

# Initiation à l'algorithme

Adrien Basse

# Table des matières



<b>I - Structures conditionnelles</b>	<b>3</b>
1. Cas d'utilisation des structures conditionnelles .....	3
1.1. <i>Cas d'utilisation de structures conditionnelles</i> .....	3
2. Mise en œuvre des structures conditionnelles .....	8
2.1. <i>La structure conditionnelle de choix simple si .. sinon</i> .....	9
2.2. <i>La structure conditionnelle de sélection multiple</i> .....	23
3. Exercices sur les structures conditionnelles .....	29
<b>II - Corrigés des exercices</b>	<b>34</b>
1. Solutions des exercices sur les structures conditionnelles .....	34
<b>Conclusion</b>	<b>48</b>

# Structures conditionnelles

I

Les structures conditionnelles permettent l'exécution sous condition d'un ensemble d'actions. Si la condition définie se réalise (la condition est vraie), alors les actions contenues dans la structure conditionnelle sont exécutées. Si la condition définie n'est pas réalisée (la condition est fausse), les actions contenues dans la structure conditionnelle ne sont pas exécutées. L'utilisation des structures conditionnelles est fréquente en algorithme et en programmation. A la première section de ce chapitre, nous proposons des problèmes simples nécessitant l'utilisation des instructions conditionnelles. La seconde section présente les syntaxes de mise en œuvre des instructions conditionnelles en pseudo code et en algorithme. La troisième section présente quelques algorithmes de base avec des modifications nécessitant l'utilisation des structures conditionnelles.

## 1. Cas d'utilisation des structures conditionnelles

Nous présentons dans cette section un ensemble de petits problèmes dont la résolution nécessite l'utilisation des structures conditionnelles. Il est important pour chaque problème de réfléchir sur les **actions à exécuter sous condition** et sur les **conditions** de leur exécution.

### 1.1. Cas d'utilisation de structures conditionnelles

#### *Affichage d'une information suivant une condition*

Nous disposons d'informations sur le nom et l'âge d'une personne et nous désirons afficher ces informations uniquement si l'âge est supérieur à 18.

#### *Méthode*

Nous devons réaliser un test sur l'âge de la personne. Si celui-ci est supérieur à 18 ( $\text{age} \geq 18$ ) alors nous effectuons un affichage du nom et de l'âge. Dans le cas contraire ( $\text{age} < 18$ ), aucune action n'est réalisée.

Une seule action sous condition :

1. L'affichage du nom et de l'âge.
  - La condition d'exécution :  $\text{age} \geq 18$ .

### *Affichage des données positives parmi plusieurs valeurs*

Nous disposons de 3 valeurs entières (a, b et c) et nous voulons afficher les valeurs entières positives parmi ces valeurs.

#### *Méthode*

---

Pour chaque valeur, nous réalisons un test pour savoir si elle est positive. Si tel est le cas (par exemple  $a > 0$ ), la valeur correspondante (a sur l'exemple) est affichée. Dans le cas contraire ( $a \leq 0$  sur l'exemple), aucune action n'est réalisée.

### *Calcul et affichage du nombre de valeurs négatives*

Nous disposons de 3 valeurs entières (a, b et c) et nous voulons calculer et afficher le nombre de valeurs entières négatives parmi ces valeurs.

#### *Méthode*

---

Pour chaque valeur, nous réalisons un test pour savoir si elle est négative. Si tel est le cas (par exemple  $a < 0$ ), cette valeur (a sur l'exemple) est comptabilisée parmi les valeurs négatives. Dans le cas contraire ( $a \geq 0$ ), aucune action n'est réalisée.

### *Calcul et affichage du nombre de valeurs non nulles*

Nous disposons de 3 valeurs entières (a, b et c) et nous voulons calculer et afficher le nombre de valeurs entières non nulles parmi ces valeurs.

#### *Méthode*

---

Pour chaque valeur, nous réalisons un test pour savoir si elle est non nulle. Si tel est le cas (par exemple b est différente de 0), cette valeur (b sur l'exemple) est comptabilisée parmi les valeurs négatives. Dans le cas contraire ( $b = 0$  sur l'exemple), aucune action n'est réalisée.

### *Calcul et affichage de la somme des valeurs positives*

Nous disposons de 3 valeurs entières (a, b et c) et nous voulons calculer et afficher la somme des valeurs entières positives parmi celles ci.

#### *Méthode*

---

Pour chaque valeur, nous réalisons un test pour savoir si elle est positive. Si tel est le cas (par exemple  $a > 0$ ), cette valeur (a sur l'exemple) est comptabilisée dans la somme. Dans le cas contraire ( $a < 0$ ), aucune action n'est réalisée. Notons également que la prise en compte ou non des valeurs nulles dans la somme ne change pas le résultat car zéro est l'élément neutre de l'addition.

### *Calcul et affichage du plus grand entre deux valeurs*

Nous disposons de deux valeurs entières (a et b) et nous désirons afficher le plus grand des deux.

### Méthode

---

Nous réalisons un test pour savoir si la première valeur est plus grande que la deuxième (Es ce que  $a > b$  par exemple). Si tel est le cas ( $a$  est effectivement plus grand que  $b$ ) alors la première valeur est affichée. Dans le cas contraire, la deuxième valeur est affichée.

#### *Calcul et affichage de plusieurs sommes*

Nous disposons de 3 valeurs entières ( $a$ ,  $b$  et  $c$ ) et nous voulons calculer et afficher la somme des valeurs entières positives, la somme des valeurs négatives, la somme de toutes les valeurs.

### Méthode

---

Pour chaque valeur, nous réalisons un test pour savoir si elle est positive. Si tel est le cas (par exemple  $a > 0$ ), cette valeur ( $a$  sur l'exemple) est comptabilisée dans la somme des valeurs positives. Dans le cas contraire ( $a < 0$ ), la valeur correspondante est comptabilisée dans la somme des valeurs négatives. Toutes les valeurs, sans aucun test, sont comptabilisées dans la somme totale. Une autre solution pour obtenir la somme totale est de faire à la fin la somme des sommes positives et négatives obtenues.

#### *Affichage d'informations majeur ou mineur suivant une condition*

Nous disposons d'informations sur le nom ( $nom$ ) et l'âge ( $age$ ) d'une personne et nous désirons afficher le nom de la personne avec la mention « majeur » ou « mineur » selon que son âge est supérieur ou inférieur à 18.

### Méthode

---

Nous réalisons un test pour savoir si l'âge est supérieure à 18 ( $age > 18$ ). Si tel est le cas alors le nom de la personne est affiché avec la mention « majeur ». Dans le cas contraire, le nom est affiché avec la mention « mineur ».

Deux actions sous condition :

1. L'affichage du nom avec la mention majeur.
  - La condition d'exécution : **age  $\geq$  18.**
2. L'affichage du nom avec la mention mineur
  - La condition d'exécution : **age  $<$  18.**

Nous pouvons noter ici que **les deux conditions sont opposées**.

#### *Calcul et affichage du prix du panier d'un client*

Un magasin de vente de légumes propose les prix suivants pour la carotte selon la quantité à acheter : Si la quantité achetée

- est inférieure strictement à 10 kg, le prix du kg est de 350 frs ;
- est comprise entre 10 et 19 kg, le prix du kg est de 325 frs ;
- est comprise entre 20 et 30 kg, le prix du kg est de 300 frs ;
- est au delà de 31kg, le prix du kg est de 250frs.

Nous disposons d'informations sur la quantité (quantite) de carotte commandée par un client et nous désirons calculer et afficher le montant de la facture.

### Méthode

---

Nous pouvons déterminer d'abord le prix du kg de carotte à appliquer au client. Pour cela, nous réalisons un test pour savoir si la quantité de carotte commandée par le client est inférieure à 10 (prix = 350), comprise entre 10 et 19 (prix = 325), comprise entre 20 et 30 (prix = 300), ou supérieure à 31 (prix = 250). Le prix ainsi déterminé, il est ensuite possible de calculer le montant de la facture (facture \* prix).

#### *Test et affichage des signes de deux nombres*

Nous disposons de deux valeurs entières (a et b) et nous désirons afficher l'un des quatre messages suivants : « a et b sont positifs », « a et b sont négatifs », « a est positif et b est négatif », « a est négatif et b est positif ».

### Méthode

---

Contrairement aux cas précédents, le test ne porte pas sur une seule valeur mais sur plusieurs (2 exactement). La condition de test est plus complexe. Par exemple, le premier message « a et b sont positifs » est affiché uniquement après un test réussi du signe positif de a et un test identique pour b (le message est affiché si a positif et si b positif). Il est important de noter que les deux composantes de la condition sont reliées par un **et**. L'utilisation du **et** signifie que les deux composantes doivent être vraie pour que la condition soit vraie. D'ailleurs, si lors du test, la première composante est fausse nul besoin alors de vérifier la seconde composante car quel que soit sa valeur (vrai ou faux) la condition sera fausse. Si la première composante est vraie alors la seconde est vérifiée et son résultat sera le résultat de la condition globale. Lorsque nous avons à mettre en œuvre une condition complexe reliée par un **et** il est préférable pour éviter de faire des tests inutiles de toujours placer en premier la condition ayant le plus de chance d'être fausse.

#### *Comparaison et affichage du signe de deux nombres*

Nous disposons de deux valeurs entières (a et b) et nous désirons afficher si cela est vrai : « a et b sont de signes contraires ».

### Méthode

---

Une première solution consiste à faire un test complexe comme dans le cas précédent sur le signe de a et de b. Pour afficher le message il faut que a soit positif et b négatif ou bien que a soit négatif et b positif. La condition de test est encore plus complexe que dans le cas précédent. Les deux grandes composantes de la condition sont reliées par un **ou** et dans chaque grande composante nous avons deux parties reliées par un **et** comme dans le cas précédent. L'utilisation du **ou** signifie qu'il suffit qu'une des deux composantes soit vraie pour que la condition soit vraie. Ainsi, si la première grande composante de la condition (a positif et b négatif) est vraie le message est affiché sans qu'il y ait besoin de vérifier la seconde partie de la condition car les deux parties sont reliées par un **ou**. Dans le cas où la première grande composante de la condition est fausse alors la seconde partie de la condition est vérifiée et elle déterminera si le message sera affiché ou non. Lorsque nous avons à mettre en œuvre une condition complexe reliée par un **ou** il est préférable pour éviter de faire des tests inutiles de toujours placer en premier la condition ayant le plus de chance d'être vraie.

Une deuxième solution moins intuitive mais plus simple consiste à utiliser le fait que lorsque deux nombres sont de signes contraires leur produit est négatif. Appliqué dans notre cas, nous testons si le produit de  $a$  et de  $b$  est négatif ou non pour afficher ou non le message. Le test réalisé dans ce cas est différent des précédents dans le sens où ce n'est pas la valeur d'une variable unique qui est testée mais une expression qui est le résultat d'un calcul pouvant inclure plusieurs variables.

### *Affichage des données dans un intervalle parmi plusieurs valeurs*

Nous disposons de 3 valeurs entières ( $a$ ,  $b$  et  $c$ ) et nous voulons calculer et afficher les valeurs entières comprises entre 0 (inclus) et 20 (non inclus) parmi ces valeurs.

#### *Méthode*

---

Pour chaque valeur, nous réalisons un test complexe pour savoir si elle est supérieure ou égal à zéro et strictement inférieur à 20. Si tel est le cas (par exemple  $a = 15$  soit  $0 \leq a < 20$ ), la valeur correspondante ( $a$  sur l'exemple) est affichée. Dans le cas contraire ( $a = 21$  par l'exemple), aucune action n'est réalisée.

### *Affichage d'informations complémentaires suivant une condition*

Nous disposons du genre (Homme, Femme), du nom d'une personne et nous voulons réaliser un affichage du nom en le faisant précéder de Monsieur ou de Madame suivant le genre.

#### *Méthode*

---

Nous réalisons un test pour savoir quel est le genre de la personne. Si le genre est égal à Homme le message Monsieur est affiché. Si le genre est Femme le message Madame est affiché. Ensuite nous affichons le nom de la personne pour disposer de l'affichage complet. Il est possible également de faire un affichage complet dès le début en ajoutant directement le nom dans les 2 précédents messages.

### *Affichage d'informations complémentaires suivant une condition (Bis)*

Nous disposons du genre (Homme, Femme), du nom d'une personne et de son état matrimonial (0 célibataire, 1 marié). Nous voulons réaliser un affichage du nom en le faisant précéder de « Monsieur », de « Madame » ou de « Mademoiselle » suivant le genre et éventuellement l'état matrimonial.

#### *Méthode*

---

Si le genre est égal à « Homme » le message « Monsieur » suivi du nom de la personne est affiché. Si le genre est égal à « Femme » un second test est réalisé et va porter sur l'état matrimonial. Ce second test n'est pas nécessaire dans le cas où le genre est égal à « Homme ». Si lors de ce second test l'état matrimonial est égal à 0 alors le message « Mademoiselle » suivi du nom de la personne est affiché. Sinon, dans le cas où l'état matrimonial est égal à 1 alors le message « Madame » suivi du nom de la personne est affiché.

