UNIVERSITÉ ABOU BAKR BEL-KAID–TLEMCEN
FACULÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE
Première année licence



Algorithmique et structure de données 1

Chapitre 4: Les structures répétitives

Mme HABRI née BENMAHDI Meryem Bochra

Année universitaire: 2024-2025

Plan du cours

1. **Importance de la répétition dans la résolution de problèmes (exemples)**
2. **Notion de compteur de boucle et de condition de répétition (ou d'arrêt)**
3. **La boucle (TantQue …. FinTQ)**
4. **La boucle (Répéter …. Jusqu'à ….)**
5. **La boucle (Pour …. FinPour)**
6. **Imbrication de boucles**
7. **Récapitulatif**

L'importance de la répétition dans la résolution de problèmes

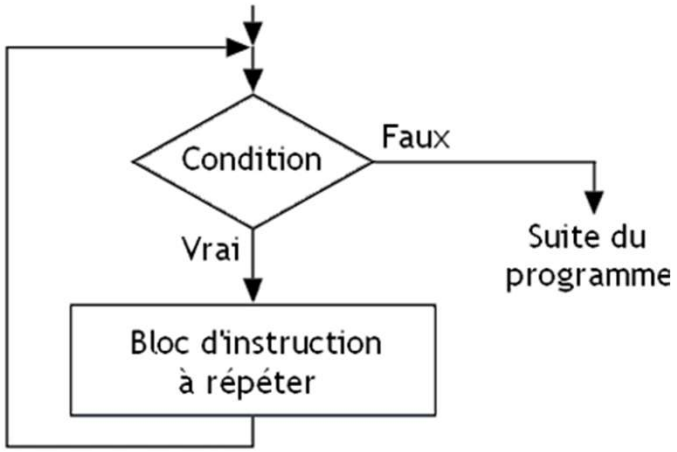
- **En programmation, la répétition joue un rôle fondamental dans la résolution de problèmes.**
- **Elle permet d'exécuter plusieurs fois une même série d'instructions.**
- **Cette structure de contrôle rend le programme plus efficace, plus rapide et plus facile à écrire.**
- **Grâce a la répétition, le programmeur peut automatiser des tâches répétitives, parcourir des ensembles de données, effectuer des calculs complexes ou rechercher une solution progressivement.**
- **En résumé, la répétition est un outil essentiel pour concevoir des programmes puissants, clairs et adaptables à des situations variées.**

La répétition en programmation

- En programmation, la structure de contrôle de **répétition**, aussi appelée **itération**, désigne l'action d'exécuter **plusieurs fois** une ou plusieurs instructions tant qu'une **condition** est vraie ou pour un **nombre déterminé de fois**.
- Cette structure de contrôle de répétition est appelée structure répétitive (ou itérative) ou boucle.
- Il y'a principalement trois boucles en programmation qui sont :
    - Tant que….FinTQ → while
    - Répéter… jusqu'à… → do… while
    - Pour → for

Boucle Tant que… FinTQ → while

- **La boucle tant que est utilisée** lorsque la condition doit être vérifiée avant chaque exécution **.**
- **Syntaxe:**



En pseudocode:

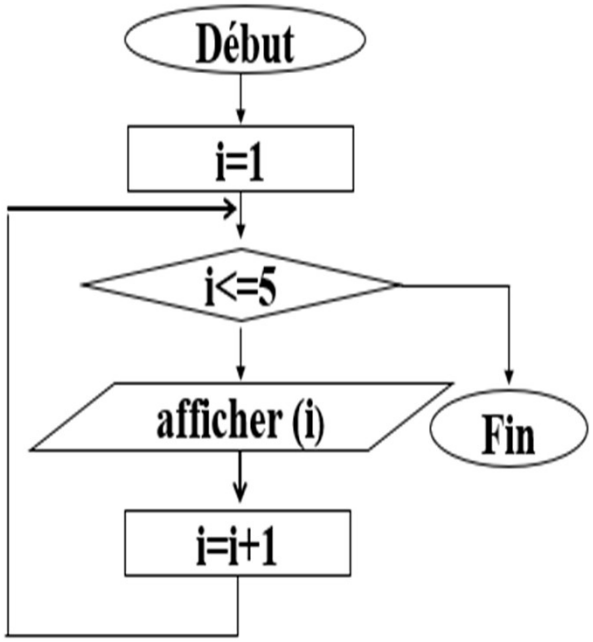
Tant que (condition) faire
//suite d'instruction
FinTQ

En langage c:

while (condition)
{
//suite d'instruction
}

- la condition (dite condition de contrôle de la boucle) est évaluée avant chaque itération
- Si la condition est vraie, on exécute le bloc d'instructions (corps de la boucle), puis, on retourne tester la condition. Si elle est encore vraie, on répète l'exécution,...
- Si la condition est fausse, on sort de la boucle et on exécute l'instruction qui se trouve juste après la fin de la boucle.

Exemple 1

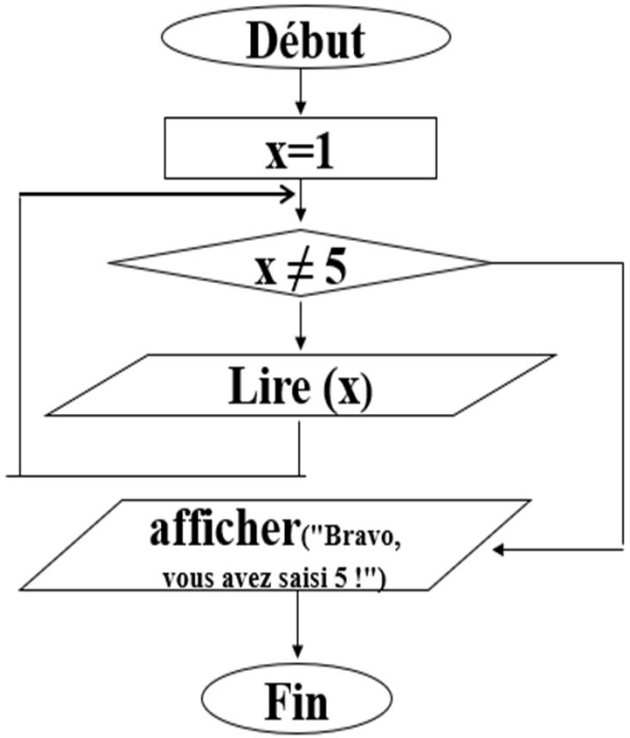



```
Algorithme ……
Var i:enttier
Début
i ← 1
Tant que (i ≤ 5) faire
   Afficher(i)
   i ← i + 1
FinTQ
fin
```

```
#include <stdio.h>
int main() {
    int i = 1;
    while (i <= 5) {
        printf("%d\n", i);
        i = i + 1;
    }
    return 0;
}
```

Cet exemple permet d'afficher les nombres de 1 à 5

Exemple 2




```
Algorithme ……..
Var x:entier
Début
x ← 1
 Tant que (x≠5) faire
   Lire(x)
 FinTQ
Afficher ("Bravo,
vous avez saisi 5 !")
Fin
```

```
#include <stdio.h>
int main() {
    int x = 1;
    while (x != 5) {
        printf("Entrez un
nombre : ");
        scanf("%d", &x);
    }
    printf("Bravo, vous
avez saisi 5 !\n");
    return 0;
}
```

Ici, le programme **demande à l'utilisateur de saisir un nombre** et **ne s'arrête que lorsqu'il entre 5**.

