DUT MMI – IUT de Marne-la-Vallée
02/10/2013
M1202 - Algorithmique

Cours 2
Variables et affectations, type et codage

UNIVERSITÉ PARIS-EST MARNE-LA-VALLÉE

iut Marne-la-Vallée

Philippe Gambette

Sources

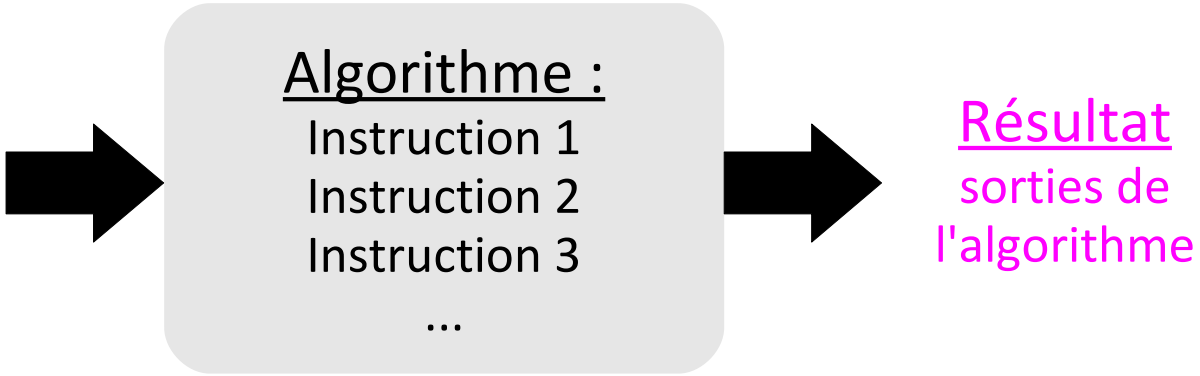
- *Le livre de Java premier langage*, d'A. Tasso
- Cours INF120 de J.-G. Luque
- Cours FLIN102 de l'Université Montpellier 2
- Cours de J. Henriet : http://julienhenriet.olympe-network.com/Algo.html
- http://xkcd.com, http://xkcd.free.fr

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

- Résumé des épisodes précédents
- Premier algorithme
- 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



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

Pour tester un algorithme : on fait la **trace**.

Noms des variables

Dans un **algorithme**, choisir pour les variables :

- un nom composé de **lettres** et éventuellement de **chiffres**
- un nom **expressif**, par exemple :
  - *chaine*, *requête1...* pour une chaîne de caractères
  - *n*, *a*, *b*, *compteur*, *nbOperations*, *longueur...* pour un entier
  - *x*, *y*, *température* pour un réel
  - *estEntier*, *testEntier*, *trouvé...* pour un booléen
- un nom **assez court** (il faut l'écrire !)
- éviter les **noms réservés** : *pour*, *tant que*, *si*...

Dans un **programme** :

- **éviter** les lettres accentuées et la ponctuation
- préférer l'**anglais** si votre code source est diffusé largement
- être **expressif** et **lisible** :
  - *est_entier* ou *estEntier* plutôt que *estentier*

Votre code sera relu, par vous ou par d'autres...

Mon premier vrai algorithme

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

Intuition :

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

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

Mon premier vrai algorithme

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

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sortie : le produit de *entier1* et *entier2*
Variables :

Début

Fin

pseudo-code

Mon premier vrai algorithme

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

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sortie : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *produit*

```
Début
    compteur ← 0
    produit ← entier1
    compteur ← 1
    Tant que compteur ≤ entier2 faire :
        produit ← addition(produit, entier1)
    Fin tant que
    renvoyer produit
Fin
```

Mon premier vrai algorithme

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

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sortie : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *produit*

```
Début
    compteur ← 0
    produit ← entier1
    compteur ← 1
    Tant que compteur ≤ entier2 faire :
        produit ← addition(produit, entier1)
    Fin tant que
    renvoyer produit
Fin
```

Terminaison ?

La condition de sortie de boucle n'est jamais vérifiée car *compteur* ne varie pas.

L'algorithme **ne termine pas**.

Mon premier vrai algorithme

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

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sortie : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *produit*

```
Début
    compteur ← 0
    produit ← entier1
    compteur ← 1
    Tant que compteur ≤ entier2 faire :
        produit ← addition(produit, entier1)
        compteur ← addition(compteur, 1)
    Fin tant que
    renvoyer produit
Fin
```

Mon premier vrai algorithme

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

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sortie : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *produit*

```
Début
    compteur ← 0
    produit ← entier1
    compteur ← 1
    Tant que compteur ≤ entier2 faire :
        produit ← addition(produit, entier1)
        compteur ← addition(compteur, 1)
    Fin tant que
    renvoyer produit
Fin
```

Terminaison ?

Oui car *compteur* augmente progressivement jusqu'à arriver à *entier2*.

Plus formellement, l'ensemble des valeurs successives de (*entier2* – *compteur*) (à la fin de chaque boucle) est une suite d'entiers positifs strictement décroissante.

L'algorithme termine.

Mon premier vrai algorithme

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

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sortie : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *produit*

```
Début
    compteur ← 0
    produit ← entier1
    compteur ← 1
    Tant que compteur ≤ entier2 faire :
        produit ← addition(produit, entier1)
        compteur ← addition(compteur, 1)
    Fin tant que
    renvoyer produit
Fin
```

Correction ?