### Calcul et affichage du salaire suivant la situation du salarié

Nous disposons du genre (Homme, Femme), du nom d'une personne, de son état matrimonial (0 célibataire, 1 marié) et si son conjoint travaille ou non (0 ne travaille pas et 1 travaille), du nombre de ses enfants, de son salaire de base, de ses indemnités de transport et de logement. Nous voulons calculer et afficher les salaires mensuels brut et net de cette personne en sachant que :

- le salaire brut est égal à la base ajoutée de la somme des indemnités de transport et de logement,
- le salaire net est égal au salaire brut moins les impôts sur le revenu ;
- l'impôt sur le revenu est égal au salaire brut multiplié par le taux d'imposition. Si la personne est célibataire et sans enfant le taux d'imposition est de 45%. Pour chaque enfant jusqu'au sixième le taux d'imposition diminue de 2%. Au delà du sixième enfant aucune diminution supplémentaire n'est effectuée. Pour les personnes mariées le taux d'imposition diminue également de 5%. Le taux diminue encore de 2% si le conjoint ne travaille pas.

#### Méthode

Nous disposons de toutes les informations pour calculer directement le salaire brut. Pour calculer le salaire net il faut au préalable calculer le taux d'imposition de la personne concernée.

D'abord nous pouvons faire un test pour délimiter le nombre d'enfants à 6. Si le nombre d'enfant est supérieur à 6 alors nous le ramenons à 6.

Ensuite, le produit du nombre d'enfants avec le taux diminué par enfant (2%) est diminué des 45% de départ.

Enfin, nous pouvons réaliser un test sur l'état matrimonial et éventuellement sur le travail du conjoint. Si la personne est mariée (1) alors le taux d'imposition diminue encore de 5%. Toujours dans le cas où la personne est mariée nous testons si le conjoint travaille ou non pour diminuer de 2% le taux d'imposition. Pour finir, nous pouvons calculer le salaire net en utilisant le salaire brut et le taux d'imposition calculés précédemment.

\* \*

\*

Les problèmes présentés ici montre la nécessité de disposer de structures algorithmiques permettant de choisir les actions à exécuter suivant une condition. Le déroulement d'un algorithme comportant des structures conditionnelles n'est presque jamais linéaire. En effet, les actions à exécuter dépendent de conditions et ne sont généralement pas les mêmes d'une exécution à une autre. Pour résoudre ces problèmes, il est essentiel d'identifier clairement les actions dont l'exécution dépend de conditions et pour chacune de ces actions, les conditions de leur exécution. Pour chaque problème proposé dans cette section, nous avons identifié ces deux éléments : Actions et conditions. Pour écrire l'algorithme il faut ensuite disposer de la syntaxe de base des structures conditionnelles en pseudo code ou en algorithme. C'est l'objet de la section suivante.

## 2. Mise en œuvre des structures conditionnelles



Dans la section précédente nous avons montré la nécessité de disposer de structures conditionnelles pour l'exécution d'actions sous condition. Nous disposons de deux structures conditionnelles : La structure conditionnelle de **choix simple** (si .. sinon) et la structure conditionnelle de **sélection multiple** (cas). Dans cette section nous présentons la mise en œuvre de ces deux structures en pseudo code et en algorithme.

## 2.1. La structure conditionnelle de choix simple si .. sinon

Une structure conditionnelle **si .. sinon** peut comporter une partie (partie **si**) ou deux (partie **si** et partie **sinon**). L'utilisation de la première partie ou des deux parties dépend de l'exercice à traiter.

Si seule la première partie de la structure conditionnelle est utilisée nous parlons de **choix simple sans alternative**. Ce cas correspond à des exercices où des actions sont à exécuter lorsqu'une condition est vraie et qu'aucune action n'est à exécuter **immédiatement après** lorsque la condition est fausse. Donc, si la condition est fausse, **aucune action n'est exécutée**. Parmi les problèmes présentés à la section précédente, vous pouvez chercher ceux qui ne nécessitent pas l'utilisation de la première partie de la structure conditionnelle. C'est le cas entre autres des problèmes d'affichage d'une information suivant une condition et de calcul et affichage de la somme des valeurs positives. Pour le premier problème cité il est possible qu'aucun affichage ne soit fait. Pour le deuxième problème, la somme peut être nulle correspondant au cas où aucune valeur positive n'a été donnée.


Lorsque les deux parties de la structure conditionnelle sont utilisées nous parlons de **choix simple avec alternative**. Il ne s'agit pas simplement d'exécuter des actions sous condition comme dans le cas précédent mais de choisir entre deux groupes d'actions celles à exécuter suivant une condition. Donc, que la condition soit vraie ou fausse, il y a à chaque fois **un et un seul groupe d'actions à exécuter parmi deux**. Parmi les problèmes présentés à la section précédente, certains nécessitent l'utilisation de choix simple avec alternative. C'est le cas entre autres du problème d'affichage d'informations majeur ou mineur suivant une condition. Si l'âge est supérieur à 18 "Majeur" est affiché. Si tel n'est pas le cas "Mineur" est affiché. Il y a donc une action à faire lorsque la condition est vraie et une action à faire lorsque la condition est fausse.

### 2.1.1. Le choix simple en pseudo code

Dans une première partie nous allons voir la syntaxe pseudo code du **choix simple sans alternative**. Dans la deuxième partie de cette section, nous présentons le pseudo code du **choix simple avec la partie alternative**. Enfin, une troisième partie permet de discuter de l'**imbrication de structures alternatives** c'est à dire des structures alternatives qui contiennent elles mêmes d'autres structures alternatives.

Des exemples simples sont présentés à chaque fois pour illustrer différents cas d'utilisation. Comme démarche, nous vous proposons pour chaque exemple de réfléchir et de répondre aux questions posées avant de consulter les résultats et les explications fournis.

#### a) Choix simple sans alternative en pseudo code

 *Syntaxe : Si en pseudo code*

**Si** condition **alors**  
**Action**

**Action** est exécutée si la **condition** est vraie. Lorsque la condition est fausse, aucune action n'est exécutée.

### Exemple : Affichage d'un entier s'il est positif.

Nous disposons d'une valeur entière donnée "a" et nous voulons l'afficher si elle est strictement positive.

```
1 afficher ('Donner un entier');
2 lire (a);
3 Si a>0 alors
4   afficher(a, ' est positif')
```

L'affichage du texte « *a est positif* » est soumis à la condition «  $a > 0$  ». Lorsque cette condition n'est pas réalisée, aucune action n'est exécutée.

### Attention

Par défaut, la structure conditionnelle concerne une seule action comme dans l'exemple précédent. Si plusieurs actions sont à exécuter lorsque la condition est vraie, il est obligatoire de les insérer dans un bloc d'actions à l'aide des mots clés **début** et **fin**. Le bloc **début ... fin** est facultatif lorsque la structure conditionnelle contient une seule action.

### Exemple

Proposer un algorithme permettant à partir de deux entiers donnés de calculer et d'afficher  $a/b$  si cela est possible ( $b$  est différent de zéro)

```
1 afficher ('Donner le dividende a');
2 lire (a);
3 afficher ('Donner le diviseur b');
4 lire (b);
5 Si b <> 0 alors
6   début
7     c:=a/b;
8     afficher ('le quotient est ',c);
9   fin
```

Nous pouvons lire le code précédent ainsi : Si la valeur de  $b$  est différente de zéro alors faire le calcul  $c=a/b$  puis l'affichage de  $c$ . Dans le cas contraire ni le calcul ni l'affichage ne sont réalisés.

Si le bloc début ... fin est enlevé du bout de code précédent la lecture est différente : Si la valeur de  $b$  est différente de zéro alors faire le calcul  $c=a/b$ . Ensuite, qu'importe la valeur de  $b$  faire l'affichage de  $c$ . En effet, par défaut la structure conditionnelle ne contient qu'une seule action (le calcul) et l'action d'affichage n'est soumise à aucune condition.

### Exemple

Quel est le résultat de l'algorithme suivant ?

```
1 a:=10;
2 Si a < 0 alors
3   afficher ('Bonjour');
4   afficher ('Bonsoir');
```

**Résultat obtenu:** Bonsoir

**Explications :** a n'étant pas négatif, 'Bonjour' n'est pas affiché. Par contre 'Bonsoir' qui n'est soumis à aucune condition est affiché. Une lecture de cet algorithme serait « *Afficher Bonjour si a est négatif puis dans tous les cas afficher Bonsoir.* »


Quel est le résultat de l'algorithme suivant ?

```
1 a:=10;
2 Si a <0 alors
3   début
4     afficher ('Bonjour');
5     afficher ('Bonsoir');
6   fin
```

**Résultat obtenu:**

**Explications :** L'affichage du 'Bonjour' comme du 'Bonsoir' sont soumis à la condition  $a < 0$ . Cette condition étant fausse, aucun affichage n'est réalisé. Une lecture de cet algorithme serait « *Afficher Bonjour puis Bonsoir si a est négatif. Ne rien faire sinon* »

## b) Choix simple avec alternative en pseudo code

 *Syntaxe : Si ... sinon en pseudo code*

---


**Si** condition **alors**

**Action1**

**Sinon**

**Action2**

**Action1** est exécutée si la **condition** est vraie. Lorsque la condition est fausse, **Action2** est exécutée.

 *Exemple : Affichage du résultat d'un étudiant.*

---

Nous disposons de la moyenne générale d'un étudiant et nous voulons afficher « *Réussite* » ou « *Echec* » suivant cette moyenne.

```
1 afficher ('Donner la moyenne');
2 lire (moyenne);
3 Si moyenne >= 10 alors
4   afficher ('Réussite');
5 Sinon
6   afficher ('Echec');
```

L'affichage de « *Réussite* » est soumis à la condition «  $moyenne \geq 10$  ». L'affichage de « *Echec* » est soumis à la condition contraire, c'est à dire «  $moyenne < 10$  ». Il est donc possible d'obtenir le même résultat en remplaçant le « *si... sinon* » par deux « *si* » avec des conditions contraires comme dans le cas si dessous. Toutefois, il est conseillé d'utiliser à chaque fois que c'est possible le « *si... sinon* » à la place de deux « *si* ». En effet, le « *si... sinon* » est plus efficace car il permet de s'affranchir d'une deuxième comparaison.

```
1 afficher ('Donner la moyenne');
2 lire (moyenne);
3 Si moyenne >= 10 alors
```

```

4  afficher ('Réussite');
5  Si moyenne <10 alors
6  afficher ('Echec');

```

### Rappel

Par défaut, la structure conditionnelle concerne une seule action dans la partie « *si* » comme dans la partie « *sinon* » comme dans l'exemple précédent. Si plusieurs actions sont à exécuter dans une des deux parties, il est obligatoire de les insérer dans un bloc d'actions à l'aide des mots clés **début** et **fin**. Le bloc **début ... fin** est facultatif lorsqu'une partie contient une seule action.

### Exemple

Proposer un algorithme permettant à partir de deux entiers donnés de calculer et d'afficher  $a/b$  si cela est possible ( $b$  est différent de zéro) ou un message d'erreur sinon.

```

1 ...
2 Si b <>0 alors
3  début
4    c:=a/b;
5    afficher ('le résultat est ',c);
6  fin
7 Sinon
8  afficher ('Division impossible');

```

Nous pouvons lire le code précédent ainsi : Si la valeur de  $b$  est différente de zéro alors faire le calcul  $c=a/b$  puis l'affichage de  $c$ . Dans le cas contraire le message d'erreur « *Division impossible* » est affiché.

Le bloc « *si* » contient deux actions. Il est donc obligatoire de le délimiter par un bloc « *début...fin* ». Sans ce bloc « *début ...fin* » l'algorithme n'est plus « *syntactiquement correct* ». En effet, le « *sinon* » est trop loin (une action de trop) pour être rattaché par défaut au « *si* ».

Le « *sinon* » ne contient qu'une seule action. Il est donc possible d'utiliser ou non un bloc « *début...fin* » pour le délimiter.

Quel est le résultat de l'algorithme suivant ?

```

1 a:=-10;
2 b:=0;
3 Si a <0 alors
4  afficher (a,' est négatif');
5 Sinon
6  afficher (a,' est positif ou nul');
7  b:=a*2;
8 afficher (b,' est son double');

```

#### Résultat obtenu:

-10 est négatif

-20 est son double

#### Explications :

$a$  étant négatif, la première action d'affichage est réalisée ( -10 est négatif).

La condition du « *si* » étant vraie, les actions du « *sinon* » (une seule action par défaut) ne sont pas à réaliser. Par contre, l'affectation de  $b$  et le dernier affichage qui ne sont soumis à aucune condition sont à réaliser.

