

# Structures conditionnelles [if]

## Exercices de cours

Karine Zampieri, Stéphane Rivière

Unisciel  algoprog  Version 14 mai 2018

### Table des matières

<b>1</b>	<b>Appréhender le cours</b>	<b>2</b>
1.1	Évaluation d'expressions logiques / qzlogiques . . . . .	2
1.2	Inverse d'un entier / pginverse . . . . .	4
1.3	Valeur absolue d'un entier / pgvabsolue . . . . .	5
<b>2</b>	<b>Appliquer le cours</b>	<b>6</b>
2.1	Différence positive de deux entiers / pgpositive . . . . .	6
2.2	Majorité d'un individu / pgmajorite . . . . .	8
2.3	Validation d'un module / pgvmodule . . . . .	10
2.4	Facturation avec remise / pgremise . . . . .	12
<b>3</b>	<b>Approfondir le cours</b>	<b>13</b>
3.1	Sexe d'un individu / pgsexe . . . . .	13
3.2	Calculatrice algébrique / pgcalcul . . . . .	15
3.3	Bonjour / pgbonjour . . . . .	17

### alg - Exercices de cours (Solution)



Mots-Clés Structures conditionnelles ■

Difficulté ●○○ (2 h) ■

# 1 Appréhender le cours

## 1.1 Évaluation d'expressions logiques / qzlogiques



### Objectif

Cet exercice évalue des expressions logiques.



Écrivez un algorithme qui saisit deux entiers dans `n1` et `n2`.  
Affichez l'invite :

```
Deux entiers?
```



On saisit 15 et 4.  
Évaluez les conditions simples (où `<-` désigne l'affectation) :

```
c1 <- n1 < n2
c2 <- 'A' < 'B'
c3 <- "Arb uste" < "Arbre"
```

### Solution simple

(c1) `eval(15<4)` donne `\linlineFaux@`.  
(c2) `eval('A'<'B')` donne `Vrai` car l'ordre alphabétique est respecté dans les codes ASCII attribués aux lettres.  
(c3) `eval("Arb uste"<"Arbre")` donne `Vrai` car l'espace est avant les lettres.



Quelle(s) est/sont la/les caractéristique(s) de l'évaluation des expressions logiques ?

### Solution simple

Il y en a deux :

- Elle s'effectue de la gauche vers la droite.
- Il y a une évaluation paresseuse des opérateurs `Et` et `Ou`.



Évaluez les expressions logiques :

```
b1 <- c1 Et c2
b2 <- c1 Ou c2
b3 <- (c1 Et c2) Ou c3
b4 <- c1 Et (c2 Ou c3)
```

### Solution simple

(b1) `c1` est `Faux` et `c2` est `Vrai` ce qui donne `Faux Et Vrai` donc `Faux`.  
(b2) Ici on a `Faux Ou Vrai` donc `Vrai`.  
(b3) Le groupe de conditions situé à gauche donne le résultat intermédiaire `Faux`, et `c3` est `Vrai`. Le résultat global est `Faux Ou Vrai` donc `Vrai`.  
(b4) `c1` est `Faux` et le groupe de conditions qui suit donne `Vrai`. Le résultat global est `Faux Et Vrai` donc `Faux`.



Écrivez les expressions logiques de la question précédente dans votre langage.



Affichez les variables (où [x] désigne le contenu de x) :

```
c1=[c1] c2=[c2] c3=c3
b1=[b1] b2=[b2] b3=[b3] b4=[b4]
```



Testez.

### Solution simple

```
Deux entiers? 15 4
c1=Faux c2=Vrai c3=Vrai
b1=Faux b2=Vrai b3=Vrai b4=Faux
```



Validez votre algorithme avec la solution.

