Le corps de programme

Le corps d'un algorithme est constitué d'un ensemble d'**actions / instructions** ordonnées de manière **séquentielle et logique**. Ces instructions se divisent en **cinq types distincts** :

1 *Lecture*

Cette opération consiste à introduire des données dans l'algorithme. Une lecture consiste à donner une valeur arbitraire à une variable.

2 *Écriture*

Cette opération implique l'affichage de données. Elle permet d'afficher des résultats ou des messages.

3 *Affectation*

Elle permet de modifier les valeurs des variables en leur assignant de nouvelles valeurs.

Structures de contrôle

Ces structures permettent de modifier la séquentialité de l'algorithme. Elles sont utilisées pour sélectionner différents chemins d'exécution ou pour répéter un traitement.

4 *Structure de* **Test alternatif simple / double / imbriqué**

5 *Structures* **répétitives (itérative)**

- *La boucle* **Pour**
- *La boucle* **Tantque**
- *La boucle* **Répéter**

Structures de contrôle

Par défaut, un programme s'exécute de façon **séquentielle**, instruction après instruction.

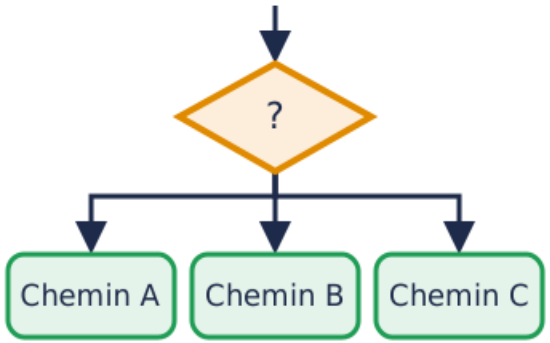
Séquentiel



Pour **choisir** un chemin ou **répéter** un traitement, on utilise deux types de structures :

Structures condit

Les tests




```
Algorithme Cercle ;
Constante PI = 3.14 ;
Variable R, P, S : Réel ;
Début
  Lire(R) ;
  P ← 2 * PI * R ;
  S ← PI * R * R ;
  Écrire(P) ;
  Écrire(S) ;
Fin.
```

›étitives

ucles

Répéter l'exécution d'un **ensemble d'instructions**.

Structures conditionnelles

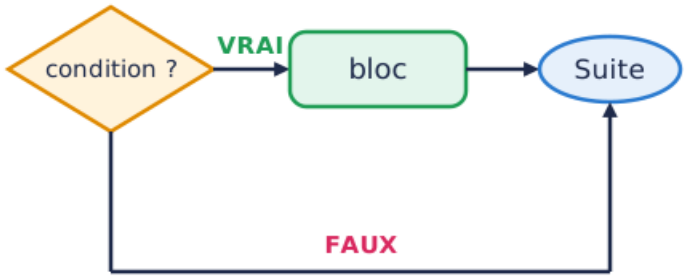
Décider si un bloc d'instructions est exécuté, ou choisir entre deux ou plusieurs blocs d'instructions.

Trois types de structures conditionnelles

a Test alternatif simple

Un seul bloc d'instructions

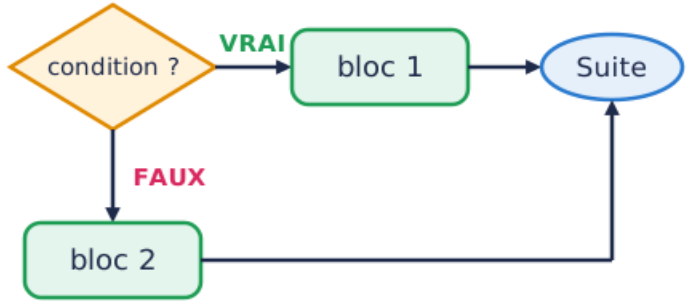
Si la condition est vraie, on exécute le bloc ; sinon, on ne l'exécute pas.



