Un **algorithme** est une description structurée de la liste des actions effectuées par un programme.

Il peut être représenté graphiquement, c'est alors un **organigramme** (ou **algorigramme**), ou écrit sous une forme littérale, appelée **langage algorithmique**. On l'appellera « l'algorithme » pour simplifier. En effet, dans les exercices, un algorithme apparaîtra le plus couramment sous forme d'une liste d'instructions, souvent dans le langage AlgoBox.

L'organigramme, qui n'est pas exigible ici, permet cependant dans certains cas de mieux comprendre certaines procédures à effectuer, car l'aspect linéaire de l'algorithme classique peut parfois manquer de clarté.

## 1. DÉBUT – FIN

**Extrémités** de l'algorithme, symbolisées par des « stades » (rectangles encadrés de deux demi-cercles).

**DÉBUT** : Doit être suivi de la définition des variables utilisées dans l'algorithme.

**FIN** : Doit théoriquement être le passage obligé à la fin de la procédure. Si ce n'est pas le cas, l'algorithme risque de « tourner » indéfiniment.

## 2. LECTURE – ÉCRITURE

**Transferts d'informations** entre l'utilisateur et la machine.

**LECTURE** : **saisie** (transfert de l'utilisateur vers la machine), à l'aide du clavier [instruction symbolisée par un parallélogramme]).

**ÉCRITURE** : **affichage** (transfert de la machine vers l'utilisateur), à l'aide de l'écran [même symbolisme]).

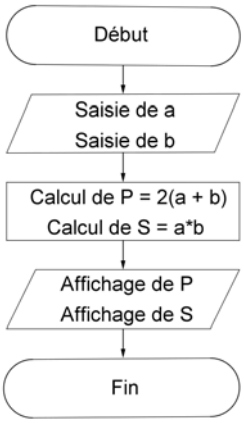
## 3. CALCULS – AFFECTATION DE VARIABLES

**CALCULS :** Traitement interne à la machine [symbolisés par un rectangle].

**AFFECTATION DE VARIABLES :** Stockage de valeurs numériques (ou autres…), repérées par leur nom [symbolisées par une lettre ou même parfois un mot].

**Exemple 3.1** : Calcul du périmètre et de la superficie d'un rectangle connaissant ses dimensions.

**Organigramme** :




**En langage AlgoBox** :

```
1   VARIABLES
2       a EST_DU_TYPE NOMBRE
3       b EST_DU_TYPE NOMBRE
4       P EST_DU_TYPE NOMBRE
5       S EST_DU_TYPE NOMBRE
6   DEBUT_ALGORITHME
7       LIRE a
8       LIRE b
9       P PREND_LA_VALEUR 2*(a+b)
10      S PREND_LA_VALEUR a*b
11      AFFICHER "Périmètre : "
12      AFFICHER P
13      AFFICHER "Superficie : "
14      AFFICHER S
15  FIN_ALGORITHME
```

**Exercice 3.2** : Construire un organigramme et un algorithme permettant de calculer la superficie d'un disque connaissant son périmètre.

**Exercice 3.3** : Dans l'algorithme ci-contre, montrer que l'on pourrait exprimer y directement en fonction de x avec une instruction ne comportant qu'une seule opération.

**Exercice 3.4** : Construire l'organigramme et l'algorithme du problème suivant : Calculer le prix HT d'un objet connaissant son prix TTC (le taux de TVA étant variable et donné par l'utilisateur).

```
1   VARIABLES
2       a EST_DU_TYPE NOMBRE
3       b EST_DU_TYPE NOMBRE
4       c EST_DU_TYPE NOMBRE
5       d EST_DU_TYPE NOMBRE
6       x EST_DU_TYPE NOMBRE
7       y EST_DU_TYPE NOMBRE
8   DEBUT_ALGORITHME
9       LIRE x
10      a PREND_LA_VALEUR x+2
11      b PREND_LA_VALEUR a*a
12      c PREND_LA_VALEUR x-2
13      d PREND_LA_VALEUR c*c
14      y PREND_LA_VALEUR (b-d)/4
15  AFFICHER y
16  FIN_ALGORITHME
```

**Exercice 3.5** : Expliciter ce que font les deux algorithmes ci-dessous et expliquer pourquoi ils ne donnent pas (en général) le même résultat.

```
1  VARIABLES
2      a EST_DU_TYPE NOMBRE
3      b EST_DU_TYPE NOMBRE
4      c EST_DU_TYPE NOMBRE
5  DEBUT_ALGORITHME
6      LIRE a
7      LIRE b
8      LIRE c
9      c PREND_LA_VALEUR a+b
10     a PREND_LA_VALEUR b+c
11     b PREND_LA_VALEUR c+a
12     AFFICHER "a = "
13     AFFICHER a
14     AFFICHER "b = "
15     AFFICHER b
16     AFFICHER "c = "
17     AFFICHER c
18 FIN_ALGORITHME
```

```
1  VARIABLES
2      a EST_DU_TYPE NOMBRE
3      b EST_DU_TYPE NOMBRE
4      c EST_DU_TYPE NOMBRE
5  DEBUT_ALGORITHME
6      LIRE a
7      LIRE b
8      LIRE c
9      a PREND_LA_VALEUR b+c
10     b PREND_LA_VALEUR c+a
11     c PREND_LA_VALEUR a+b
12     AFFICHER "a = "
13     AFFICHER a
14     AFFICHER "b = "
15     AFFICHER b
16     AFFICHER "c = "
17     AFFICHER c
18 FIN_ALGORITHME
```