### Solution alg

@[qzlogiques1.alg]

```
Algorithme PGLogiques1
Variable n1 , n2 : Entier
Variable c1 , c2 , c3 : Booléen
Variable b1 , b2 , b3 , b4 : Booléen
Début
| Afficher ( "Deux entiers? " )
| Saisir ( n1 , n2 )
| c1 <- n1 < n2
| c2 <- 'A' < 'B'
| c3 <- "Arb  uste" < "Arbre"
| b1 <- c1 Et c2
| b2 <- c1 Ou c2
| b3 <- ( c1 Et c2 ) Ou c3
| b4 <- c1 Et ( c2 Ou c3 )
| Afficher ( "c1=" , c1 , " c2=" , c2 , " c3=" , c3 )
| Afficher ( "b1=" , b1 , " b2=" , b2 , " b3=" , b3 , " b4=" , b4 )
Fin
```

## 1.2 Inverse d'un entier / pginverse



Écrivez un algorithme qui saisit un entier.  
Affichez l'invite :

Votre entier?



En utilisant une structure **Si**, affichez :

- L'inverse de l'entier s'il n'est pas nul.
- Sinon le message « l'entier est nul ».



Testez. Exemples d'exécution :

Un entier? 12  
1/12 vaut 0.0833333333

Un entier? 0  
L'entier est nul



Validez votre algorithme avec la solution.

### Solution alg

@[pginverse1.alg]

```
Algorithme PGInverse1
Variable x : Entier
Début
  | Afficher ( "Un entier? " )
  | Saisir ( x )
  | Si ( x <> 0 ) Alors
  |   | Afficher ( "1/" , x , " vaut " , 1.0 / x )
  | Sinon
  |   | Afficher ( "L'entier est nul" )
  | FinSi
Fin
```

### 1.3 Valeur absolue d'un entier / pgvabsolue



Écrivez un algorithme qui saisit un entier.  
Affichez l'invite :

```
Un entier?
```



Calculez puis affichez sa valeur absolue.



Testez. Exemples d'exécution :

```
Votre entier? 6  
==> La valeur absolue est 6
```

```
Votre entier? -8  
==> Inversion du signe  
==> La valeur absolue est 8
```



Validez votre algorithme avec la solution.

**Solution alg**    @[pgvabsolue1.alg]

```
Algorithme PGVabsolue1  
Variable n : Entier  
Début  
| Afficher ( "Votre entier? " )  
| Saisir ( n )  
| Si ( n < 0 ) Alors  
| | n <- - n  
| | Afficher ( "==> Inversion du signe" )  
| FinSi  
| Afficher ( "==> La valeur absolue est " , n )  
Fin
```

#### **Solution commentée**

L'alternative **Si-Alors** évite de mettre du code pour rien. C'est le cas ici : si la valeur est positive, il n'y a rien à faire. Par contre si elle est négative, il faut inverser le signe pour la rendre positive. C'est ce qu'effectue l'algorithme après la saisie de la valeur entière.

## 2 Appliquer le cours

### Méthode

Reprenons la méthode élémentaire d'élaboration d'algorithme vue dans le module @[Structures de base] et complétons-la en y ajoutant deux nouveaux points (N) :

- Faites attention au problème posé.....
- Résolvez les exercices...
- (N) Ayez à l'esprit le fonctionnement des nouvelles instructions étudiées.
- Transcrivez l'analyse en un algorithme.
- Soumettez votre algorithme à une ou deux traces d'exécution.
- (N) Vérifiez si votre algorithme permet de traiter correctement tous les cas permis par l'énoncé.