b Test alternatif double

Deux blocs d'instructions

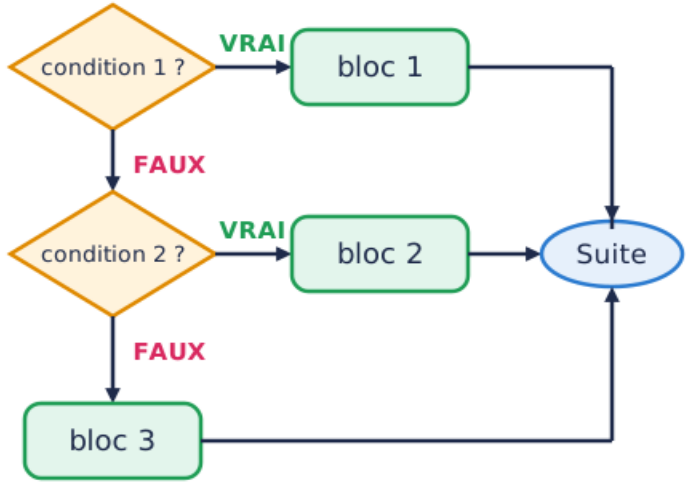
Selon la condition, on choisit entre l'exécution du bloc 1 ou du bloc 2.



c Test alternatif imbriqué

Un test placé dans un autre test

Si la condition 1 est fausse, on teste une condition 2 (choix entre 3 blocs).



1 Le test alternatif simple

Syntaxe

Le test alternatif simple

Exécuter un bloc d'instructions **uniquement si** la **condition** est vraie.

Algorithme

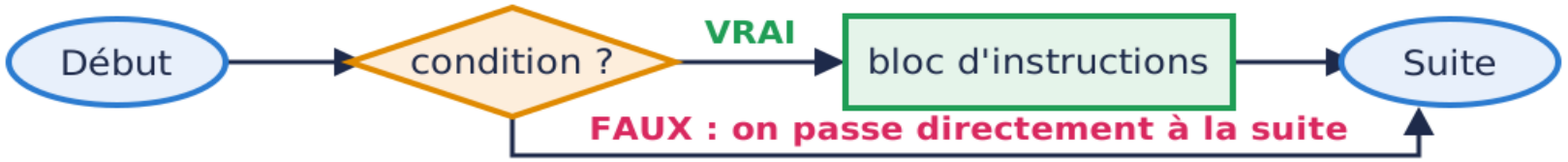
```
si <condition> alors
   <bloc_instructions_si> ;
finsi ;
```

