Tronc Commun Informatique

Dominique Perrin

X & Université de Marne la Vallée
perrin@univ-mlv.fr

Menu

1. Amphi 0
2. 10 petites classes
3. 10 t.p.
4. une composition HC
5. une composition
6. un projet

Bibliographie

Algorithmique

Danièle Beauquier, Jean Berstel, Philippe Chretienne, *Eléments d'Algorithmique*, Masson, 1992.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Algorithms*, MIT Press, 1990.

Donald E. Knuth, *The Art of Computer programming, vol. 1: Fundamental Algorithms, vol. 2: Seminumerical Algorithms, vol. 3: Sorting and Searching*, Addison Wesley, 1973.

Jan van Leeuwen,*Handbook of Theoretical Computer Science*, vols. A et B, Elsevier, 1990.

Robert Sedgewick, *Algorithms in C*, Addison Wesley, 1991.

Langage Java

Samuel N. Kamin, M. Dennis Mickunas, Edward M. Reingold, *An Introduction to Computer Science Using Java*, McGraw Hill, 1998.

Ken Arnold, James Gosling, *The Java programming Language*, Addison Wesley, 1997.

David Flanagan, *Java Examples in a Nutshell*, O'Reilly, 1997.

Patrick Niemeyer, Joshua Peck, *Exploring Java*, O'Reilly, 1997.

Thomas A. Standish, *Data Structures in Java*, Addison Wesley, 1997.

Bonjour

```
class Bonjour {
  public static void main(String[] args) {
    System.out.println("Bonjour");
  }
}
```

Pour le compiler:

```
javac Bonjour.java
```

et pour l'exécuter :

```
java Bonjour
```

Factorielle

Calcul de la factorielle :

```
class Num {
  static int factorielle (int x) {
    int f = 1;
    for (int i= 2; i <= x; i++)
      f = f* i;
    return f;
  }

  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    System.out.println(factorielle(n));
  }
}
```

la fonction factorielle peut aussi s'écrire :

```
  static int factorielle (int x) {
    if (x <= 0) return 1;
    else return x * factorielle(x-1);
  }
```

L'algorithme d'Euclide

Le calcul du pgcd de deux entiers m, n par l'algorithme d'Euclide repose sur la division entière de m par n comme $m = nq + r$ et la règle facile à vérifier

$$\mathrm{pgcd}(m, n) = \mathrm{pgcd}(n, r)$$

On en déduit directement la méthode :

```
static int pgcd (int m, int n) {
  if (n == 0) return m;
  else return pgcd( n, m % n);
}
```

ou encore, sans appel récursif

```
static int pgcd(int m, int n) {
  int r;
  while (n != 0) {
    r= m % n; m= n; n= r;
  }
  return m;
}
```

Programme complet

```
class Num {
  static int pgcd(int m, int n) {
    if (n == 0) return m;
    else return pgcd(n, m % n);
  }
  public static void main(String[] args) {
    int a,b;
    a= Integer.parseInt(args[0]);
    b= Integer.parseInt(args[1]);
    System.out.println(pgcd(a,b));
  }
}
```

Exécution :

```
>javac Num.java
>java Num 8 13
1
>
```

Le pgcd complet

On cherche maintenant les entiers u, v tels que

$$um + vn = pgcd(m, n)$$

On peut procéder ainsi, en rangeant dans un tableau les entiers (p, u, v) :

```
static int[] pgcd2(int m, int n) {
  int a[]=new int[3];
  if (n == 0) {
    a[0]=m; a[1]=1; a[2]=0;
  } else {
    int temp;
    a= pgcd2(n, m % n);
    temp= a[2]; a[2]= a[1]- a[2]*(m/n);
    a[1]= temp;
  }
  return a;
}
```