Vous vous apercevrez que le style de rédaction des énoncés des exercices change notamment sur trois points :

1. Les énoncés sont rédigés dans un style moins académique, plus proche du langage de tous les jours, que ceux des modules précédent. Par exemple, au lieu d'écrire « écrivez un programme qui demande un nombre représentant une température... » nous écrirons « écrivez un programme qui saisit une température.... ». Ou encore, au lieu de « écrivez un programme qui demande un nombre compris entre 0 et 20 représentant une note.... » nous écrirons plus volontiers « écrivez un programme qui saisit une note comprise entre 0 et 20.... », ou encore « écrivez un programme qui saisit une note.... ».
2. Ces énoncés ne mettent plus en avant les valeurs en entrée et les valeurs en sortie. Il faut le plus souvent les déduire de l'énoncé et de l'exemple d'exécution.
3. Ces énoncés sont plus imprécis. Cela correspond davantage à la manière dont les utilisateurs énoncent réellement leurs problèmes. C'est au concepteur de lever les ambiguïtés, soit par déduction logique, soit en se documentant, soit en interrogeant l'utilisateur. Nos imprécisions seront néanmoins limitées. Par exemple, s'il n'y a pas d'ambiguïté, les chaînes de caractères ne sont pas toujours entourées de guillemets.

### 2.1 Différence positive de deux entiers / pgpositive



Écrivez un algorithme qui saisit deux entiers dans `n1` et `n2`.  
Affichez l'invite :

Deux entiers?



Calculez la différence **positive** des deux entiers dans `rs` (entier).

**Aide méthodologique**

Au choix :

- Si  $n1 < n2$  alors c'est  $n2 - n1$  sinon c'est  $n1 - n2$ .
- Calculez la différence des deux entiers dans `rs` puis utilisez une structure **Si-Alors** pour positiver le résultat.



Affichez (où `[x]` désigne le contenu de `x`) :

La difference positive est `[rs]`



Testez. Exemples d'exécution :

Deux entiers? -6 3

La difference positive est 9

Deux entiers? 10 2

La difference positive est 8



Validez votre algorithme avec la solution.

**Solution alg : Première méthode**

@[pgpositive1.alg]

Algorithme PGPositive1

Variable `n1` , `n2` : Entier

Variable `rs` : Entier

Début

```
| Afficher ( "Deux entiers? " )
| Saisir ( n1 , n2 )
| Si ( n1 < n2 ) Alors
|   | rs <- n2 - n1
| Sinon
|   | rs <- n1 - n2
| FinSi
| Afficher ( "La difference positive est " , rs )
```

Fin

**Solution alg : Deuxième méthode**

@[pgpositive2.alg]

Algorithme PGPositive2

Variable `n1` , `n2` : Entier

Variable `rs` : Entier

Début

```
| Afficher ( "Deux entiers? " )
| Saisir ( n1 , n2 )
| resultat <- n2 - n1
| Si ( rs < 0 ) Alors
|   | rs <- - rs
| FinSi
| Afficher ( "La difference positive est " , rs )
```

Fin

## 2.2 Majorité d'un individu / pgmajorite



Écrivez un algorithme qui saisit l'âge d'un individu dans `age` (entier).  
Affichez l'invite :

```
Votre age?
```



Définissez une constante `MAJORITE=18` (âge de la majorité).



En utilisant une structure `Si`, affichez :

- S'il est ou non majeur (âge supérieur ou égal à `MAJORITE`).
- Ainsi que le nombre d'années depuis ou dans pour être majeur.



Testez. Exemples d'exécution :

```
Votre age? 12
==> Vous n'êtes pas majeur
==> Majorite dans 6 annees
```

```
Votre age? 40
==> Vous êtes majeur
==> Majorite depuis 22 annees
```



Définissez maintenant les constantes :

- `RETRAITE=62` (âge minimum du départ à la retraite).
- `COTISEE=42` (nombre d'années de cotisation).



Saisissez la durée cotisée dans `duree` (entier).  
Affichez l'invite :

```
Durée cotisation?
```



Affichez alors s'il peut partir à la retraite (âge supérieur ou égal à `RETRAITE` et durée cotisée supérieure ou égale à `COTISEE`). Dans le cas de la négative, affichez le nombre d'années restants.