Une lecture de cet algorithme serait « *afficher d'abord la valeur de a avec son signe (négatif ou positif ou nul).* Ensuite calculer et afficher le double de a sans aucune condition. »

Quel est le résultat de l'algorithme suivant ?

```
1 a:=-10;
2 b:=0;
3 Si a <0 alors
4   afficher (a,' est négatif');
5 Sinon
6   début
7     afficher (a,' est positif ou nul');
8     b:=a*2;
9   fin
10 afficher (b,' est son double');
```

**Résultat obtenu:**

-10 est négatif

-0 est son double

**Explications :**

La seule différence avec l'algorithme précédent est que l'affectation du b est insérée au bloc sinon. La condition du « *si* » étant vraie, cette affectation n'est pas faite et le b garde sa valeur initiale (0).

Quel est le résultat de l'algorithme suivant ?

```
1 a:=-10;
2 b:=0;
3 Si a <0 alors
4   afficher (a,' est négatif');
5   b:=a*3;
6   a:=-a;
7 Sinon
8   début
9     afficher (a,' est positif ou nul');
10    b:=a*2;
11  fin
12 afficher (b,' est son double');
```

**Résultat obtenu:** Erreur de syntaxe...

**Explications :**

Le bloc « *sinon* » est trop loin (deux actions trop loin) pour être rattaché par défaut au bloc « *si* ». Pour résoudre ce problème il faut délimiter les trois actions qui appartiennent au bloc « *si* » par un bloc début...fin comme sur l'algorithme suivant.

Quel est le résultat de l'algorithme suivant ?

```
1 a:=-10;
2 b:=0;
3 Si a <0 alors
4   début
5     afficher (a,' est négatif');
6     b:=a*2;
7     a:=-a;
8   fin
```

```

9 Sinon
10 début
11     afficher (a, ' est positif ou nul');
12     b:=a*2;
13 fin
14 afficher (b, ' est son double');

```

#### Résultat obtenu:

-10 est négatif  
-20 est son double

#### Explications :

La condition du « *si* » étant vraie, le premier affichage est réalisé ainsi que les deux affectations. Le dernier affichage qui n'est soumis à aucune condition est également réalisé avec comme valeur de « *b* » le double de la valeur de « *a* ».

Notons également que même si la valeur de *a* est devenue positive dans le *si*, cela n'entraîne pas la réalisation des actions dans le *sinon*.

Quel est le résultat de l'algorithme suivant ?

```

1 a:=-10;
2 b:=0;
3 Si a <0 alors
4 début
5     afficher (a, ' est négatif');
6     b:=a*2;
7     a:=-a;
8 fin
9 Si a>=0 alors
10 début
11     afficher (a, ' est positif ou nul');
12     b:=a*2;
13 fin
14 afficher (b, ' est son double');

```

#### Résultat obtenu:

-10 est négatif  
10 est positif ou nul  
20 est son double

#### Explications :

Le « *si...sinon* » est généralement équivalent en terme de résultat à deux « *si* » avec des conditions opposées. Ce n'est pas le cas ici à cause du changement de valeur de « *a* » dans le bloc d'actions du premier « *si* ». Ce changement fait que la condition du deuxième « *si* » est également vraie. Si la valeur du « *a* » était positive dès le début, seul le deuxième « *si* » aurait vu sa condition vraie.

Le dernier affichage qui n'est soumis à aucune condition est également réalisé avec comme valeur de « *b* » le double de la valeur de « *a* » après la première affectation (« *a* := -*a* ») qui la rend positive.

### c) Imbrication de structures alternatives en pseudo code

Le bloc « *si* » comme le bloc « *sinon* » peut contenir d'autres blocs « *si* » et/ou « *si...sinon* ». Nous parlons dans ce cas d'imbrication de structures alternatives.

#### Exemple : Affichage de l'intervalle d'un entier.

Nous disposons d'un entier *a* et nous voulons afficher son intervalle d'appartenance parmi les possibilités suivante : « *a est plus grand ou égal à 100* », « *a est compris entre 1 et 99* », « *a est égal à 0* », « *a est négatif* ».

```
1 ...
2 Si a >=100 alors
3   afficher (a, ' est plus grand ou égal à 100');
4 Sinon
5   Si a >=1 alors
6     afficher (a, ' est compris entre 1 et 99');
7   Sinon
8     Si a=0 alors
9       afficher (a, ' est égal à 0');
10    Sinon
11      afficher (a, 'est négatif');
```

Le premier bloc « *sinon* » contient un bloc « *si ...sinon* ». Le « *sinon* » de ce dernier bloc contient à son tour un « *si ... sinon* ».

Chaque bloc « *si* » et « *sinon* » contient une seule action, d'où la possibilité de ne pas utiliser de blocs « *début ... fin* ». En effet, un bloc « *si* » ou un bloc « *si ... sinon* » est vu comme un seul élément et ne nécessite pas d'être encadré par un bloc « *début ... fin* ». L'algorithme précédent est donc équivalent au suivant avec l'utilisation des blocs « *début ... fin* » facultatifs. Pour ne pas surcharger le code nous n'avons pas encadré les affichages par des blocs « *début ... fin* ».

```
1 ...
2 Si a >=100 alors
3   afficher (a, ' est plus grand ou égal à 100');
4 Sinon
5   début
6     Si a >=1 alors
7       afficher (a, ' est compris entre 1 et 99');
8     Sinon
9       début
10        Si a=0 alors
11          afficher (a, ' est égal à 0');
12        Sinon
13          afficher (a, 'est négatif');
14      fin
15   fin
```

Par contre, dans l'algorithme suivant le bloc « *début ... fin* » est obligatoire. En effet, le premier bloc « *si* » contient deux éléments : un affichage et un bloc « *si* ».

```
1 ...
2 Si a >b alors
3   début
4     afficher (a, ' est supérieur à ', b);
5     Si b>c alors
6       afficher (b, ' est compris entre', a, ' et ', c);
```

```
7 fin
```

Quel est le résultat de l'algorithme suivant avec et sans le bloc « *début ... fin* » ?

```
1 a:=65;
2 b:=75;
3 c:=55;
4 Si a > b alors
5   début
6     afficher (a, ' est supérieur à ', b);
7   Si b > c alors
8     afficher (b, ' est compris entre ', a, ' et ', c);
9   fin
```

**Résultat obtenu avec le bloc « *début...fin* »:**

Aucun affichage...

**Explications :**

La première condition («  $a > b$  ») étant fausse, aucune des deux actions dans le bloc « *début...fin* » n'est réalisée.

**Résultat obtenu sans le bloc « *début...fin* »:**

75 est compris entre 65 et 55

**Explications :**

Nous sommes en présence d'une erreur sémantique. La première condition («  $a > b$  ») est fausse et donc le premier affichage n'est pas réalisé. Par contre, la deuxième condition («  $b > c$  ») est vraie et entraîne la réalisation du deuxième affichage. Ce deuxième test est fait puisque ce bloc « *si* » n'est soumis à aucune condition contrairement au cas précédent.

## 2.1.2. Le choix simple en algorithme

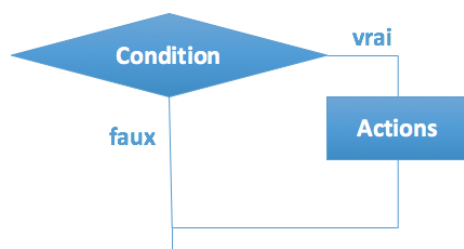
Cette section est décomposée en trois parties. Nous présentons dans la première partie la syntaxe générale du **choix simple sans alternative** en algorithme. Dans la deuxième partie de cette section, nous présentons l'algorithme du **choix simple avec la partie alternative**. Enfin, une troisième partie permet de présenter des exemples d'algorithmes comportant **des structures conditionnelles imbriquées** c'est à dire des structures conditionnelles qui contiennent elles mêmes d'autres structures conditionnelles.

Des exemples simples sont présentés dans chaque partie pour illustrer différents cas d'utilisation. Comme démarche, nous vous proposons pour chaque exemple de réfléchir et de répondre aux questions posées avant de consulter les résultats et les explications fournis.

### a) Choix simple sans alternative en algorithme



*Syntaxe : Si en algorithme*

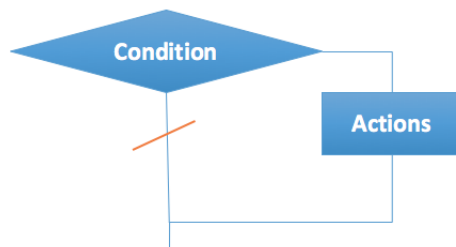


Le **losange** permet de spécifier la condition à tester. Deux lignes partent du losange :

- La ligne disposant du label **vrai**, portent les actions à réaliser lorsque la condition est vraie ;
- La ligne portant le label **faux** ne dispose d'aucune action car il s'agit d'un choix simple sans alternative.



Les **Actions** sont exécutées si la **condition** est **vraie**. Lorsque la condition est **fausse**, aucune action n'est exécutée.

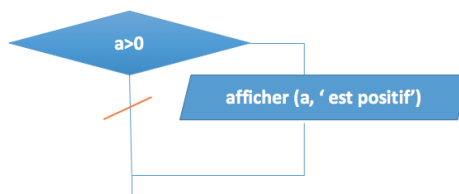


La figure précédente est équivalente à celle ci. La ligne conduisant à la **condition fausse est barrée**. Cela signifie que l'autre ligne représente la condition vraie.

A travers des exercices d'application simples, nous allons maintenant voir la mise en œuvre concrète de cette syntaxe.

#### Exemple : Affichage d'un entier s'il est positif.

Nous disposons d'une valeur entière donnée "a" et nous voulons l'afficher si elle est strictement positive.

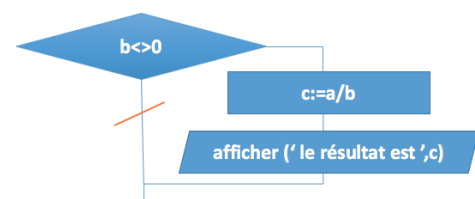


L'affichage du texte « *a est positif* » est soumis à la condition «  $a > 0$  ». Lorsque cette condition n'est pas réalisée, aucune action n'est exécutée.

#### Exemple

Proposer un algorithme permettant à partir de deux entiers donnés de calculer et d'afficher  $a/b$  si cela est possible (b est différent de zéro)

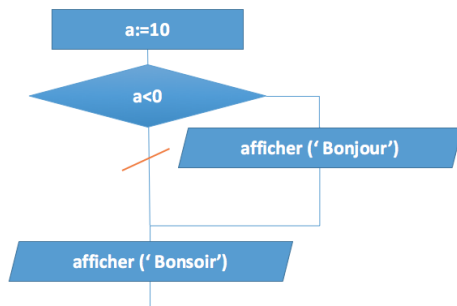
Nous pouvons lire le code précédent ainsi : Si la valeur de b est différente de zéro alors faire le calcul  $c=a/b$  puis l'affichage de c. Dans le cas contraire ni le calcul ni l'affichage ne sont réalisés.



#### Exemple

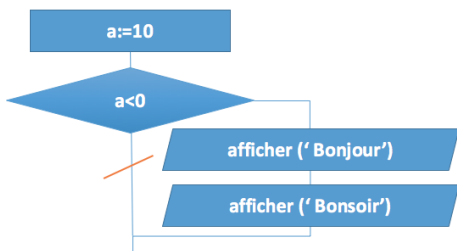
Quel est le résultat de l'algorithme suivant ?

**Résultat obtenu:** Bonsoir



**Explications :** a n'étant pas négatif, 'Bonjour' n'est pas affiché. Par contre, 'Bonsoir' qui n'est soumis à aucune condition est affiché. Une lecture de cet algorithme serait « *Afficher Bonjour si a est négatif puis dans tous les cas afficher Bonsoir.* »

Quel est le résultat de l'algorithme suivant ?



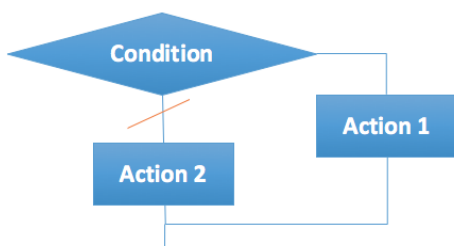
**Résultat obtenu:**

**Explications :** L'affichage du 'Bonjour' comme du 'Bonsoir' sont soumis à la condition  $a < 0$ . Cette condition étant fausse, aucun affichage n'est réalisé. Une lecture de cet algorithme serait « *Afficher Bonjour puis Bonsoir si a est négatif. Ne rien faire sinon* »

## b) Choix simple avec alternative en algorithme



*Syntaxe : Si ... sinon en algorithme*

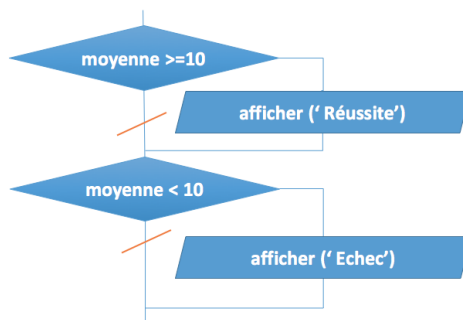


**Action1** est exécutée si la **condition** est vraie. Lorsque la condition est fausse, **Action2** est exécutée. Dans chaque partie nous pouvons avoir autant d'actions à réaliser que nécessaire.