⇄

Langage C

```
if (condition)
{
   <bloc_instructions_si>;
}
```

En résumé

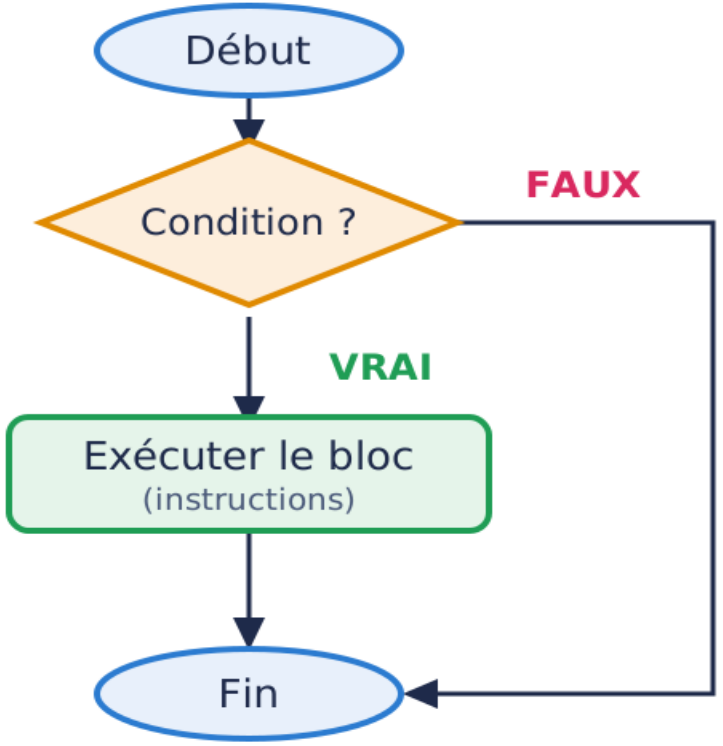


Organigramme

Le test alternatif simple

Le bloc n'est exécuté que si la condition est vraie.

Organigramme



Comment lire ce schéma ?

1. On évalue la condition.
2. **VRAI** : le bloc d'instructions est exécuté, puis on continue.
3. **FAUX** : le bloc est ignoré, on passe directement à la suite.

Écriture en algorithme

```
si <condition> alors
   <bloc_instructions_si> ;
finsi ;
```

Exemple

Un test alternatif simple

Lire x ; si x > 2, lui ajouter 3 ; puis afficher x.

Algorithme

```
lire(x)
si x > 2 alors
   x ← x + 3 ;
finsi
écrire(x)
```

⇄

Langage C

```
scanf("%d", &x);
if (x > 2)
{
   x = x + 3;
}
printf("%d", x);
```

Remarque

Un bloc est délimité par les accolades { et }.
S'il ne contient qu'une instruction, elles sont facultatives.

Essai

x=5 → 8

x=1 → 1

2

Le test alternatif double

Syntaxe

Le test alternatif double

Deux blocs d'instructions : on exécute **l'un ou l'autre**, selon la condition.

Algorithme

```
si <condition> alors
    <bloc_instructions_si> ;
sinon
    <bloc_instructions_sinon> ;
finsi ;
```

⇄

Langage C

```
if (condition)
{
    <bloc_instructions_si>;
}
else
{
    <bloc_instructions_sinon>;
}
```