**Exercice 3.6** : Simplifier l'algorithme ci-dessous en n'utilisant que la variable *x*.

```
1  VARIABLES
2      x EST_DU_TYPE NOMBRE
3      y EST_DU_TYPE NOMBRE
4      z EST_DU_TYPE NOMBRE
5  DEBUT_ALGORITHME
6      LIRE x
7      y PREND_LA_VALEUR 2*x+1
8      z PREND_LA_VALEUR y*y-1
9      x PREND_LA_VALEUR z/4
10     AFFICHER x
11 FIN_ALGORITHME
```

**Exercice 3.7** : Écrire un algorithme permettant d'échanger les valeurs de deux variables.

**Exercice 3.8** : Un algorithme contient dans cet ordre les instructions suivantes :

« *a prend la valeur 5* », « *b prend la valeur 2* », « *a prend la valeur b* » et « *b prend la valeur a* ».
Déterminer les valeurs finales de *a* et de *b* et faire une remarque concernant l'une des instructions.

**Exercice 3.9** : Écrire l'algorithme suivant et commenter le résultat : choisir un entier, calculer le carré de l'entier suivant, puis le carré de l'entier précédent, puis la différence (positive) de ces deux carrés, puis enfin le quotient de ce dernier résultat par l'entier de départ.

**Exercice 3.10** : Écrire l'algorithme permettant de calculer la moyenne (sans coefficients) de trois notes de contrôle, données par l'utilisateur.

**Exercice 3.11** : Même exercice que précédemment avec une moyenne avec coefficients, donnés aussi par l'utilisateur.

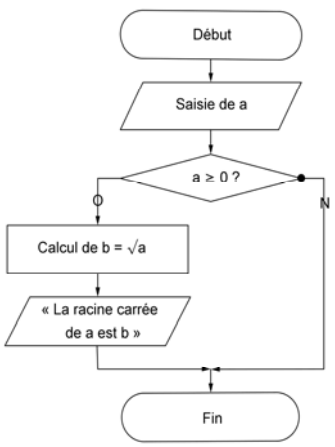
**Exercice 3.12** : Écrire l'algorithme permettant de calculer le pourcentage de variation entre un premier nombre et un deuxième, donnés dans l'ordre par l'utilisateur.

**Exercice 3.13** : Écrire l'algorithme permettant de calculer le terme de rang *n* d'une suite arithmétique, l'entier *n*, le premier terme et la raison étant donnés par l'utilisateur.
Même question avec une suite géométrique.

## 4. TEST

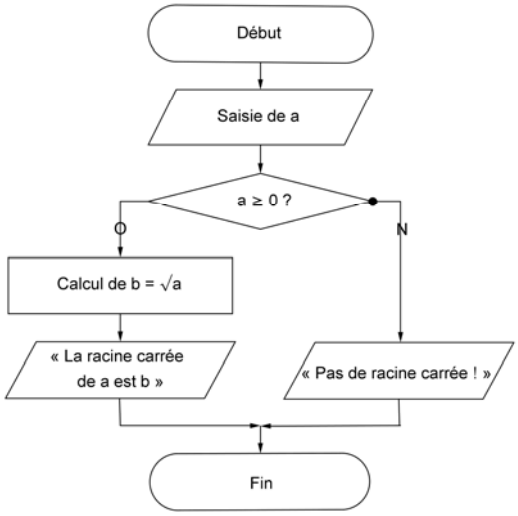
**TEST** « SI… ALORS… [SINON] » : permet un choix ayant deux réponses possibles (VRAI ou FAUX), avec des instructions à réaliser si la réponse est VRAI, et éventuellement si la réponse est FAUX, par l'intermédiaire d'un « SINON » [symbolisée par un losange – le branchement pour FAUX est parfois repéré par un rond noir].

**Exemple 4.1** : Calcul de la racine carrée d'un réel positif, et sans réaction lorsqu'il est négatif.




```
1  VARIABLES
2      a EST_DU_TYPE NOMBRE
3      b EST_DU_TYPE NOMBRE
4  DEBUT_ALGORITHME
5      LIRE a
6      SI (a>=0) ALORS
7          DEBUT_SI
8              b PREND_LA_VALEUR sqrt(a)
9              AFFICHER "La racine carrée de a est : "
10             AFFICHER b
11         FIN_SI
12 FIN_ALGORITHME
```