Attention aux boucles infinies

- **Le nombre d'itérations dans une boucle while n'est pas forcément connu à l'avance. Il dépend de l'évaluation de la condition.**
- **Lorsque vous créez une boucle, assurez-vous toujours qu'elle peut s'arrêter à un moment! Si la condition est toujours vraie, votre programme ne s'arrêtera jamais!**

Exemple :

```
 int compteur = 0;
while (compteur >= 0) {
printf("La variable compteur vaut %d \n", compteur);
compteur++; }
```

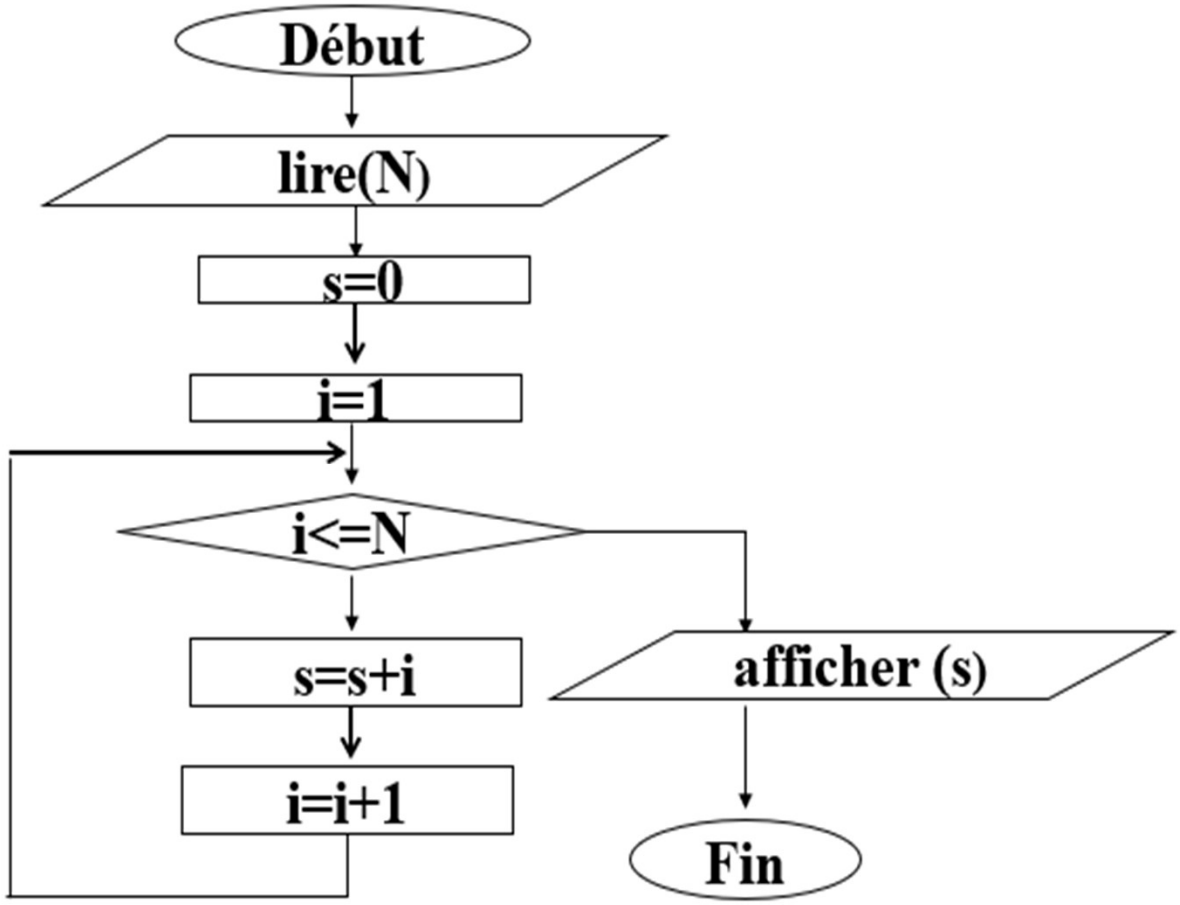
- **Une des instructions du corps de la boucle doit absolument changer la valeur de condition de vrai à faux (après un certain nombre d'itérations), sinon le programme tourne indéfiniment.**

Exercice d'application 1

Trouve l'organigramme et écrire un algorithme et un programme qui demande à l'utilisateur de saisir un entier N, puis calcule la **somme des nombres de 1 à N** en utilisant une boucle Tant que.

Exemple : Si N = 5, le programme affiche :La somme est 15

L'organigramme



L'algorithme

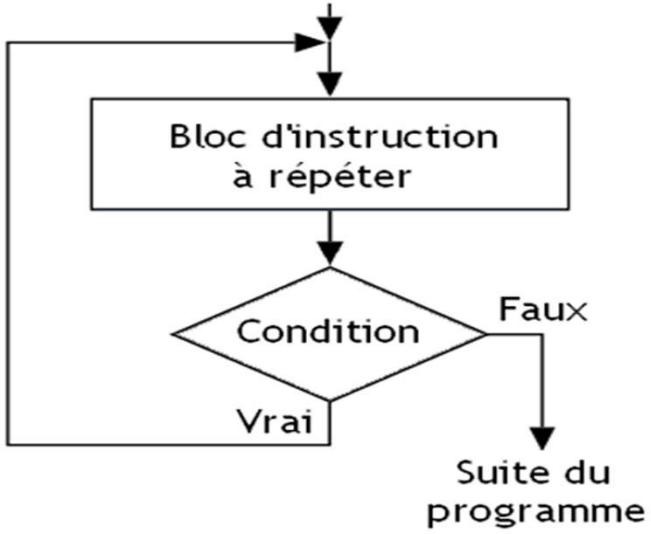
```
Algorithme Somme_TantQue
    Var N, i, somme : Entier
    Début
        Écrire ("Entrez un entier N : ")
        Lire (N)
        i ← 1
        somme ← 0
        Tant que (i ≤ N) faire
            somme ← somme + i
            i ← i + 1
        FinTQ
        Écrire ("La somme est ", somme)
    Fin
```

Le programme C

```
#include <stdio.h>
int main() {
    int N, i = 1, somme = 0;
    printf("Entrez un entier N : ");
    scanf("%d", &N);
    while (i <= N) {
        somme = somme + i;
        i = i + 1;
    }
    printf("La somme est %d\n", somme);
    return 0;
}
```

Boucle Répéter → do while

- Cette boucle est utilisée lorsque un bloc d'instructions est exécuté avant de vérifier la condition.
- Syntaxe :



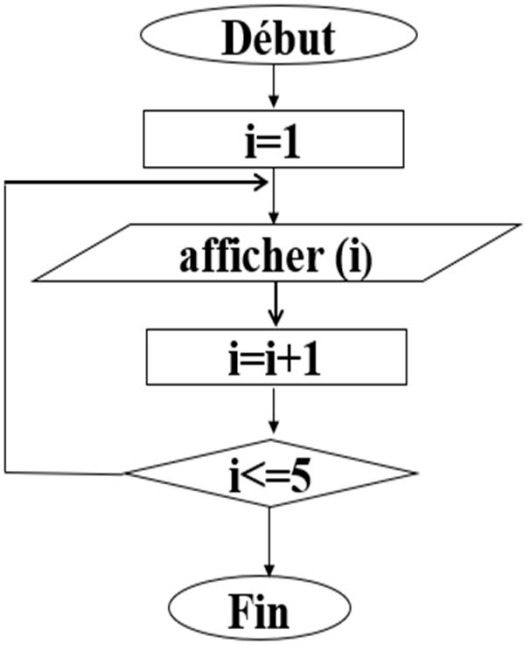
En pseudocode:
Répéter
//suite d'instruction
Jusqu'a
(non(condition))