En résumé la condition est une expression logique (booléenne), vraie ou fausse

VRAI → on exécute le premier bloc

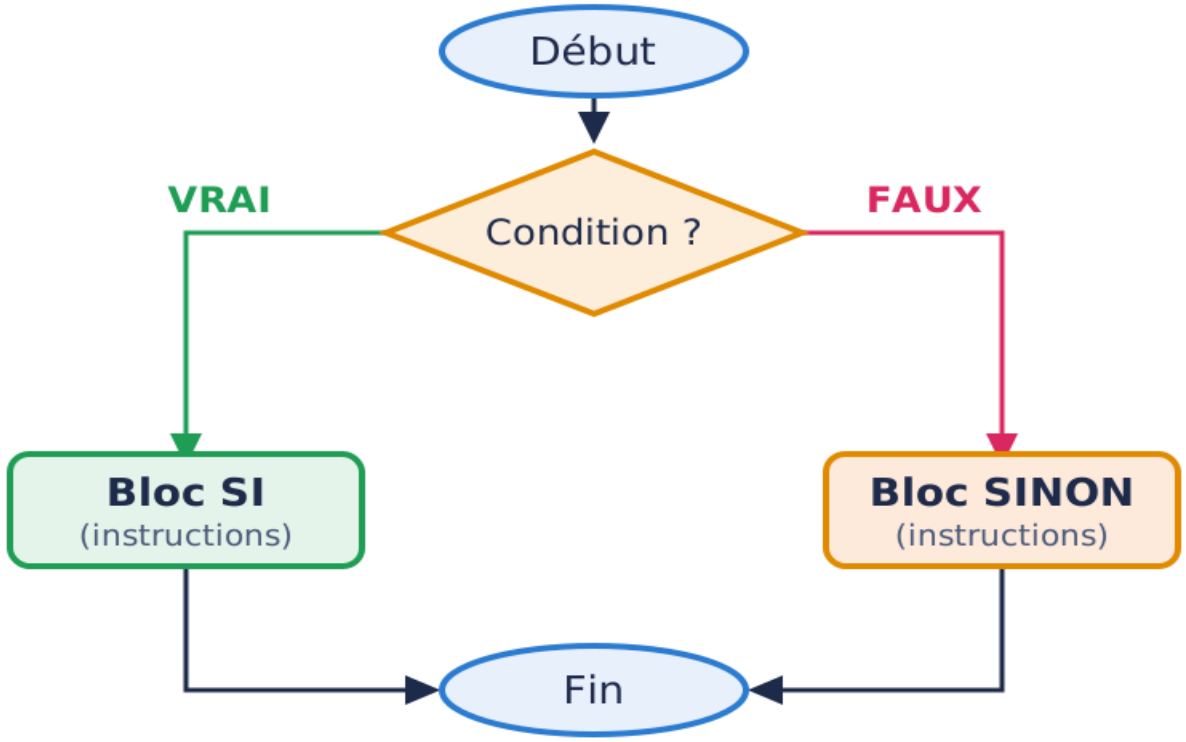
FAUX → on exécute le second bloc

Organigramme

Le test alternatif double

Un seul des deux blocs est exécuté, selon la condition.

Organigramme



Comment lire ce schéma ?

1. On évalue la condition.
2. **VRAI** : le bloc SI est exécuté.
3. **FAUX** : le bloc SINON est exécuté.
4. Dans les deux cas, le programme continue après le test.

Écriture en algorithme

```
si <condition> alors
   <bloc_instructions_si> ;
sinon
   <bloc_instructions_sinon> ;
finsi ;
```

Exemple

Un test alternatif double

Si x > 2, on ajoute 3 à x ; sinon, on lui retire 2.

Algorithme

```
lire(x)
si (x > 2) alors
    x ← x + 3
sinon
    x ← x - 2
finsi
écrire(x)
```

⇄

Langage C

```
scanf("%d", &x);
if (x > 2)
{
    x = x + 3;
}
else
{
    x = x - 2;
}
printf("%d", x);
```

Essai

x=5 **vrai** → x + 3 = 8

x=1 **faux** → x - 2 = -1

3 Le test alternatif imbriqué

Syntaxe

Le test alternatif imbriqué

Plusieurs conditions enchaînées : on imbrique un **si...sinon** dans la branche **sinon** d'un autre.

Algorithme

```
si <condition1> alors
    <bloc_instructions_1> ;
sinon
    si <condition2> alors
        <bloc_instructions_2> ;
    sinon
        <bloc_instructions_3> ;
    finsi ;
finsi ;
```

⇄

Langage C

```
if (condition1)
{
    <bloc_instructions_1> ;
}
else if (condition2)
{
    <bloc_instructions_2> ;
}
else
{
    <bloc_instructions_3> ;
}
```