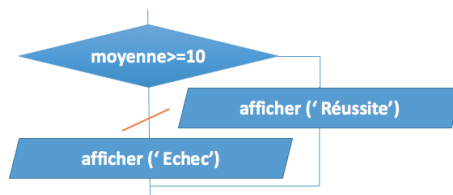


*Exemple : Affichage du résultat d'un étudiant.*

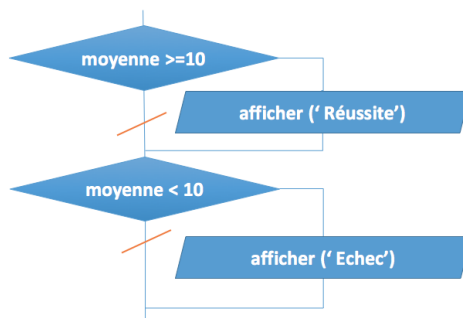
Nous disposons de la moyenne générale d'un étudiant et nous voulons afficher « *Réussite* » ou « *Echec* » suivant cette moyenne.



L'affichage de « *Réussite* » est soumis à la condition « *moyenne >= 10* ». L'affichage de « *Echec* » est soumis à la condition contraire, c'est à dire « *moyenne < 10* ».



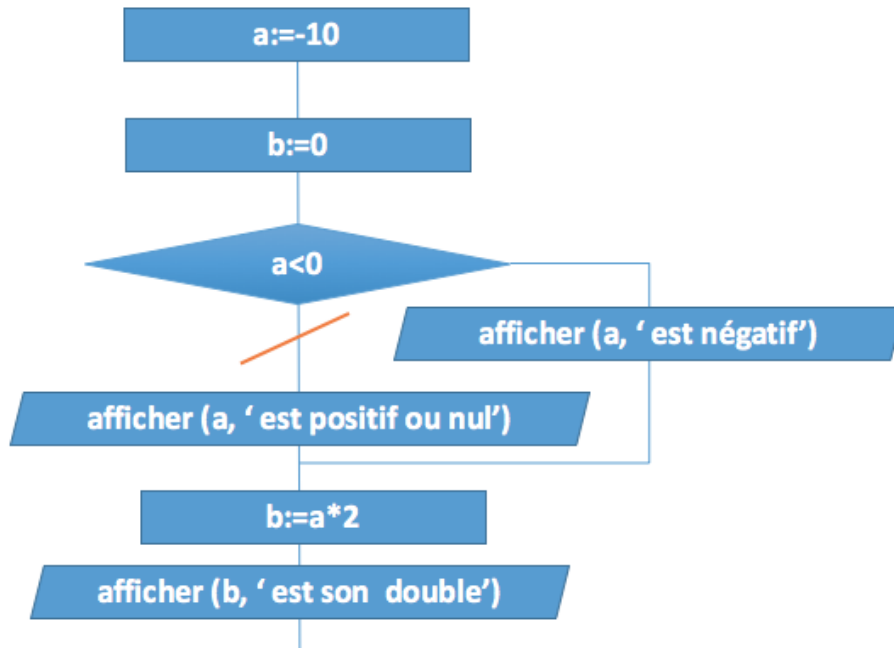
L'affichage de « *Réussite* » est soumis à la condition « *moyenne >= 10* ». L'affichage de « *Echec* » est soumis à la condition contraire, c'est à dire « *moyenne < 10* ».



Il est possible d'obtenir le même résultat que précédemment en remplaçant le « *si... sinon* » par deux « *si* » avec des conditions contraires. Toutefois, il est conseillé d'utiliser à chaque fois que c'est possible le « *si... sinon* » à la place de deux « *si* ». En effet, le « *si... sinon* » est plus efficace car il permet de s'affranchir d'une deuxième comparaison.

### Exemple

Quel est le résultat de l'algorithme suivant ?



**Résultat obtenu:**

-10 est négatif

-20 est son double

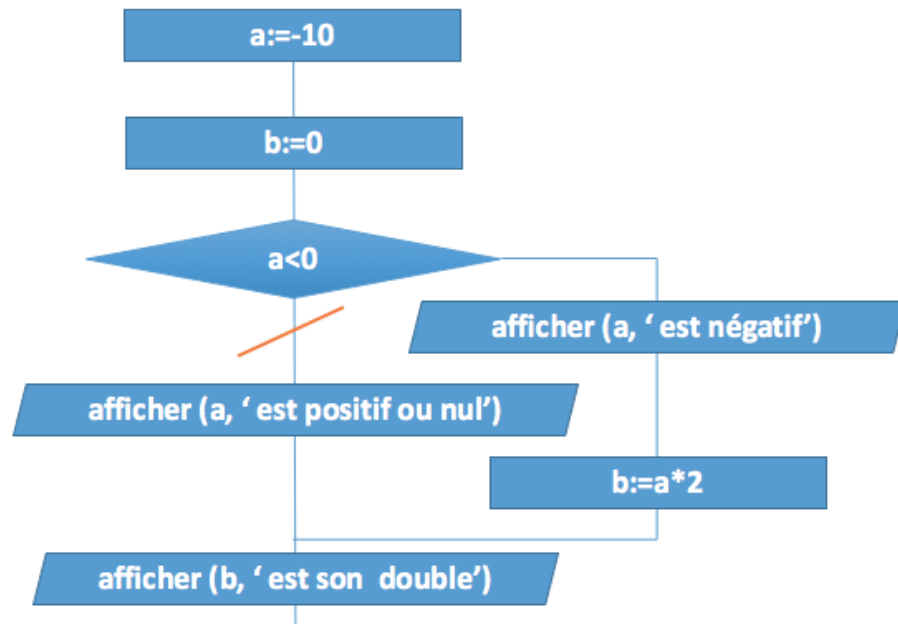
**Explications :**

a étant négatif, la première action d'affichage est réalisée ( -10 est négatif).

La condition du « si » étant vraie, les actions du « sinon » (une seule action par défaut) ne sont pas à réaliser. Par contre, l'affectation de b et le dernier affichage qui ne sont soumis à aucune condition sont à réaliser.

Une lecture de cet algorithme serait « afficher d'abord la valeur de a avec son signe (négatif ou positif ou nul). Ensuite calculer et afficher le double de a sans aucune condition. »

Quel est le résultat de l'algorithme suivant ?



**Résultat obtenu:**

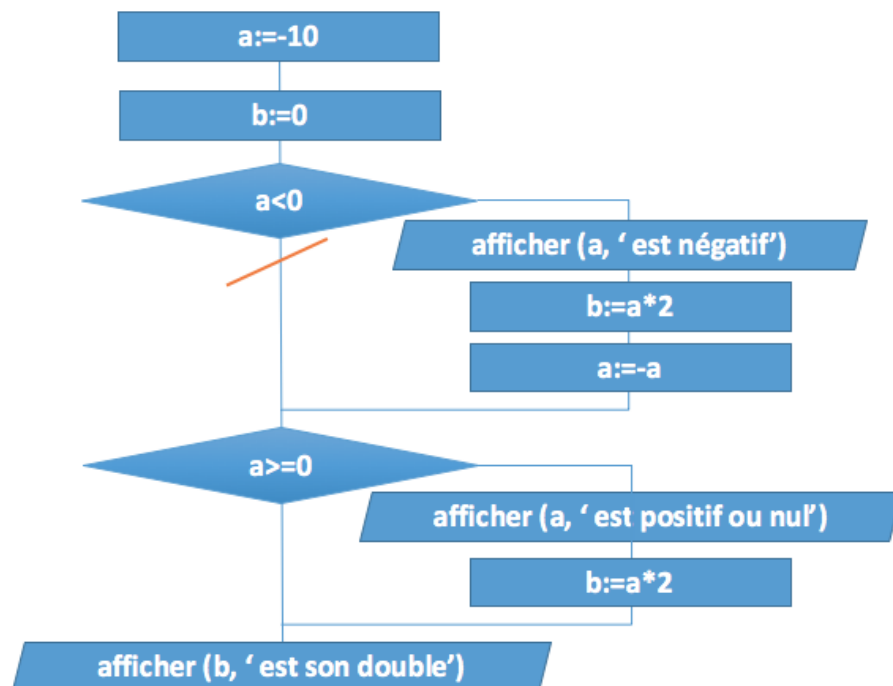
-10 est négatif

-0 est son double

**Explications :**

La seule différence avec l'algorithme précédent est que l'affectation du b est insérée au bloc sinon. La condition du « si » étant vraie, cette affectation n'est pas faite et le b garde sa valeur initiale (0).

Quel est le résultat de l'algorithme suivant ?



```
-10 est négatif
10 est positif ou nul
20 est son double
```

**Explications :**

Le « *si...sinon* » est généralement équivalent en terme de résultat à deux « *si* » avec des conditions opposées. Ce n'est pas le cas ici à cause du changement de valeur de « *a* » dans le bloc d'actions du premier « *si* ». Ce changement fait que la condition du deuxième « *si* » est également vraie. Si la valeur du « *a* » était positive dès le début, seul le deuxième « *si* » aurait vu sa condition vraie.

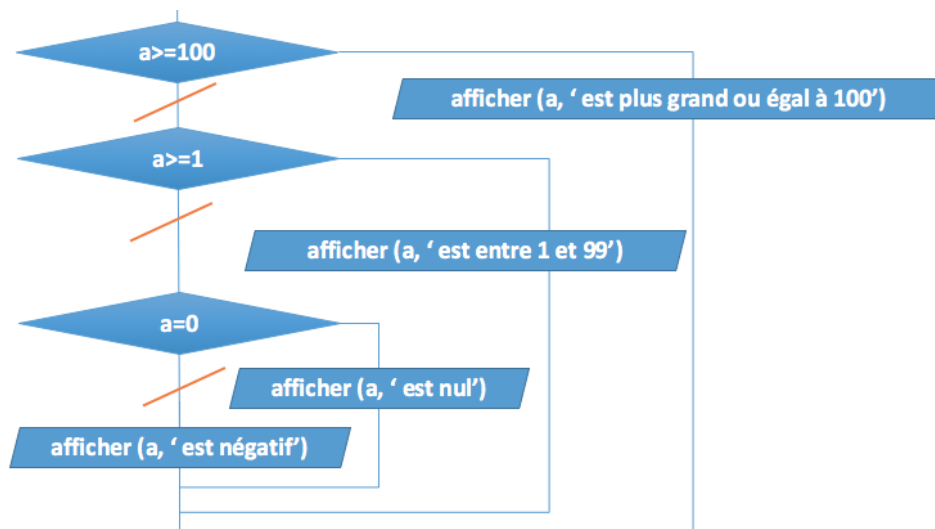
Le dernier affichage qui n'est soumis à aucune condition est également réalisé avec comme valeur de «  $b$  » le double de la valeur de «  $a$  » après la première affectation («  $a := -a$  ») qui la rend positive.

### c) Imbrication de structures conditionnelles en algorithme

La partie « *si* » comme la partie « *sinon* » d'une structure conditionnelle peut contenir d'autres blocs « *si* » et/ou « *si...sinon* ». Nous parlons dans ce cas d'imbrication de structures conditionnelles.

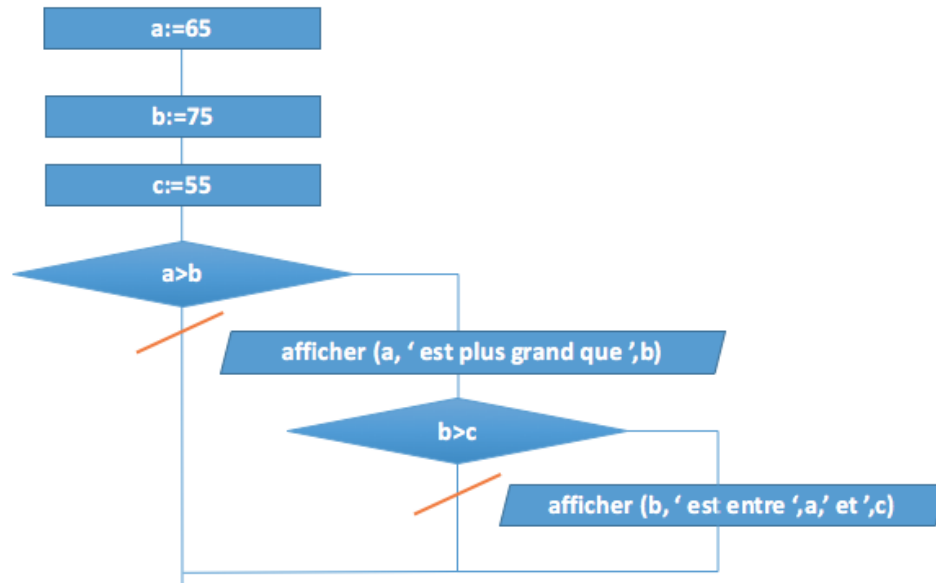
👉 Exemple : Affichage de l'intervalle d'un entier.

