Prédicats de triangles [ge02] - Exercice résolu

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 22 mai 2018

Table des matières

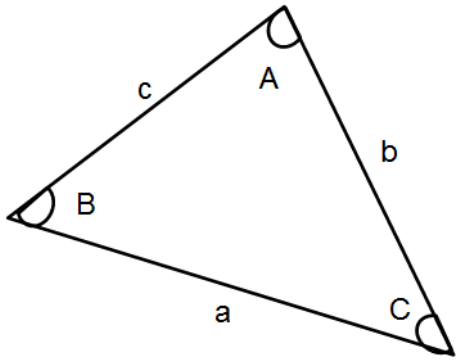
Java - Prédicats de triangles (Solution)

Mots-Clés Géométrie, Prédicats ■
Requis Structures de base, Structures conditionnelles, Algorithmes paramétrés ■
Difficulté • ○ ○ (1 h 30) ■

Objectif
Cet exercice travaille uniquement avec des fonctions booléennes, aussi nommées *prédicats*. Les valeurs de vérité qu'elles renvoient sont utilisées par un algorithme de caractérisation de familles de triangles.

1 Énoncé

Le triangle est une forme géométrique dont on a démontré une quantité incroyable de propriétés.



En voici quelques-unes :

- Les côtés a, b, c forment un triangle si la longueur du plus grand côté est inférieure à la somme des longueurs des deux autres côtés.
- La loi du cosinus :
$$\cos(A) = \frac{b^2 + c^2 - a^2}{2bc}$$
- Un triangle est rectangle si le carré de l'hypoténuse (le côté le plus long) est égal à la somme des carrés des deux autres côtés.
- Un triangle est équilatéral si ses trois côtés sont égaux.
- Un triangle est isocèle s'il possède deux et uniquement deux côtés égaux.
- Un triangle est scalène si les trois cotés sont de longueurs différentes.

Objectif

Écrire des fonctions puis les utiliser afin de caractériser la nature d'un triangle à partir des longueurs des trois côtés. De plus, dans le cas de triangles isocèle et scalène, préciser s'ils sont rectangles ou s'il possèdent un angle obtus (supérieur à 90°) ou seulement des angles aigus.

...(suite page suivante)...

2 Algorithmique, Programmation

2.1 Prédicat triangle

Il n'est pas toujours possible de prendre trois segments de longueur quelconque et de construire un triangle ayant ces segments pour côtés. Par exemple, le triplet (1,2,5) ne correspond pas à un triangle.

Propriété
Pour construire un triangle dont les côtés ont pour longueur trois réels donnés, il faut et il suffit que le plus grand soit inférieur ou égal à la somme des deux autres : **Inégalité triangulaire**.

Pré-conditions
Pour simplifier l'écriture des fonctions, les valeurs des trois paramètres doivent être strictement positives **et données dans un ordre non décroissant**, c.-à-d. que si $v1, v2, v3$ sont ces valeurs, elles doivent vérifier $0 < v1 \leq v2 \leq v3$. Cette contrainte fait partie de la *spécification* de la fonction, elle n'a pas à être vérifiée par la fonction.

Écrivez le **profil** d'une fonction `trtriangle(a,b,c)` qui renvoie `Vrai` si le triplet de réels `(a,b,c)` peuvent représenter les longueurs des trois côtés d'un triangle, `Faux` sinon.

Analyse
Définir son corps revient à formaliser la propriété d'*inégalité triangulaire* pour le plus grand des trois réels passés en paramètres. D'après la contrainte d'ordre imposée par l'énoncé, le plus grand des trois nombres est la valeur du troisième paramètre. La propriété se traduit donc sous la forme d'une simple inégalité entre les paramètres : $c \leq a+b$.

Écrivez le corps de la fonction.

Validez votre fonction avec la solution.

Solution Java @[pgtrpredicats.java]

```
/**
  Prédicat de triangle
  @param[in] a - longueur de segment
  @param[in] b - longueur de segment
  @param[in] c - longueur de segment
  @return Vrai si (0 < a <= b <= c) peuvent constituer les longueurs des cotés d'un
    triangle
*/

public static boolean trtriangle(double a, double b, double c)
{
  return (c <= a + b);
```

```
}
```

Écrivez une procédure `test_triangle` qui teste cette fonction avec les séries de valeurs suivantes :

- (1.0,2.1,2.5) : `Vrai` car $2.5 \leq 1 + 2.1$ est vrai
- (0.6,0.9,1.8) : `Faux` car $1.8 \leq 0.6 + 0.9$ est faux
- (0.8,1.7,2.5) : `Vrai` car $2.5 \leq 0.8 + 1.7$ est vrai (égalité)

Testez.

