Algorithmique

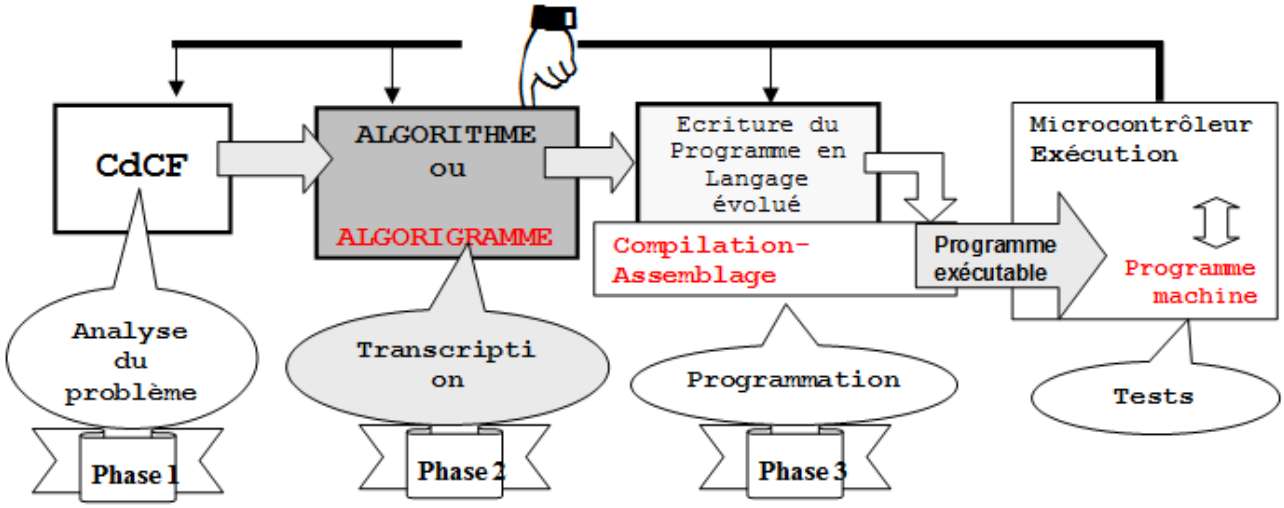
Table des matières

Un algorithme énonce une résolution sous la forme d'une série d'opérations à effectuer. La mise en œuvre de l'algorithme consiste en l'écriture de ces opérations dans un langage de programmation et constitue alors la brique de base d'un programme informatique.

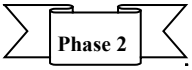
1. Introduction

Un programme informatique a pour objectif de faire exécuter une suite de travaux à une machine mais pour cela, il faut que la machine puisse comprendre les ordres d'un opérateur humain.

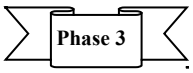
Le développement d'un programme nécessite trois phases.



Cahier Des Charges Fonctionnel (**CdCF**) : expression en français ou avec des outils de spécification du besoin.



ALGORITHMIQUE (Analyse structurée du problème) : représentation sous forme graphique ou en pseudo code.



Traduction dans un langage « de programmation »
Les mots-clés ou les symboles graphiques sont traduits en langage de programmation.

2. Algorithmique

L'algorithmique est l'ensemble des règles et des techniques qui sont impliquées dans la définition et la conception d'algorithmes.

Un algorithme est une suite finie et non-ambiguë d'instructions permettant d'arriver, en un temps fini, à un résultat déterminé, à partir d'une situation donnée.

Pour concevoir un algorithme trois étapes sont nécessaires :

1. **La préparation du traitement** : recherche des données nécessaires à la résolution d'un problème.
2. **Le traitement** : résolution pas à pas du problème après décomposition en plusieurs sous-ensembles si nécessaire.
3. **L'édition** des résultats.

2.1. Modes d'expression d'un algorithme

Un algorithme peut être écrit :

- en langage littéral (pseudo code) par l'utilisation de mots-clés et de délimiteurs. On parle de langage de description d'algorithme.