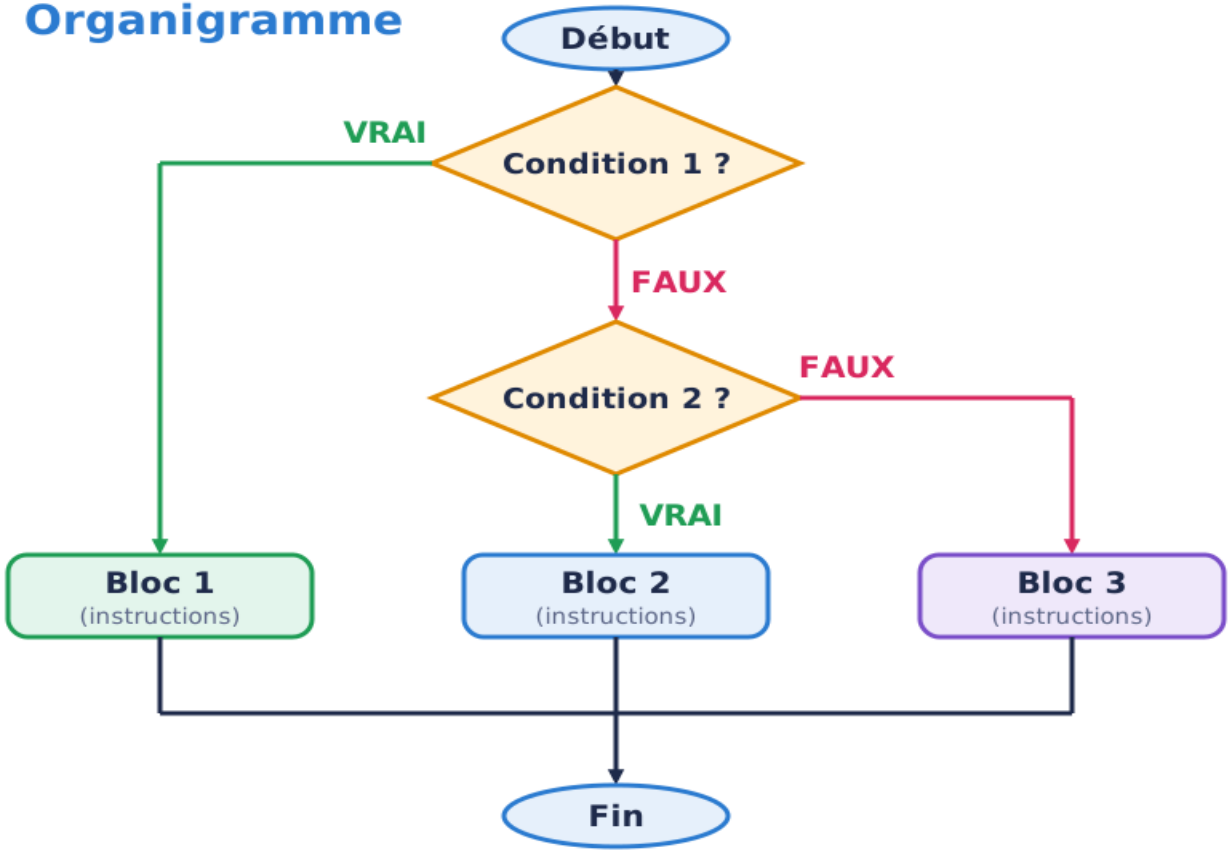
Remarque : on peut imbriquer autant de niveaux que nécessaire — chaque **sinon** peut contenir un nouveau **si...sinon**.

Organigramme

Le test alternatif imbriqué

Plusieurs conditions enchaînées : le second test se trouve dans la branche FAUX du premier.

Organigramme



Comment lire ce schéma ?

1. On évalue la condition 1.
2. **VRAI** : le bloc 1 est exécuté.
3. **FAUX** : on évalue la condition 2, VRAI → bloc 2, FAUX → bloc 3.
4. Un seul bloc est exécuté, puis le programme continue après le test.

Écriture en algorithme

```
si <condition1> alors
    <bloc_instructions_1> ;
sinon
    si <condition2> alors
        <bloc_instructions_2> ;
    sinon
        <bloc_instructions_3> ;
    finsi ;
finsi ;
```

Exemple

Classer un entier : négatif, nul ou positif

Algorithme

```
Algorithme Signe ;
Variables N : entier ;
Début
    //Les entrées
    Écrire("Entrer un entier :") ;
    Lire(N) ;
    //Les sorties
    si (N < 0) alors
        Écrire(N, " est négatif") ;
    sinon
        si (N = 0) alors
            Écrire(N, " est nul") ;
        sinon
            Écrire(N, " est positif") ;
        finsi ;
    finsi ;
Fin.
```

⇄

Langage C

```
#include <stdio.h>
int main()
{
    int N ;
    //Les entrées
    printf("Entrer un entier :") ;
    scanf("%d", &N) ;
    //Les sorties
    if (N < 0)
        printf("%d est negatif", N) ;
    else if (N == 0)
        printf("%d est nul", N) ;
    else
        printf("%d est positif", N) ;
    return 0 ;
}
```

Traces d'exécution : N = -5 → "-5 est négatif" | N = 0 → "0 est nul" | N = 7 → "7 est positif"

À retenir

Remarques

Deux points d'attention sur le test **if ... else** en langage C.