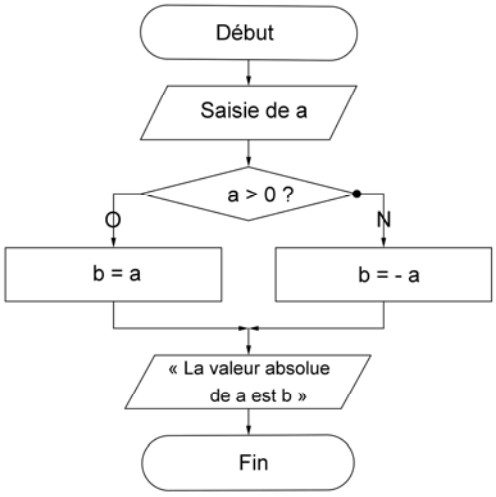
**Exemple 4.2** : Calcul de la racine carrée d'un réel positif, et affichage d'un message d'erreur lorsqu'il est négatif.




```
1   VARIABLES
2       a EST_DU_TYPE NOMBRE
3       b EST_DU_TYPE NOMBRE
4   DEBUT_ALGORITHME
5       LIRE a
6       SI (a>=0) ALORS
7           DEBUT_SI
8               b PREND_LA_VALEUR sqrt(a)
9               AFFICHER "La racine carrée de a est : "
10              AFFICHER b
11          FIN_SI
12          SINON
13              DEBUT SINON
14                  AFFICHER « Pas de racine carrée ! »
15              FIN SINON
16  FIN_ALGORITHME
```

**Remarque 4.3** : « sqrt(*a*) » renvoie en langage AlgoBox la racine carrée (positive) de *a* (lorsque *a* est positif).

**Exercice 4.4** : Voici l'organigramme et l'algorithme du calcul de la valeur absolue d'un réel.
Refaire l'un et l'autre sans utiliser l'instruction « SINON ».




```
1   VARIABLES
2       a EST_DU_TYPE NOMBRE
3       b EST_DU_TYPE NOMBRE
4   DEBUT_ALGORITHME
5       LIRE a
6       SI (a>0) ALORS
7           DEBUT_SI
8               b PREND_LA_VALEUR a
9           FIN_SI
10          SINON
11              DEBUT_SINON
12              b PREND_LA_VALEUR -a
13          FIN_SINON
14      AFFICHER "La valeur absolue de "
15      AFFICHER a
16      AFFICHER " est : "
17      AFFICHER b
18  FIN_ALGORITHME
```

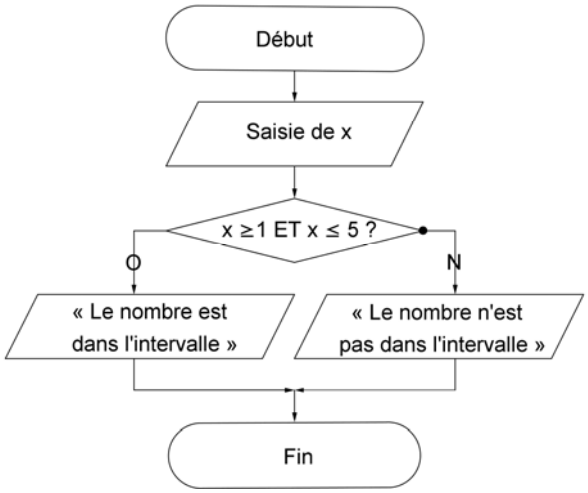
**Exercice 4.5** : Construire un organigramme et l'algorithme correspondant, indiquant le plus grand de deux réels donnés par l'utilisateur. Prévoir un message si les deux nombres sont égaux.

**Exercice 4.6** : Écrire un algorithme saisissant cinq réels donnés par l'utilisateur et indiquant, au fur et à mesure, le plus grand des réels saisis.

## 5. TEST AVEC CONNECTEURS

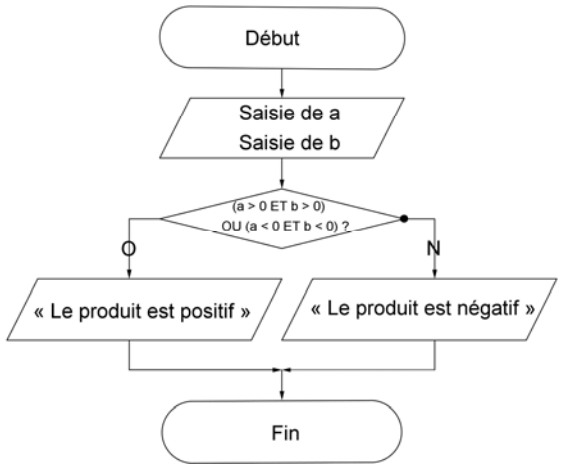
**Remarque 5.1** : Si A et B sont deux affirmations pouvant être vraies ou bien fausses, alors l'affirmation **(A ET B)** est vraie si et seulement si les deux affirmations sont vraies simultanément, et **(A OU B)** est vraie si et seulement si au moins l'une des deux affirmations est vraie.