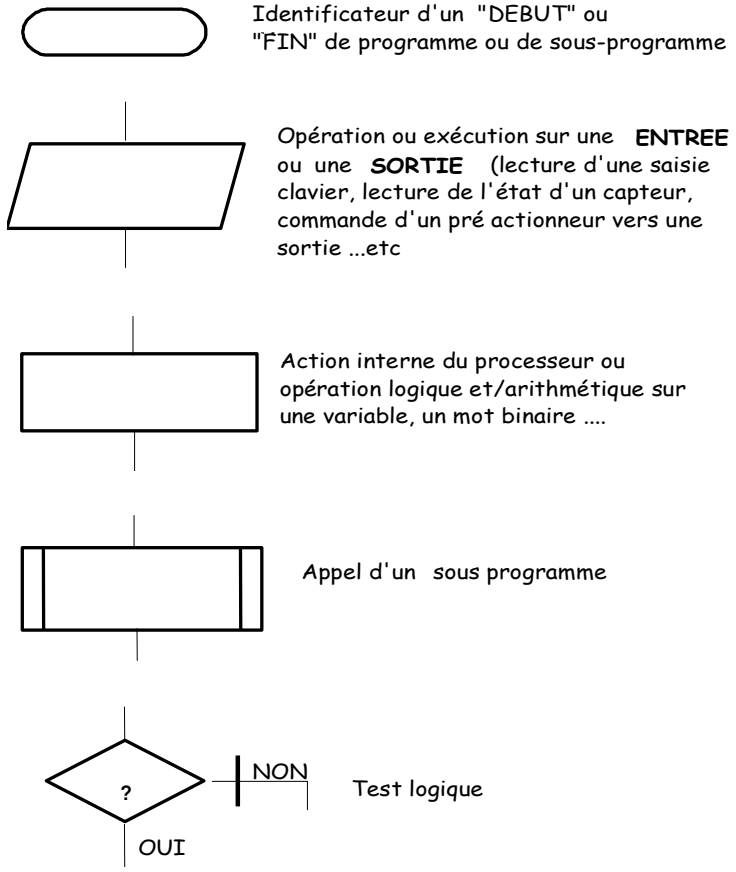
  L'algorithme est bien adapté pour le codage dans un langage de haut niveau (Ex : Java, C#, C, C++, Python, PHP, Javascript).

Exemples de mots-clés	Exemples de mots délimiteurs
Lire – Ecrire **si… alors… sinon** **tant que …… faire**	Ils fixent les bornes des entrées et des sorties des structures algorithmiques : **début - fin - finsi**

- graphiquement (utilisation de symboles normalisés), on parle d'algorigramme ou d'organigramme de programmation.

  L'algorigramme est plus adapté au codage dans un langage de bas niveau (Ex : assembleur) ou pour l'apprentissage de l'algorithmique (Ex : logiciel Flowcode).

PRINCIPAUX SYMBOLES UTILISES (NF Z 67-010)



2.2. Organisation d'un algorithme

2.2.1. L'en tête

Dans cette partie le concepteur donne un nom à l'algorithme. Il définit le traitement effectué et les données auxquelles il se rapporte.

Exemple : Algorithme Calcul_Surface_Disque
{ Diamètres D et d existent }
{ Affichage de la surface du disque }

2.2.2. La partie déclarative

Dans cette partie, le concepteur décrit les différents « objets » que l'algorithme utilise.

2.2.2.1. Les constantes

Une constante est un emplacement de stockage qui contient une valeur constante dans tout l'algorithme.

Constante

*(Nom_Constante **:** type) := valeur **;***

Exemple :

Constante

(Pi : réel) := 3,1416

(c : entier) := 299 792 458

La déclaration de constantes symboliques permet de donner un nom à un objet constant dans tout l'algorithme et ensuite de faire référence à cet objet par son nom plutôt que par sa valeur.

2.2.2.2. Les variables

Une variable est un emplacement de stockage qui contient une valeur pouvant changer au cours de l'exécution de l'algorithme. Un programme peut-être amené à manipuler différents types de données :

- booléen : valeur pouvant être soit Vraie, soit Fausse
- entiers : valeur numériques entières pouvant être signées ou non signées
- réels : valeurs numériques codées avec une mantisse et un exposant
- caractère : octet correspondant à un code ASCII
- chaîne de caractères : ensemble de caractères
- tableau de données : ensemble de données de même type

Variable

Nom_Variable **:** type

Exemple :

Variable

vitesse : réel

nbr_buts : entier

Une variable est caractérisée par :

- son nom,
- son type qui représente l'ensemble (au sens mathématique) dans lequel la variable prend ses valeurs,
- sa valeur.

2.2.3. La partie exécutive

2.2.3.1. Les procédures et fonctions

Les procédures et fonctions peuvent nécessiter éventuellement un ou plusieurs paramètres d'entrée ou de sortie.

Un paramètre d'entrée est la référence à une variable manipulée par la procédure ou la fonction.

