Trois p'tits tours et puis s'en vont

Jean-Jacques.Levy@inria.fr

http://www.jeanjacques-levy.com/

tel: 01 39 63 56 89

Catherine Bensoussan

cb@lix.polytechnique.fr

Aile 00, LIX

tel: 34 67

http://w3.edu.polytechnique.fr/informatique/

Plan

1. **Instructions d'itérations**
2. **Grammaire BNF**
3. **Tableaux**
4. **Ligne de commande**
5. **Fonctions statiques**
6. **Fonctions prenant des tableaux en arguments**

Algorithme d'Euclide

```
class PGCD {
  public static void main (String[ ] args) {
    if (args.length != 2)
      System.out.println ("Mauvais nombre d'arguments.");
    else {
      int a = Integer.parseInt (args[0]);
      int b = Integer.parseInt (args[1]);
      if (b == 0)
        System.out.println (a);
      else {
        while (a % b != 0) {
          int x = a % b;
          a = b;
          b = x;
        }
        System.out.println (b);
      }
    }
  }
}
```

Instructions d'itération

- `while (`e`)` *instruction*
- `for (`*affectation*$_1$`;` e`;` *affectation*$_2$`)` *instruction*
- cette dernière instruction est équivalente à

  affectation$_1$;
  `while (`e`) {`
     instruction
     affectation$_2$;
  `}`

Exemple: calcul de factorielle

```
long b = 1;
for (long a = 1; a <= n; ++a)
    b = b * a;
```

Remarque: `++a` équivalent à `a = a + 1`

Exemples d'itération

Calcul de la plus petite valeur dont la factorielle est négative!

```
while (b > 0) {
    b = b * a;
    ++a;
}
```

(Pour `int`**,** $18! < 0$ **et pour** `long`**,** $22! < 0$**) Dans la même veine, la suite harmonique converge (lentement) en informatique. Pourquoi?**

Autre exemple, calcul de $\sqrt{a}$**, par la formule**

$$u_{n+1} = \frac{1}{2}(u_n + \frac{a}{u_n})$$

Toute boucle `while` peut s'écrire avec un `for`

Exemple du pgcd: la boucle interne

```
while (a % b != 0) {
  int x = a % b;
  a = b;
  b = x;
}
```

peut aussi s'écrire

```
for ( ; a % b != 0; ) {
  int x = a % b;
  a = b;
  b = x;
}
```

ou encore

```
for ( ; a % b != 0; int x = a % b, a = b, b = x)
    ;
```

(L'instruction du corps de la boucle est alors l'instruction vide)

Grammaire BNF de Java (*Backus Naur Form*)

La grammaire de Java est complètement spécifiée, comme pour ∼tous les langages de programmation.

```
ForStatement:
        for ( ForInit_opt ; Expression_opt ; ForUpdate_opt )
                Statement

ForInit:
        StatementExpressionList
        LocalVariableDeclaration

ForUpdate:
        StatementExpressionList

StatementExpressionList:
        StatementExpression
        StatementExpressionList , StatementExpression
```

StatementExpression:

- Assignment
- PreIncrementExpression
- PreDecrementExpression
- PostIncrementExpression
- PostDecrementExpression
- MethodInvocation
- ClassInstanceCreationExpression

Tableaux

Les tableaux décrivent des ensembles de valeurs d'un même type. Deux notations sont tolérées. (Nous préférerons la 1ère).

- *type*`[ ]` *nom_du_tableau*`;`
- *type nom_du_tableau*`[ ];`

Deux manières pour initialiser un tableau

- `int[ ] x = {1, 4, 9, 16, 25, 36, 49 };`
- `int[ ] x = new int [7];`

Dans le deuxième cas, `x` a 7 éléments dont on ne connait pas la valeur.

Les éléments `x[0]`, `x[1]`, `x[2]`, `x[3]`, `x[4]`, `x[5]`, `x[6]` du tableau `x` s'obtiennent par indexation à partir de 0.