En programme c:
do
//suite d'instruction
while (condition);

Ne pas oublier le points virgule

- **La boucle do...while est très similaire à while La seule chose qui change par rapport à while, c'est la position de la condition. Au lieu d'être au début de la boucle, la condition est à la fin. Cette boucle s'exécutera donc toujours au moins une fois.**

Exemple 1

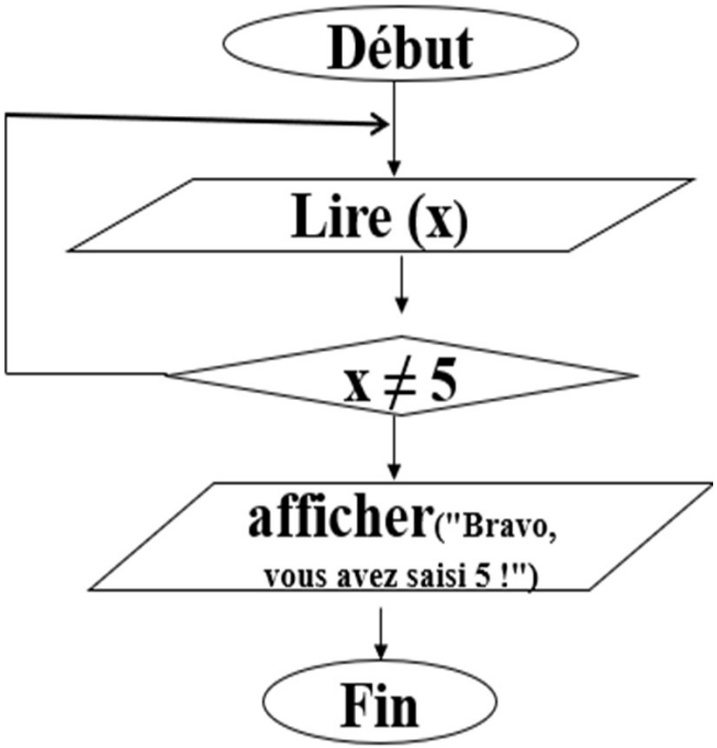



```
Algorithme ……
Var i : entier
Début
   i ← 1
   Répéter
      Afficher(i)
      i ← i + 1
   Jusqu'à (i > 5)
Fin
```

```
#include <stdio.h>
int main() {
    int i = 1;
    do {
        printf("%d\n", i);
        i = i + 1;   //
     } while (i <= 5);
    return 0;
}
```

Cet exemple permet d'afficher les nombres de 1 à 5

Exemple 2




```
Algorithme ……..
Var x : entier
Début
   Répéter
      Lire(x)
   Jusqu'à (x=5)
   Afficher("Bravo, vous avez saisi 5 !")
Fin
```

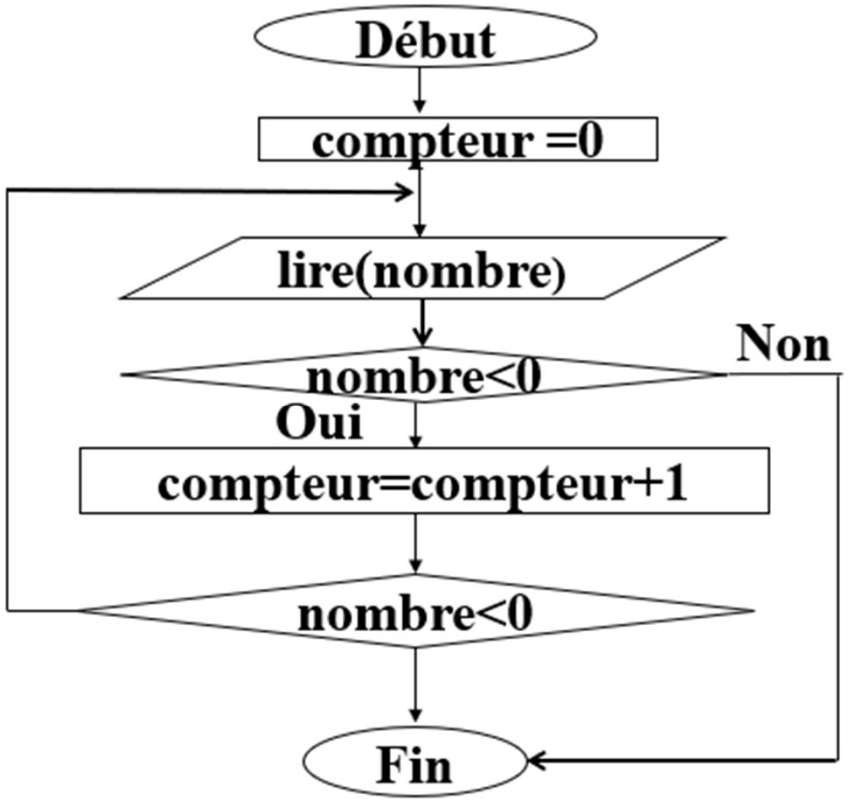
```
#include <stdio.h>
int main() {
    int x;
    do {
     printf("Entrez un nombre : ");
        scanf("%d", &x);
    } while (x != 5);
printf("Bravo, vous avez saisi 5!\n");
 return 0;
}
```

Ici, le programme **demande à l'utilisateur de saisir un nombre** et **ne s'arrête que lorsqu'il entre 5**.

Exercice d'application 2

Trouve l'organigramme et écrire un algorithme et un programme qui demande à l'utilisateur de contrôler la saisie un réel positif ou nul, puis calcule le **nombre de fois que l'utilisateur a entré un nombre négatif**.

L'organigramme



L'algorithme

Algorithme pour saisir un nombre positif ou nul
Début
    compteur ← 0
    Répéter
        Ecrire ("Entrez un nombre réel (négatif pour tester, positif pour arrêter) :")
        Lire (nombre)
        Si (nombre < 0) Alors
            compteur ← compteur + 1
        FinSi
    Jusqu'à (nombre > =0) // on arrête quand un réel positif est saisi

    Ecrire ("Vous avez saisi ", compteur, " nombres négatifs.")
Fin

Le programme

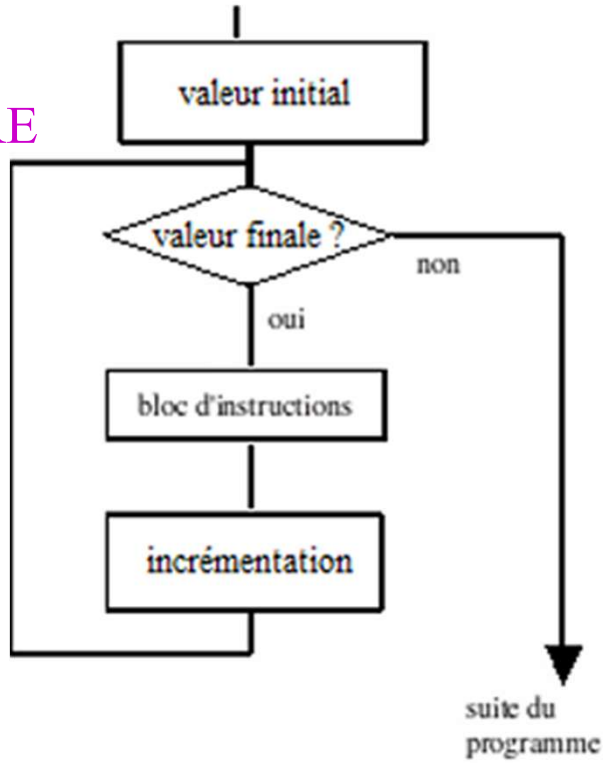
```
#include <stdio.h>
int main() {
    float nombre;
    int compteur = 0;
    do {
        printf("Entrez un nombre réel (positif pour arrêter) : ");
        scanf("%f", &nombre);
        if (nombre < 0) {
            compteur++;
        }
    } while (nombre <0);
    printf("Vous avez saisi %d nombres négatifs.\n", compteur);
    return 0;
}
```