Nous disposons d'un entier  $a$  et nous voulons afficher son intervalle d'appartenance parmi les possibilités suivante : «  $a$  est plus grand ou égal à 100 », «  $a$  est compris entre 1 et 99 », «  $a$  est égal à 0 », «  $a$  est négatif ».



Le premier bloc « *sinon* » contient un bloc « *si ...sinon* ». Le « *sinon* » de ce dernier bloc contient à son tour un « *si ... sinon* ».

Quel est le résultat de l'algorithme suivant ?

**Résultat obtenu :**

Aucun affichage...

**Explications :**

La première condition («  $a > b$  ») étant fausse, aucune des deux actions dans la partie « *si* » n'est réalisée.

## 2.2. La structure conditionnelle de sélection multiple

La sélection multiple **cas ... parmi** permet de choisir des actions à exécuter selon différentes valeurs d'une expression entière ou caractère. La sélection multiple comprend :

- **une entête** permettant de préciser l'expression à tester. Cette partie est obligatoire ;
- **une ou plusieurs parties** contenant chacune une ou plusieurs valeurs à tester et le bloc d'actions à réaliser si le test réussit. Le test consiste à faire correspondre à la valeur de l'expression en entête soit une valeur constante simple, une liste de valeurs ou un intervalle de valeurs;
- **une dernière partie « *sinon* » optionnelle** pour préciser le bloc d'actions à exécuter si aucun des autres tests n'a pu se réaliser.

Le nombre de parties à utiliser ainsi que le choix de la dernière partie optionnelle dépendent de l'exercice à traiter.

Un exercice traité avec la sélection multiple peut toujours être réécrit à l'aide du choix simple. Le contraire n'est pas toujours vrai. En effet, lorsque la condition est complexe (plusieurs données et/ou plusieurs opérateurs (et, ou,...)) la sélection multiple n'est pas utilisable.

### 2.2.1. La sélection multiple en pseudo code et en algorithme

Dans une première partie nous allons voir la syntaxe générale en pseudo code et en algorithme de la **sélection multiple**. La deuxième partie de cette section permet de discuter de l'**imbrication de structures conditionnelles** c'est à dire des structures conditionnelles qui contiennent elles mêmes d'autres structures conditionnelles.

Des exemples simples sont présentés à chaque fois pour illustrer différents cas d'utilisation. Comme démarche, nous vous proposons pour chaque exemple de réfléchir et de répondre aux questions posées avant de consulter les résultats et les explications fournis.

#### a) Sélection multiple en pseudo code



##### *Syntaxe : Sélection multiple en pseudo code*

###### **Cas expression parmi**

**Constante 1: Bloc d'actions A**

**Constante 2, constante 3: Bloc d'actions B**

**Intervalle: Bloc d'actions C**

**Sinon: Bloc d'actions N**

**finCas**

« Le bloc d'actions A » est exécuté si la valeur de l'expression est égale à la « constante 1 ». Le test renvoie « vrai » pour une seule valeur qui est ici « la constante 1 ».

« Le bloc d'actions B » est exécuté si la valeur de l'expression est égale à la « constante 2 » **ou** à la « constante 3 ». Le test renvoie « vrai » pour deux valeurs qui sont la « constante 2 » et la « constante 3 ». Il est possible de spécifier autant de valeurs que nécessaire. Les valeurs sont séparées par une virgule.

« Le bloc d'actions C » est exécuté si la valeur de l'expression appartient à l'intervalle donné. Le test renvoie « vrai » pour toutes les valeurs de l'intervalle.

« Le bloc d'actions N » correspond à la partie **sinon** et est exécuté si aucun des autres tests n'a pu renvoyer « vrai ». Donc, le test renvoie « vrai » pour toutes les autres valeurs possible.

Une sélection multiple comporte au plus une seule partie **sinon**, et autant des autres parties que nécessaire.

##### *Code du choix simple équivalent au code de sélection multiple*

A partir de la syntaxe présentée précédemment, nous pouvons aisément écrire un code équivalent en utilisant la structure conditionnelle de choix simple. Le code suivant utilise le choix simple pour réécrire le code de sélection multiple précédent.

**si expression = Constante 1 alors**

**Bloc d'actions A**

**sinon si expression = Constante 2 ou expression=Constante 3**

**alors**

**Bloc d'actions B**

**sinon si expression dans Intervalle alors**

**Bloc d'actions C**

**sinon**

**Bloc d'actions N**

« Le bloc d'actions A » est exécuté si la valeur de l'expression est égale à la « constante 1 »;

« Le bloc d'actions B » est exécuté si la valeur de l'expression est égale à la « constante 2 » **ou** à la « constante 3 »;

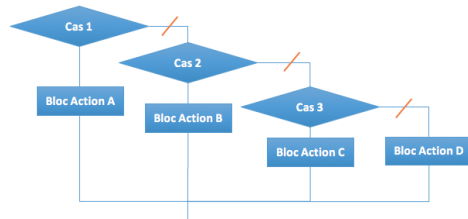
« Le bloc d'actions C » est exécuté si la valeur de l'expression appartient à l'intervalle donné ;



Si aucune des conditions précédente n'est vraie, « le bloc d'actions *N* » est exécuté.

Avec l'équivalence précédente, nous comprenons qu'une sélection multiple n'est en fait qu'une succession de « si ... sinon si ... » pouvant se terminer par un « sinon si... » ou un « sinon » avec des conditions de test particulières. D'où l'utilisation des mêmes symboles graphiques pour représenter en algorithme **la sélection multiple** et le **choix simple** (cf. p.16) .

### Syntaxe : Sélection multiple en algorithme



Pour que cela représente une sélection multiple les différentes conditions (cas1, cas2, ...) portent sur une même expression à comparer à une valeur particulière à une liste de valeurs ou un intervalle de valeurs.

Si la condition « cas 1 » est vraie alors « le bloc d'actions A » est exécuté et la sélection multiple est terminée. Si la condition « cas 1 » est fausse, un test similaire est réalisé avec la condition « cas 2 » et ainsi de suite.

Sur cet exemple nous avons choisi d'ajouter un bloc « sinon » à travers « le bloc d'actions D » qui s'exécute lorsque aucune des conditions précédente n'est vraie. Lorsque la sélection multiple ne contient pas de bloc « sinon », « le bloc d'actions D » (le dernier bloc) est à supprimer.

### Exemple : Calculatrice.

Nous disposons de deux valeurs entières donnée "a" et "b" et d'un caractère "op" égal à +, -, \*, ou /. Nous voulons calculer et afficher a op b.

```
1 afficher ('Donner deux entiers');
2 lire (a,b);
3 afficher ('Donner un opérateur');
4 lire(op);
5 cas op parmi
6 '+':
7   resultat:=a+b;
8   afficher (a,op,b,' = ',resultat);
9 '-':
10  resultat:=a-b;
11  afficher (a,op,b,' = ',resultat);
12 '/':
13  resultat:=a/b;
14  afficher (a,op,b,' = ',resultat);
15 '*':
16  resultat:=a*b;
17  afficher (a,op,b,' = ',resultat);
18 finCas
```

Le test porte sur l'opérateur « *op* ». Selon sa valeur, le calcul correspondant est réalisé puis le résultat affiché. Nous pouvons noter ici que la sauvegarde du calcul intermédiaire dans la donnée « *résultat* » n'est pas nécessaire. Nous pouvons réécrire ce code comme suit sans donnée intermédiaire.

```
1 afficher ('Donner deux entiers');
2 lire (a,b);
3 afficher ('Donner un opérateur');
4 lire(op);
5 cas op parmi
6   '+':
7     afficher (a,op,b,' = ',a+b);
8   '-':
9     afficher (a,op,b,' = ',a-b);
10  '/':
11    afficher (a,op,b,' = ',a/b);
12  '*':
13    afficher (a,op,b,' = ',a*b);
14 finCas
```

Il serait tentant, en remarquant que l'affichage est identique pour les quatre cas, de vouloir faire un seul affichage à la fin de la sélection multiple comme dans le code ci dessous. Quelle est le problème du code suivant par rapport au deux exemples précédents?

```
1 afficher ('Donner deux entiers');
2 lire (a,b);
3 afficher ('Donner un opérateur');
4 lire(op);
5 cas op parmi
6   '+':
7     resultat:=a+b;
8   '-':
9     resultat:=a-b;
10  '/':
11    resultat:=a/b;
12  '*':
13    resultat:=a*b;
14 finCas
15 afficher (a,op,b,' = ',resultat);
```

Que se passe-t-il si l'opérateur donné par l'utilisateur n'est pas parmi les quatre testés ?

Dans les deux premiers exemples rien n'est passé. Aucun calcul et aucun affichage ne se réalise.

Dans le troisième exemple, l'affichage va se faire et de manière incorrecte car la donnée « *résultat* » n'est pas bien renseignée. Ce troisième exemple n'est donc pas correct.

Comment modifier les deux premiers exemples pour prendre en compte le cas où l'utilisateur donne un opérateur incorrect en affichant un message d'erreur?

Nous vous proposons la solution suivante.

```
1 afficher ('Donner deux entiers');
2 lire (a,b);
3 afficher ('Donner un opérateur');
4 lire(op);
5 cas op parmi
6   '+':
7     afficher (a,op,b,' = ',a+b);
```

```

8  '-':
9    afficher (a,op,b,' = ',a-b);
10 '/':
11   afficher (a,op,b,' = ',a/b);
12 '*':
13   afficher (a,op,b,' = ',a*b);
14 sinon:
15   afficher ('opérateur incorrect');
16 finCas

```

Si l'opérateur n'est pas parmi les quatre choisis, c'est le bloc sinon qui se réalise donc l'affichage du message d'erreur « *opérateur incorrect* ».

Cette solution montre plus clairement pourquoi il n'était pas possible de mettre l'affichage du résultat à la fin de la sélection. En effet, il y a une différence entre les quatre premiers affichages et le dernier qui rend impossible « la factorisation » de l'affichage à la fin.

Quelles modifications du code précédent proposez vous pour permettre à l'utilisateur de donner à sa guise l'opérateur '/' ou ':' pour faire la division ?

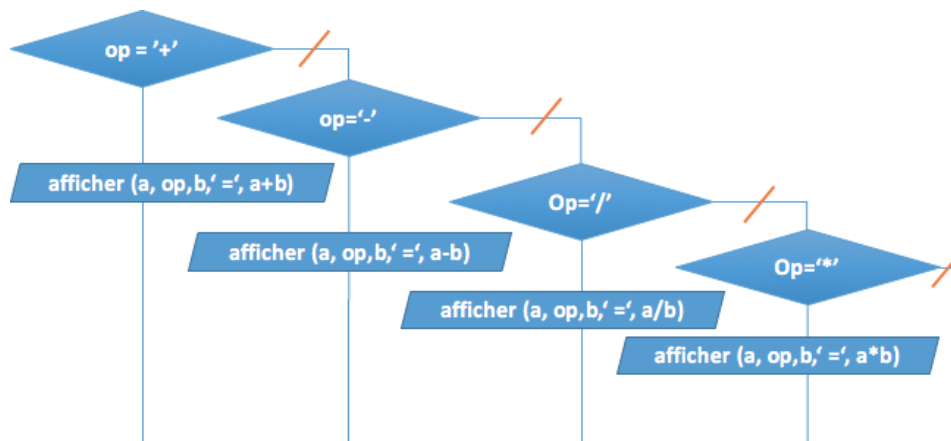
```

1 afficher ('Donner deux entiers');
2 lire (a,b);
3 afficher ('Donner un opérateur');
4 lire(op);
5 cas op parmi
6 '+':
7   afficher (a,op,b,' = ',a+b);
8 '-':
9   afficher (a,op,b,' = ',a-b);
10 '/',':':
11   afficher (a,op,b,' = ',a/b);
12 '*':
13   afficher (a,op,b,' = ',a*b);
14 sinon:
15   afficher ('opérateur incorrect');
16 finCas

```

Il aurait été également possible d'ajouter un cinquième cas (':') et de lui mettre le même code que pour l'opérateur '/'. Pour éviter cette répétition de code la solution proposée utilise le fait que plusieurs valeurs peuvent être assignées au même cas. En effet, en plus de la possibilité de donner un intervalle de valeurs, il est tout à fait possible de lister différentes valeurs séparées par une virgule.

Proposez maintenant un algorithme correspondant au premier pseudo code pour la gestion d'une calculatrice simple sans message d'erreurs.



Quatre tests sont réalisés et portent tous sur la valeur de la donnée « *op* ». Selon la valeur de « *op* », le résultat de l'opération correspondante est affiché.