**Exemple 5.2** : Détermination si un nombre appartient à l'intervalle [1 ; 5].




```
1   VARIABLES
2       x EST_DU_TYPE NOMBRE
3   DEBUT_ALGORITHME
4       LIRE x
5       SI ((x>=1) ET (x<=5)) ALORS
6           DEBUT_SI
7               AFFICHER "Le nombre est dans l'intervalle"
8           FIN_SI
9           SINON
10              DEBUT_SINON
11              AFFICHER "Le nombre n'est pas dans l'intervalle"
12              FIN_SINON
13  FIN_ALGORITHME
```

**Exemple 5.3** : Détermination du signe du produit de deux nombres non nuls sans faire le calcul du produit.




```
1   VARIABLES
2       a EST_DU_TYPE NOMBRE
3       b EST_DU_TYPE NOMBRE
4   DEBUT_ALGORITHME
5       LIRE a
6       LIRE b
7       SI ((a>0 ET b>0) OU (a<0 ET b<0)) ALORS
8           DEBUT_SI
9               AFFICHER "Le produit est positif"
10          FIN_SI
11          SINON
12              DEBUT_SINON
13                  AFFICHER "Le produit est négatif"
14              FIN_SINON
15  FIN_ALGORITHME
```

**Exercice 5.4** : Écrire un algorithme permettant de vérifier si un réel a son carré supérieur ou égal à 1, sans calculer effectivement le carré.

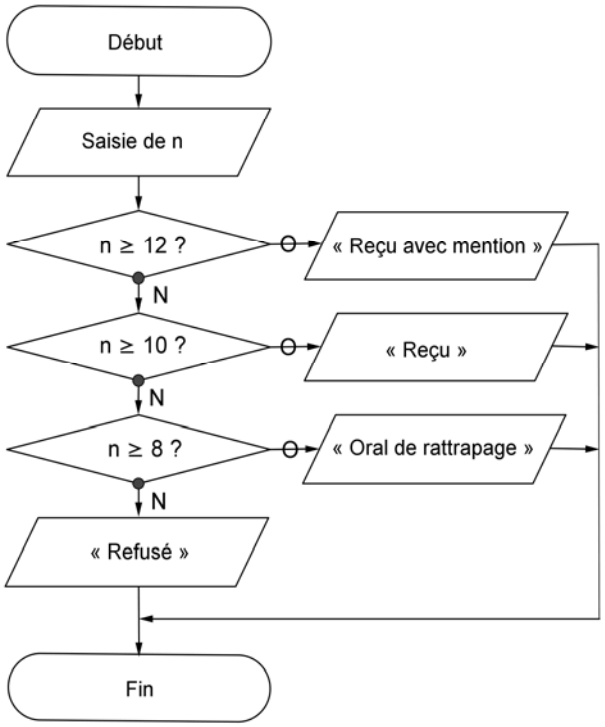
**Exercice 5.5** : Écrire un algorithme saisissant trois réels donnés par l'utilisateur et indiquant s'ils ont été saisis dans l'ordre croissant, décroissant ou dans le désordre.

**Exercice 5.6** : Une année est **bissextile** lorsqu'elle est divisible par 4, sans être divisible par 100, sauf si elle est divisible par 400. Écrire un algorithme indiquant si une année, saisie par l'utilisateur, est bissextile ou non.

## 6. TESTS IMBRIQUÉS

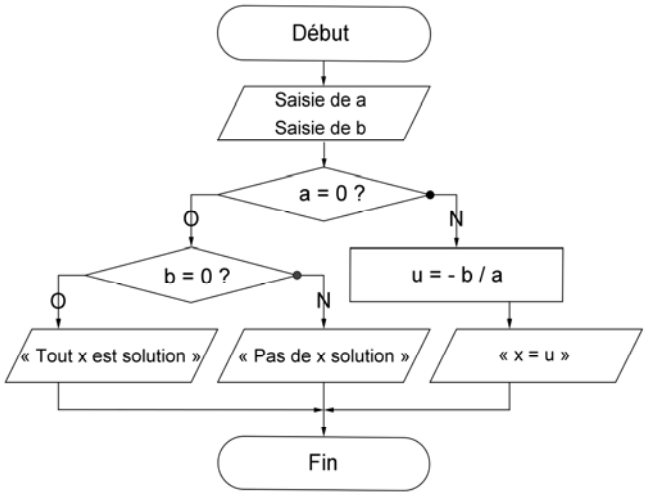
**Exemple 6.1** : Affichage des résultats à un examen selon la moyenne obtenue par le candidat.

**Remarque 6.2** : L'organigramme indique après chaque affichage un transfert « ALLER À » vers « FIN ». Cependant, dans un algorithme, il est classique d'imbriquer les tests pour éviter ces transferts.




```
1   VARIABLES
2       n EST_DU_TYPE NOMBRE
3   DEBUT_ALGORITHME
4       LIRE n
5       SI (n>=12) ALORS
6           DEBUT_SI
7           AFFICHER "Reçu avec mention"
8           FIN_SI
9           SINON
10              DEBUT_SINON
11              SI (n>=10) ALORS
12                  DEBUT_SI
13                  AFFICHER "Reçu"
14                  FIN_SI
15                  SINON
16                      DEBUT_SINON
17                      SI (n>=8) ALORS
18                          DEBUT_SI
19                          AFFICHER "Oral de rattrapage"
20                          FIN_SI
21                          SINON
22                              DEBUT_SINON
23                              AFFICHER "Refusé"
24                              FIN_SINON
25                      FIN_SINON
26              FIN_SINON
27  FIN_ALGORITHME
```