Essayons avec l'exemple :
entier1 = 5 et *entier2* = 3

Tableau des valeurs des variables avant le début de la *i*-ième boucle Tant que :

i	1	2	3	4
compteur	1	2	3	4
résultat	5	10	15	20
entier1	5	5	5	5
entier2	3	3	3	3

La boucle n'est exécutée que 3 fois mais on renvoie 20 : l'algorithme **n'est pas correct !**

Mon premier vrai algorithme

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

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sortie : le produit de *entier1* et *entier2*
Variables : entiers *compteur* et *produit*

```
Début
    compteur ← 0
    produit ← entier1
    compteur ← 1
    Tant que compteur < entier2 faire :
        produit ← addition(produit, entier1)
        compteur ← addition(compteur, 1)
    Fin tant que
    renvoyer produit
Fin
```

Correction ?

Essayons avec l'exemple :
entier1 = 5 et *entier2* = 3

Tableau des valeurs des variables avant le début de la *i*-ième boucle Tant que :

i	1	2	3	4
compteur	1	2	3	-
résultat	5	10	15	-
entier1	5	5	5	-
entier2	3	3	3	-

La boucle n'est exécutée que 2 fois et on renvoie 15 : l'algorithme semble **correct...**

... mais ne l'est pas pour *entier2*=0

Mon premier vrai algorithme

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

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sortie : 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
```

Mon premier vrai algorithme

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

Multiplication :

Entrées : deux entiers *entier1* et *entier2*
Sortie : 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
```

Correction ?

entier1 = 5 et *entier2* = 3

Tableau des valeurs des variables avant le début de la *i*-ième boucle Tant que :

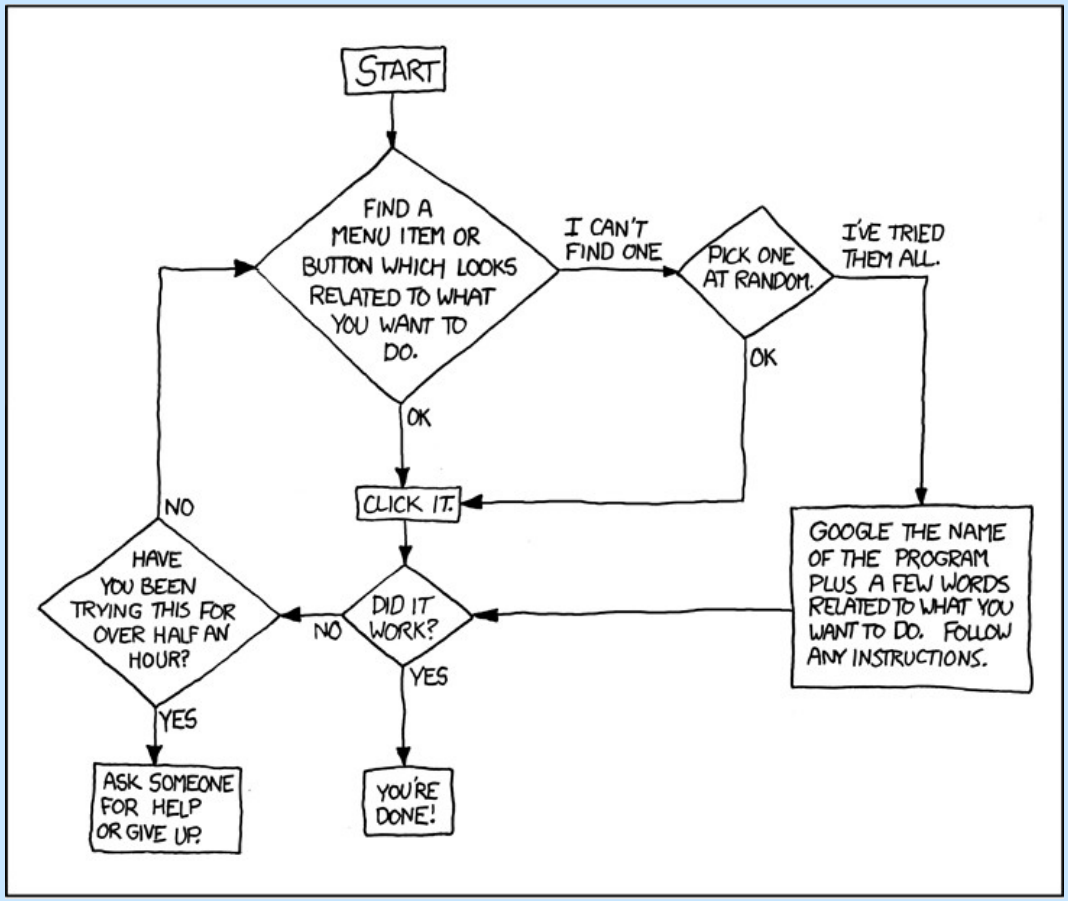
i	1	2	3	4
compteur	0	1	2	3
produit	0	5	10	15
entier1	5	5	5	5
entier2	3	3	3	3

On remarque que *produit* est égal à *compteur* x *entier1* tout au long de l'algorithme.
Or à la fin de l'algorithme *compteur*=*entier2* donc l'algorithme est correct.

