

DUT SRC – IUT de Marne-la-Vallée
07/10/2011
INF120 - Algorithmique

Cours 2
***Variables et affectations,
type et codage***

Sources

- *Le livre de Java premier langage*, d'A. Tasso
- Cours INF120 de J.-G. Luque
- Cours FLIN102 de l'Université Montpellier 2
- Cours de J. Henriet : <http://julienhenriet.olymp-network.com/Algo.html>
- <http://xkcd.com>, <http://xkcd.free.fr>

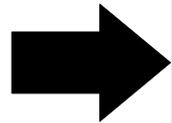
Plan du cours 2 – Variables et affectations, type et codage

- Résumé des épisodes précédents
- Correction du QCM0
- Le pseudo-code
- De l'organigramme au code Java
- Codage des données
- Codage binaire des entiers
- Codage des flottants
- Autres codages
- Codage hexadécimal
- Booléens et opérations de base

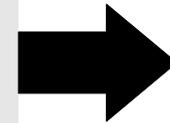
Résumé des épisodes précédents

Algorithme : suite d'instructions pour résoudre un problème

Données du problème
entrées de l'algorithme



Algorithme :
Instruction 1
Instruction 2
Instruction 3
...



Résultat
sorties de l'algorithme

Un algorithme utilise plusieurs types d'**instructions** :

- des **affectations** dans des **variables** (mémoires)
- des **appels** à d'autres **algorithmes**
- des “**lectures**” d'**entrées** et “**renvois**” de **sorties**
- des **boucles**
- des **tests**

On peut décrire un algorithme :

- en français
- en pseudo-code
- par un organigramme
- dans un langage de programmation

Dictionnaire pseudo-code / Java

	Pseudo-code	Java
Déclaration d'un algorithme	Addition Entrées : entiers i et j Sortie : somme $i+j$ (entier) Début ... renvoyer ... Fin	public static int Addition (int i , int j){ ... return ... }
Déclaration d'une variable	Variables : entier i	int i ;
Affectation	$i \leftarrow 1$	$i = 1$;
Test	Si $i=1$ alors ... Sinon ... FinSi	if ($i==1$){ ... } else { ... }
Boucle	Tant que $i<3$: ... Fin TantQue	while ($i<3$) { ... }

L'organigramme de la multiplication "simple"

Je connais l'algorithme d'**addition** de deux entiers positifs.
Comment écrire un algorithme de **multiplication** de deux entiers ?

Intuition :

$$5 \times 3 = \underbrace{5 + 5 + 5}_{3 \text{ fois}}$$

$$\textit{entier1} \times \textit{entier2} = \underbrace{\textit{entier1} + \textit{entier1} + \textit{entier1} + \dots + \textit{entier1}}_{\textit{entier2} \text{ fois}}$$

En français :

J'ajoute *entier2* fois l'entier *entier1* à l'entier 0.

L'organigramme de la multiplication "simple"

Intuition :

$$5 \times 3 = 5 + 5 + 5$$

En français :

J'ajoute *entier2* fois l'entier *entier1* à l'entier 0.

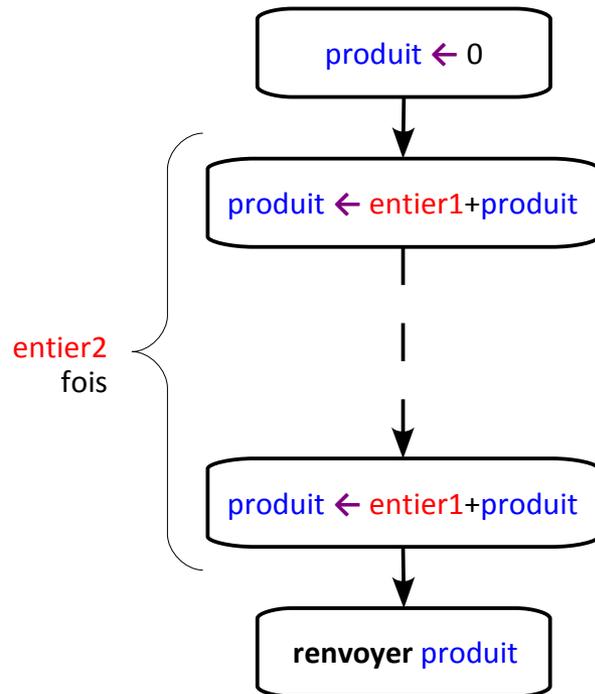
L'organigramme de la multiplication "simple"

Intuition :

$$5 \times 3 = 5 + 5 + 5$$

En français :

J'ajoute *entier2* fois l'entier *entier1* à l'entier 0.