**Exemple 6.3** : Résolution de l'équation $ax + b = 0$, dont l'inconnue est $x$.




```
1       VARIABLES
2           a EST_DU_TYPE NOMBRE
3           b EST_DU_TYPE NOMBRE
4           u EST_DU_TYPE NOMBRE
5       DEBUT_ALGORITHME
6           SI (a==0) ALORS
7               DEBUT_SI
8                   SI (b==0) ALORS
9                       DEBUT_SI
10                          AFFICHER "Tout x est solution"
11                      FIN_SI
12                  SINON
13                      DEBUT_SINON
14                          AFFICHER "Pas de x solution"
15                  FIN_SINON
16              FIN_SI
17          SINON
18              DEBUT_SINON
19                  u PREND_LA_VALEUR -b/a
20                  AFFICHER " x = "
21                  AFFICHER u
22          FIN_SINON
23      FIN_ALGORITHME
```

**Exercice 6.4** : Écrire un algorithme permettant la résolution, dans l'ensemble des réels, de l'équation du second degré $ax^2 + bx + c = 0$ (avec $a$ non nul).

**Exercice 6.5** : Écrire deux algorithmes (l'un avec conversion en minutes et l'autre non) permettant de faire le calcul suivant : l'utilisateur indique un horaire (deux variables : l'une indiquant les heures, l'autre les minutes), et l'algorithme indique l'horaire 15 minutes plus tard (même configuration). Ne pas oublier d'étudier le changement de jour éventuel.

**Exercice 6.6** : Écrire deux algorithmes (comme précédemment) permettant le calcul suivant : l'utilisateur indique l'horaire de départ d'un voyage (deux variables : l'une indiquant les heures, l'autre les minutes), et l'horaire d'arrivée (tout a lieu dans la même journée). L'algorithme doit indiquer en réponse la durée du voyage (heures et minutes).

**Exercice 6.7** : Écrire deux algorithmes permettant d'afficher l'état chimique de l'eau (glace, liquide, vapeur) selon sa température en Celsius, l'un avec des tests séparés et l'autre, des tests imbriqués.

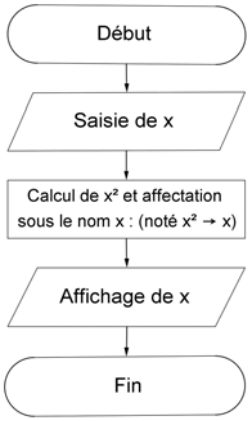
**Exercice 6.8** : Écrire un algorithme permettant de calculer le prix payé par utilisateur de photocopieuse sachant que les 10 premières photocopies coutent 10 centimes chacune, les 20 photocopies suivantes au prix de 9 centimes chacune et au-delà au prix de 8 centimes chacune.

**Exercice 6.9** : Écrire un algorithme permettant de calculer les frais de livraison d'une marchandise selon son prix : Pour la tranche comprise entre 0 et 500 Euros, les frais sont de 5 %, et pour la tranche au-delà, ils sont de 3 % de la valeur au dessus de 500 Euros.

## 7. BOUCLE « POUR »

**Remarque 7.1** : On peut faire un calcul avec la valeur d'une variable, et affecter le résultat sous le même nom de variable (utile pour les suites définies par récurrence).

**Exemple 7.2** : Calcul du carré d'un réel.

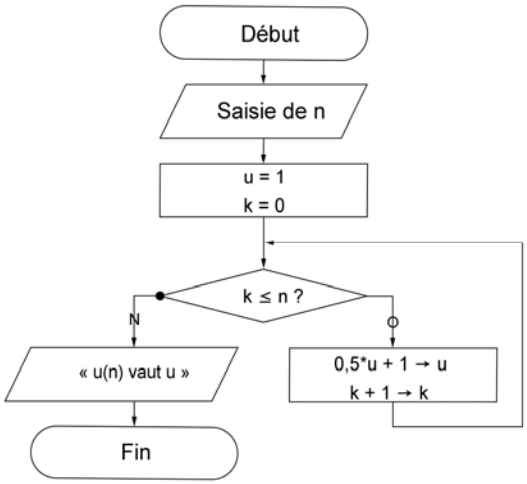



```
1  VARIABLES
2      x EST_DU_TYPE NOMBRE
3  DEBUT_ALGORITHME
4      LIRE x
5      x PREND_LA_VALEUR x*x
6      AFFICHER x
7  FIN_ALGORITHME
```

**Exercice 7.3** : Écrire un algorithme permettant d'effectuer trois fois de suite (sans utiliser de boucle) le calcul suivant : multiplier par 2 et ajouter 1, en utilisant une seule variable. Refaire l'algorithme, toujours avec une seule variable, et un seul calcul.

**Exemple 7.4** : Détermination du terme $u_n$ d'une **suite** $(u_n)$ **définie par récurrence** : $u_0 = 1$ et, pour tout $n$, $u_{n+1} = 0{,}5\ u_n + 1$.