Validez votre procédure avec la solution.

Solution Java @[pgtrpredicats.java]

```
/**
  @test
*/

public static void test_triangle()
{
  trtriangle(1.0,2.1,2.5);
  trtriangle(0.6,0.9,1.8);
  trtriangle(0.8,1.7,2.5);
}
```

2.2 Les prédicats

Ce problème définit les fonctions booléennes qui prennent en paramètres trois réels, représentant les longueurs de segments d'un triangle, et qui renvoient `Vrai` ou `Faux` selon qu'ils définissent ou non un triangle ayant la propriété désignée par le nom de la fonction : `trplat`, `trequilateral`, `trrectangle` ou `trisocele`. Elles prennent toutes trois valeurs supposées **strictement positives données dans un ordre non décroissant**. Rappel : Les contraintes imposées aux valeurs passées aux paramètres n'ont pas à être vérifiées par les fonctions.

Propriété
Les trois valeurs définissent un **triangle plat** si et seulement si la plus grande est égale à la somme des deux autres.

L'égalité de deux réels se fera à epsilon près.
Définissez la constante `EPSILON=1e-8`.

Définition
Deux réels x et y sont **égaux à epsilon près** si $x - y < \varepsilon$.

Écrivez une fonction `egalite(x,y)` qui teste et renvoie `Vrai` si des réels `x` et `y` sont égaux à `EPSILON` près, `Faux` sinon.

Outil Java

L'opération $\sqrt{x}$ s'écrit `Math.abs(x)`.

Validez votre fonction avec la solution.

Solution Java @[pgtrpredicats.java]

```
/**
  Précision
*/
final static double EPSILON = 1e-8;
```

```
/**
  Prédicat d'égalité de deux réels
  @param[in] x - un réel
  @param[in] y - un réel
  @return Vrai si x et y sont égaux à EPSILON près, Faux sinon
*/

public static boolean egalite(double x, double y)
{
  return (Math.abs(x - y) < EPSILON);
}
```

Écrivez une fonction `trplat(a,b,c)` qui teste et renvoie `Vrai` si le triplet de réels `(a,b,c)` (avec `0<a<=b<=c`) définit un triangle plat, `Faux` sinon.

Validez votre fonction avec la solution.

Solution Java @[pgtrpredicats.java]

```
/**
  Prédicat de triangle plat
  @param[in] a - longueur de segment
  @param[in] b - longueur de segment
  @param[in] c - longueur de segment
  @return Vrai si (0 < a <= b <= c) constituent un triangle plat
*/

public static boolean trplat(double a, double b, double c)
{
  return egalite(c,a + b);
}
```

Propriété
Les trois valeurs définissent un **triangle équilatéral** si et seulement si elles sont égales entre elles.

Écrivez une fonction `trequilateral(a,b,c)` qui teste et renvoie `Vrai` si le triplet de réels `(a,b,c)` (avec `0<a<=b<=c`) définit un triangle équilatéral, `Faux` sinon.

Solution simple
Intuitivement on se rend compte que si les trois valeurs données dans un ordre non décroissant sont telles que la première est égale à la dernière, alors celle du milieu leur est aussi égale. Ceci est facile à montrer. Soient trois valeurs a, b, c telles que $(a \leq b \leq c)$ **et** $a = c$:

- $b \leq c$ **et** $a = c$ entraîne $b \leq a$ (en remplaçant c par a)
- $b \leq a$ **et** $a \leq b$ entraîne $a = b$
- $a = b$ **et** $a = c$ entraîne $b = c$

Sous la contrainte de l'ordre non décroissant, il suffit donc que a soit égal à c pour définir un triangle équilatéral.

Validez votre fonction avec la solution.

Solution Java @[pgtrpredicats.java]

```
/**
  Prédicat de triangle équilateral
  @param[in] a - longueur de segment
  @param[in] b - longueur de segment
  @param[in] c - longueur de segment
  @return Vrai si (0 < a <= b <= c) constituent un triangle équilateral
*/

public static boolean trequilateral(double a, double b, double c)
{
  return egalite(a,c);
}
```

Propriété
Les trois valeurs définissent un **triangle rectangle** si et seulement si le carré de la plus grande (qui correspondra à l'*hypoténuse*) est égal à la somme des carrés des deux autres.

Écrivez une fonction `trrectangle(a,b,c)` qui teste et renvoie `Vrai` si le triplet de réels `(a,b,c)` (avec `0<a<=b<=c`) définit un triangle rectangle, `Faux` sinon.