Organigramme de résolution de tout problème logiciel

La “minute xkcd”

Chers parents, grands parents, collègues, et autres non-informaticiens variés.
Nous ne savons pas tout faire dans tous les logiciels comme par magie.
Quand on vous aide, en général on ne fait que ça :



http://xkcd.com/627
http://xkcd.free.fr?id=627

Merci d'imprimer cet organigramme et de le scotcher à côté de votre écran. Félicitations, vous êtes maintenant l'expert du coin en informatique !

Dictionnaire pseudo-code / Java

	Pseudo-code	Java

Déclaration d'un algorithme	**Addition**
	Entrées : entiers *i* et *j*
	Sortie : entier
	Début
	…
	renvoyer …
	Fin
		public static int **Addition**(int i, int j){
		…
		
		return …
		}

Déclaration d'une variable	Variables : entier *i*	int *i*;
Affectation	*i* ← 1	*i* = 1;

Test	Si *i*=1 alors :
	…
	Sinon :
	…
	FinSi
		if (*i*==1){
		…
		} else {
		…
		}

Boucle	Tant que *i*<3 :
	…
	Fin TantQue
		while (*i*<3) {
		…
		}

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 : entier
Début
…
renvoyer …
Fin
	public static int **Addition**(int *i*, int *j*){
	…
	
	return …
	}

Variables : entier *i*	int *i*;
i ← 1	*i* = 1;

Si *i*=1 alors
…
Sinon
…
FinSi
	if (*i*==1){
	…
	} else {
	…
	}

Tant que *i*<3 :
…
Fin TantQue
	while (*i*<3) {
	…
	}

Le calcul des puissances de 2

Je connais le calcul de la multiplication par 2 (en Java : *2).

Comment calculer les puissances de 2 ? 1, 2, 4, 8, 16, 32, …

DeuxPuissance(4) =

Intuition :

Tant qu'on n'est pas à la puissance voulue, on ...

→ Au total, pour **DeuxPuissance**(*a*), on fait ... multiplications par 2.

Le calcul des puissances de 2

Je connais le calcul de la multiplication par 2 (en Java : *2).

Comment calculer les puissances de 2 ? 1, 2, 4, 8, 16, 32, …

DeuxPuissance(4) = $\underbrace{2*2*2*2}_{\text{4 fois}}$ = 16

Intuition :

Tant qu'on n'est pas à la puissance voulue, on multiplie par 2.

→ Au total, pour **DeuxPuissance**(*a*), on fait *a* multiplications par 2.

En français :

Je multiplie *a* fois par 2 l'entier 1.

Le calcul des puissances de 2

Intuition :

DeuxPuissance(4) = 2 * 2 * 2 * 2

En français :

Je multiplie *a* fois par 2 l'entier 1.

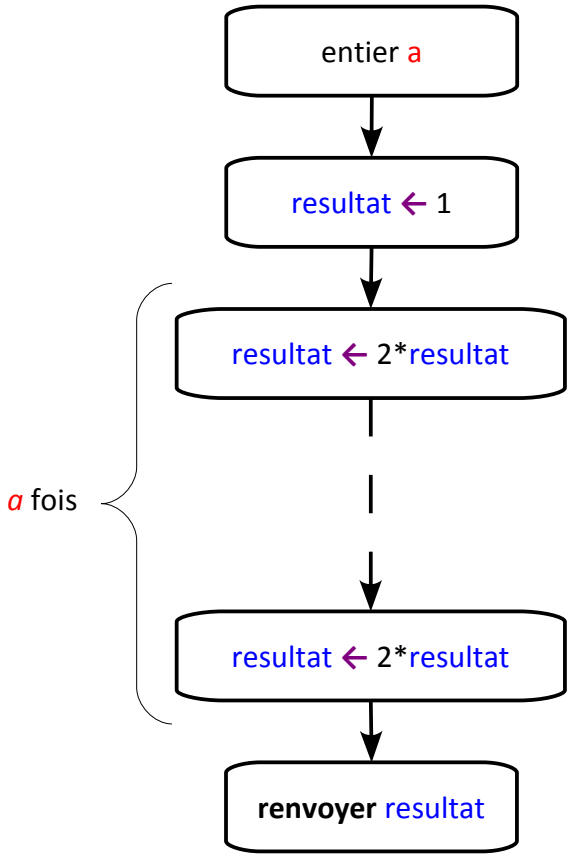
Le calcul des puissances de 2

Intuition :

DeuxPuissance(4) = 2 * 2 * 2 * 2

En français :

Je multiplie *a* fois par 2 l'entier 1.



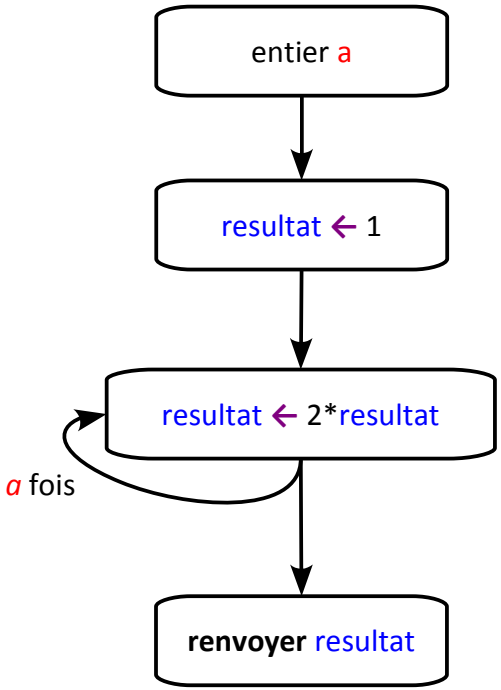
Le calcul des puissances de 2

Intuition :

DeuxPuissance(4) = 2 * 2 * 2 * 2

En français :

Je multiplie *a* fois par 2 l'entier 1.