La longueur du tableau `x` est `x.length`

Initialisation dynamique

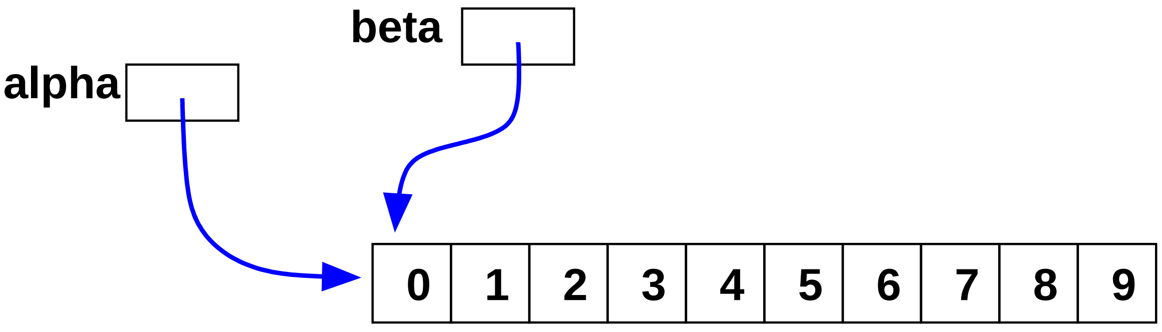
```
int n = 100;
int[ ] alpha = new int[n];
for (int i = 0; i < n; ++i)
   alpha[i] = i;
```

ou

```
int[ ] alpha = {0, 1, 2, 3, 4, 5, 6, 7, 9};

int[ ] beta = alpha;
```

Graphiquement:



Arguments de la ligne de commande

Le paramètre `args` de la fonction principale `main` est par convention un tableau de chaînes de caractères, contenant les arguments de la ligne de commande.

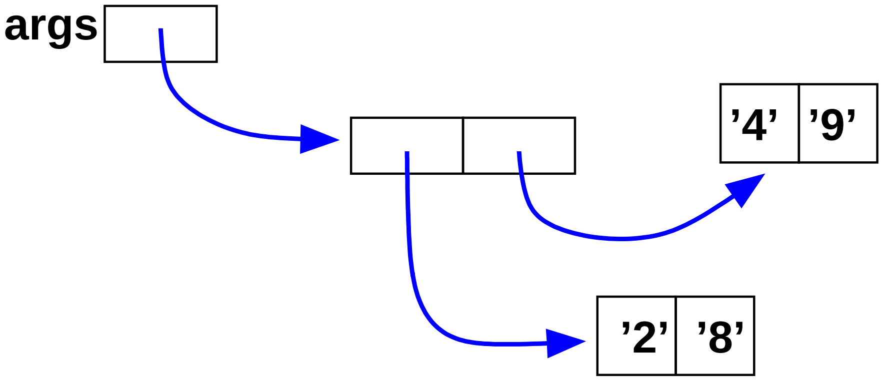
```
class PGCD {
  public static void main (String[ ] args) {
    if (args.length != 2)
      System.out.println ("Mauvais nombre d'arguments.");
    else {
      int a = Integer.parseInt (args[0]);
      int b = Integer.parseInt (args[1]);
      if (b == 0)
        System.out.println (a);
      else {
        while (a % b != 0) { ... }
        System.out.println (b);
      }
    }
  }
}
```

Arguments sur la ligne de commande – bis

Dans le cas précédent, les arguments de la ligne de commande sont ceux qui figurent sur la ligne après le nom de la classe contenant la fonction `main`

```
java PGCD 28 49
```

- Le nombre d'arguments est donc `args.length`
  Dans l'exemple, `args.length == 2`.
- Le 1er argument est `arg[0]`, le 2ème `arg[1]`, etc
  Dans l'exemple `arg[0]` vaut `"28"` et `arg[1]` vaut `"49"`.
- Le type chaîne de caractères est `String` (cf plus tard)



Fonctions

Les fonctions permettent d'abstraire un ensemble d'instructions. Comme en math, une fonction a des arguments et un résultat.