Le dernier test ne dispose d'aucune action dans sa partie « *sinon* », cela veut dire que la sélection multiple ne dispose pas également d'un bloc *sinon*. Pour gérer le premier message d'erreur correspondant à la saisie d'un opérateur incorrect, il suffit donc d'ajouter l'affichage du message d'erreur dans le dernier *sinon*.

La gestion de deux opérateurs pour la division ('/' et ':' ) est quant à elle réalisée en modifiant la condition « *op* = '/' » par « *op* = '/' ou *op* = ':' ».

## b) Imbrication de structures conditionnelles en pseudo code

Chaque cas d'une structure de sélection multiple peut à son tour contenir d'autres structures conditionnelles. Nous parlons dans ce cas d'imbrication de structures conditionnelles.

**Exemple : Calculatrice avec la gestion de la division par zéro.**

Proposer une modification de la calculatrice pour afficher un message d'erreur lorsque l'utilisateur choisit la division avec un « *b* » nul.

```

1 afficher ('Donner deux entiers');
2 lire (a,b);
3 afficher ('Donner un opérateur');
4 lire(op);
5 cas op parmi
6 '+':
7   afficher (a,op,b,' = ',a+b);
8 '-':
9   afficher (a,op,b,' = ',a-b);
10 '/',':':
11   si b<>0 alors
12     afficher (a,op,b,' = ',a/b);
13   sinon
14     afficher ('Division par zéro impossible');
15 '*':
16   afficher (a,op,b,' = ',a*b);
17 sinon:
18   afficher ('opérateur incorrect');
19 finCas
  
```

Un si ... sinon nous permet de gérer le cas où le b est nul dans le bloc de la division ('/', ':').

Es ce qu'il est possible de trouver une sélection multiple équivalente au « *si...sinon* » proposé ?

```

1 afficher ('Donner deux entiers');
2 lire (a,b);
3 afficher ('Donner un opérateur');
4 lire(op);
5 cas op parmi
6 '+':
7   afficher (a,op,b,' = ',a+b);
8 '-':
9   afficher (a,op,b,' = ',a-b);
10 '/',':':
11   cas b parmi
12     0: afficher ('Division par zéro impossible');
13   sinon: afficher (a,op,b,' = ',a/b);
14   finCas
15 '*':
16   afficher (a,op,b,' = ',a*b);
17 sinon:
18   afficher ('opérateur incorrect');
19 finCas

```

Le bloc sinon d'une sélection multiple permet de gérer un nombre infini de valeurs. Ce n'est pas le cas pour les autres blocs.

Par rapport à l'exercice nous avons deux blocs : Un bloc avec une valeur (zéro) et un autre bloc avec une infinité de valeurs (différent de zéro). Ces deux blocs forment un tout (zéro + différent de zéro = tous les entiers). D'où la possibilité de proposer cette solution.

\* \*  
\*

Nous avons présentés dans cette section les syntaxes en pseudo code et en algorithme des structures conditionnelles de choix simple et de sélection multiple. Des exercices pratiques nécessitant l'utilisation des structures conditionnelles ont également été présentés et des solutions proposées en pseudo code et en algorithme. Ces exercices ont permis de mettre en exergue différents cas d'utilisation et quelques erreurs courantes.

Pour vous permettre de mettre en pratique les syntaxes étudiées ici, nous vous proposons à la prochaine section plusieurs exercices qui nécessitent l'utilisation des structures conditionnelles.

### 3. Exercices sur les structures conditionnelles

Nous vous présentons dans cette section plusieurs exercices nécessitant l'utilisation des structures conditionnelles. Il est important de proposer pour tous ces exercices une solution en pseudo code, en algorithme ou dans un langage de programmation donné. Comme démarche, nous vous proposons :

- de bien tester manuellement chaque solution que vous proposerez. Pour cela, il est important de bien choisir des données de test et de vérifier si avec ces données le résultat obtenu est en adéquation avec le résultat attendu ;
- de réfléchir à des solutions alternatives et de les comparer en terme d'efficacité à la solution retenue ;

- de bien tester l'implémentation de la solution avec des logiciels comme flowgorithm (algorithme) ou dans un environnement de programmation donné. Vous pourrez vous assurer lors de ce test final que l'algorithme donne non seulement le bon résultat mais se déroule comme prévu par les tests manuels.

### *Double sous condition*

Proposer un code permettant d'afficher le double d'un nombre donné si celui est inférieur à un nombre seuil.

### *Carré ou rectangle*

Proposer un code permettant de récupérer deux dimensions et d'afficher le type de figure concerné. Lorsque les deux dimensions sont égales et strictement positives le message est de la sorte : "Il s'agit d'un carré de côté ...".

Lorsque les deux dimensions ne sont pas égales et sont strictement positives, le message est de la sorte : "Il s'agit d'un rectangle de longueur ... et de largeur ...".

Dans le cas contraire un message d'erreur est affiché : "figure invalide".

### *Double ou triple*

Proposer un code permettant d'afficher

- le double d'un nombre donné si celui est compris dans un intervalle donné ;
- son triple s'il est en dehors de cette intervalle ;
- un message d'erreur « *intervalle incorrect* », si en donnant l'intervalle la première borne est supérieure ou égale à la deuxième borne.

### *Le plus grand des trois*

Proposer un code qui dispose de trois valeurs entières et qui affiche le plus petit des trois.

### *Le moins bon*

Proposer un code permettant de récupérer trois notes d'étudiants avec leurs noms et qui affiche le nom de l'étudiant ayant la plus petite note des trois. Si plusieurs étudiants partagent la plus mauvaise note, le message « *plus d'un étudiant avec la plus petite note* » est affiché.

### *Les très bons*

Proposer un code permettant de récupérer quatre notes d'étudiants avec leurs noms et qui affiche les noms des étudiants ayant une note supérieure strictement à seize.

### *Bosseurs non boursiers*

Proposer un code permettant de récupérer quatre notes d'étudiants avec leurs noms et leurs bourses (0 : pas de bourse, 1: demi bourse, 2: bourse entière) et qui affiche les noms des étudiants n'ayant pas de bourse et qui ont obtenu une note supérieure strictement à seize.

*Madame, mademoiselle et monsieur*

Proposer un code permettant de récupérer des informations sur un étudiant (noms, genre (homme :0, femme : 1), situation matrimoniale (célibataire :0, marié :1) et qui affiche son nom en les faisant précéder de Monsieur, Madame ou Mademoiselle selon son genre et sa situation matrimoniale.

*Lot supérieur*

Proposer un code permettant de récupérer quatre notes d'étudiants avec leurs noms et qui affiche les noms des étudiants ayant une note supérieure à la moyenne des notes des quatre étudiants.

*Au milieu des trois.*

Proposer un code qui dispose de trois valeurs entières et qui affiche celui qui se trouve au milieu des deux autres. Si aucun n'est réellement au milieu, aucun message n'est affiché.

*Périmètre et surface d'un carré*

Proposer un code permettant de calculer le périmètre et la surface d'un carré lorsque le côté est strictement positif. Dans le cas contraire un message d'erreur est affiché : "Carré invalide".

*Périmètre et surface rectangle*

Proposer un code permettant de calculer le périmètre et la surface d'un rectangle lorsque la longueur donnée est supérieure strictement à la largeur et que les deux sont strictement positif. Dans le cas contraire un message d'erreur est affiché : "Rectangle invalide".

*Périmètre et surface rectangle ou carré*

Proposer un code permettant de calculer le périmètre et la surface d'un rectangle ou d'un carré. L'utilisateur doit d'abord choisir la figure à utiliser. 1 signifie qu'il veut un rectangle et 0 qu'il veut un carré. S'il s'agit d'un rectangle, la longueur donnée est supérieure strictement à la largeur et que les deux sont strictement positif. Si tel n'est pas le cas, un message d'erreur est affiché : "Rectangle invalide".

S'il s'agit d'un carré, le côté est strictement positif. Si tel n'est pas le cas, un message d'erreur est affiché : "Carré invalide".

*Affichage de nombres si ordre croissant*

Proposer un code qui dispose de trois valeurs entières et qui les affiche si elles ont été données suivant l'ordre croissant.

*Saisie dans l'ordre croissant*

Proposer un code qui permet de saisir trois valeurs entières uniquement dans un ordre croissant. Si lors de la saisie un nombre ne respecte pas l'ordre croissant, la saisie est immédiatement arrêtée et un message d'erreur affiché : « *Saisie incorrect* ». Si la saisie est correcte le message suivant est affiché : « *Saisie correct* ».

### Ordre de trois nombres

*Mention*

*Table de multiplication*

*Devinez*

*Mesdames puis mesdemoiselles puis messieurs*

\*



Avec les structures conditionnelles, les structures itératives constituent l'autre grande partie des structures algorithmiques. Ces structures itératives sont présentées au chapitre suivant.

# Corrigés des exercices

## II

Nous proposons dans ce chapitre un ensemble de corrigés d'exercices présentés dans les chapitres précédents. Il est important, avant de consulter les corrigés des exercices, de proposer ses propres solutions, de les discuter et éventuellement de les implémenter.

## 1. Solutions des exercices sur les structures conditionnelles

### *Double sous condition*

Proposer un code permettant d'afficher le double d'un nombre donné si celui est inférieur à un nombre seuil.

```
1 Entrées:
2   a : entier
3   seuil : entier
4 Sorties:
5   doubleA: entier
6 Début
7   écrire ('Donner un nombre');
8   lire(a);
9   écrire ('Donner un seuil');
10  lire(seuil);
11  si a<seuil alors
12    début
13      doubleA:=2*a;
14      écrire ('le double est ',doubleA);
15    fin;
16 Fin.
```

Il est possible de ne pas faire de calcul intermédiaire et de réaliser l'affichage en même temps (« écrire ('le double est ', 2\*a) »). La donnée « doubleA », dans ce cas, n'est pas nécessaire.

Notons également que la structure « si » comporte deux actions. Cela rend le bloc d'actions « début... fin ; » obligatoire. Lorsque le calcul intermédiaire n'est pas réalisé, le « si » comporte une seule action. Dans ce cas le bloc « début... fin » n'est pas obligatoire.

Dans cet exercice rien n'est à faire lorsque la condition est fausse.

### *Carré ou rectangle*

Proposer un code permettant de récupérer deux dimensions et d'afficher le type de figure concerné. Lorsque les deux dimensions sont égales et strictement positives le message est de la sorte : "Il s'agit d'un carré de côté ...".

Lorsque les deux dimensions ne sont pas égales le message est de la sorte : "Il s'agit d'un rectangle de longueur ... et de largeur ...".

longueur donnée est supérieure strictement à la largeur et que les deux sont strictement positif. Dans le cas contraire un message d'erreur est affiché : "figure invalide".

```

1 Entrées:
2  dimension1 : entier
3  dimension2 : entier
4 Début
5  écrire ('Donner les deux dimensions');
6  lire(dimension1,dimension2);
7  si dimension1=dimension2 et dimension1>0 alors
8      écrire ('Il s''agit d''un carré de côté',dimension1);
9  sinon si dimension2 > dimension1 et dimension1>0 alors
10     écrire ('Il s''agit d'un rectangle de longueur',dimension2,' et de largeur ',
        dimension1 );
11  sinon si dimension1 > dimension2 et dimension2>0 alors
12     écrire ('Il s''agit d'un rectangle de longueur',dimension1,' et de largeur ',
        dimension2 );
13  sinon
14     écrire ('Figure invalide');
15
16 Fin.

```

Les sorties sont sous forme d'affichage et il n'y a aucun calcul à faire. Il n'y a donc pas lieu d'utiliser des données de sortie.

Les deux premiers tests peuvent s'écrire en ajoutant à chaque fois « *et dimension2>0* ». Cette condition n'est pas nécessaire car dans les deux cas lorsque « *dimension1>0* », cela implique que « *dimension2>0* » également.

L'utilisation du « *sinon* » permet d'avoir un code plus facile à exécuter (donc plus efficace) et rend son écriture plus facile. Le « *sinon* » correspond bien ici aux actions à faire lorsque aucune autre des trois conditions n'est réalisée.

### Double ou triple

Proposer un code permettant d'afficher

- le double d'un nombre donné si celui est compris dans un intervalle donné ;
- son triple s'il est en dehors de cette intervalle ;
- un message d'erreur« *intervalle incorrect* », si en donnant l'intervalle la première borne est supérieure ou égale à la deuxième borne.

```

1 Entrées:
2  nombre : entier
3  borneA : entier
4  borneB : entier
5 Début
6  écrire ('Donner un nombre');
7  lire(nombre);
8  écrire ('Donner les bornes A et B de l''intervalle');
9  lire(borneA, borneB);
10 si borneA>=borneB alors
11     écrire ('Intervalle incorrect');
12 sinon si nombre > borneA et nombre<borneB alors
13     écrire ('Le double est',2*nombre);

```

```

14  sinon
15      écrire ('Le triple est',3*nombre);
16 Fin.

```

Les calculs du double et du triple peuvent être fait avant leur affichage. Il faut pour cela ajouter au moins une autre donnée au code pour la sauvegarde de ces deux calculs intermédiaires.