Le calcul des puissances de 2

Intuition :

DeuxPuissance(4) = 2 * 2 * 2 * 2

En français :

Je multiplie *a* fois par 2 l'entier 1.

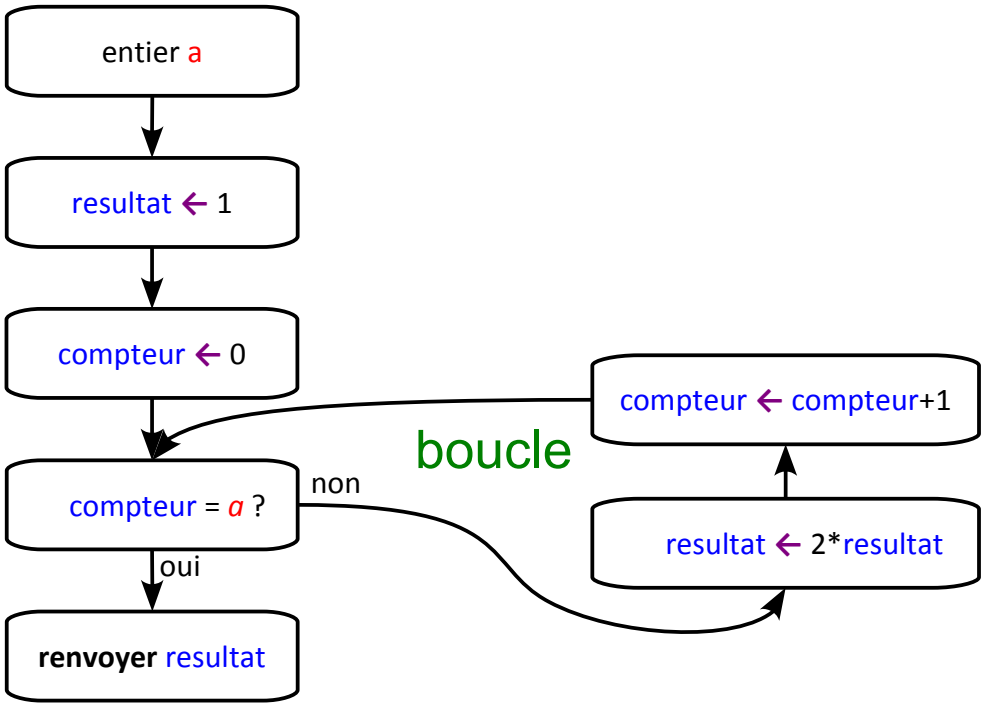
Le calcul des puissances de 2

Intuition :

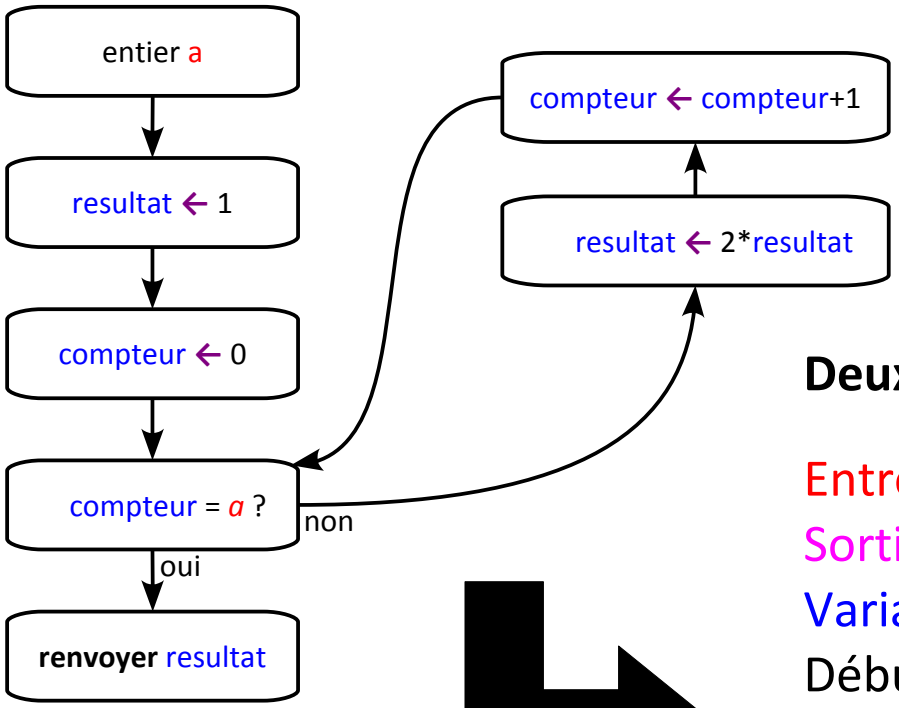
DeuxPuissance(4) = 2 * 2 * 2 * 2

En français :

Je multiplie *a* fois par 2 l'entier 1.