1 Pas de ; après la condition

Erreur logique : le point-virgule termine le if, donc le bloc est toujours exécuté.

✗
```
if (x > 2);
{ x = x + 3; }
```

✓
```
if (x > 2)
{ x = x + 3; }
```

2 Accolades facultatives

Une seule instruction par bloc : on peut enlever les { } du if et du else.

```
if (x > 2)
    x = x + 3;
else
    x = x - 2;
```

01

Exercice N°01

Écrire un programme C qui saisit un nombre entier et détecte si ce nombre est pair ou impair.

pair

impair

Solution

Corrigé de l'exercice N°01

R = N mod 2 ; R = 0 ⇒ pair

Saisir un entier et détecter s'il est pair ou impair.

Algorithme

```
Algorithme Partie ;
Variables
    N, R : entier;
Début
    //Les entrées
    Écrire ("Donner un entier : ") ;
    Lire (N) ;
    //Traitement
    R ← N mod 2;
    //Les sorties
    Si (R = 0) alors
        Écrire (N, " est pair")
    Sinon
        Écrire (N, " est impair") ;
    Fin-Si ;
Fin.
```

⇄

Programme C

```c
#include <stdio.h>
int main()
{
   int N, R;
   //Les entrées
   printf("Donner un entier : ");
   scanf("%d", &N);
   //Traitement
   R= N % 2 ;
   //Les sorties
   if(R==0)
     printf("%d est pair", N) ;
   else
     printf("%d est impair", N) ;
   return 0;
}
```

Exercice N°02

Donner l'organigramme (algorigramme) de l'algorithme ci-dessous :

Algorithme

```
Algorithme Vitesse ;
Variables
    V, D : réel ;
    T : entier ;
Début
    Lire (D, T) ;
    V ← D/T ;
    Si (V <= 80) alors
        Écrire ('Vitesse normale')
    Sinon
        Écrire ('Vitesse élevée') ;
    FinSi ;
Fin.
```