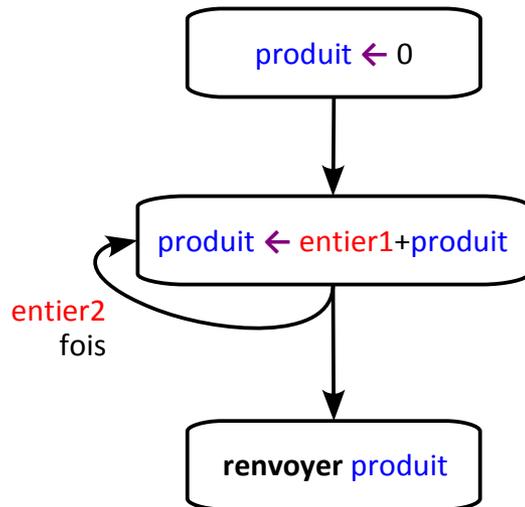
L'organigramme de la multiplication "simple"

Intuition :

$$5 \times 3 = 5 + 5 + 5$$

En français :

J'ajoute *entier2* fois l'entier *entier1* à l'entier 0.



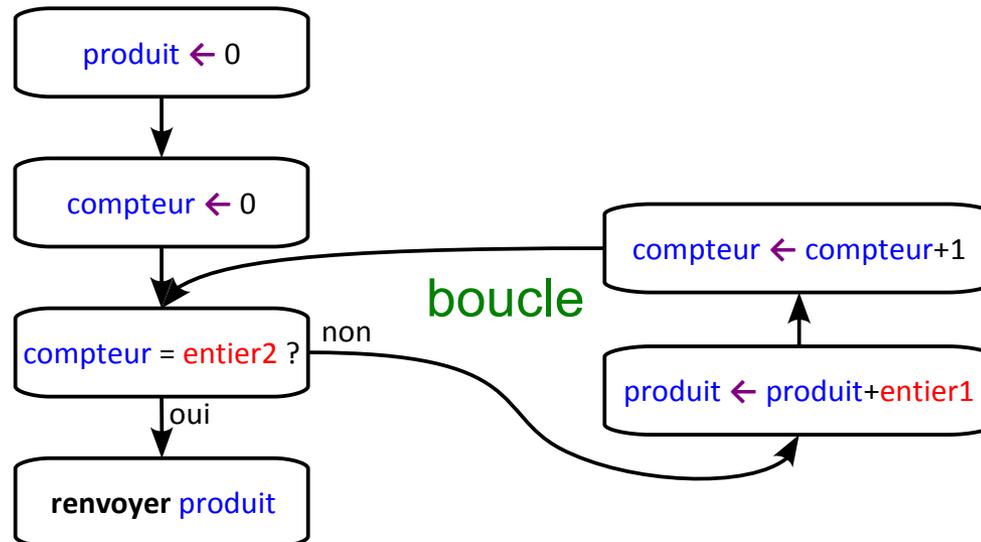
L'organigramme de la multiplication "simple"

Intuition :