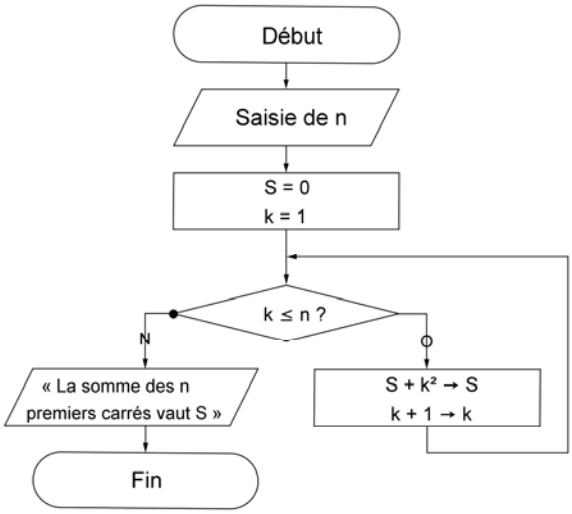
**Remarque 7.5** : Pour traduire exactement cet organigramme en langage algorithmique, il faudrait effectuer un RENVOI avant le test, ce qui est faisable (instruction « ALLER À » mais inutilement compliqué.
De plus, puisque la variable $k$ prend successivement TOUTES les valeurs entières de 1 à $n$, on peut traduire plus simplement cet organigramme en langage algorithmique grâce à une **boucle « Pour… De… À »**, qui utilise successivement les valeurs de la variable citée de la première à la dernière, avec un pas de 1, c'est-à-dire en ajoutant à chaque fois 1 à la variable, et sort de la boucle lorsque la dernière valeur est dépassée.




```
1  VARIABLES
2      u EST_DU_TYPE NOMBRE
3      k EST_DU_TYPE NOMBRE
4      n EST_DU_TYPE NOMBRE
5  DEBUT_ALGORITHME
6      LIRE n
7      u PREND_LA_VALEUR 1
8      POUR k ALLANT_DE 1 A n
9          DEBUT_POUR
10             u PREND_LA_VALEUR 0.5*u+1
11         FIN_POUR
12     AFFICHER "u("
13     AFFICHER n
14     AFFICHER ") = "
15     AFFICHER u
16 FIN_ALGORITHME
```

**Remarque 7.6** : Attention, au sortir de la boucle, $k$ vaut $n + 1$.

**Exemple 7.7** : Calcul de la somme des carrés des entiers entre 1 et *n*, entier donné par l'utilisateur

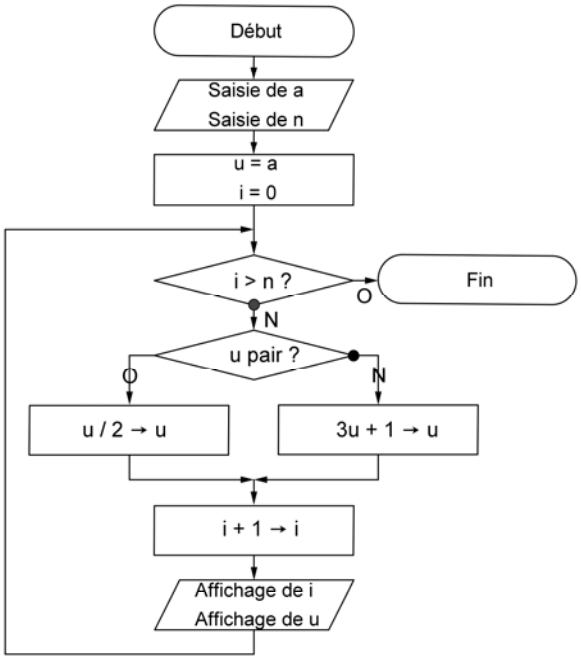



Là aussi, on utilise une boucle « POUR » :

```
1   VARIABLES
2       n EST_DU_TYPE NOMBRE
3       S EST_DU_TYPE NOMBRE
4       k EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6       LIRE n
7       S PREND_LA_VALEUR 0
8       POUR k ALLANT_DE 1 A n
9           DEBUT_POUR
10              S PREND_LA_VALEUR S+k*k
11          FIN_POUR
12      AFFICHER "La somme des "
13      AFFICHER n
14      AFFICHER " premiers carrés vaut : "
15      AFFICHER S
16  FIN_ALGORITHME
```

**Exemple 7.8** : Détermination des premières valeurs de la *suite* dite *de Syracuse* :

À partir d'un entier $u_0 = a$, on calcule $u_{n+1}$ ainsi : si $u_n$ est pair, $u_{n+1}$ vaut $u_n / 2$, sinon, $u_{n+1}$ vaut $3u_n + 1$




```
1   VARIABLES
2       u EST_DU_TYPE NOMBRE
3       i EST_DU_TYPE NOMBRE
4       a EST_DU_TYPE NOMBRE
5       n EST_DU_TYPE NOMBRE
6   DEBUT_ALGORITHME
7       LIRE a
8       LIRE n
9       u PREND_LA_VALEUR a
10      POUR i ALLANT_DE 1 A n
11          DEBUT_POUR
12              SI (u%2==0) ALORS
13                  DEBUT_SI
14                      u PREND_LA_VALEUR u/2
15                  FIN_SI
16                  SINON
17                      DEBUT_SINON
18                          u PREND_LA_VALEUR 3*u+1
19                  FIN_SINON
20              AFFICHER i
21              AFFICHER " -> "
22              AFFICHER u
23      FIN_POUR
24  FIN_ALGORITHME
```