Validez votre fonction avec la solution.

Solution Java @[pgtrpredicats.java]

```
/**
  Prédicat de triangle rectangle
  @param[in] a - longueur de segment
  @param[in] b - longueur de segment
  @param[in] c - longueur de segment
  @return Vrai si (0 < a <= b <= c) constituent un triangle rectangle
*/

public static boolean trrectangle(double a, double b, double c)
{
  return egalite(c * c,a * a + b * b);
}
```

Propriété
Les trois valeurs, qui doivent respecter l'**inégalité triangulaire**, définissent un **triangle isocèle** si et seulement si deux au moins d'entre elles sont égales, la troisième correspondant à la *base* du triangle.

Écrivez une fonction `trisocele(a,b,c)` qui teste et renvoie `Vrai` si le triplet de réels `(a,b,c)` (avec `0<a<=b<=c`) définit un triangle isocèle, `Faux` sinon.

Solution simple
On peut distinguer trois manières de construire un triangle isocèle avec les valeurs (a, b, c) données en ordre non décroissant :

- La base est a, donc $b = c$
- La base est b, donc $a = c$: il s'agit d'un triangle équilatéral
- La base est c, donc $a = b$

La condition à renvoyer est donc $a = b$ **ou** $b = c$, condition qui est encore vérifiée si les trois valeurs sont égales et définissent un triangle équilatéral.

Commentaires
La spécification imposant que les trois valeurs constituent les longueurs d'un triangle est primordiale. En effet, l'appel de la fonction `trisocele(2,2,10)` renvoie `Vrai` alors qu'il ne s'agit pas d'un triangle.

Validez votre fonction avec la solution.

Solution Java @[pgtrpredicats.java]

```
/**
  Prédicat de triangle isocèle
  @param[in] a - longueur de segment
  @param[in] b - longueur de segment
  @param[in] c - longueur de segment
  @return Vrai si (0 < a <= b <= c) constituent un triangle isocèle
```

```
*/

public static boolean trisocele(double a, double b, double c)
{
  return (egalite(a,b) || egalite(b,c));
}
```

Écrivez une procédure `test_predicats` qui demande trois réels strictement positifs dans un ordre non décroissant puis affiche le résultat des fonctions.

Testez avec les séries de valeurs suivantes :

(a, b, c)	plat	équilatéral	isocèle	rectangle
$(3, 7, 10)$	Vrai	F	F	F
$(5, 5, 10)$	Vrai	F	Vrai	F
$(6, 6, 6)$	F	Vrai	Vrai	F
$(3, 3, 5)$	F	F	Vrai	F
$(3, 5, 5)$	F	F	Vrai	F
$(3, 4, 5)$	F	F	F	Vrai
$(1, 1, \sqrt{2})$	F	F	Vrai	Vrai
$(4, 5, 6)$	F	F	F	F

Validez votre procédure avec la solution.

Solution Java @[pgtrpredicats.java]

```
/**
  @test
*/

public static void test_predicats()
{
  Scanner input = new Scanner(System.in);
  input.useLocale(Locale.US);
  double c1, c2, c3;
  System.out.print("Trois reels strictement positifs dans un ordre non decroissant? ");
  c1 = input.nextDouble();
  c2 = input.nextDouble();
  c3 = input.nextDouble();
  System.out.println(trplat(c1,c2,c3));
  System.out.println(trequilateral(c1,c2,c3));
  System.out.println(trisocele(c1,c2,c3));
  System.out.println(trrectangle(c1,c2,c3));
}
```

2.3 Nature de triangle

Ce problème définit une fonction qui renvoie une chaîne indiquant la nature d'un triangle à partir de trois valeurs strictement positives données par l'utilisateur dans un ordre non

décroissant. Si l'utilisateur ne respecte pas les contraintes sur les valeurs, elle renvoie le message « Consignes non respectées ». Si les valeurs sont fournies correctement, elle renvoie **une seule** des propositions suivantes en respectant les règles de priorité précisées :

- « Ce n'est pas un triangle » lorsque la plus grande valeur est strictement supérieure à la somme des deux autres.
- « Triangle plat » lorsque la longueur du plus grand côté est égale à la somme des deux autres.
- « Triangle équilatéral » lorsque les trois côtés ont la même longueur.
- « Triangle isocèle » lorsque seuls deux des trois côtés ont la même longueur.
- « Triangle rectangle » lorsque le carré du côté le plus long est égal à la somme des carrés des deux autres côtés.
- « Triangle isocèle rectangle » lorsque le triangle est à la fois rectangle et isocèle.
- « Triangle quelconque » dans tous les autres cas.