Le « *si... sinon* » est à utiliser de préférence ici. Si aucune des deux premières conditions ne se réalisent alors les actions du troisième bloc sont à exécuter.

### *Le plus grand des trois*

Proposer un code qui dispose de trois valeurs entières et qui affiche le plus petit des trois.

```

1 Entrées:
2  val1 : entier
3  val2 : entier
4  val3 : entier
5 Début
6  écrire ('Donner trois nombres');
7  lire(val1,val2,val3);
8  si val1<=val2 et val1<=val3 alors
9      écrire (val1,' est le plus petit');
10 sinon si val2<=val1 et val2<=val3 alors
11     écrire (val2,' est le plus petit');
12 sinon
13     écrire (val3,' est le plus petit');
14 Fin.

```

Il faut juste faire attention ici à ne pas utiliser la transitivité («  $val1 \leq val2$  et  $val2 \leq val3$  »). Si cela implique que «  $val1$  » est le plus petit, ce n'est pas la seule condition pour qu'il le soit. Pour prendre en compte tous les cas, une possibilité serait la condition suivante : «  $(val1 \leq val2 \text{ et } val2 \leq val3) \text{ ou } (val1 \leq val3 \text{ et } val3 \leq val2)$  » . La condition proposée reste donc la plus facile et la plus efficace.

L'ajout de l'égalité dans les conditions permet d'afficher la plus petite valeur dans le cas où plusieurs données partagent cette valeur.

### *Le moins bon*

Proposer un code permettant de récupérer trois notes d'étudiants avec leurs noms et qui affiche le nom de l'étudiant ayant la plus petite note des trois. Si plusieurs étudiants partagent la plus mauvaise note, le message « *plus d'un étudiant avec la plus petite note* » est affiché.

```

1 Entrées:
2  note1 : réel
3  nom1: chaîne de caractères
4  note2 : réel
5  nom2: chaîne de caractères
6  note3 : réel
7  nom3: chaîne de caractères
8 Début
9  écrire ('Donner trois notes avec les noms des étudiants');
10 lire(note1,nom1,note2,nom3,nom3);
11 si note1<note2 et note1<note3 alors
12     écrire (nom1,' a la plus mauvaise note');
13 sinon si note2<note1 et note2<note3 alors

```

```

14     écrire (nom2,' a la plus mauvaise note');
15 sinon si note3<note2 et note3<note2 alors
16     écrire (nom3,' a la plus mauvaise note');
17 sinon
18     écrire (' Plus d'un étudiant avec la plus petite note');
19 Fin.

```

Il est similaire au précédent sauf dans l'affichage. Les conditions sont en effet identiques mais les données testées (notes) ne sont pas celles affichées (noms). Il faut donc garder le lien entre une note et le nom correspondant : « *note1* » et « *nom1* » appartiennent au même étudiant, ...

### *Les très bons*

Proposer un code permettant de récupérer quatre notes d'étudiants avec leurs noms et qui affiche les noms des étudiants ayant une note supérieure strictement à seize.

```

1 Entrées:
2  note1 : réel
3  nom1: chaîne de caractères
4  note2 : réel
5  nom2: chaîne de caractères
6  note3 : réel
7  nom3: chaîne de caractères
8  note4 : réel
9  nom4: chaîne de caractères
10 Début
11 écrire ('Donner quatre notes avec les noms des étudiants');
12 lire(note1,nom1,note2,nom2,note3, nom3, note4,nom4);
13 si note1>16 alors
14     écrire (nom1,' a plus de 16');
15 si note2>16 alors
16     écrire (nom2,' a plus de 16');
17 si note3>16 alors
18     écrire (nom3,' a plus de 16');
19 si note4>16 alors
20     écrire (nom4,' a plus de 16');
21 Fin.

```

Comme pour l'exercice précédent, il faut garder le lien entre la note et le nom. Cela permet de tester sur les notes et de pouvoir afficher à chaque fois les noms correspondants.

Il s'agit bien de plusieurs « *si* » et pas de « *si ... sinon* » car les quatre affichages peuvent se faire ensemble : Tous les quatre étudiants ont une note supérieure à 16.

Comme dit dans l'énoncé, il faut prendre en compte uniquement les notes strictement supérieures. Dans les conditions il faut donc utiliser « *>* » et non pas « *>=* ».

### *Bosseurs non boursiers*

Proposer un code permettant de récupérer quatre notes d'étudiants avec leurs noms et leurs bourses (0 : pas de bourse, 1: demi bourse, 2: bourse entière) et qui affiche les noms des étudiants n'ayant pas de bourse et qui ont obtenu une note supérieure strictement à seize.

```

1 Entrées:
2  note1 : réel
3  nom1: chaîne de caractères

```

```

4  bou1: entier
5  note2 : réel
6  nom2: chaîne de caractères
7  bou2: entier
8  note3 : réel
9  nom3: chaîne de caractères
10 bou3: entier
11 note4 : réel
12 nom4: chaîne de caractères
13 bou4: entier
14 Début
15 écrire ('Donner quatre noms d''étudiants');
16 lire(nom1,nom2, nom3, nom4);
17 écrire ('Donner leurs notes respectives');
18 lire(note1,note2,note3,note4);
19 écrire ('Donner leurs bourses respectives');
20 lire(bou1,bou2,bou3,bou4);
21 si bou1=0 et note1>16 alors
22     écrire (nom1,' est bosseur non boursier');
23 si bou2=0 et note2>16 alors
24     écrire (nom2,' est bosseur non boursier');
25 si bou3=0 et note3>16 alors
26     écrire (nom3,' est bosseur non boursier');
27 si bou4=0 et note4>16 alors
28     écrire (nom4,' est bosseur non boursier');
29 Fin.

```

Il est important de gérer comme dans les exercices précédents le lien entre les données (le nom avec sa note et sa bourse).

Le test porte à la fois sur la bourse et la note. Lorsque ce test est positif, le nom correspondant est affiché.

Plusieurs si sont à utiliser car les conditions sont indépendantes ( le fait qu'un étudiant soit bosseur et boursier n'a aucun lien avec le statut des autres étudiants) et il est possible de réaliser plusieurs affichages (d'avoir plusieurs étudiants bosseurs non boursiers).

### *Madame, mademoiselle et monsieur*

Proposer un code permettant de récupérer des informations sur un étudiant (nom, genre (homme :0, femme :1), situation matrimonial (célibataire :0, marié :1) et qui affiche son nom en les faisant précéder de Monsieur, Madame ou Mademoiselle selon son genre et sa situation matrimoniale.

```

1 Entrées:
2  nom: chaîne de caractères
3  genre: entier
4  situation:entier
5 Début
6 écrire ('Donner le nom, genre et situation de l''étudiant');
7 lire(nom,genre, situation);
8 si genre=0 alors
9     écrire ('Monsieur ',nom);
10 sinon si situation=0 alors
11     écrire ('Mademoiselle ',nom);
12 sinon
13     écrire ('Madame ',nom);
14 Fin.

```

Nous sommes soit monsieur, soit madame, soit mademoiselle (les trois conditions sont disjointes et forment un tout). Le « *si ... sinon* » est donc à utiliser de préférence.

### *Lot supérieur*

Proposer un code permettant de récupérer quatre notes d'étudiants avec leurs noms et qui affiche les noms des étudiants ayant une note supérieure à la moyenne des notes des quatre étudiants.

```

1 Entrées:
2  note1 : réel
3  nom1: chaîne de caractères
4  note2 : réel
5  nom2: chaîne de caractères
6  note3 : réel
7  nom3: chaîne de caractères
8  note4 : réel
9  nom4: chaîne de caractères
10 intermediaires:
11  moyenne : réel
12 Début
13 écrire ('Donner quatre notes avec les noms des étudiants');
14 lire(note1,nom1,note2,nom2,note3, nom3, note4,nom4);
15 moyenne:=(note1+note2+note3+note4)/4;
16 si note1>moyenne alors
17     écrire (nom1,' est dans le lot supérieur');
18 si note2>moyenne alors
19     écrire (nom2,' est dans le lot supérieur');
20 si note3>moyenne alors
21     écrire (nom3,' est dans le lot supérieur');
22 si note4>moyenne alors
23     écrire (nom4,' est dans le lot supérieur');
24 Fin.
```

Les conditions de test concernent les notes et la moyenne. Celle ci doit donc être calculée avant de débiter les tests.

Les quatre affichages ne sont pas liés et peuvent se faire en même temps, d'où l'utilisation de « *si* » séparés à la place de « *si ... sinon* ».

### *Au milieu des trois.*

Proposer un code qui dispose de trois valeurs entières et qui affiche celui qui se trouve au milieu des deux autres. Si aucun n'est réellement au milieu, aucun message n'est affiché.

```

1 Entrées:
2  val1 : entier
3  val2 : entier
4  val3 : entier
5 Début
6 écrire ('Donner trois nombres');
7 lire(val1,val2,val3);
8 si (val1<val2 et val1>val3) ou (val1<val3 et val1>val2) alors
9     écrire (val1,' est au milieu');
10 sinon (val2<val1 et val2>val3) ou (val2<val3 et val2>val1) alors
11     écrire (val2,' est au milieu');
12 sinon si(val3<val2 et val3>val1) ou (val3<val1 et val3>val2) alors
```

```

13     écrire (val3, ' est au milieu');
14 Fin.

```

Pour écrire les conditions de test, il faut prendre en compte que les deux autres données ne sont pas toujours ordonnées.

Les trois conditions sont disjointes et forment un tout, d'où l'utilisation du « *si ... sinon* ».

### *Périmètre et surface d'un carré*

Proposer un code permettant de calculer le périmètre et la surface d'un carré lorsque le côté est strictement positif. Dans le cas contraire un message d'erreur est affiché : "Carré invalide".

```

1 Entrées:
2  cote : entier
3 Sorties:
4  perimetre : entier
5  surface : entier
6 Début
7  écrire ('Donner le côté');
8  lire(cote);
9  si (cote>0) alors
10   début
11     perimetre:=cote*4;
12     surface:=cote*cote;
13     écrire ('Le périmètre est ',perimetre);
14     écrire ('La surface est ',surface);
15   fin;
16 sinon
17   écrire (' Carré invalide');
18 Fin.

```

Le bloc « *si* » comporte plusieurs actions d'où la nécessité de les mettre dans un bloc « *début ... fin* ».

Les deux actions d'affichage (« *surface* », « *perimetre* ») sont obligatoirement dans le bloc. Les calculs intermédiaires sont optionnels et peuvent être faits avant le test. Comme ils ne sont utilisés que dans le bloc « *si* », il est plus efficace de les mettre à l'intérieur. En effet, les calculs ne sont alors fait que si la condition (« *cote>0* ») est vraie.

### *Périmètre et surface rectangle*

Proposer un code permettant de calculer le périmètre et la surface d'un rectangle lorsque la longueur donnée est supérieure strictement à la largeur et que les deux sont strictement positif. Dans le cas contraire un message d'erreur est affiché : "Rectangle invalide".

```

1 Entrées:
2  longueur : entier
3  largeur : entier
4 Sorties:
5  perimetre : entier
6  surface : entier
7 Début
8  écrire ('Donner la longueur et la largeur');
9  lire(longueur,largeur);
10 si longueur>largeur et largeur>0 alors
11   début

```



```

12     perimetre:=(longueur + largeur)*2;
13     surface:=longueur*largeur;
14     écrire ('Le périmètre est ',perimetre);
15     écrire ('La surface est ',surface);
16     fin;
17 sinon
18     écrire (' Rectangle invalide');
19 Fin.

```

Il n'est pas nécessaire d'ajouter à la première condition « *longueur>0* » car « *longueur>largeur* » et « *largeur>0* ».

Le bloc « *si* » comporte plusieurs actions, d'où la nécessité de les mettre dans un bloc « *début ... fin* ».

### *Périmètre et surface rectangle ou carré*

Proposer un code permettant de calculer le périmètre et la surface d'un rectangle ou d'un carré. L'utilisateur doit d'abord choisir la figure à utiliser. 1 signifie qu'il veut un rectangle et 0 qu'il veut un carré. S'il s'agit d'un rectangle, la longueur donnée est supérieure strictement à la largeur et que les deux sont strictement positifs. Si tel n'est pas le cas, un message d'erreur est affiché : "Rectangle invalide".

