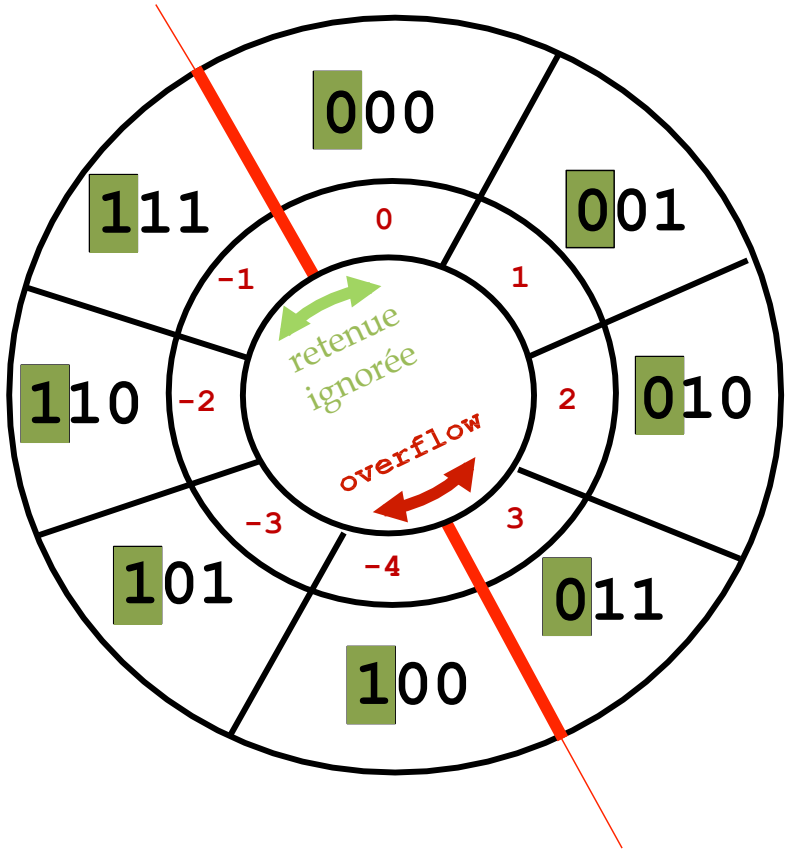
Représentation binaire des nombres entiers relatifs

complément à deux

- Le premier bit est le bit de signe
  - `0` : nombre positif
  - `1` : nombre négatif
- Avec 8 bits, les nombres représentables vont de -128 à +127 :
  - -128 = `10000000`
  - +127 = `01111111`
- -1 = `11111111` avec cette représentation
- +128 n'est pas représentable avec 8 bits!

Opérations binaires : addition et soustraction

complément à deux



Les entiers en Java (2/3)

Arithmétique en **complément à deux** (voir la vidéo fournie)

Les opérations + et - se font modulo les plus grand/petit nombres représentables :

- Pas de débordement
  - Si un byte vaut 127, lui additionner 1 donnera -128
  - Si un byte vaut -128 lui retrancher 1 donnera 127

Soyez attentifs à la promotion entière : les calculs sur des byte renvoient toujours un int

- **transtypage nécessaire**

```
byte b = 10;
byte sum = b + b;                // compile error
byte sum = (byte) (b + b);       // cast necessary
```

Les entiers en Java (3/3)

Un `int` est représenté sur 4 octets (32 bits).

Transtyper un `int` en `byte` tronque les 3 octets les plus à gauche.

Transtyper un `byte` en `int` remplit le 3 octets les plus à gauche de 1 si le `byte` est négatif et de 0 sinon (complément à deux).

```
byte b = 127;          // 0b01111111
int i = 127 + 1;       // 0b00000000_00000000_00000000_10000000 = 128
byte b2 = (byte) i;    // 0b00000000_00000000_00000000_10000000 = -128
int i2 = (byte)-128;   // 0b11111111_11111111_11111111_10000000 = -128
```

Utilisez le debugger pour examiner le contenu des variables (représentation binaire)

Les opérateurs binaires &, |, ^, ~, <<, >>, >>>

Ils s'appliquent à la représentation binaire des nombres.