Testez. Exemple d'exécution :

```
Votre age? 40
==> Vous êtes majeur
Majorite depuis 22 annees
Durée cotisation? 22
Vous ne pouvez pas encore partir à la retraite
Depart prévu dans 22 annees
```





Validez votre algorithme avec la solution.

**Solution alg**

@[pgmajorite1.alg]

```
Algorithme PGMajorite1
Constante MAJORITE <- 18
Constante RETRAITE <- 62
Constante COTISEE <- 42
Variable age , duree : Entier
Début
| Afficher ( "Votre age? " )
| Saisir ( age )
| Si ( age >= MAJORITE ) Alors
| | Afficher ( "=> Vous êtes majeur" )
| | Afficher ( "Majorite depuis " , age - MAJORITE , " annees" )
| Sinon
| | Afficher ( "=> Vous n'êtes pas majeur" )
| | Afficher ( "Majorite dans " , MAJORITE - age , " annees" )
| FinSi
| Afficher ( "Durée cotisation? " )
| Saisir ( duree )
| Si ( age >= RETRAITE Et duree >= COTISEE ) Alors
| | Afficher ( "Vous pouvez partir à la retraite" )
| Sinon
| | Afficher ( "Vous ne pouvez pas encore partir à la retraite" )
| | Afficher ( "Depart prévu dans " , RETRAITE - age , " annees" )
| FinSi
Fin
```

## 2.3 Validation d'un module / pgvmodule



### Objectif

Un module est sanctionné par une note d'oral de coefficient 1 et une note d'écrit de coefficient 2. La moyenne obtenue doit être supérieure ou égale à 10 pour valider le module. Cet exercice calcule le résultat (reçu, refusé) d'un étudiant à un module.



Écrivez un algorithme qui saisit :

- La note d'orale d'un étudiant dans `no` (réel).
- Sa note d'écrit dans `ne` (réel).

Affichez les invites :

```
Note d'écrit?
```

```
Note d'oral?
```

Supposez des notes positives valides entre 0 et 20.



Calculez la moyenne du module dans `moyenne` (réel) définie par :

$$moyenne = (2ne + no)/3$$



Affichez (où `[x]` désigne le contenu de `x`) :

```
==> Moyenne [moyenne]
```



Affichez le résultat (`recu`, `refus`) en comparant sa moyenne à 10 au moyen de l'alternative `Si`.



Testez. Exemples d'exécution :

```
Note d'écrit? 8.5
Note d'oral? 14
==> Moyenne 10.3333333333
==> reçu
```

```
Note d'écrit? 11
Note d'oral? 5
==> Moyenne 9
==> refusé
```



Validez votre algorithme avec la solution.

**Solution alg**    @[pgvmodule1.alg]

```
Algorithme pg_vmodule1
Variable ne , no : Réel
Variable moyenne : Réel
Début
|  Afficher ( "Note d'écrit? " )
|  Saisir ( ne )
|  Afficher ( "Note d'oral? " )
|  Saisir ( no )
|  moyenne <- ( 2 * ne + no ) / 3
|  Afficher ( "=> Moyenne " , moyenne )
|  Si ( moyenne >= 10 ) Alors
|  |  Afficher ( "=> reçu" )
|  Sinon
|  |  Afficher ( "=> refusé" )
|  FinSi
Fin
```

## 2.4 Facturation avec remise / pgremise



Écrivez un algorithme qui saisit un prix hors taxes.



Calculez le prix TTC correspondant (avec un taux de TVA constant de 19.6%).



Établissez ensuite une remise dont le taux dépend de la valeur ainsi obtenue, à savoir :

- 0% pour un montant inférieur à 1 000€.
- 1% pour un montant supérieur ou égal à 1 000€ et inférieur à 2 000€.
- 3% pour un montant supérieur ou égal à 2 000€ et inférieur à 5 000€.
- 5% pour un montant supérieur ou égal à 5 000€.



Affichez le prix TTC ainsi que la remise.