**Remarque 7.9** : « **u%2** » renvoie en langage AlgoBox le reste de la division de *u* par 2. S'il vaut 0, c'est que *u* est un entier pair. Dans les exemples 7.5 et 7.7, les affichages finaux sont en dehors de la boucle car on affiche seulement le dernier terme. En revanche, dans l'exemple 7.8, les affichages sont dans la boucle car les valeurs sont affichées pour chaque valeur de *n*.

**Exercice 7.10** : Expliquer ce que fait l'algorithme ci-dessous, c'est-à-dire ce que représente *P* pour les nombres *a* et *b*.

```
1       VARIABLES
2           P EST_DU_TYPE NOMBRE
3           a EST_DU_TYPE NOMBRE
4           b EST_DU_TYPE NOMBRE
5           k EST_DU_TYPE NOMBRE
6       DEBUT_ALGORITHME
7           LIRE a
8           LIRE b
9           P PREND_LA_VALEUR 0
10          POUR k ALLANT_DE 1 A a
11              DEBUT_POUR
12                  P PREND_LA_VALEUR P+b
13              FIN_POUR
14          AFFICHER "P = "
15          AFFICHER P
16      FIN_ALGORITHME
```

**Exercice 7.11** : Écrire un algorithme calculant le produit de tous les entiers entre *n* et 2*n*, où *n* est un entier positif donné par l'utilisateur.

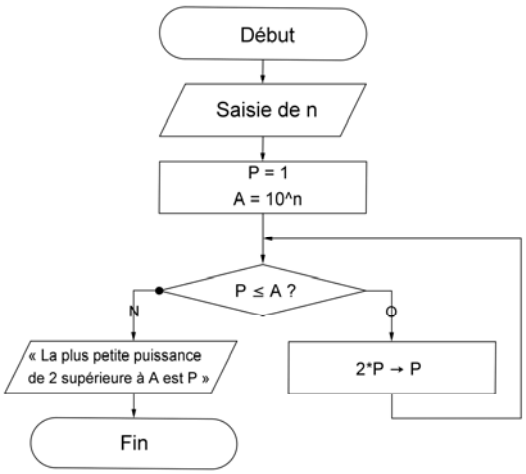
**Exercice 7.12** : Écrire un algorithme calculant le *n*-ième terme de la **suite de Fibonacci**, définie par :

$u_0 = 1$, $u_1 = 1$ et, pour tout *n* positif, $u_{n+2} = u_{n+1} + u_n$

## 8. BOUCLE « TANT QUE »

**Exemple 8.1** : Détermination de la plus petite puissance de 2 supérieure à $10^n$, l'entier *n* étant donné par l'utilisateur.

**Remarque 8.2** : Cet organigramme ressemble beaucoup à ceux des exemples 7.5 et 7.7, mais ici le nombre de fois que le test sera effectué n'est pas connu à l'avance, donc on ne peut faire une « boucle POUR ». On utilise alors une « **boucle TANT QUE** », qui effectue des instructions tant que certaines conditions sont vérifiées, et s'arrêtant lorsque ce n'est plus le cas.




```
1   VARIABLES
2       n EST_DU_TYPE NOMBRE
3       P EST_DU_TYPE NOMBRE
4       A EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6       LIRE n
7       P PREND_LA_VALEUR 1
8       A PREND_LA_VALEUR pow(10,n)
9       TANT_QUE (P<=A) FAIRE
10          DEBUT_TANT_QUE
11              P PREND_LA_VALEUR 2*P
12          FIN_TANT_QUE
13      AFFICHER "La plus petite puissance de 2 supérieure à "
14      AFFICHER A
15      AFFICHER " est : "
16      AFFICHER P
17  FIN_ALGORITHME
```

**Exemple 8.3 : Méthode de dichotomie** pour déterminer un encadrement d'amplitude $10^{-p}$ de la solution dans [1 ; 2] de l'équation $x^2 = 2$. *(algorithme ci-contre)*

**Remarque 8.4** : « pow(*a* , *b*) » renvoie en langage AlgoBox le nombre $a^b$.

**Exercice 8.5** : Écrire un algorithme donnant la liste de tous les entiers positifs dont le cube est inférieur ou égal à un réel donné, grâce à une instruction « Tant que ».

```
1   VARIABLES
2       a EST_DU_TYPE NOMBRE
3       b EST_DU_TYPE NOMBRE
4       m EST_DU_TYPE NOMBRE
5       p EST_DU_TYPE NOMBRE
6   DEBUT_ALGORITHME
7       a PREND_LA_VALEUR 1
8       b PREND_LA_VALEUR 2
9       LIRE p
10      TANT_QUE (b-a>pow(10,-p)) FAIRE
11          DEBUT_TANT_QUE
12              m PREND_LA_VALEUR (a+b)/2
13              SI ((m*m-2)*(b*b-2)>0) ALORS
14                  DEBUT_SI
15                      b PREND_LA_VALEUR m
16                  FIN_SI
17              SINON
18                  DEBUT_SINON
19                      a PREND_LA_VALEUR m
20                  FIN_SINON
21              AFFICHER a
22              AFFICHER " < solution < "
23              AFFICHER b
24          FIN_TANT_QUE
25  FIN_ALGORITHME
```