Boucle Pour → for

- Cette boucle est utilisée lorsque le nombre d'itérations est connu à l'avance.
- Syntaxe:

En pseudocode:

Pour variable ← valeur_initiale jusqu'a valeur_finale [PAS pas] FAIRE
    // Instructions à répéter
FINPOUR



- POUR → début de la boucle
- variable → compteur (souvent appelée i, j, ou k)
- valeur_initiale → point de départ
- valeur_finale → point d'arrêt
- PAS (optionnel) → incrément ou décrément
- FINPOUR → fin de la boucle

Boucle Pour → for

En langage C:

```
for (initialisation ; condition ; pas )
 {
//instructions á répéter
}
```

Il y a trois instructions condensées, chacune séparée par un point-virgule:

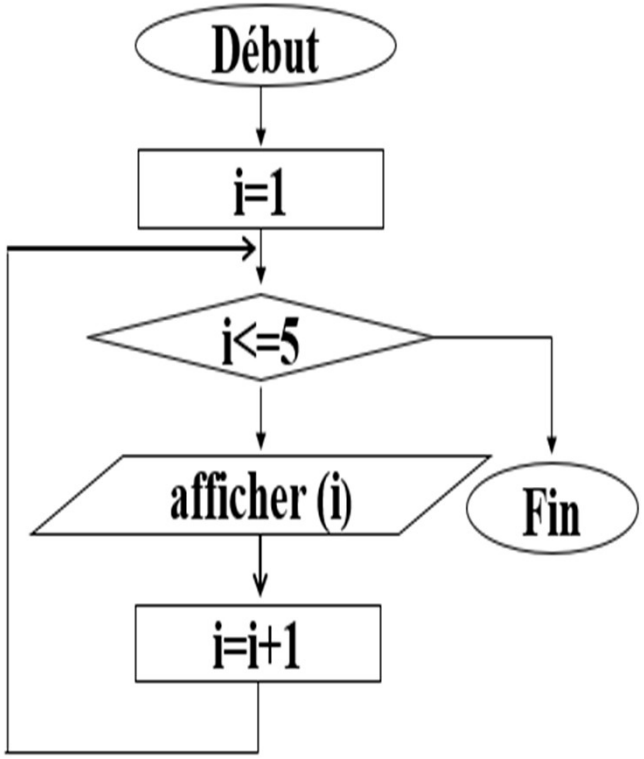
- La première est l'initialisation, cette première instruction est utilisée pour préparer notre variable compteur.
- La seconde est la condition : comme pour la boucle while, c'est la condition qui dit si la boucle doit être répétée ou non. Tant que la condition est vraie, la boucle for continue.
- Enfin, il y a l'incrémentation ou décrémentation : cette dernière instruction est exécutée à la fin de chaque tour de boucle pour mettre à jour la variable compteur. par exemple).

Principe de la boucle

1. La valeur initiale est affectée à la variable i;
2. On compare la valeur du i et la valeur finale :
3. Si la condition est fausse, on sort de la boucle et on continue avec l'instruction qui suit la fin de la boucle (l'accolade fermante en C et Finpour en pseudocode).
4. Si la condition est vraie alors :
   a. Les instructions de la boucle seront exécutées.
   b. Ensuite, la valeur de i est incrémentée de la valeur du pas (sinon 1 par défaut).
   c. On recommence l'étape 2 : La comparaison entre i et la valeur finale est de nouveau effectuée, et ainsi de suite.

```
Algorithme ……..
Var i : entier
Début
    Pour i ← 1 à 5 faire
        Afficher(i)
    FinPour
Fin
```

Ici la valeur finale est 5



Exemple 1

```
Algorithme ……..
Var i : entier
Début
   Pour i ← 1 à 5 faire
      Afficher(i)
   FinPour
Fin
```

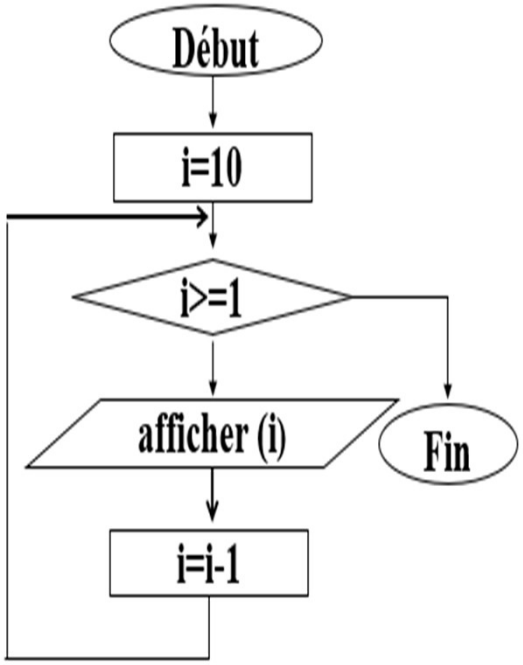
Le pas prend la valeur 1 par défaut quand il n'est pas mentionné

```
Algorithme ……..
Var i : entier
Début
   Pour i ← 1 à 5 [1] faire
      Afficher(i)
   FinPour
Fin
```

```
#include <stdio.h>
int main() {
   int i;
   for (i = 1; i <= 5; i++) {
      printf("%d\n", i);
   }
   return 0;
}
```

Exemple 2

Trouver l'organigramme et écrire un algorithme et le traduire en programme qui affiche les nombres de 10 á 1