Étapes de l'organigramme

1. Début → Lire (D, T)
2. Traitement : V ← D/T
3. Test : V <= 80 ?
4. Oui / Non → Écrire → Fin

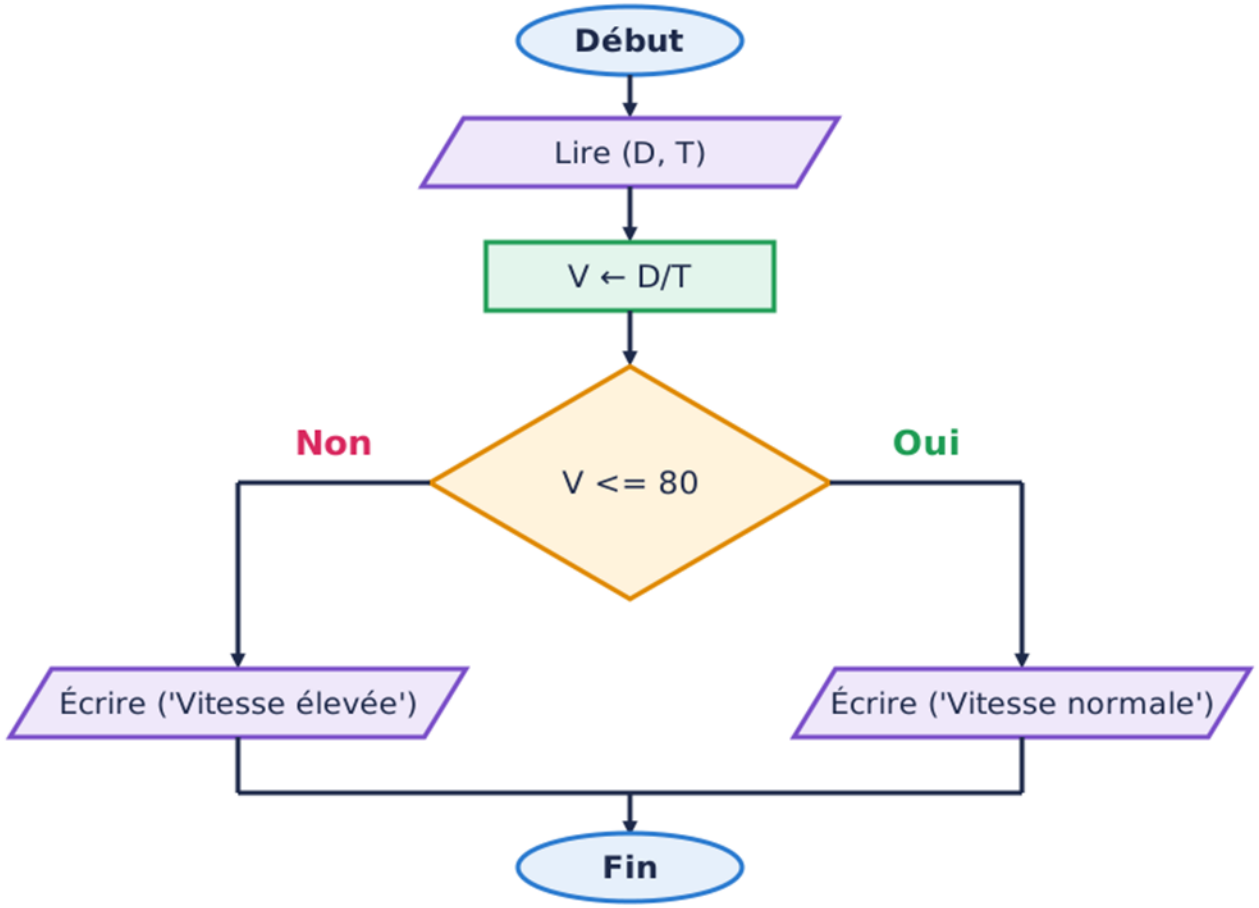
Corrigé de l'exercice N°02

Donner l'organigramme (algorigramme) de l'algorithme ci-dessous :

Algorithme

```
Algorithme Vitesse ;
Variables
    V, D : réel ;
    T : entier ;
Début
    Lire (D, T) ;
    V ← D/T ;
    Si (V <= 80) alors
        Écrire ('Vitesse normale')
    Sinon
        Écrire ('Vitesse élevée') ;
    FinSi ;
Fin.
```

Organigramme (corrigé)



Exercices N°03 et N°04

Exercice N°03

Écrire un algorithme/programme en langage C qui permet de lire deux nombres entiers (a, b), calcule et affiche la valeur de c comme suit :

$$c = \begin{cases} 0 & \text{si } a = b \\ a - b & \text{si } a > b \\ b - a & \text{si } a < b \end{cases}$$

À faire : algorithme + programme C

Exercice N°04

Soit un service d'impression qui établit le prix d'impression d'une page selon le nombre de pages (nb_pages) :

a	nb_pages ≤ 10	5 D.A.
b	entre 11 et 20	4.5 D.A.
c	entre 21 et 60	3 D.A.
d	nb_pages > 60	2.5 D.A.