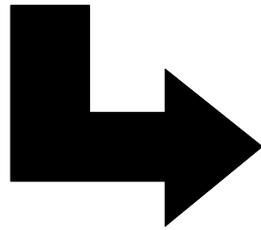
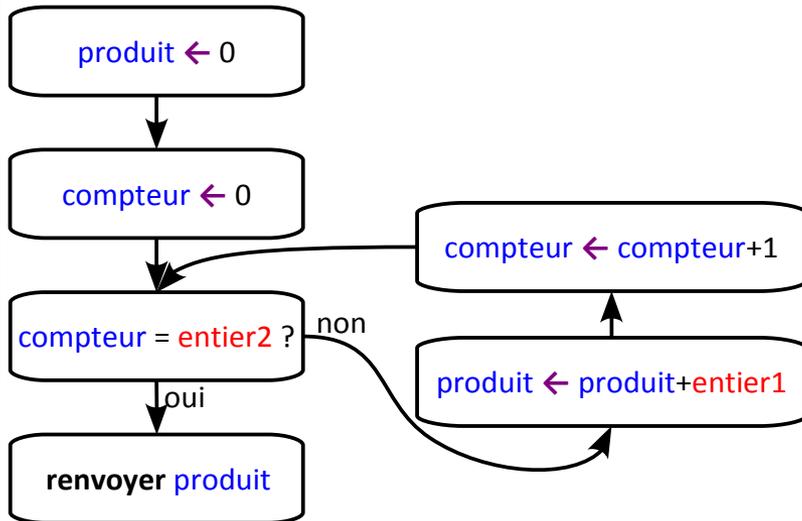
$$5 \times 3 = 5 + 5 + 5$$

En français :

J'ajoute *entier2* fois l'entier *entier1* à l'entier 0.



De l'organigramme au pseudo-code



Multiplication :

Entrées : deux entiers *entier1* et *entier2*

Sorties : le *produit* de *entier1* et *entier2*

Variables : entiers *compteur* et *produit*

Début

compteur ← 0

produit ← 0

Tant que *compteur* < *entier2* faire :

produit ← addition(*produit*, *entier1*)

compteur ← addition(*compteur*, 1)

Fin tant que

renvoyer produit

Fin

Du pseudo-code au code Java

Multiplication :

Entrées : deux entiers *entier1* et *entier2*

Sorties : le produit de *entier1* et *entier2*

Variables : entiers *compteur* et *produit*

Début

compteur ← 0

produit ← 0

Tant que *compteur* < *entier2* faire :

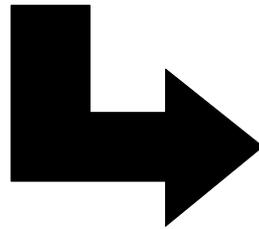
produit ← addition(*produit*, *entier1*)

compteur ← addition(*compteur*, 1)

Fin tant que

renvoyer *produit*

Fin



```
public static int multiplication(int entier1,  
int entier2){
```

```
    int compteur=0;
```

```
    int produit=0;
```

```
    while (compteur<entier2){
```

```
        produit=produit+entier1;
```

```
        compteur=compteur+1;
```

```
    }
```

```
    return produit;
```

```
}
```

Le codage

La “minute votes SMS”

Programme Java :

```
import java.io.*;

public class Boucle{

    public static void main(String [] arg){

        int i=1;

        while(i>0){

            i=i*2;

        }

        System.out.print("J'ai fini !");

    }

}
```

Est-ce que ce programme affiche “J'ai fini !” ?

Le codage

La “minute votes SMS”

Programme Java :

```
import java.io.*;

public class Boucle{

    public static void main(String [] arg){

        int i=1;

        while(i>0){

            i=i*2;

        }

        System.out.print("J'ai fini !");

    }

}
```

Algorithme en pseudo-code :

Boucle :

Variable : entier i

Début

$i \leftarrow 1$

Tant que $i > 0$ faire :

$i \leftarrow 2 \times i$

Fin tant que

afficher("J'ai fini!")

Fin

Est-ce que ce programme affiche “J'ai fini !” ?

Le codage

La “minute votes SMS”

Programme Java :

```
import java.io.*;

public class Boucle{

    public static void main(String [] arg){

        int i=1;

        while(i>0){

            i=i*2;

        }

        System.out.print("J'ai fini !");

    }

}
```

Est-ce que ce programme affiche “J'ai fini !” ?

Algorithme en pseudo-code :

Boucle :

Variable : entier i

Début

$i \leftarrow 1$

Tant que $i > 0$ **faire :**

$i \leftarrow 2 \times i$

Fin tant que

afficher("J'ai fini!")

Fin

Est-ce que cet algorithme se termine ?

Le codage

La “minute votes SMS”

Programme Java :

```
import java.io.*;

public class Boucle{

    public static void main(String [] arg){

        int i=1;

        while(i>0){

            i=i*2;

        }

        System.out.print("J'ai fini !");

    }

}
```

Est-ce que ce programme affiche “J'ai fini !” ?