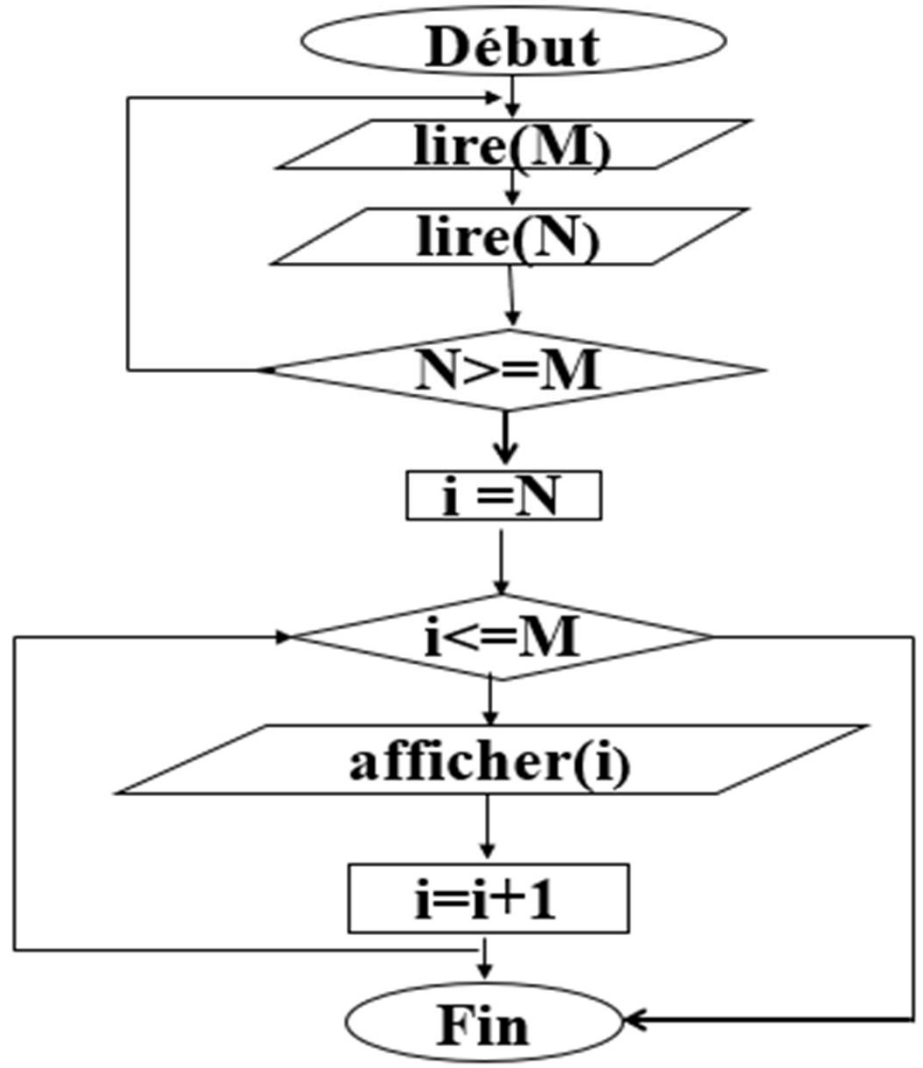

```
Algorithme de décrémentation
Var i : entier
Début
   Pour i ← 10 à 1 pas -1 faire
      Afficher(i)
   FinPour
Fin
```

```
#include <stdio.h>
int main() {
    int i;
    for (i = 10; i >= 1; i--)
    {
     printf("%d\n", i);
    }
    return 0;
}
```

Exercice d'application

Trouve l'organigramme et écrire un algorithme et un programme qui demande à l'utilisateur d'afficher les entiers de N à M. (Il faut vérifier la saisie de N inferieure à M).

Organigramme



L'algorithme

```
Algorithme de contrôle de saisie et d'affichage
Var M, N: réel;
Début
   Répéter
      Ecrire ("Entrez N : ")
      Lire (N)
      Ecrire ("Entrez M : ")
      Lire (M)
   Jusqu'à (N < M)

   Pour i ← N à M faire
      Ecrire (i)
   FinPour
Fin
```

Le programme

```
#include <stdio.h>
int main() {
   int N, M;
do {
      printf("Entrez N : ");
      scanf("%d", &N);
      printf("Entrez M : ");
      scanf("%d", &M);
      if (N >= M) {
         printf("Erreur : N doit être inférieur à M.\n");
      }
   } while (N >= M);
printf("Les entiers superieur a %d et inferieurs ou egaux a %d sont :\n", N, M);
   for (int i = N; i <= M; i++) {
      printf("%d ", i);
   }
   printf("\n");
   return 0;
}
```

Attention

Il est fortement déconseillé de modifier la valeur du compteur (et/ou la valeur de finale) à l'intérieur de la boucle. En effet, une telle action :

- **perturbe le nombre d'itérations prévu par la boucle Pour**
- **présente le risque d'aboutir à une boucle infinie**

Exemples :

```
 int compteur;
for (compteur = 0 ; compteur < 10 ; compteur++) {
printf("La variable compteur vaut %d !\n", compteur);
compteur = compteur+2; }

for (compteur = 9 ; compteur > 0 ; compteur--) {
printf("La variable compteur vaut %d !\n", compteur);
compteur = compteur+2;}
```

La boucle For Vs. La boucle While

La boucle Pour est un cas particulier de la boucle While (cas où le nombre d'itérations est connu et fixé). Tout ce qu'on peut écrire avec For peut être remplacé par une boucle While (la réciproque n'est pas forcément vraie).

Exemple :

```
int compteur;
for (compteur = 0 ; compteur < 10 ; compteur++)
{printf("La variable compteur vaut %d !\n", compteur); }
// est equivalent a :
int compteur = 0;
while (compteur < 10)
{printf("La variable compteur vaut %d !\n", compteur);
compteur++;}
```

Les boucles imbriquées

Le bloc d'instructions d'une boucle peut contenir lui même une autre boucle. C'est ce qu'on appelle des boucles imbriquées.

Exemple :

```
int i;
 int j;
 for (i = 0 ; i <3 ; i++)
 {
j=1;
while (j <= 4)
 {
 printf("i=%d et j=%d \n", i,j);
 j++;
}
}
```

Itération	i	j	affichage
1	0	1	i=0 et j=1
2	0	2	i=0 et j=2
3	0	3	i=0 et j=3
4	0	4	i=0 et j=4
5	1	1	i=1 et j=1
6	1	2	i=1 et j=2
7	1	3	i=1 et j=3
8	1	4	i=1 et j=4
9	2	1	i=2 et j=1
10	2	2	i=2 et j=2
11	2	3	i=2 et j=3
12	2	4	i=2 et j=4

Exemple 2

Donner l'affichage de ce code

```
int i;
 int j;
 for (i = 1 ; i <= 3 ; i++)
 {
 printf("ligne =%d: ", i);
 j=1;
while (j <= 4)
 {
 printf("colonne=%d; ", j);
 j++;
}
printf("\n");
 }
```

```
ligne =1: colonne=1; colonne=2; colonne=3; colonne=4;
ligne =2: colonne=1; colonne=2; colonne=3; colonne=4;
ligne =3: colonne=1; colonne=2; colonne=3; colonne=4;
```

Récapitulatif

Utilisez la structure qui répète le mieux l'idée du programme que vous voulez réaliser :

- **Si le bloc d'instructions ne doit pas être exécuter si la condition est fausse, alors utilisez while ou for.**
- **Si le bloc d'instructions doit être exécuté au moins une fois, alors utilisez do- while.**
- **Si le nombre d'exécutions du bloc d'instructions est connu à l'avance alors utilisez for.**
- **Si le bloc d'instructions doit être exécuté aussi longtemps qu'une condition extérieure est vraie alors utilisez while.**
- **Généralement, Le choix entre les trois boucles n'est souvent qu'une question de préférence ou d'habitudes.**

Exercice d'application 4

Ecrire un programme qui permet de faire le produit des N premiers nombres sachant que N doit être strictement positif.