Un paramètre de sortie est une valeur renvoyée par une fonction.

Une fonction ou une procédure peut elle-même appeler une ou plusieurs fonctions et procédures.

Fonction NomFonction(NomEntrée1 : [Type], NomEntrée2 : [Type],…) : [TypeDuRésultat]

Constante { déclaration des constantes locales }

Variable { déclaration des variables locales }

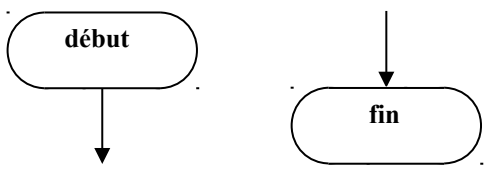
Début

{ description des actions effectuées par la fonction }

Fin

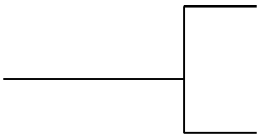
2.2.3.2. Le programme principal

Le programme principal consiste en une suite d'opérations élémentaires faisant souvent appel à des fonctions ou procédures.Il est délimitée par des primitives algorithmiques (mots clés) ou des schémas :

Algorithme	**Algorigramme** (NF Z 67-010)
début .... **fin**	

2.2.4. Les commentaires

Des commentaires doivent être insérés dans le programme afin d'en faciliter la relecture.

Algorithme	**Algorigramme** (NF Z 67-010)
{ Commentaire **}**	

2.3. Les structures algorithmiques

Les structures algorithmiques sont réparties en 3 catégories :

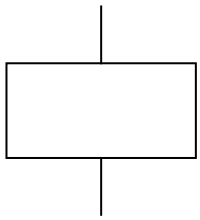
- structures linéaire d'opérations
- structures alternatives (ou conditionnelles) ou de choix : en fonction d'une condition, le programme exécute des opérations différentes
- structures itératives ou répétitives: sous contrôle d'une condition, une séquence d'opérations est exécutée répétitivement

2.3.1. Les structures linéaires

- **L'affectation**

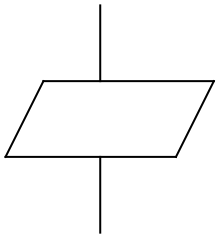
  Cette action attribue une **valeur** (une constante ou le résultat d'un traitement) à une variable.

  On notera cette action par le symbole : ← ou :=

Algorithme	**Algorigramme** (NF Z 67-010)
Nom_Variable **:=** *valeur*	

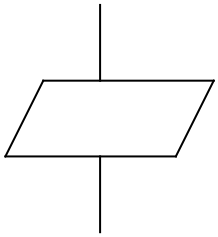
  L'affectation n'a de sens que si les deux objets de part et d'autre du signe := sont de même type.

- **La saisie d'une valeur sur un terminal**

Algorithme	**Algorigramme** (NF Z 67-010)
Lire( )	

On pourra lire une variable seule, de type quelconque, ou un ensemble de variables séparées par des virgules, de type identique ou différent.

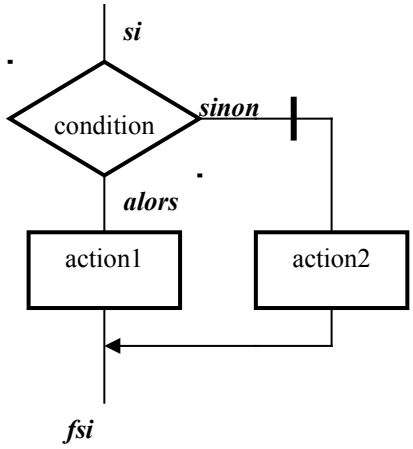
- **L'édition de résultats sur un périphérique de sortie**

Algorithme	**Algorigramme** (NF Z 67-010)
Ecrire( )	

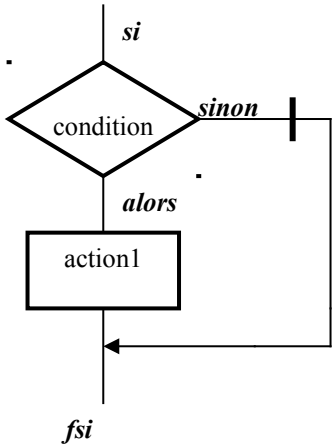
On pourra écrire les valeurs d'une ou plusieurs variables.