De l'organigramme au pseudo-code



DeuxPuissance :

Entrée :

Sortie :

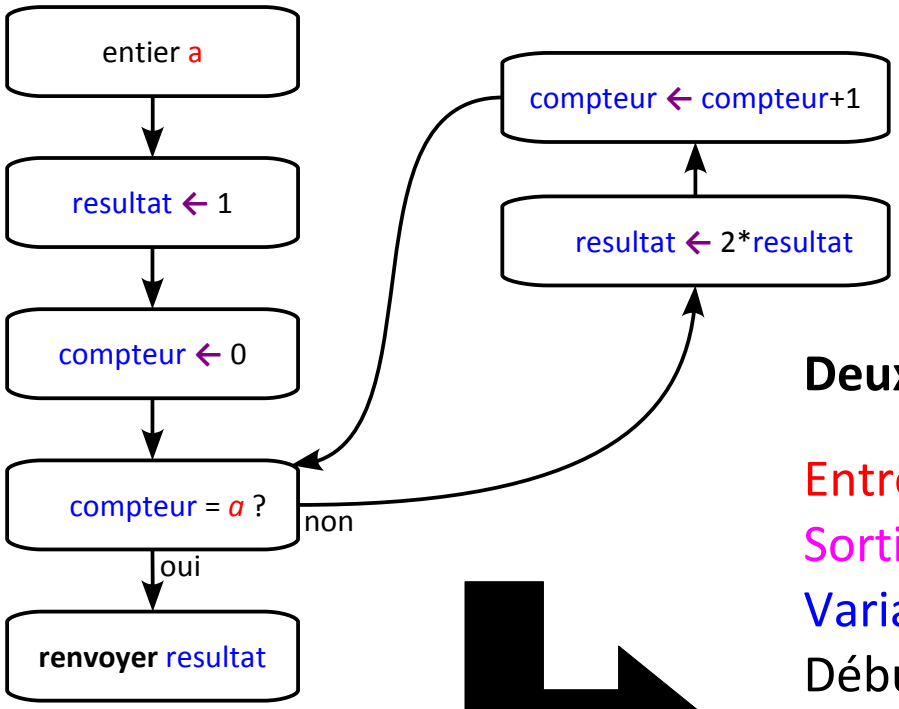
Variables :

Début

renvoyer

Fin

De l'organigramme au pseudo-code



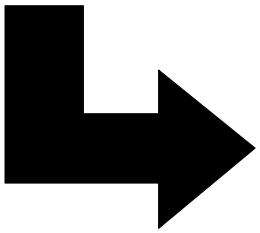
DeuxPuissance :

Entrée : un entier *a*
Sortie : un entier
Variables : entiers *compteur* et *resultat*
Début
    compteur ← 0
    resultat ← 1
    Tant que *compteur* < *a* faire :
        resultat ← 2**resultat*
        compteur ← *compteur* +1
    Fin tant que
    renvoyer *resultat*
Fin

Du pseudo-code au code Java

DeuxPuissance :

Entrée : un entier *a*
Sortie : un entier
Variables : entiers *compteur* et *resultat*
Début
    compteur ← 0
    resultat ← 1
    Tant que *compteur* < *a* faire :
        resultat ← 2**resultat*
        compteur ← *compteur* +1
    Fin tant que
    renvoyer *resultat*
Fin

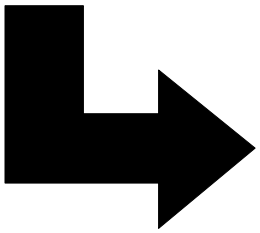


```
public static int
DeuxPuissance(          ){

   return            ;

}
```

Du pseudo-code au code Java

DeuxPuissance :

Entrée : un entier *a*
Sortie : un entier
Variables : entiers *compteur* et *resultat*
Début
    compteur ← 0
    resultat ← 1
    Tant que *compteur* < *a* faire :
        resultat ← 2**resultat*
        compteur ← *compteur* +1
    Fin tant que
    renvoyer *resultat*
Fin

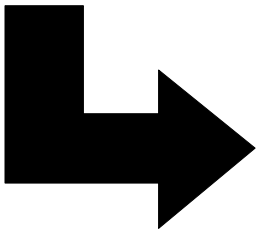


```
public static int
DeuxPuissance(int a){

    int compteur=0;
    int resultat=1;
    while (compteur<a){
        resultat=2*resultat;
        compteur=compteur+1;
    }
    return resultat;

}
```

Du pseudo-code au code Java

DeuxPuissance :

Entrée : un entier *a*
Sortie : un entier
Variables : entiers *compteur* et *resultat*
Début
    compteur ← 0
    resultat ← 1
    Tant que *compteur* < *a* faire :
        resultat ← 2**resultat*
        compteur ← *compteur* +1
    Fin tant que
    renvoyer *resultat*
Fin



```
public static int
DeuxPuissance(int a){

    int compteur=0;
    int resultat=1;
    while (compteur<a){
        resultat=2*resultat;
        compteur=compteur+1;
    }
    return resultat;

}
```

Trace de **DeuxPuissance**(4) :

Valeurs des variables `compteur` et `resultat` après le i-ième passage dans la boucle `while` :

i	compteur	resultat
0	0	1
1	1	2
2	2	4
3	3	8
4	4	16

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 ← 1
    Tant que *i* > 0 faire :
        i ← 2 x *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 ← 1
    Tant que *i* > 0 faire :
        i ← 2 x *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 ← 1
    Tant que *i* > 0 faire :
        i ← 2 x *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
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!!