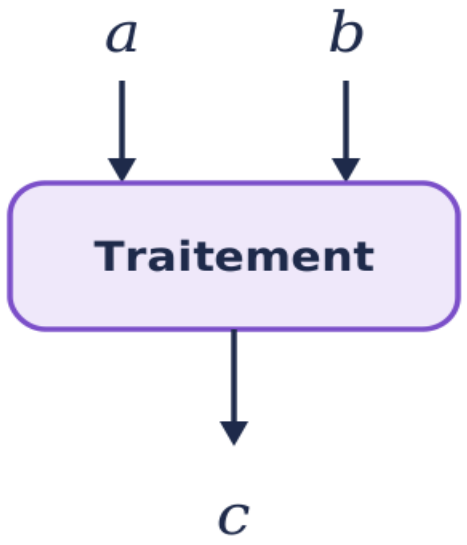
Écrire un algorithme, puis le programme en langage C, qui permet de calculer le prix d'impression pour un nombre de pages quelconque.

À faire : algorithme + programme C

Corrigé de l'exercice N°03

Schéma : entrées, traitement et sorties

Schéma général



Entrées :

```
Lire (a,b) ;
```

Traitements :

```
Si a=b alors
    c ← 0;
    Sinon-Si a>b alors
        c ← a-b;
    Sinon
        c ← b-a;
    Fin-Si
Fin-Si
```

Sortie :

```
Écrire (c) ;
```

Corrigé de l'exercice N°03

Algorithme

```
Algorithme exo2;
Variables
    a, b, c : réel;
Début
    //-*-*- Entrées -*-*-
    Lire (a,b);
    //-*-*- Traitements -*-*-
    Si a=b alors
        c ← 0;
    Sinon-Si a>b alors
        c ← a-b;
    Sinon
        c ← b-a;
    Fin-Si
    Fin-Si
    //-*-*- Sorties -*-*-
    Écrire ('c = ', c:0:3);
Fin.
```

⇄

Programme C

```c
#include <stdio.h>
int main()
{
    float a, b, c;
    //-*-*- Entrées -*-*-
    scanf("%f %f", &a, &b);
    //-*-*- Traitements -*-*-
    if (a==b)
        c = 0;
    else if (a>b)
        c = a-b;
    else
        c = b-a;
    //-*-*- Sorties -*-*-
    printf("c = %.3f", c);
    return 0;
}
```

Corrigé de l'exercice N°04

Algorithme

```
Algorithme Impression;
Variables
    nb_pages : entier;
    prix_impression : réel;
Début
    //-*-*- Entrées -*-*-
    Écrire ("Donner le nombre de pages : ");
    Lire (nb_pages);
    //-*-*- Traitements -*-*-
    Si nb_pages <= 10 alors
        prix_impression ← nb_pages * 5;
    Sinon-Si (nb_pages >= 11) et (nb_pages <= 20) alors
        prix_impression ← nb_pages * 4.5;
    Sinon-Si (nb_pages >= 21) et (nb_pages <= 60) alors
        prix_impression ← nb_pages * 3;
    Sinon-Si nb_pages > 60 alors
        prix_impression ← nb_pages * 2.5;
    Fin-Si
    Fin-Si
    Fin-Si
    Fin-Si
    //-*-*- Sorties -*-*-
    Écrire ("Prix d'impression = ", prix_impression, " D.A.");
Fin.
```

Programme C

```c
#include <stdio.h>
int main()
{
    int nb_pages;
    float prix_impression;
    //-*-*- Entrées -*-*-
    printf("Donner le nombre de pages : ");
    scanf("%d", &nb_pages);
    //-*-*- Traitements -*-*-
    if (nb_pages <= 10)
        prix_impression = nb_pages * 5;
    else if (nb_pages >= 11 && nb_pages <= 20)
        prix_impression = nb_pages * 4.5;
    else if (nb_pages >= 21 && nb_pages <= 60)
        prix_impression = nb_pages * 3;
    else if (nb_pages > 60)
        prix_impression = nb_pages * 2.5;
    //-*-*- Sorties -*-*-
    printf("Prix d'impression = %.1f D.A.", prix_impression);
    return 0;
}
```


Merci