2.3.2. Les structures alternatives

- **La structure alternative de base**

Algorithme	**Algorigramme** (NF Z 67-010)
si (*condition vraie)* ***alors*** *action1* **sinon** *action2* **finsi**	

Si **action2** est vide, on écrira :

Algorithme	**Algorigramme (**NF Z 67-010)
si (*condition vraie)* ***alors*** *action1* ***finsi***	

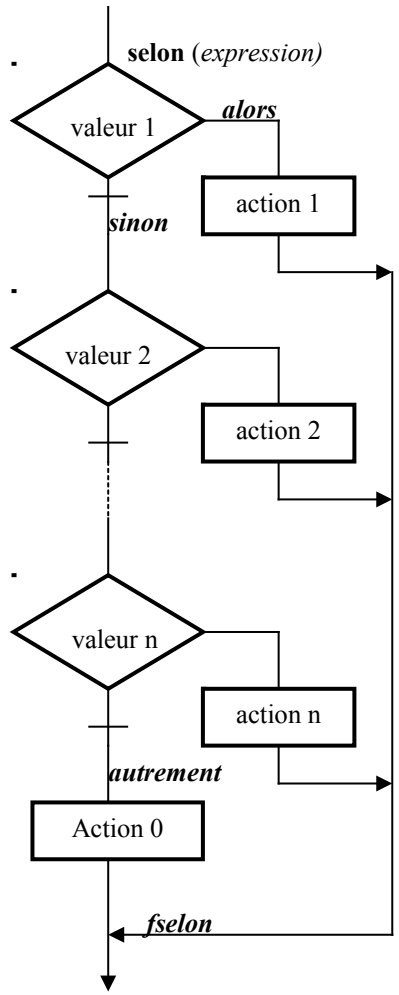
- **Les structures alternatives imbriquées**

Il peut arriver que « action1 » et « action2 » soient elles-mêmes des actions comportant un choix, par exemple :

si (*condition1)*
alors
    si *(condition2)*
    alors
        action1
    sinon
        action2
    finsi
sinon
    si *(condition3)*
    alors
        action4
    sinon
        action5
    finsi
finsi

- **Structure de choix multiple**

  Lorsqu'on peut choisir entre plusieurs possibilités se rapportant à une même expression, on utilisera la primitive suivante :

Algorithme	**Algorigramme** (NF Z 67-010)
Selon (*expression*) ***Faire*** *valeur 1 :* *action_1* ***FinFaire*** *valeur 2 :* *action_2* ***FinFaire*** … *valeur n :* *action_n* ***FinFaire*** **Sinon** : *action_par-defaut* **FinSelon**	

2.3.3. Les structures itératives ou répétitives

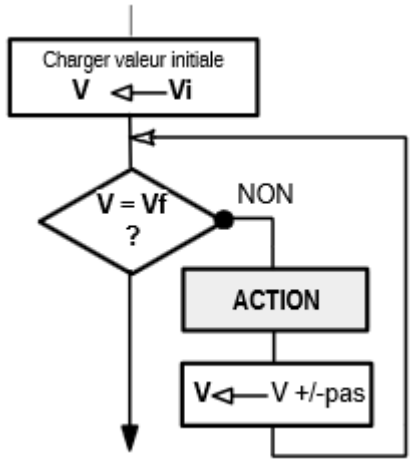
Ces structures permettent d'exécuter plusieurs fois une séquence d'instructions.

Pour que l'itération puisse se terminer, il faut que l'exécution de l'action modifie la valeur de l'expression conditionnelle évaluée.

Pour cela, il faut veiller à initialiser la ou les variables intervenant dans la condition d'évolution (test) de l'itération.

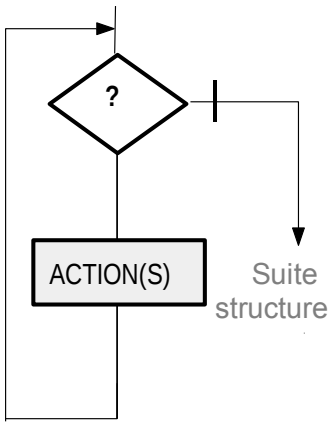
- Le nombre d'itérations est connu (boucle de comptage)

  Lorsque le nombre d'itérations est connu, on peut utiliser une variable auxiliaire (compteur de boucle) dont la valeur caractérise le nombre de passages dans la boucle.

Algorithme	**Algorigramme** (NF Z 67-010)
Variables i : entier; **pour i** <*valeur_initiale*> **à** <*valeur_finale*> **par pas de** <*n*> **Faire** *Action(s);* **FinFaire**	

Remarque : pour des valeur de n = 1, l'instruction « par pas de » est optionnelle.