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!!

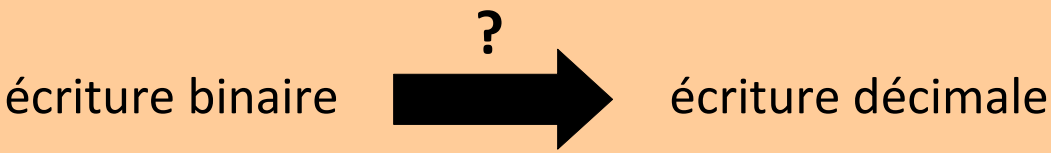
Les entiers "int" Java ne sont pas de "vrais entiers" mais des entiers entre -2 147 483 648 et 2 147 483 647

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

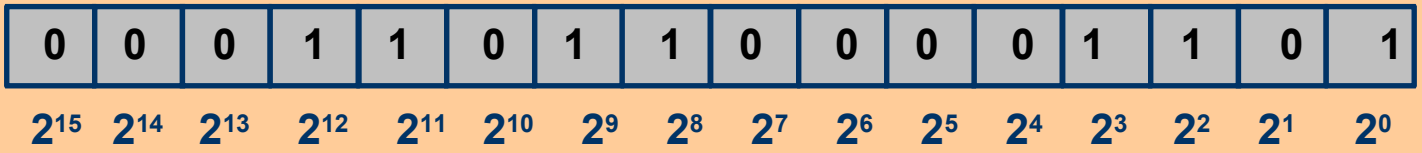


Le codage des entiers en binaire

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

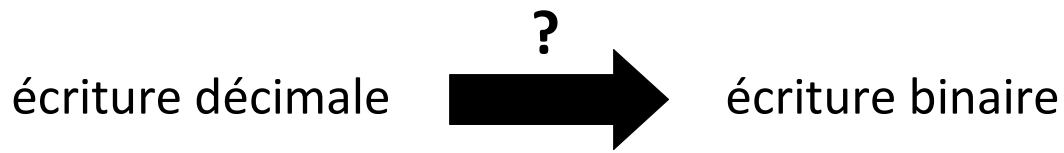


$2^{12} + 2^{11} + 2^{9} + 2^{8} + 2^{3} + 2^{2} + 2^{0} = 4096 + 2048 + 512 + 256 + 8 + 4 + 1 = 6925$

écriture binaire → écriture décimale

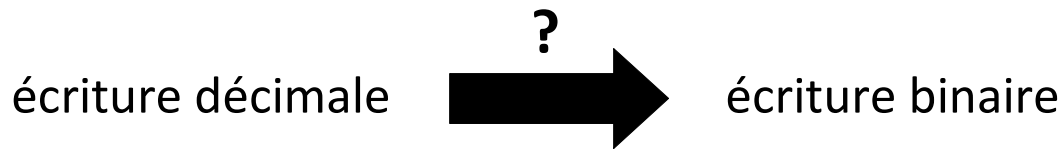
Le codage des entiers en binaire

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**.



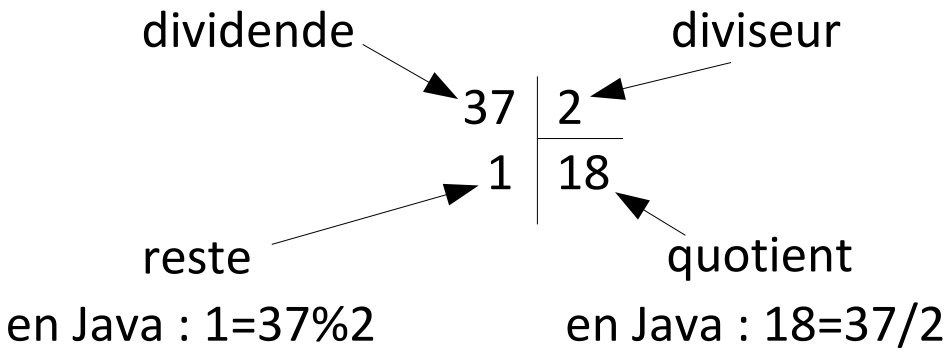
Le codage des entiers en binaire

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 ?

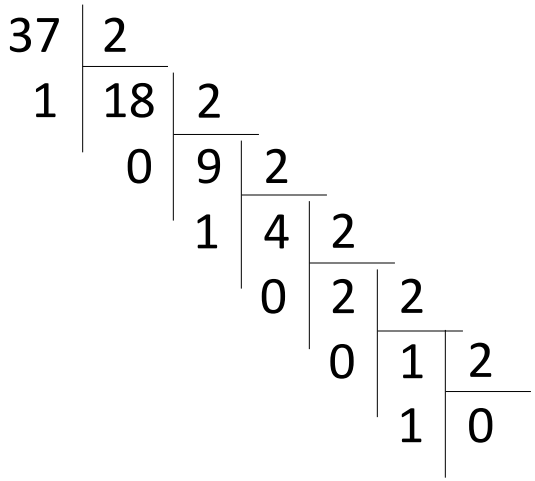
Division euclidienne :



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**.