En Java, les fonctions sont aussi appelées méthodes car c'est la terminologie dans les langages orientés-objets. Nous utilisons les deux appelations.

```
class PGCD {
  public static void main (String[ ] args) {
      int a = Integer.parseInt (args[0]);
      int b = Integer.parseInt (args[1]);
      System.out.println (plus (a, b));
  }

  static int plus (int x, int y) {
      return x + y;
  }
}
```

La déclaration d'une fonction commence par le mot-clé `static`, puis vient le type du résultat, puis le nom de la fonction, puis la déclaration des paramètres, et enfin le corps de la définition entre accolades.

Deux fonctions sur les tableaux

```
static int sigma (int[ ] a) {
  int s = 0;
  for (int i = 0; i < a.length; ++i)
    s = s + a[i];
  return s;
}

static int[ ] somme (int[ ] a, int[ ] b) {
  int[ ] c = new int [Math.min (a.length, b.length)];

  for (int i = 0; i < c.length; ++i)
    c[i] = a[i] + b[i];
  return c;
}
```

sigma: $\texttt{int}[\,] \mapsto \texttt{int}$

somme: $\texttt{int}[\,] \times \texttt{int}[\,] \mapsto \texttt{int}[\,]$

Exercice calculer le produit scalaire.

Encore une autre fonction sur les tableaux

```
static void imprimer (int[ ] a) {
  for (int i = 0; i < a.length; ++i) {
    System.out.print(a[i]);
    System.out.print(" ");
  }
  System.out.println();
}
```

ou encore

```
static void imprimer (int[ ] a) {
  for (int i = 0; i < a.length; ++i)
    System.out.print(a[i] + " ");
  System.out.println();
}
```

Remarque: + est le seul opérateur surchargé de Java. Le sens de + est normal quand ses arguments sont numériques, mais dès qu'un de ses arguments est une chaîne de caractères, le 2ème argument est transformé en chaîne de manière naturelle et le résultat est la concaténation des deux chaînes de caractères.

Et Encore une autre

```
static int[ ] tableauAleatoire (int n) {
  int a[ ] = new int[n];
  for (int i = 0; i < a.length; ++i)
      a[i] = (int) (Math.random() * 100);
  return a;
}
```

Arguments des fonctions

- Java ne copie jamais implicitement des données complexes. Donc, un tableau passé en argument d'une fonction n'est jamais copié. (En fait, c'est son adresse mémoire qui est passée en argument).
- Java implémente l'appel par valeur. Autrement dit, on ne fait que passer la valeur des arguments.

Matrices

- **Une matrice est un tableau de tableaux (lignes puis colonnes)**
- **La déclaration ressemble à celle des tableaux unidimensionnels**

```
float[ ][ ] matrice;
```

- **L'initialisation a deux formes:**

```
float[ ][ ] matrice = {{1, 2}, {3, 4}};
float[ ][ ] matrice = new float[2][2];
```

Fonctions sur les matrices

```
static float[ ][ ] somme (float[ ][ ] a, float[ ][ ] b) {
  float[ ][ ] c = new float [a.length][a[0].length];

  for (int i = 0; i < c.length; ++i)
    for (int j = 0; j < b[0].length; ++j) {
      c[i][j] = a[i][j] + b[i][j];
  return c;
}

static float[ ][ ] produit (float[ ][ ] a, float[ ][ ] b) {
  int m = a.length, n = b[0].length, p = a[0].length;
  float[ ] c = new float [m][n];

  for (int i = 0; i < m; ++i)
    for (int j = 0; j < n; ++j) {
      c[i][j] = 0;
      for (int k = 0; i < p; ++k)
          c[i][j] = c[i][j] + a[i][k] * b[k][j];
    }
  return c;
}
```

Vérifier dimensions compatibles dans les deux fonctions.

Le beau monde du multimédia