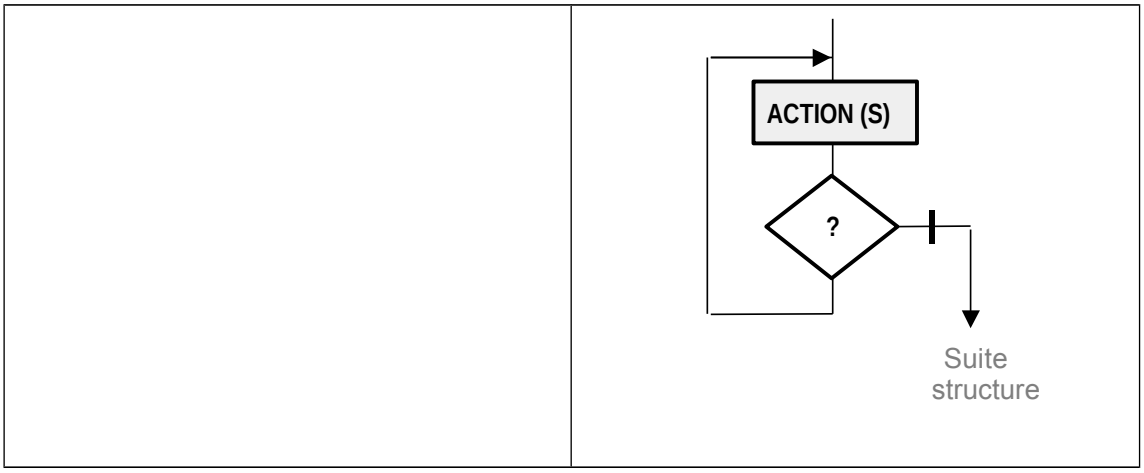
- Le nombre d'itérations est inconnu (test en tête de boucle)

Algorithme	**Algorigramme** (NF Z 67-010)
Tantque (*condition vraie*) **Faire** *Action(s)* **FinFaire**	

- Le nombre d'itérations est inconnu (test en fin de boucle)

  L'instruction est réalisée au moins UNE fois.

Algorithme	**Algorigramme** (NF Z 67-010)
Faire *Action(s)* **Tantque** (*condition vraie)*	



2.4. Structures de données : les vecteurs (tableaux à une dimension)

Soient :

- un ensemble quelconque : l'ensemble des valeurs E
- un entier n : nombre d'éléments du vecteur
- un intervalle I de N tel que I = [0..n], avec (n ≥ 0)

Un vecteur est une application V de I dans E ; I est appelé ensemble des indices.

Exemple :

	0	1	2	3	4	5
V	A	B	Z	A	C	D

V[0..5] := ('A','B','Z','A','C','D')
V[2] := 'Z'
V[6] : non défini

On note par ← l'action d'empiler une valeur dans un vecteur : V ← valeur

et par → l'action de dépiler une valeur d'un vecteur : V → variable

3. Traduction dans un langage de programmation

Cette dernière étape devrait être celle à laquelle le concepteur consacre le moins de temps (dans l'hypothèse ou les deux étapes précédentes ont été correctement développées !).

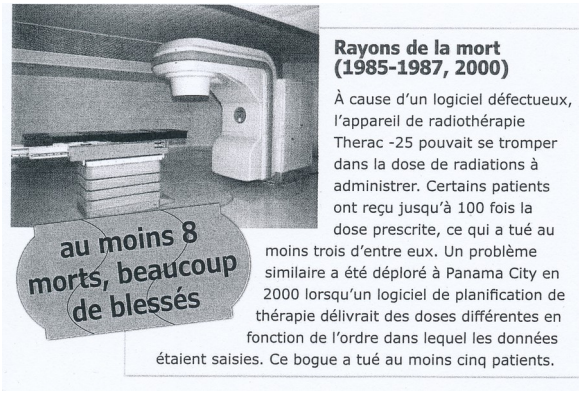
« *Contrairement aux idées reçues, il n'est pas facile du tout de produire un bon logiciel : cela demande beaucoup de* ***précision et de soin*** »

Pour éviter les défauts logiciels (bogues), chaque ligne de code écrite doit répondre à **trois exigences** :

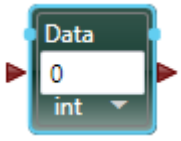
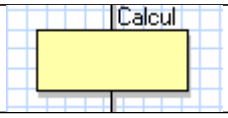
1. une **bonne compréhension du problème** et de la solution que la ligne est supposée mettre en œuvre
2. la capacité d'**exprimer** cette solution correctement dans un langage de programmation
3. l'absence totale de la moindre faute de frappe