Propriété

Les trois **règles de priorité** sont :

- Un triangle *plat* ayant deux côtés égaux sera qualifié de *plat* et non d'*isocèle*.
- Un triangle *équilatéral* ne sera pas qualifié d'*isocèle*.
- Un triangle *isocèle rectangle* ne sera pas qualifié d'*isocèle* ni de *rectangle*.

Exemple : Exemples

- (2,3,10) affiche « Ce n'est pas un triangle ».
- (5,5,10) affiche « Triangle plat » même s'il a deux côtés égaux.
- (3,4,5) affiche « Triangle rectangle ».
- (3.6,6.2,6.2) affiche « Triangle isocèle ».

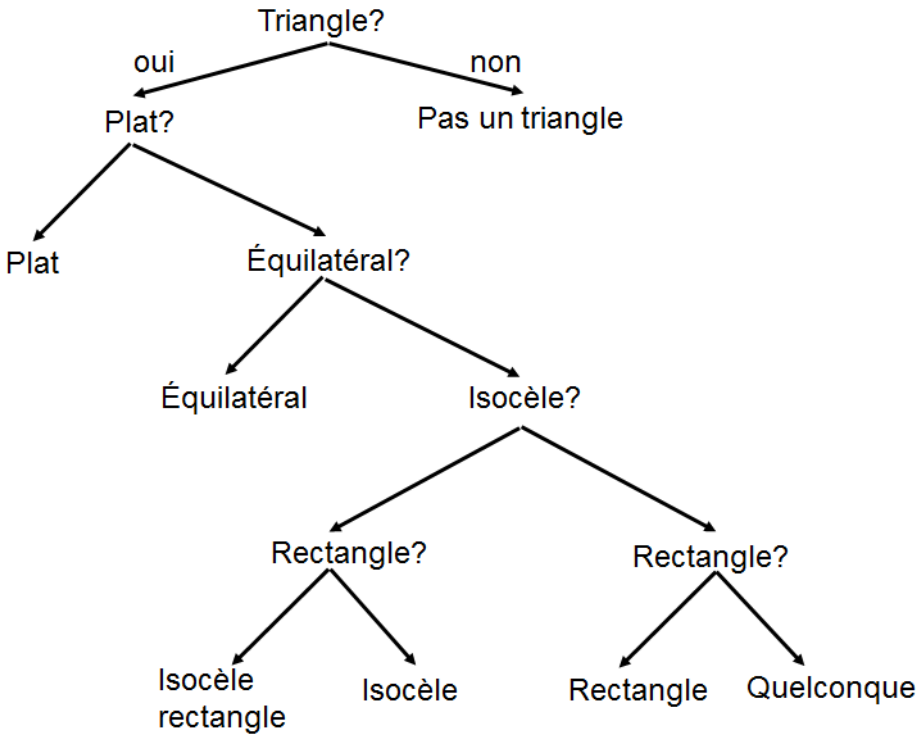
Analyse

Commençons par examiner les trois triplets qui correspondent à des triangles ayant deux propriétés. Le seul qui soit explicitement prévu par l'énoncé est le triangle isocèle rectangle. Un triangle équilatéral est aussi isocèle sans qu'il soit nécessaire de le préciser mais sa propriété équilatérale est prédominante. Il en est de même d'un triangle plat dont la propriété d'être isocèle ne présente aucun intérêt puisqu'il s'agit d'un triangle dégénéré.

Les variables nécessaires sont les trois variables réelles pour les trois valeurs entrées par l'utilisateur. Nous ajoutons une variable `message` de type `chaîne` pour mémoriser la réponse.

L'énoncé demande de filtrer les mauvaises réponses, c.-à-d. de refuser de traiter des entrées dès que l'une n'est pas positive ou qu'elles ne sont pas données dans un ordre non décroissant. Il s'agit donc de refuser le traitement si l'expression $0 \leq a$ Et $a \leq b$ Et $b \leq c$ est fausse, c.-à-d. si sa négation **non** est vraie.

La fonction doit organiser les appels de fonctions en commençant par éliminer le cas où les paramètres ne définissent par un triangle. Ensuite l'organisation des appels peut suivre l'arbre de décision suivant qui s'appuie sur l'énoncé :



Écrivez une fonction `trnature(a,b,c)` qui calcule et renvoie le message (une `chaîne`) de la nature d'un triplet de réels strictement positifs `(a,b,c)` (donnés en ordre non décroissant) qui peuvent constituer les longueurs des côtés d'un triangle.