Ils vont être utilisés de façon omniprésente :

- par certains algorithmes de **chiffrement**
- pour **accéder à un bit particulier** dans le but d'y stocker de l'information
- pour coder des fonctionnalités utiles sur les **images**, par exemple :
    - extraction des canaux de couleurs.
    - produire une représentation en niveau de gris d'une image.

Voir la documentation de Java pour une liste exhaustive de ces opérateurs.

Les opérateurs binaires : << et |

```
int val1 = 1 << 7;
```


```
// 0b10000000
```

```
int val2 = val1 | 1;

// 0b10000001
```

OR		
0	0	0
0	1	1
1	0	1
1	1	1

Les opérateurs binaires : << et |

```
int val1 = 0b10;
int val2 = 0b10001;
```

But : Obtenir `0b10010001`.

1. Shift left avec insertion de 0s à droite (<<)

```
int val1_sl = val1 << 6;
// 0b10000000
```

2. Bitwise OR

```
int val3 = val1_sl | val2;
// 0b10010001
```

Les opérateurs binaires : >>>, >> et &

```
byte val1 = -128;          // 0b10000000
byte val2 = val1 >> 7;     // 0b11111111 = -1
```

7×1 

Décalage arithmétique
conserve le signe

```
byte val3 = val1 >>> 7;    // 0b00000001 = 1
```

7×0 

Décalage logique
ne conserve pas le signe

```
int val2 = val2 & 0b11;
         // 0b00000011
```

AND		
0	0	0
0	1	0
1	0	0
1	1	1

Les opérateurs binaires : >>>, >> et &

```
int val1 = 0b10010001;
```

But : Extraire les bits rouges et bleus séparement.

1. Shift right avec insertion de 0s à gauche (>>>)

```
int val1_sr = val1 >>> 6;
// 0b10
```

2. Bitwise AND avec une masque

```
int val2 = val1 & 0b11111;
// 0b10001
```

>>> insertion de 0s à gauche.
>> insertion de 0s pour nombres positifs et de 1s pour les négatifs sur la gauche.

Objectifs du cours d’aujourd’hui

- Opérateurs binaires en Java
- Les pièges communs et conseils

Les pièges

1. Ordre de bits.
2. Chiffres positifs / négatifs.
3. Cas limites.

Piège 1 : Ordre de bits

On vous demande de transformer un octet en un tableau de booléens, et vice-versa.
Par exemple :

droite à gauche

Indices aux opérateurs binaires
<<, **>>>** et **>>** : 76543210.

```
byte value = 0b00110110;
boolean[] bits = toBitArray(value);
// bits = {false, false, true, true, false, true, true, false}
```

gauche à droite

Piège 2 : Nombres positifs / négatifs

Comment faire de l'arithmétique non signée entre des nombres de type `byte` ?

```
byte b = 0b10010000;          // 144 (non signé) = -112 (signé)
System.out.println(b / 12); // affiche -9
```

Comment on fait pour que le résultat soit 12 ?

```
System.out.println(Byte.toUnsignedInt(b) / 12); // affiche 12
```

Piège 3 : Cas limites

Il faudra s'assurer que les paramètres des méthodes sont corrects en utilisant ce que l'on appelle des **assertions**.

```
public static int[][] embedBW(int[][] cover, boolean[][] image)
{
        assert (cover != null);
        assert (cover.length >= image.length);
        //...
}
```

Questions ?

Nous sommes à disposition sur le forum **Ed** pendant la semaine.
N’hésitez pas à poser des questions.

Bon projet à toutes et tous !

rafael.pires@epfl.ch

Merci