Fichier `justifier.rb`

```ruby
def justifier( donnees, largeur )
  resultat = []

  sorte_donnees = :filename if donnees.class == String

  (PRuby::Pipeline.source( donnees, sorte_donnees ) |
   supprimer_blancs_inutiles |
   supprimer_commentaires |
   supprimer_lignes_vides_inutiles |
   generer_mots |
   paqueter_mots( largeur ) |
   ajouter_blancs( largeur ) |
   PRuby::Pipeline.sink( resultat ) ).
    run

  resultat
end
```

```ruby
def supprimer_blancs_inutiles
  lambda do |cin, cout|
    cin.each do |ligne|
      ligne.chomp!   # On supprime le '\n' si present.
      ligne.strip!   # On supprime les blancs en debut/fin de ligne
      cout << ligne
    end
    cout.close
  end
end

#
```

```ruby
def supprimer_commentaires
  lambda do |cin, cout|
    cin.each do |ligne|
      # On emet une ligne vide si vide: les lignes vides sont
      # importantes pour indiquer les paragraphes.
      if ligne.empty?
        cout << ""
      elsif ligne !~ /^%/
        cout << ligne.split("%").first.strip
      else
        # Ligne avec un juste on commentaire: on ne l'emet pas!
      end
    end
    cout.close
  end
end

#
```

```ruby
def supprimer_lignes_vides_inutiles
  lambda do |cin, cout|
    ligne_precedente_vide = false
    cin.each do |ligne|
      unless ligne.empty?
        cout << "" if ligne_precedente_vide
        cout << ligne
      end
      ligne_precedente_vide = ligne.empty?
    end
    cout.close
  end
end

#
```

```ruby
def generer_mots
  lambda do |cin, cout|
    cin.each do |ligne|
      DBC.assert ligne !~ /^\s+$/, "*** Ligne avec juste des blancs

      if ligne.empty?
        cout << ""
      else
        ligne.split(" ").each do |mot|
          cout << mot
        end
      end
    end
    cout.close
  end
end

#
```

```
def paqueter_mots( largeur )
  lambda do |cin, cout|
    nb_cars_sur_ligne = 0
    mots = []

    cin.each do |mot|
      if mot.empty? # Fin de paragraphe: on emet le paquet de mots
        cout << mots unless mots.empty?
        cout << [] # On indique le paragraphe.
        nb_cars_sur_ligne = 0
        mots = []
      else # Pas une fin de paragraphe.
        if nb_cars_sur_ligne + mot.size + mots.size > largeur
          # Ligne trop longue si on ajoutait le mot, et ce en tenan
          # compte des espaces inter-mots: on emet les mots.
          cout << mots
          nb_cars_sur_ligne = 0
          mots = []
        end
        mots << mot
        nb_cars_sur_ligne += mot.size
      end
    end

    # On emet la derniere ligne, possiblement incomplete.
    cout << mots unless mots.empty?
    cout.close
  end
end

#
```

```ruby
def ajouter_blancs( largeur )
  lambda do |cin, cout|
    cin.each do |mots|
      if mots.empty?
        cout << ""
      elsif mots.size == 1 || cin.peek == PRuby::EOS || cin.peek.em
        # Ligne suivie d'un saut de paragraphe (ou fin du texte):
        # emet les mots sans mettre d'espaces additionneles
        # (i.e., sans justification a droite).
        cout << mots.join( " " )
      else
        nb_blancs_a_distribuer =
          largeur - mots.map(&:size).reduce(&:+)
        DBC.assert nb_blancs_a_distribuer >= mots.size - 1

        min_blancs = nb_blancs_a_distribuer / (mots.size - 1)
        blancs_en_trop = nb_blancs_a_distribuer % (mots.size - 1)
        res = (0...mots.size - 1).reduce("") do |res, i|
          nb_blancs = min_blancs + (i < blancs_en_trop ? 1 : 0)
          res << mots[i] << " " * nb_blancs
        end
        res << mots.last

        DBC.ensure( res.size == largeur,
                    "*** Erreur ajouter_blancs: res = '#{res}' (#{r
                     largeur = #{largeur}" )
        cout << res
      end
    end
    cout.close
  end
end
```