Algorithme en pseudo-code :

Boucle :

Variable : entier i

Début

$i \leftarrow 1$

Tant que $i > 0$ faire :

$i \leftarrow 2 \times i$

Fin tant que

afficher("J'ai fini!")

Fin

Est-ce que cet algorithme se termine ?

OUI si i est **VRAIMENT** un entier

Le codage

La "minute votes SMS"

Programme Java :

```
import java.io.*;

public class Boucle{

    public static void main(String [] arg){

        int i=1;

        while(i>0){

            i=i*2;

        }

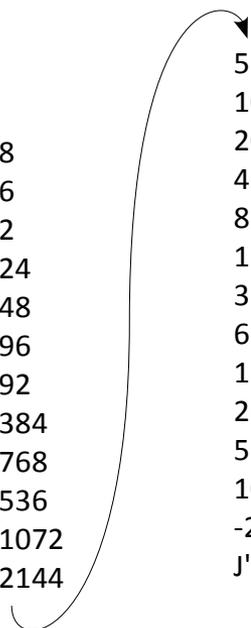
        System.out.print("J'ai fini !");

    }

}
```

Résultat du programme :

2	
4	
8	
16	
32	524288
64	1048576
128	2097152
256	4194304
512	8388608
1024	16777216
2048	33554432
4096	67108864
8192	134217728
16384	268435456
32768	536870912
65536	1073741824
131072	-2147483648
262144	J'ai fini!!



Est-ce que ce programme affiche "J'ai fini !" ? **OUI !**

Le codage

La “minute votes SMS”

Programme Java :

```
import java.io.*;

public class Boucle{

    public static void main(String [] arg){

        int i=1;

        while(i>0){

            i=i*2;

        }

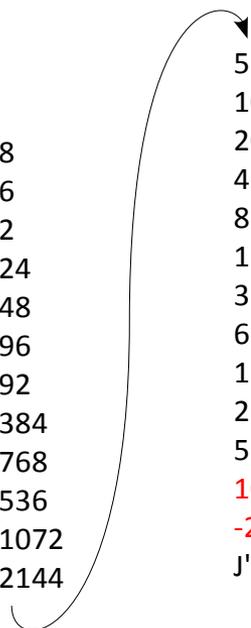
        System.out.print("J'ai fini !");

    }

}
```

Résultat du programme :