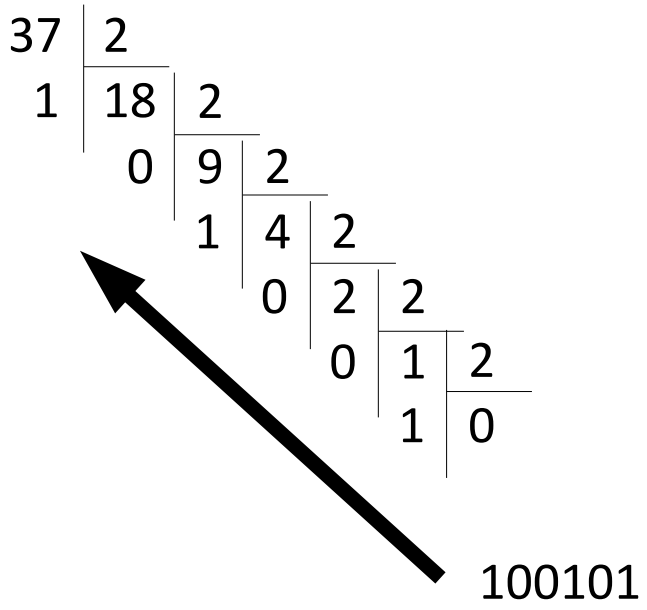
écriture décimale **?** écriture binaire



Le 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

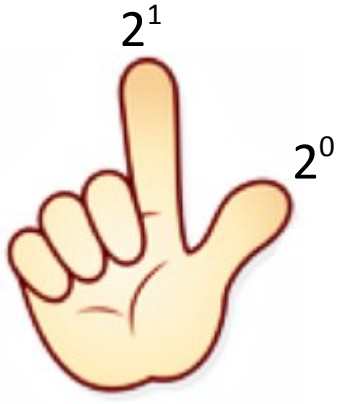
Pour faire son geek :

- 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}



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

Source image : http://www.so-xperts.com/blog/2011/04/performance-web-et-marketing-le-pouce-et-lindex/

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 données en binaire : $2^{10} = 1024$ donc $2^{10} \approx 1000$

③ → $2^{32} \approx ?$

① $a^{b \times c} = (a^b)^c$

② $a^{b+c} = a^b \times a^c$

$2^{32} = 2^{30+2}$ ②

$= 2^{30} \times 2^2$

$= 2^{10 \times 3} \times 4$

$= (2^{10})^3 \times 4$ ①

$\approx 1000^3 \times 4$

$\approx$ 4 milliards

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 :

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.