for

```
#include <stdio.h>
int main() {
    int N;
    long produit = 1; // utiliser long pour éviter le débordement
do {
        printf("Entrez un entier N strictement positif : ");
        scanf("%d", &N);
    } while (N <= 0);
for (int i = 1; i <= N; i++) {
        produit = produit * i;
    }
    printf("Le produit des %d premiers nombres est : %ld\n", N, produit);
    return 0;
}
```

while

```
#include <stdio.h>
int main() {
    int N, i = 1;
    long produit = 1;
do {
        printf("Entrez un entier N strictement positif : ");
        scanf("%d", &N);
    } while (N <= 0);
while (i <= N) {
        produit = produit * i;
        i++;
    }
    printf("Le produit des %d premiers nombres est : %ld\n", N, produit);
    return 0;
}
```

do while

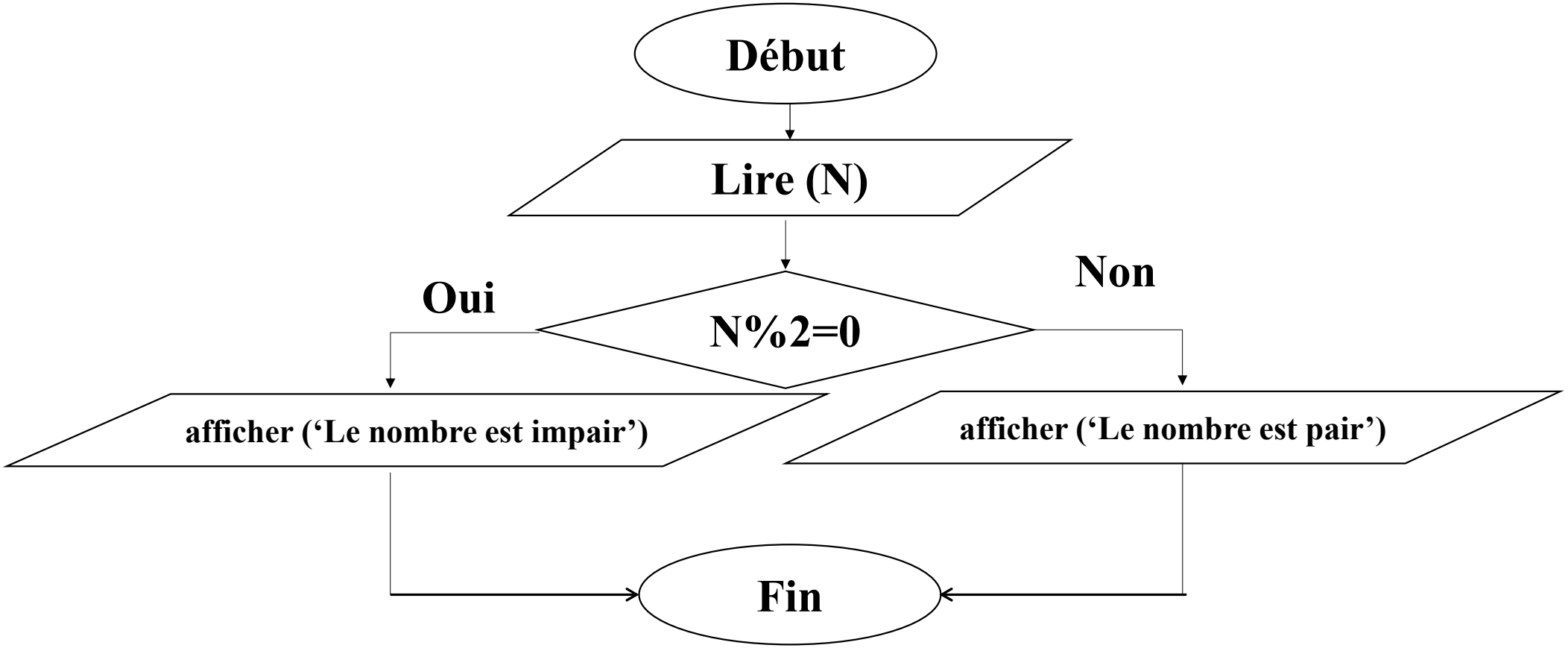
```
#include <stdio.h>
int main() {
    int N, i = 1;
    long produit = 1;
do {
        printf("Entrez un entier N strictement positif : ");
        scanf("%d", &N);
    } while (N <= 0);
do {
        produit = produit * i;
        i++;
    } while (i <= N);
    printf("Le produit des %d premiers nombres est : %ld\n", N, produit);
    return 0;
}
```

Exercice d'application 5

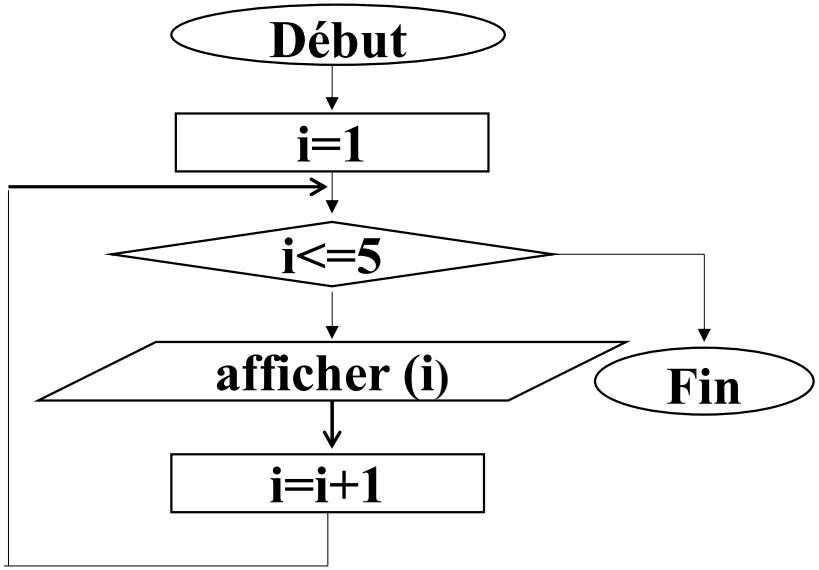
L'algorithme d'Euclide permet de calculer le PGCD de deux entiers naturels non nuls g et p, écrire un programme qui permet de trouver le PGCD en traduisant l'algorithme suivant :
On effectue la division euclidienne de g par p. On note r le reste (on n'utilise pas le quotient).
On remplace ensuite g par p et p par r.
Tant que le p est différent de 0, on répète .
Après un certain nombre d'itérations, on obtiendra un p égal à 0.
Le PGCD de g et p de est alors le reste précédent (c'est à dire le g).

```
#include <stdio.h>
int main() {
    int a, b, p, g, r;
    printf("Donner le premier entier : ");
    scanf("%d", &a);
    printf("Donner le deuxième entier : ");
    scanf("%d", &b);
    if(a<b){
        p=a;
        g=b; }
    else {
        p=b;
        g=a;}
while (p != 0) {
        r = g % p;
        g = p;
        p = r; }
   printf("Le PGCD entre %d et %d est : %d\n",a,b, g);
  return 0;}
```