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
-2147483648
J'ai fini!!
```



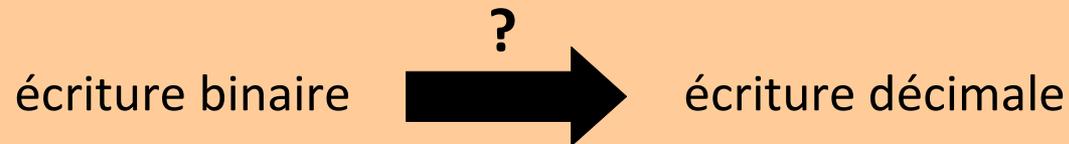
Les entiers “int” Java ne sont **pas de “vrais entiers”** mais des entiers **entre -2 147 483 648 et 2 147 483 647**

Le codage des entiers en binaire

La "minute mathématique"

Pour le stockage comme pour le traitement d'instructions, il est nécessaire que toutes les données traitées par un ordinateur soient codées en **binaire**, par des **0** et des **1**.

Exemple de nombre entier en binaire : 1101100001101



Le codage des entiers en binaire

La "minute mathématique"

Pour le stockage comme pour le traitement d'instructions, il est nécessaire que toutes les données traitées par un ordinateur soient codées en **binaire**, par des **0** et des **1**.

Exemple de nombre entier en binaire : 1101100001101

0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$2^{12} + 2^{11} + 2^9 + 2^8 + 2^3 + 2^2 + 2^0 = 4096 + 2048 + 512 + 256 + 8 + 4 + 1 = 6925$$

écriture binaire



écriture décimale

Le codage des entiers en binaire

Pour le stockage comme pour le traitement d'instructions, il est nécessaire que toutes les données traitées par un ordinateur soient codées en **binaire**, par des **0** et des **1**.



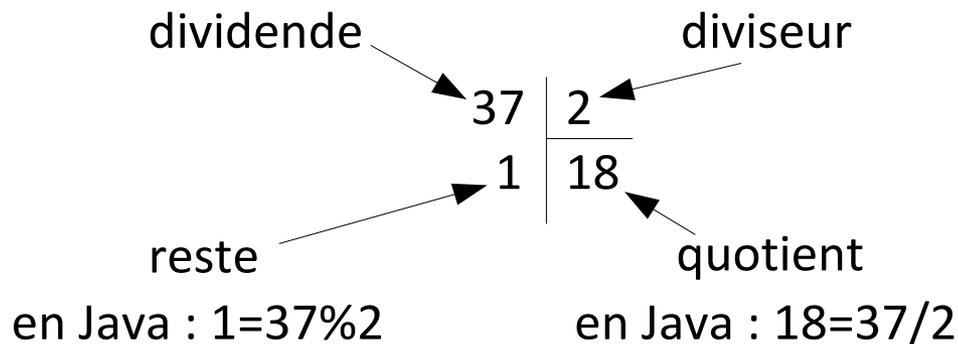
Le codage des entiers en binaire

Pour le stockage comme pour le traitement d'instructions, il est nécessaire que toutes les données traitées par un ordinateur soient codées en **binaire**, par des **0** et des **1**.



Exemple : écrire 37 en binaire ?

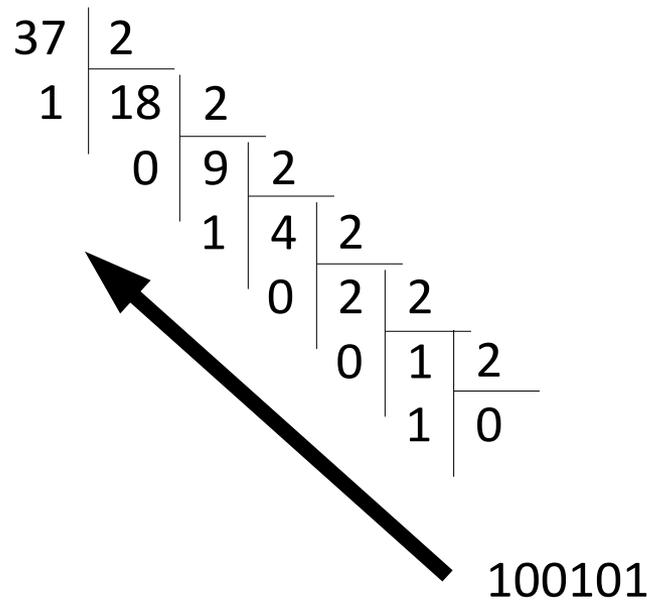
Division euclidienne :