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

Le codage des entiers en mémoire

1 bit = 0 ou 1

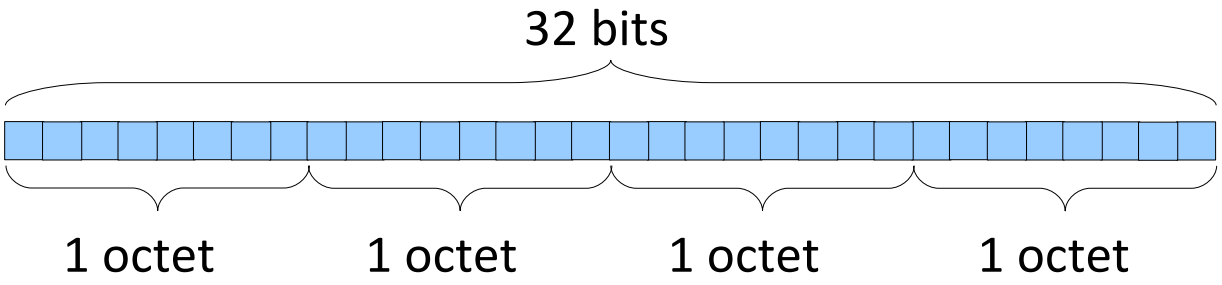
1 octet = 8 bits

1 Ko (kilooctet) = 1024 octets

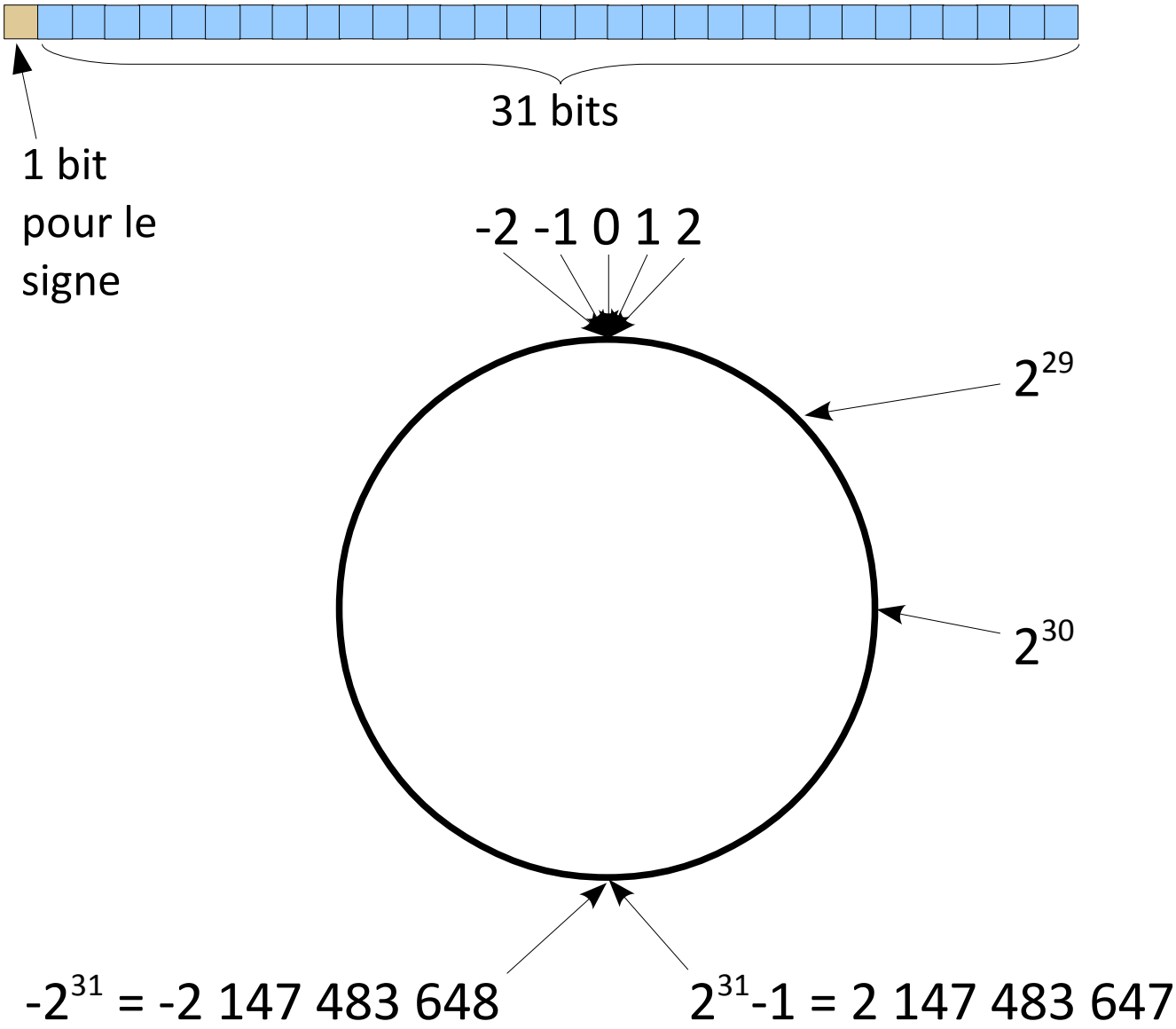
1 Mo (mégaoctet) = 1024 Ko (disquette)

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

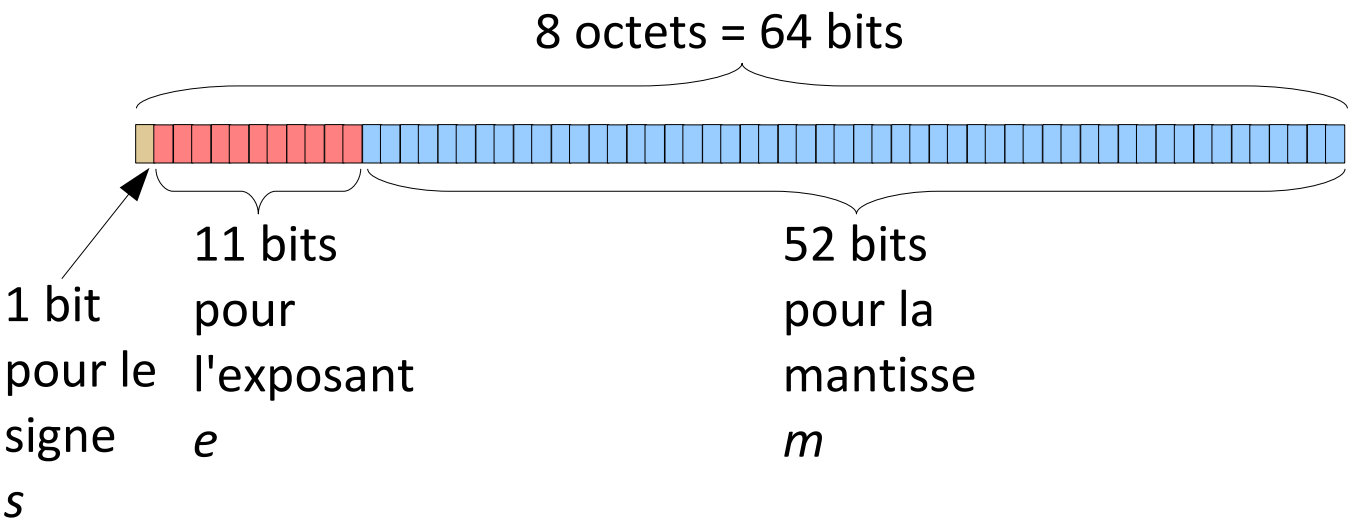
1 To (téraoctet) = 1024 Go (disque dur externe)



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

Codage hexadécimal

La “minute culturelle”

Hexadécimal : en base 16 (ἑξάς : 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 (ἑξάς : 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 ? 

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
	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
	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”)