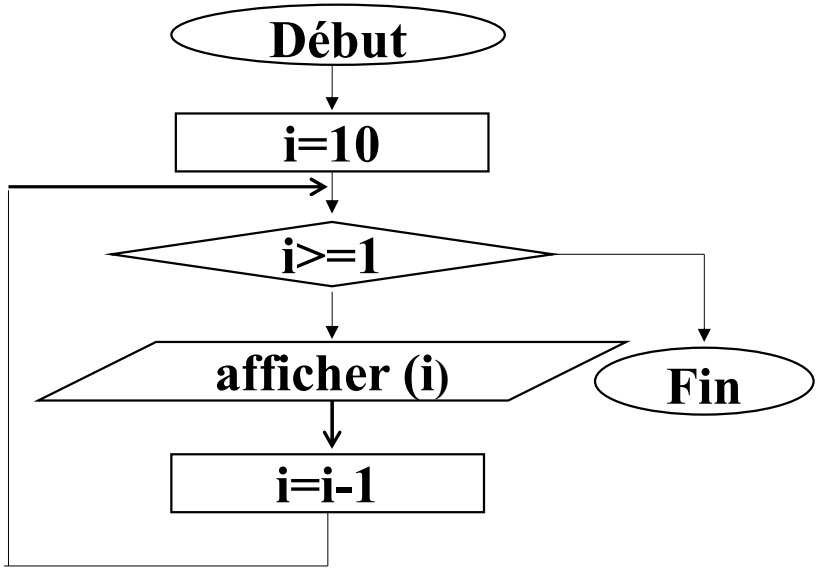
Organigramme d'une structure conditionnelle simple

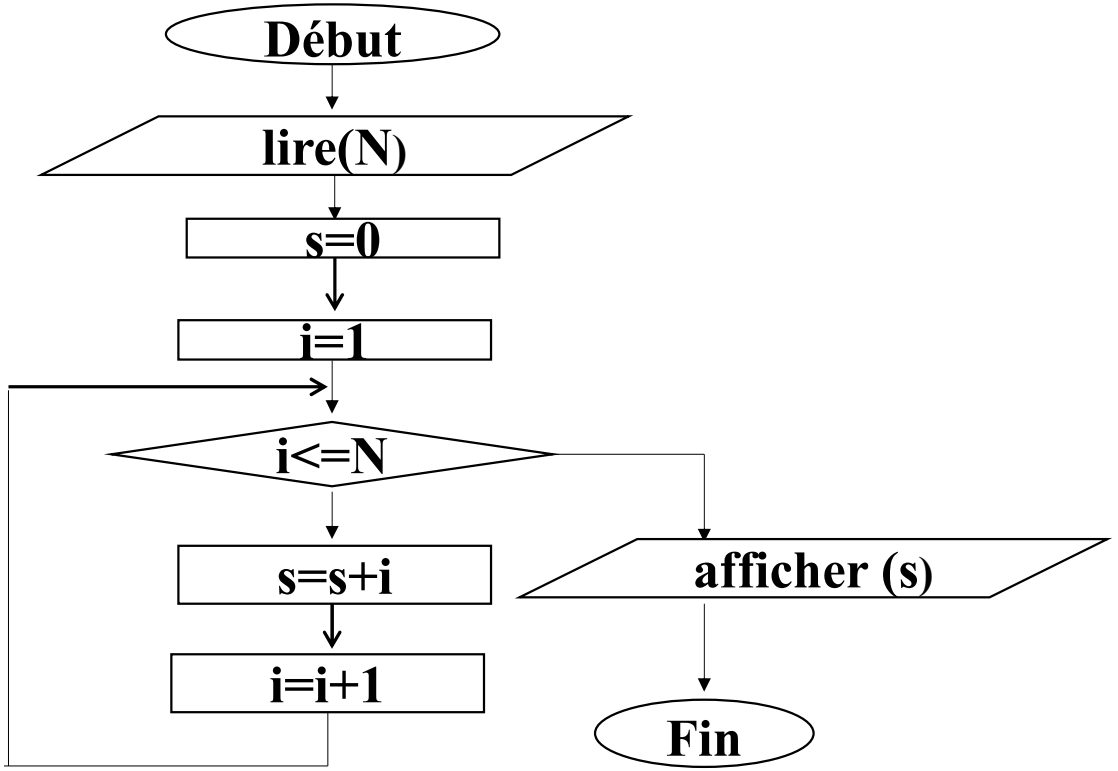


Organigramme d'une structure conditionnelle simple

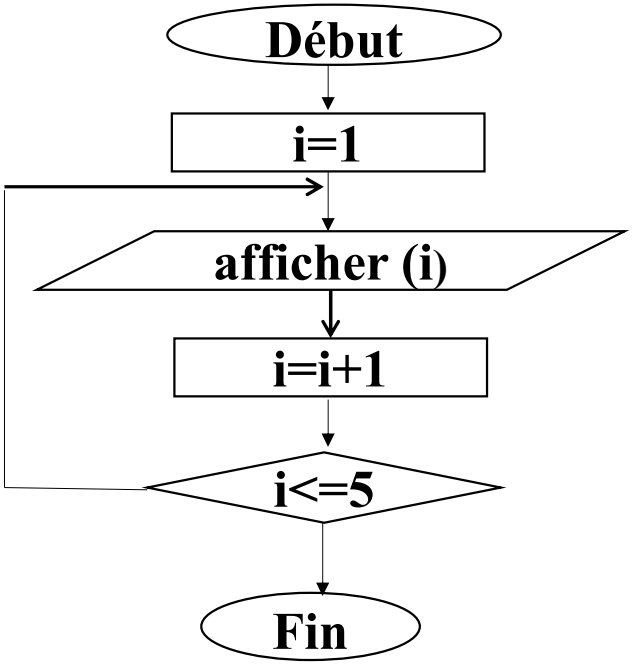


Organigramme d'une structure conditionnelle simple

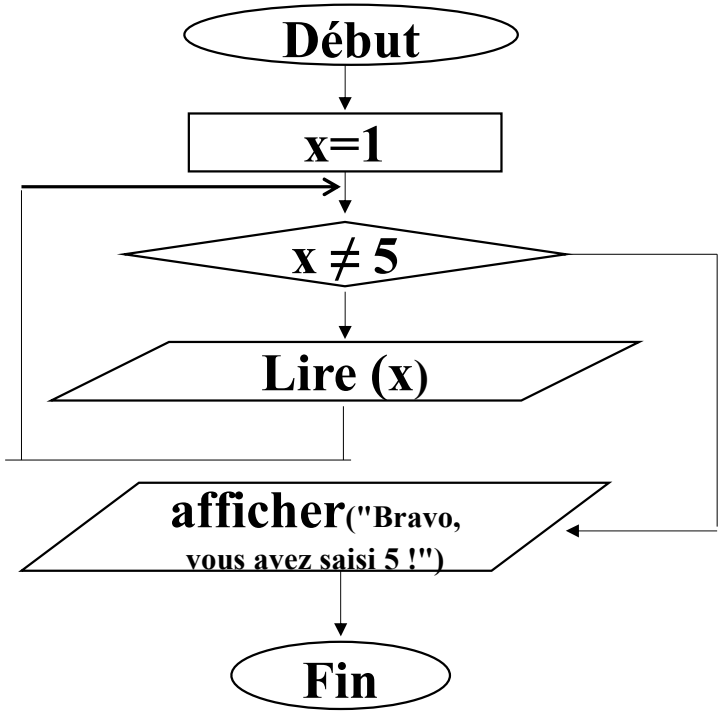



Début
lire(N)
s=0
i=1
i<=N
s=s+i
i=i+1
afficher (s)
Fin

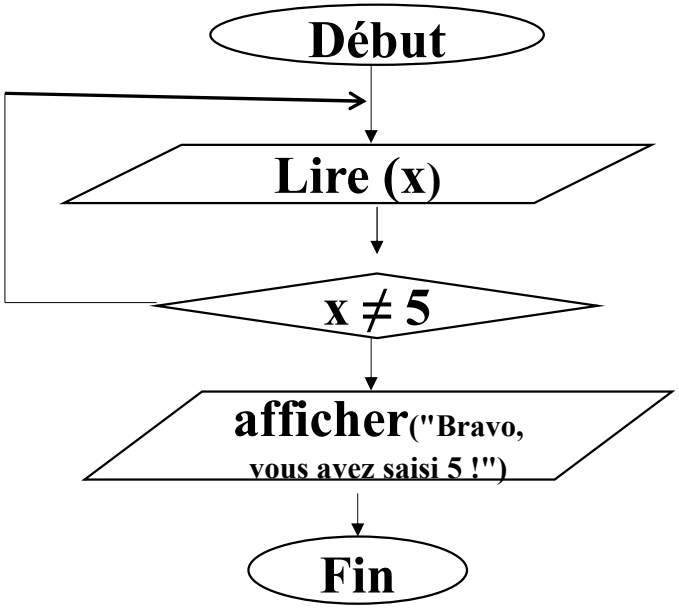
Organigramme d'une structure conditionnelle simple