Le codage des entiers en binaire

Pour le stockage comme pour le traitement d'instructions, il est nécessaire que toutes les données traitées par un ordinateur soient codées en **binaire**, par des **0** et des **1**.

écriture décimale  écriture binaire



Le calcul rapide en binaire

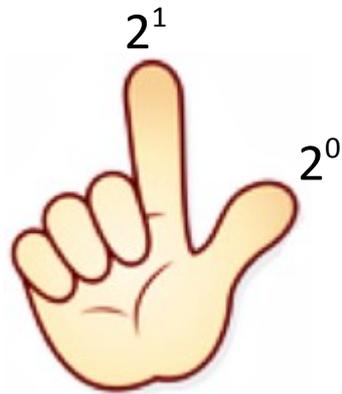
Pour faire son geek :

- compter sur ses doigts en binaire, jusqu'à 2^{10}

Le calcul rapide en binaire

Pour faire son geek :

- compter sur ses doigts en binaire, jusqu'à 2^{10}



$$2^{0+} + 2^1 = 3$$

Le calcul rapide en binaire

Pour faire son geek :

- compter sur ses doigts en binaire, jusqu'à 2^{10}

- faire des estimations de nombres données en binaire : $2^{10} = 1024$ donc $2^{10} \approx 1000$

➔ $2^{32} \approx ?$

Le calcul rapide en binaire

Pour faire son geek :

- compter sur ses doigts en binaire, jusqu'à 2^{10}

- faire des estimations de nombres donnés en binaire : $2^{10} = 1024$ donc $2^{10} \approx 1000$

③ $\longrightarrow 2^{32} \approx ?$

① $a^{b \times c} = (a^b)^c$

② $a^{b+c} = a^b \times a^c$

$$\begin{aligned} 2^{32} &= 2^{30+2} && \textcircled{2} \\ &= 2^{30} \times 2^2 \\ &= 2^{10 \times 3} \times 4 \\ &= (2^{10})^3 \times 4 && \textcircled{1} \\ &\approx 1000^3 \times 4 \\ &\approx 4 \text{ milliards} \end{aligned}$$

Le calcul rapide en binaire

Pour faire son geek :

- compter sur ses doigts en binaire, jusqu'à 2^{10}

- faire des estimations de nombres données en binaire : $2^{10} = 1024$ donc $2^{10} \approx 1000$

➔ $2^{32} \approx 4$ milliards

$$2^{32} = 4\,294\,967\,296$$

Le codage binaire

La "minute xkcd"

De 1 à 10 :