S'il s'agit d'un carré, le côté est strictement positif. Si tel n'est pas le cas, un message d'erreur est affiché : "Carré invalide".

```

1 Entrées:
2  longueur : entier
3  largeur  : entier
4  typeFig:entier
5 Sorties:
6  perimetre : entier
7  surface   : entier
8 Début
9  écrire ('Donner le type de figure à saisir');
10 lire(typeFig);
11 si typeFig=0 alors
12     début
13         écrire ('Donner la longueur et la largeur');
14         lire(longueur,largeur);
15         si longueur>largeur et largeur>0 alors
16             début
17                 perimetre:=(longueur + largeur)*2;
18                 surface:=longueur*largeur;
19                 écrire ('Le périmètre est ',perimetre);
20                 écrire ('La surface est ',surface);
21             fin;
22         sinon
23             écrire (' Rectangle invalide');
24         fin;
25 sinon si typeFig=1 alors
26     début
27         écrire ('Donner le côté');
28         lire(longueur);
29         si (longueur>0) alors
30             début
31                 perimetre:=longueur*4;
32                 surface:=longueur*longueur;
33                 écrire ('Le périmètre est ',perimetre);
34                 écrire ('La surface est ',surface);

```

```

35         fin;
36     sinon
37         écrire (' Carré invalide');
38     fin;
39 Fin.

```

Le code comporte deux grands blocs qui ont chacun plusieurs actions : un bloc pour chaque type de figure. Les codes dans chaque bloc sont similaires aux exercices précédents sur chaque figure.

### *Affichage de nombres si ordre croissant*

Proposer un code qui dispose de trois valeurs entières et qui les affiche si elles ont été données suivant l'ordre croissant.

```

1 Entrées:
2   val1 : entier
3   val2 : entier
4   val3 : entier
5 Début
6   écrire ('Donner trois nombres');
7   lire(val1,val2,val3);
8   si val1<=val2 et val2<=val3 alors
9       écrire (val1,' ',val2,' ',val3);
10 Fin.

```

Un seul test est fait pour s'assurer que les nombres sont saisis dans l'ordre croissant. Si c'est le cas, l'affichage est fait. Dans le cas contraire rien n'est fait.

### *Saisie dans l'ordre croissant*

Proposer un code qui permet de saisir trois valeurs entières uniquement dans un ordre croissant. Si lors de la saisie un nombre ne respecte pas l'ordre croissant, la saisie est immédiatement arrêtée et un message d'erreur affiché : « *Saisie incorrect* ». Si la saisie est correcte le message suivant est affiché : « *Saisie correct* ».

```

1 Entrées:
2   val1 : entier
3   val2 : entier
4   val3 : entier
5 Début
6   écrire ('Donner deux nombres');
7   lire(val1,val2);
8   si val1<=val2 alors
9       début
10          écrire ('Donner un nombre');
11          lire(val3);
12          si val2<=val3 alors
13              écrire ('Saisie correcte');
14          sinon
15              écrire ('Saisie incorrecte');
16      fin;
17 sinon
18     écrire('Saisie incorrecte');
19 Fin.

```

La saisie des deux premières données se fait sans condition. Tel n'est pas le cas pour la troisième saisie qui ne se fait que lorsque les deux premières données ont été correctement saisies ( $val1 \leq val2$ ).

Le bloc « *si* » comporte trois actions (le « *si...sinon* » peut être considéré comme un bloc), et doit disposer d'un bloc « *début...fin* ».

Deux possibilités d'affichage d'un message incorrecte, d'où les deux « *sinon* » : « *val2* » mal saisie ou bien « *val3* » mal saisie.

### *Seul contre tous*

Proposer un code qui dispose de trois valeurs entières et qui affiche celui des trois qui est de signe contraire aux deux autres. S'il n'existe pas un message d'erreur est affiché : "Tous pareils..."

```
1 Entrées:
2   val1 : entier
3   val2 : entier
4   val3 : entier
5 Début
6   écrire ('Donner trois nombres');
7   lire(val1,val2,val3);
8   si (val1<0 et val2>0 et val3>0) ou (val1>0 et val2<0 et val3<0) alors
9     écrire (val1, ' est seul dans ce cas');
10  sinon si (val2<0 et val1>0 et val3>0) ou (val2>0 et val1<0 et val3<0) alors
11    écrire (val2, ' est seul dans ce cas');
12  sinon si (val3<0 et val2>0 et val1>0) ou (val3>0 et val2<0 et val1<0) alors
13    écrire (val3, ' est seul dans ce cas');
14  sinon
15    écrire ('Tous pareil');
16 Fin.
```

Quatre possibilités d'affichage, d'où les quatre blocs : Une des trois valeurs ou le message d'erreur.

Pour les tests, il faut prendre en compte que chaque donnée peut être soit positive, soit négative.

### *Ordre de trois nombres*

Proposer un code qui dispose de trois valeurs entières et qui affiche l'ordre dans lequel ils ont été saisies: Soit «Ordre croissant», «Ordre décroissant » ou « Désordonné ».

```
1 Entrées:
2   val1 : entier
3   val2 : entier
4   val3 : entier
5 Début
6   écrire ('Donner trois nombres');
7   lire(val1,val2,val3);
8   si val1<val2 et val2<val3 alors
9     écrire ('Ordre croissant');
10  sinon si val1>val2 et val2>val3 alors
11    écrire ('Ordre décroissant');
12  sinon
13    écrire ('Désordonné');
14 Fin.
```

Nous avons fait le choix de sortir les égalités, des ordres croissant et décroissant.

Trois affichages sont possibles. Ils sont disjoints et forment un tout ; d'où l'utilisation de « *si...sinon* ».

## Mention

Proposer un code qui dispose de la moyenne d'un étudiant et qui affiche sa mention parmi les suivantes: Soit «Très bien», «Bien », « Abien », « Passable », « Echec ».

```

1 Entrées:
2  moyenne : réel
3 Début
4  écrire ('Donner la moyenne');
5  lire(moyenne);
6  si moyenne>=16 alors
7    écrire ('Très bien');
8  sinon si moyenne >=14 alors
9    écrire ('Bien');
10 sinon si moyenne >=12 alors
11   écrire ('Abien');
12 sinon si moyenne>=10 alors
13   écrire ('Passable');
14 sinon
15   écrire('Echec');
16 Fin.

```

L'utilisation du « *si...sinon* » permet d'éviter l'ajout de conditions redondantes. Exemple : Il n'est pas nécessaire d'ajouter la condition « *moyenne<16* » au deuxième test (« *moyenne>=14* ») car le sinon signifie déjà cela.

## Table de multiplication

Proposer un code qui demande à l'utilisateur le résultat d'une multiplication (Exemple : Combien font 4\*7 ?) et qui affiche lorsque l'utilisateur donne un bon résultat « *Très bien* », sinon « *Incorrect* ». Les nombres à multiplier sont choisis aléatoirement entre 1 et 12.

```

1 Entrées:
2  nb1 : entier
3  nb2 : entier
4  resultatProp: entier
5 Intermédiaires:
6  resultatV : entier
7 Début
8 nb1:=7;
9 nb2:=8;
10 resultatV:=nb1*nb2;
11 écrire (nb1, 'x', nb2, '=');
12 lire(resultatProp);
13 si resultatV=resultatProp alors
14   écrire ('Très bien');
15 sinon
16   écrire ('Incorrecte');
17 Fin.

```

En programmation il existe généralement une fonction pour générer aléatoirement un nombre. Elle est à utiliser pour initialiser « *nb1* » et « *nb2* » à la place des affectations simples faites ici.

Le calcul du bon résultat doit être fait avant le test qui consiste à le comparer avec le résultat proposé par l'utilisateur.

*Devinez*

Proposer un code qui demande à l'utilisateur de deviner un nombre fixé au préalable et qui affiche suivant la valeur donnée l'un des messages suivants : « *Trop grand* », « *Trop petit* », « *Bravo* ». Le nombre à deviner est choisi aléatoirement entre 1 et 10.

```

1 Entrées:
2  nb : entier
3  proposition: entier
4 Début
5 nb:=5;
6 écrire('Donner un nombre');
7 lire(proposition);
8 si nb=proposition alors
9   écrire ('Bravo');
10 sinon si nb<proposition alors
11   écrire ('Trop grand');
12 sinon
13   écrire ('Trop petit');
14 Fin.

```

Ici aussi, l'affectation de la donnée « *nb* » doit être remplacée en programmation par l'utilisation d'une fonction de génération aléatoire d'un nombre.

Les conditions de test sont disjointes et forment un tout ; d'où l'utilisation de « *si...sinon* ».

*Mesdames puis mesdemoiselles puis messieurs*

Proposer un code permettant de récupérer des informations sur quatre étudiants (noms, genre (homme :0, femme :1), situation matrimoniale (célibataire :0, marié :1) et qui affiche ces étudiants séparément suivant leur genre et leur situation matrimoniale. Vous afficherez d'abord les mesdames, ensuite les mesdemoiselles et enfin les messieurs.

Modifier ce code pour faire précéder chaque groupe d'aucune mention (aucun élément dans ce groupe), de la mention madame, mademoiselle ou monsieur (un seul élément), ou de la mention mesdames, mesdemoiselles ou messieurs.

```

1 Entrées:
2  nom1: chaîne de caractères
3  genre1: entier
4  situation1:entier
5  nom2: chaîne de caractères
6  genre2: entier
7  situation2:entier
8  nom3: chaîne de caractères
9  genre3: entier
10 situation3:entier
11 nom4: chaîne de caractères
12 genre4: entier
13 situation4:entier
14 Début
15 écrire ('Donner quatre noms d''étudiants');
16 lire(nom1,nom2, nom3, nom4);
17 écrire ('Donner leurs genres respectifs');
18 lire(genre1,genre2,genre3,genre4);
19 écrire ('Donner leurs situations respectives');

```

```

20 lire(situation1,situation2,situation3,situation4);
21 écrire("Mesdames");
22 si genre1=1 et situation1=1 alors
23   écrire (nom1);
24 si genre2=1 et situation2=1 alors
25   écrire (nom2);
26 si genre3=1 et situation3=1 alors
27   écrire (nom3);
28 si genre4=1 et situation4=1 alors
29   écrire (nom4);
30 écrire("Mesdemoiselles");
31 si genre1=1 et situation1=0 alors
32   écrire (nom1);
33 si genre2=1 et situation2=0 alors
34   écrire (nom2);
35 si genre3=1 et situation3=0 alors
36   écrire (nom3);
37 si genre4=1 et situation4=0 alors
38   écrire (nom4);
39 écrire("Messieurs");
40 si genre1=0 alors
41   écrire (nom1);
42 si genre2=0 alors
43   écrire (nom2);
44 si genre3=0 alors
45   écrire (nom3);
46 si genre4=0 alors
47   écrire (nom4);
48 Fin.

```

Il faut gérer le lien entre les données : L'étudiant « *nom1* » a comme genre « *genre1* » et comme situation matrimoniale « *situation1* ».

Le test porte soit sur le genre et la situation s'il s'agit d'une femme, soit sur le genre s'il s'agit d'un homme.

Il est difficile d'utiliser des « *si...sinon* » puisque les affichages sont regroupés.

```

1 Entrées:
2  nom1: chaîne de caractères
3  genre1: entier
4  situation1:entier
5  nom2: chaîne de caractères
6  genre2: entier
7  situation2:entier
8  nom3: chaîne de caractères
9  genre3: entier
10 situation3:entier
11 nom4: chaîne de caractères
12 genre4: entier
13 situation4:entier
14 Début
15 écrire ('Donner quatre noms d''étudiants');
16 lire(nom1,nom2, nom3, nom4);
17 écrire ('Donner leurs genres respectifs');
18 lire(genre1,genre2,genre3,genre4);
19 écrire ('Donner leurs situations respectives');
20 lire(situation1,situation2,situation3,situation4);
21 mada=0;
22 made=0;

```

```

23 mons=0;
24 si genre1=0 alors
25   mons=mons+1;
26 sinon si situation1=0
27   mada=mada+1;
28 sinon
29   mada=mada+1;
30 si genre2=0 alors
31   mons=mons+1;
32 sinon si situation2=0
33   made=made+1;
34 sinon
35   mada=mada+1;
36 si genre3=0 alors
37   mons=mons+1;
38 sinon si situation3=0
39   made=made+1;
40 sinon
41   mada=mada+1;
42 si genre4=0 alors
43   mons=mons+1;
44 sinon si situation4=0
45   made=made+1;
46 sinon
47   mada=mada+1
48 Si mada >1 alors
49   écrire("Mesdames");
50 sinon si mada =1 alors
51   écrire ("Madame");
52 Si mada>0 alors
53   //code identique au code précédent pour afficher les mesdames ou rien du
   tout...
54 Si made >1 alors
55   écrire("Mesdemoiselles");
56 sinon si made =1 alors
57   écrire ("Mademoiselle");
58 si made>0 alors
59   //code identique au code précédent pour afficher les mesdemoiselles ou rien du
   tout ...
60 Si mons >1 alors
61   écrire("Messieurs");
62 sinon si mons =1 alors
63   écrire ("Monsieur");
64 Si mons >0 alors
65   //code identique au code précédent pour afficher les messieurs ou rien du
   tout...
66 Fin.

```

La grande différence avec le code précédent, est qu'il faut au préalable calculer le nombre de chaque catégorie pour connaître la mention à afficher. Le contenu de chacun des trois blocs reste identique au cas précédent. Ce contenu peut être mis dans un bloc « *si* » pour vérifier si c'est vraiment la peine d'essayer d'afficher.