**Exercice 8.6** : L'algorithme ci-contre permet de calculer les termes de **deux suites** $(u_n)$ et $(v_n)$ **définies par des récurrences simultanées** :

$u_0 = 5$, $v_0 = 3$ et,

pour tout $n$, $u_{n+1} = 0{,}4\ u_n + 0{,}8\ v_n$ et $v_{n+1} = 0{,}6\ u_n + 0{,}2\ v_n$

1) Expliquer pourquoi on est ici obligé, dans les lignes 14 et 16, de passer par l'intermédiaire de la variable *c*.

2) Que se serait-il passé si on avait écrit cela ? :

```
14      a PREND_LA_VALEUR 0.4*a+0.8*b
15      b PREND_LA_VALEUR 0.6*a+0.2*b
```

3) Proposer une autre possibilité en complétant les pointillés des lignes suivantes :

```
14          c PREND_LA_VALEUR a
15          a PREND_LA_VALEUR ……
16          b PREND_LA_VALEUR ……
```

4) Réécrire l'algorithme en utilisant une boucle « POUR » à la place de la boucle « TANT QUE ».

```
1  VARIABLES
2      n EST_DU_TYPE NOMBRE
3      a EST_DU_TYPE NOMBRE
4      b EST_DU_TYPE NOMBRE
5      c EST_DU_TYPE NOMBRE
6      i EST_DU_TYPE NOMBRE
7  DEBUT_ALGORITHME
8      LIRE n
9      a PREND_LA_VALEUR 5
10     b PREND_LA_VALEUR 3
11     i PREND_LA_VALEUR 0
12     TANT_QUE (i<n) FAIRE
13         DEBUT_TANT_QUE
14             c PREND_LA_VALEUR 0.4*a+0.8*b
15             b PREND_LA_VALEUR 0.6*a+0.2*b
16             a PREND_LA_VALEUR c
17             i PREND_LA_VALEUR i+1
18         FIN_TANT_QUE
19     AFFICHER "a = "
20     AFFICHER a
21     AFFICHER "b = "
22     AFFICHER b
23 FIN_ALGORITHME
```

**Exercice 8.7** : Indiquer ce que fait l'algorithme ci-contre, c'est-à-dire ce que représente l'affichage finale (on étudiera les deux cas selon le signe de $a - b$).

**Exercice 8.8** : Écrire un algorithme calculant le nombre d'années qu'il faudra à une commune de 300 000 individus de n'avoir plus que 50 000 individus, sachant qu'elle perd 10 % de sa population par an.

**Exercice 8.9** : On appelle « **partie entière** » d'un réel le plus grand entier inférieur ou égal à ce réel.
Écrire un algorithme calculant la partie entière d'un réel positif.

**Remarque 8.10 :** En langage AlgoBox, la partie entière de *x* se note « floor(x) »

```
1  VARIABLES
2      a EST_DU_TYPE NOMBRE
3      b EST_DU_TYPE NOMBRE
4  DEBUT_ALGORITHME
5      LIRE a
6      LIRE b
7      TANT_QUE (a>=b) FAIRE
8          DEBUT_TANT_QUE
9              a PREND_LA_VALEUR a-b
10         FIN_TANT_QUE
11     AFFICHER a
12 FIN_ALGORITHME
```

**Exercice 8.11** : Écrire un algorithme donnant le nombre d'entiers entre deux réels donnés.

**Exemple 8.12** : **PGCD de deux entiers positifs par la méthode dite de « l'Algorithme d'Euclide »**
(Attention ! pour ne pas alourdir la procédure, il n'y a pas de vérification de positivité des deux entiers. Avec des entiers négatifs, le résultat peut être faux.)

**Remarque 8.13** : « B!=0 » signifie « *B* différent de 0 » et « A%B » est le reste de la division euclidienne de *A* par *B*.

```
1      VARIABLES
2          A EST_DU_TYPE NOMBRE
3          B EST_DU_TYPE NOMBRE
4          R EST_DU_TYPE NOMBRE
5      DEBUT_ALGORITHME
6          LIRE A
7          LIRE B
8          AFFICHER "Le PGCD de "
9          AFFICHER A
10         AFFICHER " et "
11         AFFICHER B
12         AFFICHER " est "
13         TANT_QUE (B!=0) FAIRE
14             DEBUT_TANT_QUE
15                 R PREND_LA_VALEUR A%B
16                 A PREND_LA_VALEUR B
17                 B PREND_LA_VALEUR R
18             FIN_TANT_QUE
19         AFFICHER A
20     FIN_ALGORITHME
```