Sur une échelle de 1 à 10,
quelle est la probabilité
que cette question utilise
du binaire ?



<http://xkcd.com/953>
<http://xkcd.free.fr?id=953>

Si vous obtenez une note de 11/100 à un examen d'informatique, mais que vous dites qu'il devrait être compté comme un 15/20, alors on décidera probablement que vous le méritez.

Le codage des entiers en mémoire

1 bit = 0 ou 1

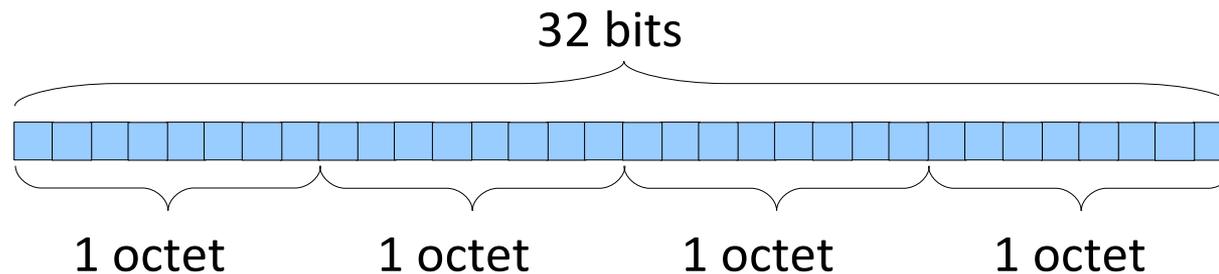
1 octet = 8 bits

1 Ko (kiloctet) = 1024 octets

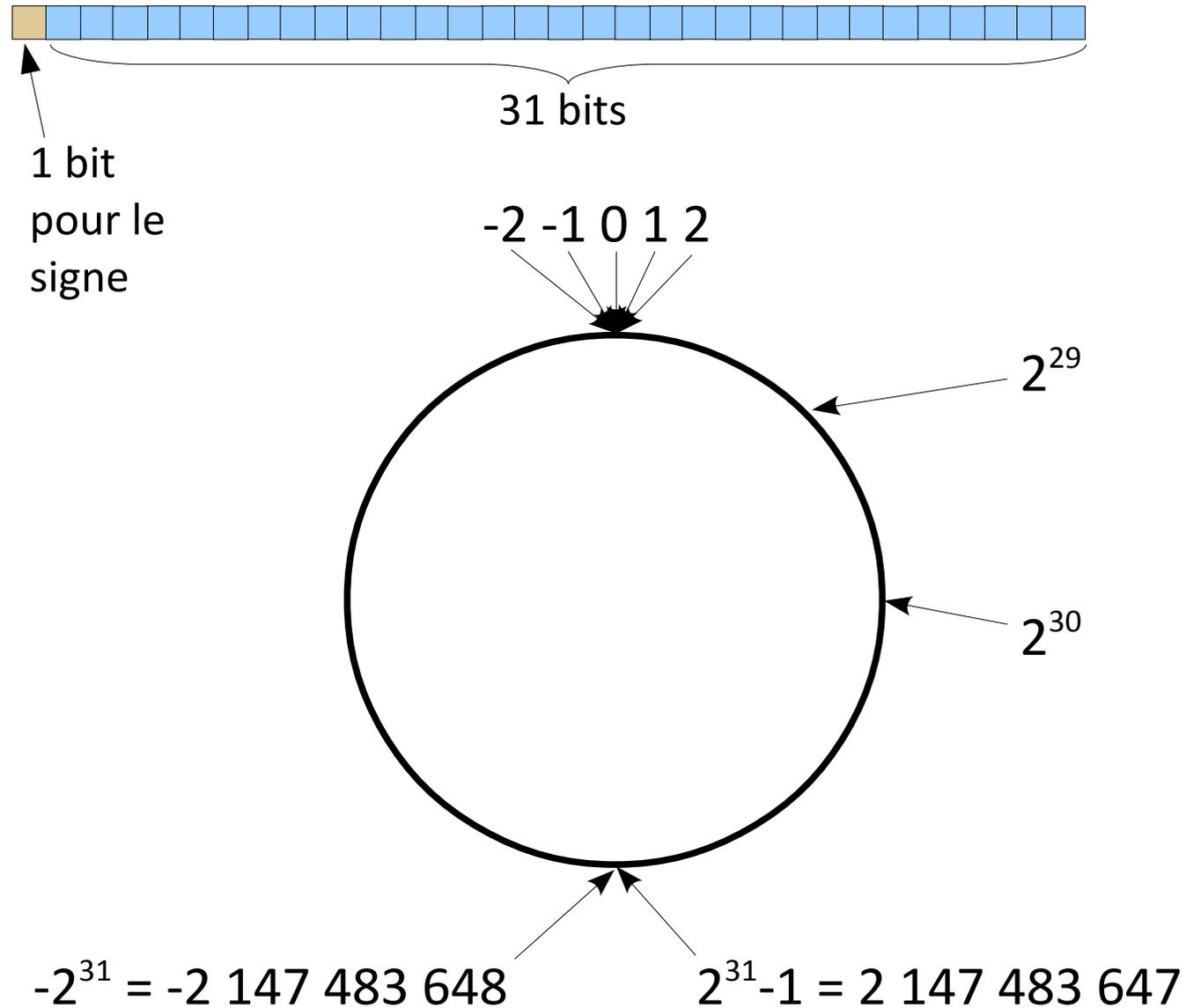
1 Mo (mégaoctet) = 1024 Ko (disquette)

1 Go (gigaoctet) = 1024 Mo (carte mémoire, 2h de vidéo en DivX)

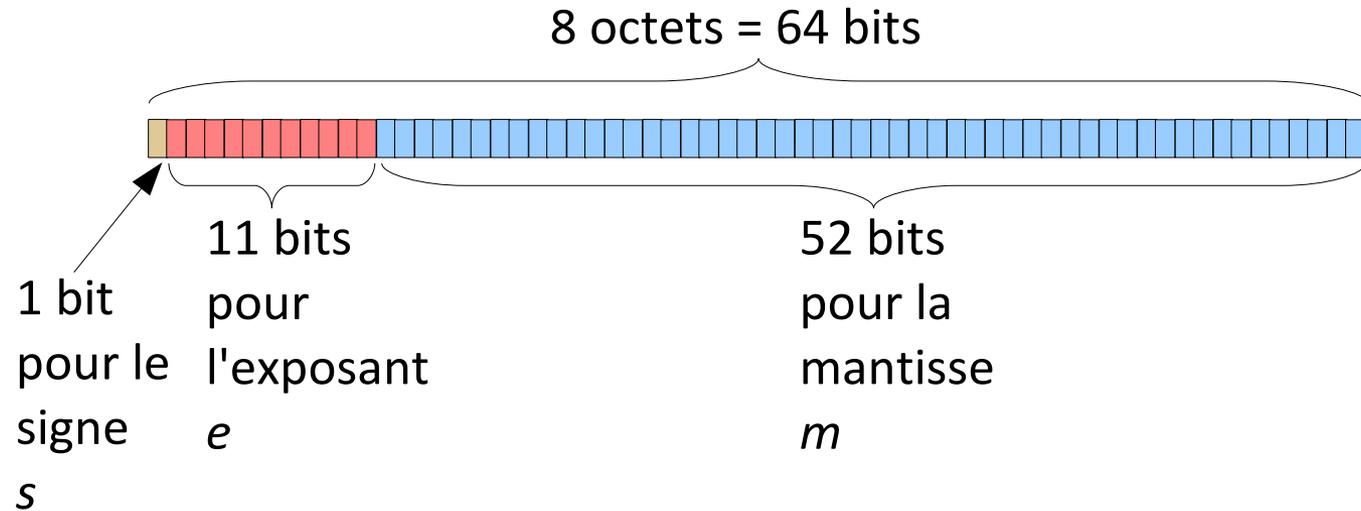
1 To (téraoctet) = 1024 Go (disque dur externe)



Le codage des entiers Java



Le codage des flottants double précision



$$x = (-1)^s m 2^{e-1023}$$

Autres codages

- **Chaînes de caractères**

- ASCII : 7 bits, caractères simples codés de 32 à 127
- ANSI : 8 bits, caractères simples codés de 32 à 127, caractères accentués de 128 à 255
- UTF-8 : de 1 à 4 octets

- **Couleurs d'une image**

- RGB : “red, blue, green”, 1 octet pour chacun :
 - valeurs entre 0 et 255
 - codage hexadécimal avec 2 symboles

Codage hexadécimal

La "minute culturelle"

Hexadécimal : en base 16 (ἕξάς : six, decem : dix)

Codé par les chiffres de 0 à 9 et les lettres

A	B	C	D	E	F
↓	↓	↓	↓	↓	↓
10	11	12	13	14	15

- Deux symboles pour un octet :

16^2 valeurs possibles = 256

- Utilisé pour coder les couleurs en HTML :

couleur="#RRGGBB"

rouge="#FF0000", vert="#00FF00"

#800080 ?

Codage hexadécimal

La "minute culturelle"

Hexadécimal : en base 16 (ἕξάς : six, decem : dix)