Validez votre fonction avec la solution.

Solution Java : Si en séquence @[pgtrpredicats.java]

```
/**
  Nature d'un triangle
  @param[in] a - longueur de segment
  @param[in] b - longueur de segment
  @param[in] c - longueur de segment
  @return la chaine représentant la nature du triangle (0 < a <= b <= c)
*/

public static String trnature1(double a, double b, double c)
{
  String msg = "";
  if (!(0.0 < a && a <= b && b <= c))
  {
    msg = "Consignes non respectees";
  }
  else if (!trtriangle(a,b,c))
  {
```

```
    msg = "Pas un triangle";
  }
  else if (trplat(a,b,c))
  {
    msg = "Triangle plat";
  }
  else if (trequilateral(a,b,c))
  {
    msg = "Triangle equilateral";
  }
  else if (trisocele(a,b,c))
  {
    if (trrectangle(a,b,c))
    {
      msg = "Triangle isocele rectangle";
    }
    else
    {
      msg = "Triangle isocele";
    }
  }
  else if (trrectangle(a,b,c))
  {
    msg = "Triangle rectangle";
  }
  else
  {
    msg = "Triangle quelconque";
  }
  return msg;
}
```

Solution Java : Factorisation de branches @[pgtrpredicats.java]

```
/**
  Nature d'un triangle (version sequence)
  @param[in] a - longueur de segment
  @param[in] b - longueur de segment
  @param[in] c - longueur de segment
  @return la chaine représentant la nature du triangle (0 < a <= b <= c)
*/

public static String trnature2(double a, double b, double c)
{
  String msg = "";
  if (!(0.0 < a && a <= b && b <= c))
  {
    msg = "Consignes non respectees";
  }
  else if (!trtriangle(a,b,c))
  {
    msg = "Pas un triangle";
  }
  else
  {
    msg = "Triangle ";
    if (trplat(a,b,c))
```

```
    {
      msg += "plat";
    }
    else if (trequilateral(a,b,c))
    {
      msg += "equilateral";
    }
    else if (trisocele(a,b,c))
    {
      msg += "isocele ";
    }
    if (trrectangle(a,b,c))
    {
      msg += "rectangle";
    }
    // cas du message non modifié
    if (msg.equals("Triangle "))
    {
      msg += "quelconque";
    }
  }
  return msg;
}
```

Soit la procédure `verifNature` qui vérifie la fonction avec plusieurs jeux d'essais.

Java @[pgtrpredicats.java]

```
/**
  @test
*/

public static void verifNature()
{
  System.out.println(trnature(-3,2,5));
  System.out.println(trnature(6,3,5));
  System.out.println(trnature(3,6,5));
  System.out.println(trnature(3,5,13));
  System.out.println(trnature(3,7,10));
  System.out.println(trnature(5,5,10));
  System.out.println(trnature(6,6,6));
  System.out.println(trnature(3,3,5));
  System.out.println(trnature(3,5,5));
  System.out.println(trnature(3,4,5));
  System.out.println(trnature(1,1,Math.sqrt(2)));
  System.out.println(trnature(4,5,6));
}
```

Testez.

3 Que retenir de cet exercice?

C'est au programme appelant de vérifier qu'une fonction est appelée dans des conditions qui respectent la spécification.

Les jeux d'essais ne sont pas réservés aux algorithmes. Il faut aussi tester les fonctions. Celles qui sont fournies dans des bibliothèques sont censées être justes.

Toutes les fonctions booléennes, nommées aussi **prédicats**, peuvent s'écrire de deux manières :

- En affectant explicitement la valeur booléenne `Vrai` ou `Faux` à une variable booléenne locale renvoyée par la fonction.
- En renvoyant directement une expression booléenne évaluée dans la fonction.

Une **variable locale** est déclarée à l'intérieur d'une fonction et n'est utilisable que dans celle-ci. Elle est inconnue à l'extérieur de la fonction et disparait de la mémoire dès que l'exécution de la fonction est terminée.

Il est intéressant d'utiliser les fonctions même si leur définition prend plus de place que les expressions qu'elles renferment. En effet :

- Si leur nom est bien choisi, il exprime en clair leurs propriétés, rendant ainsi l'algorithme immédiatement compréhensible sans qu'il soit nécessaire de connaître les détails des corps des fonctions.
- La programmation de ces fonctions peut être déléguée à des experts du domaine (ici la géométrie) permettant ainsi un partage du travail dan une équipe de développement.

4 Références générales

Comprend [Boudreault-CC1 :c8 :td3], [Tartier-AL1 :c6 :ex20] ■