Testez.



Validez votre algorithme avec la solution.

## 3 Approfondir le cours

### 3.1 Sexe d'un individu / pgsexe



Écrivez un algorithme qui saisit un caractère parmi 'm', 'h', 'g', 'f' dans `sexe`.  
Affichez l'invite :

Votre sexe parmi m,h,g,f?



Affichez le sexe de l'individu selon :

- « Masculin » si 'm' (Masculin), 'h' (Homme) ou 'g' (Garçon)
- « Féminin » si 'f' (Féminin, Femme, Fille)
- « Saisie incorrecte » dans tous les autres cas.



Testez. Exemples d'exécution :

Votre sexe parmi m,h,g,f? h  
Vous êtes du sexe Masculin

Votre sexe parmi m,h,g,f? x  
Saisie incorrecte



Validez votre algorithme avec la solution.

#### Solution alg : Structure Selon

@[pgsexe1.alg]

```

Algorithme PGSexe1
Variable sexe : Chaîne
Début
  | Afficher ( "Votre sexe parmi m,h,g,f? " )
  | Saisir ( sexe )
  | Selon Sexe
  |   | Cas 'm' , 'h' , 'g'
  |   |   | Afficher ( "Vous êtes du sexe Masculin" )
  |   | Cas 'f'
  |   |   | Afficher ( "Vous êtes du sexe Féminin" )
  |   | Cas Autre
  |   |   | Afficher ( "Saisie incorrecte" )
  | FinSelon
Fin
  
```

#### Solution alg : Structure Si

@[pgifsexe2.alg]

```

Algorithme PGSexe2
Variable sexe : Chaîne
Début
  | Afficher ( "Votre sexe parmi m,h,g,f? " )
  | Saisir ( sexe )
  
```

```
| Si ( sexe = 'm' Ou sexe = 'h' Ou sexe = 'g' ) Alors
|   | Afficher ( "Vous êtes du sexe Masculin" )
| Sinon Si ( sexe = 'f' ) Alors
|   | Afficher ( "Vous êtes du sexe Féminin" )
| Sinon
|   | Afficher ( "Saisie incorrecte" )
| FinSi
Fin
```

## 3.2 Calculette algébrique / pgcalcul



Écrivez un algorithme qui saisit :

- Un entier dans `n1`.
- Un caractère dans `op`.
- Un deuxième entier dans `n2`.

Affichez l'invite :

```
n1 @ n2?
```



Déclarez un entier `rs` qui mémorisera le calcul de l'opération correspondante.



Déclarez un booléen `ok` et initialisez-le à `Vrai` :  
il vaudra `Vrai` si le calcul a été effectué, `Faux` sinon.



En utilisant une structure conditionnelle (`Selon` ou `Si`) :

- Traitez les cas où `op` est l'un des opérateurs `'+'` (addition), `'-'` (soustraction), `'*'` ou `'x'` (multiplication) en mémorisant le résultat du calcul dans `rs`.
- Traitez les cas où `op` est l'opérateur `'/'` (division) ou `'%'` (modulo). Dans ces deux cas, l'opération est réalisable si `n2` n'est pas nul. Mettez `Faux` dans `ok` si elle n'a pas été effectuée.
- Enfin dans tous les autres cas, l'opération n'est pas valide : mettez `Faux` dans `ok`.



Affichez l'opération effectuée si `ok` est resté à `Vrai`, le message d'erreur sinon.

```
[n1] [op] [n2] donne [rs] # si ok
Opérateur erroné ou division par zéro # message d'erreur
```



Testez. Exemples d'exécution :

```
n1 @ n2? 3 x -3
3 x -3 donne -9
```

```
n1 @ n2? 6 / 0
Opérateur erroné ou division par zéro
```



Validez votre algorithme avec la solution.