De fait, si $u'n + v'r = p$, on aura

$$v'm + (u' - v'q)n = p.$$

Nombres complexes

```
class Complexe {
  double x, y;
  Complexe( double re, double im) {
    x= re; y= im;
  }
  double valeurAbsolue() {
    return Math.sqrt(x*x + y*y);
  }
  static Complexe
          somme(Complexe c, Complexe d) {
    return new Complexe(c.x + d.x, c.y + d.y);
  }
  static Complexe
         inverse( Complexe c) {
    double r=c.x*c.x + c.y*c*y;
    return new Complexe( c.x/r, -c.y/r);
  }
  String toString() {
     return x + " + " + y + "i";
  }
```

```
  static Complexe
        produit(Complexe c, Complexe d) {
    return
     new Complexe( c.x*d.x - c.y*d.y,
                   c.y*d.x+ c.y*d.x);
  }
  public static main (String[] args) {
    Complexe c d;
    c = new Complexe(1,2);
    System.out.println(somme(c,c));
    d= inverse(c);
    System.out.println(produit(c*d));
  }
}
```

Exécution :

```
2.0 + 4.0 i
1.0 + 0.0 i
```

Fractions continues

Un développement en *fraction continue* d'un nombre réel positif α est une suite $(a_0, a_1, \ldots)$ d'entiers naturels telle que:

$$\alpha = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots}}}$$

Quand α est un nombre rationnel m/n, la suite des quotients obtenus en appliquant l'algorithme d'Euclide à (m, n) est un développement en fraction continue de α.

En effet si $m = qn + r$, on a

$$\frac{m}{n} = q + 1/(\frac{n}{r})$$

On en déduit la méthode suivante, qui affiche les termes successifs du développement :

```
static void cont(int m, int n) {
  if (n != 0) {
    System.out.println( m / n);
    cont( n, m % n);
  }
}
```

Pour le développement d'un nombre réel α, on aura :

```
static void cont(double alpha, int n) {
  int a,i;

  for (i = 0; i< n; i++) {
    a =  alpha;
    System.out.println(a);
    alpha = 1/(alpha-a);
  }
}
```

Le résultat de l'appel de `cont(3.14159,4)` est

```
3
7
15
1
```

La classe FractionContinue

Si on veut manipuler des développements en fraction continue, on peut créer une classe dont les objets sont les suites d'entiers $(a_0, a_1, \ldots)$.

```
class FracContinue {
  int terme;
  FracContinue reste= null;

  FracContinue(double alpha, int n) {
    terme= (int) alpha;
    if (n > 0)
      suite=
       new FracContinue(1/(alpha-terme), n-1);
  }
  public  String toString() {
    String s= terme+ " ";
    if (reste != null)
      s= s + reste.toString();
  }
}
```

Continuants

La fraction

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}} = \frac{p_n}{q_n}$$

s'appelle le n-ième continuant. On le calcule par les relations de récurrence:

$$p_n = p_{n-1}a_n + p_{n-2},\ q_n = q_{n-1}a_n + q_{n-2}$$

avec $p_0 = a_0, p_1 = a_1 a_0 + 1, q_0 = 1, q_1 = a_1$ La suite des contnuants fournit une approximation rationnelle de nombres réels. Pour π, on a

$$[3, \frac{22}{7}, \frac{333}{106}, \frac{355}{113}, \frac{103993}{33102}, \frac{104348}{33215}, \frac{208341}{66317}, \frac{312689}{99532}, \frac{833719}{265381}, \frac{1146408}{364913}, \frac{4272943}{1360120}]$$

On peut aussi calculer les continuants récursivement :

```
int[] Continuant() {
 // calcul des  continuants
  int[] a= new int[2];
  if (this.reste == null) {
    a[0]=this.terme;
    a[1]=1;
  }
  else {
    int[] b= this.reste.Continuant();
    a[0]=this.terme*b[0] + b[1];
    a[1]= b[0];
  }
  return a;
}
```

L'éxecution de

```
FracContinue s= new FracContinue(Math.PI,4);
int a= s.Continuant();
System.out.println(a[0]+"/"+a[1]);
```

donne

103993/33102