Codé par les chiffres de 0 à 9 et les lettres A B C D E F

	A	B	C	D	E	F
	↓	↓	↓	↓	↓	↓
	10	11	12	13	14	15

- Deux symboles pour un octet :

16^2 valeurs possibles = 256

- Utilisé pour coder les couleurs en HTML :

couleur="#RRGGBB"

rouge="#FF0000", vert="#00FF00"

#800080 ?



Retour sur l'intérêt du pseudo-code

Intérêts du pseudo-code :

- Clair, lisible
- Pédagogique
- Indépendant du langage de programmation
- Pour distinguer le fond de la forme

Pseudo-code	Java
Addition Entrées : entiers i et j Sortie : somme $i+j$ (entier) Début ... renvoyer ... Fin	public static int Addition(int i , int j){ ... return ... }
Variables : entier i	int i ;
$i \leftarrow 1$	$i = 1$;
Si $i=1$ alors ... Sinon ... FinSi	if ($i==1$){ ... } else { ... }
Tant que $i<3$: ... Fin TantQue	while ($i<3$) { ... }

Les booléens

- Opérations sur les booléens :
et, ou, non

ET	VRAI	FAUX
VRAI	VRAI	FAUX
FAUX	FAUX	FAUX

OU	VRAI	FAUX
VRAI	VRAI	VRAI
FAUX	VRAI	FAUX

NON	VRAI	FAUX
	FAUX	VRAI

Les booléens

- Opérations sur les booléens :
et, ou, non

ET	VRAI	FAUX
VRAI	VRAI	FAUX
FAUX	FAUX	FAUX

OU	VRAI	FAUX
VRAI	VRAI	VRAI
FAUX	VRAI	FAUX

NON	VRAI	FAUX
VRAI	FAUX	VRAI

Opérations sur les booléens codées
sur les entiers 0 et 1 :

x	1	0
1	1	0
0	0	0

max(1,...+...)	1	0
1	1	1
0	1	0

1-x	1	0
1	0	1

Les opérations de base en Java

- Type entier *int*

+, -, *, /, (division entière), % (reste modulo), ^ (puissance)

- Type flottant *float*, ou *double* (plus précis)

+, -, *, /

- Type booléen *boolean*

false (faux), true (vrai), && (et), || (ou), ! (non)

- Type chaîne de caractères *string*

+ (concaténation : "INF"+"120"="INF120")