**Solution alg**    @[pgcalcul1.alg]

```
Algorithme PGCalcul1
Variable n1 , n2 : Entier
Variable op : Chaîne
Variable rs : Entier
Variable ok : Booléen
Début
|  Afficher ( "n1 @ n2? " )
|  Saisir ( n1 , op , n2 )
|  ok <- Vrai
|  Selon op
|  |  Cas '+'
|  |  |  rs <- n1 + n2
|  |  Cas '-'
|  |  |  rs <- n1 - n2
|  |  Cas '*' , 'x'
|  |  |  rs <- n1 * n2
|  |  Cas '/'
|  |  |  Si ( n2 <> 0 ) Alors
|  |  |  |  rs <- DivEnt ( n1 , n2 )
|  |  |  Sinon
|  |  |  |  ok <- Faux
|  |  |  FinSi
|  |  Cas '%'
|  |  |  Si ( n2 <> 0 ) Alors
|  |  |  |  rs <- Modulo ( n1 , n2 )
|  |  |  Sinon
|  |  |  |  ok <- Faux
|  |  |  FinSi
|  |  Cas Autre
|  |  |  ok <- Faux
|  FinSelon
|  Si ( ok ) Alors
|  |  Afficher ( n1 , " " , op , " " , n2 , " donne " , rs )
|  Sinon
|  |  Afficher ( "Opérateur erroné ou division par zéro" )
|  FinSi
Fin
```



### 3.3 Bonjour / pgbonjour



Écrivez un algorithme qui saisit une heure dans `hr` (entier).  
Affichez l'invite :

Quelle heure est-il?



Affichez la salutation selon :

- « Bonjour » si `hr` est dans  $[0..18[$
- « Bonsoir » si `hr` est dans  $[18..22[$
- « Bonne nuit » si `hr` est dans  $[22..24[$
- « Heure invalide » dans tous les autres cas



Testez. Exemples d'exécution :

Quelle heure est-il? 19  
Bonsoir

Quelle heure est-il? 25  
Heure invalide



Validez votre algorithme avec la solution.

#### Solution alg : Première solution

@[pgbonjour0.alg]

Algorithme PGBonjour0

Variable `hr` : Entier

Début

```
| Afficher ( "Quelle heure est-il? " )
| Saisir ( hr )
| Si ( 0 <= hr Et hr < 18 ) Alors
| | Afficher ( "Bonjour" )
| Sinon
| | Si ( 18 <= hr Et hr < 22 ) Alors
| | | Afficher ( "Bonsoir" )
| | | Sinon
| | | Si ( hr <= 22 Et hr < 24 ) Alors
| | | | Afficher ( "Bonne nuit" )
| | | | Sinon
| | | | Afficher ( "Heure invalide" )
| | | FinSi
| | FinSi
| FinSi
Fin
```

**Solution alg : Solution améliorée**

@[pgbonjour1.alg]

```
Algorithme pgBonjour1
Variable hr : Entier
Début
| Afficher ( "Quelle heure est-il? " )
| Saisir ( hr )
| Si ( hr < 0 Ou 24 <= hr ) Alors
| | Afficher ( "Heure invalide" )
| Sinon
| | Si ( hr < 18 ) Alors
| | | Afficher ( "Bonjour" )
| | Sinon
| | | Si ( hr < 22 ) Alors
| | | | Afficher ( "Bonsoir" )
| | | Sinon
| | | | Afficher ( "Bonne nuit" )
| | | FinSi
| | FinSi
| FinSi
Fin
```

**Solution commentée**

- Première solution : Notez qu'il faut effectivement vérifier que l'heure est dans l'intervalle. En n'écrivant que la deuxième partie de la condition, pour une heure négative (donc invalide), on afficherait :

Bonsoir

- Solution améliorée : Cette version est une bonne solution car elle effectue le minimum de tests. Chaque condition est simple, outre la première qui vérifie que l'heure est correcte, contrairement à la version naïve qui emploie trois conditions composées.