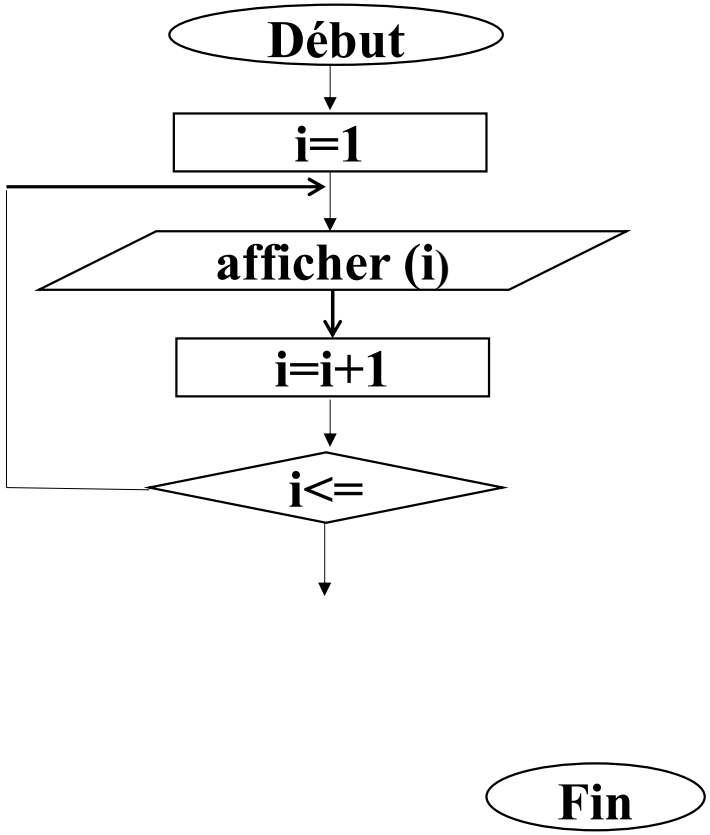
Organigramme d'une structure conditionnelle simple

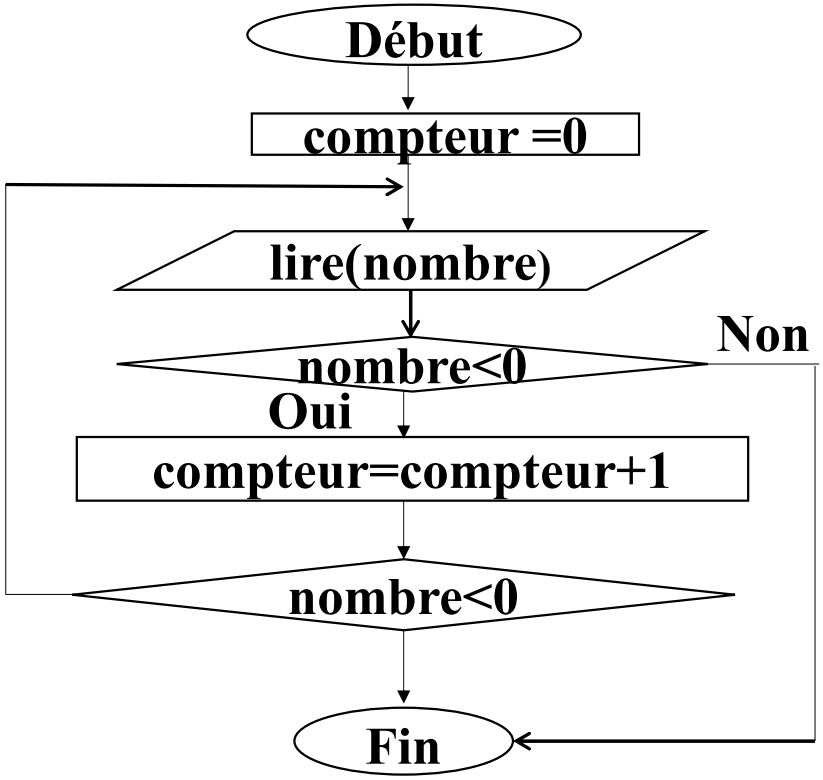


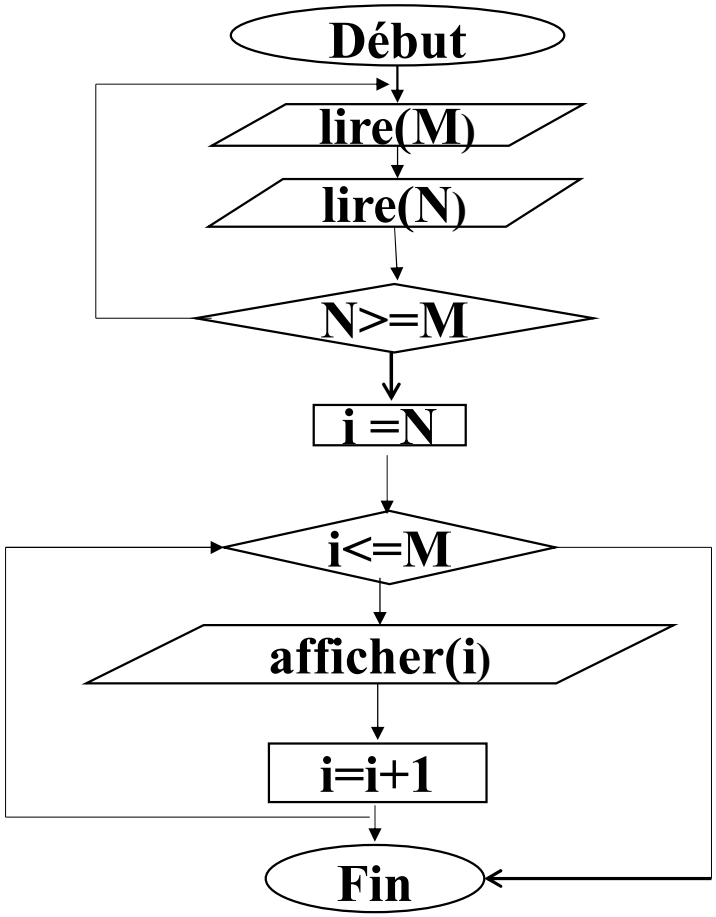
Organigramme d'une structure conditionnelle simple



Organigramme d'une structure conditionnelle simple




Début
compteur =0
lire(nombre)
nombre<0
Non
Oui
compteur=compteur+1
nombre<0
Fin



ui