Afin de réduire le risque de créer de futurs défaut de code, il est bon :

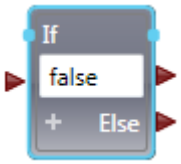
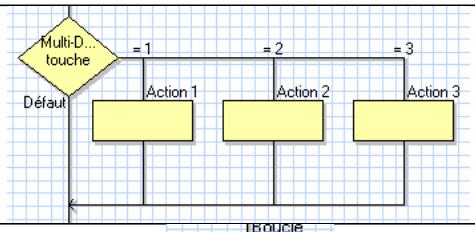
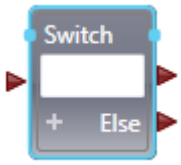
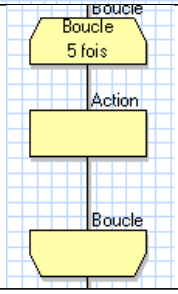
4. de s'assurer que le **code** est **clair et compréhensible**
5. d'ajouter des **commentaires** (et les maintenir à jour) pour expliquer le raisonnement sous-jacent
6. de suivre une norme de **codage (Ex : HICPP pour C++) ou des conventions (Ex : conventions de codage pour java, C# etc)**



4. Symboles utilisés dans « Flowcode » et « VPL[1] »

Algorithmique	Logiciel Flowcode	Langage VPL
Commentaires	Idem norme NFZ 67-010	
Constante		
Variable		
Affectation		Pas de symbole particulier

1 Visual Programming Language

Choix	Idem norme NFZ 67-010	
Choix multiple		
Boucle « pour… (Nombre d'itération connu)		Pas de symbole particulier

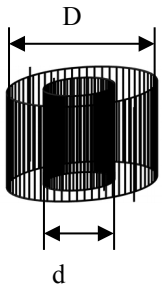
5. Exemple de traitement algorithmique

Enoncé du problème :

Une machine découpe des disques circulaires de diamètre D, percés d'un trou de diamètre d dans une plaque (d < D).

Des palpeurs mesurent D et d et transmettent ces informations à un calculateur.

Celui-ci doit déterminer la surface S du disque et l'afficher.



Phase 1 : Analyse du problème

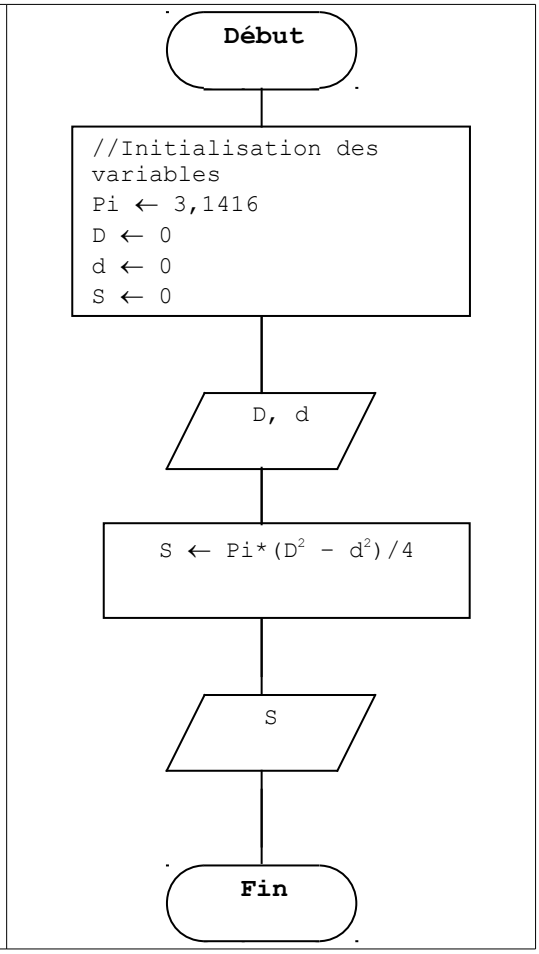
1. **Préparation du traitement** : saisie des diamètres D et d.
2. **Traitement** : calcul de la surface du disque S = π(D² - d²) / 4.
3. **Edition des résultats** : Affichage de la surface du disque S.

Phase 2 : Expression du problème en langage algorithmique

Primitives algorithmiques	Algorigramme

```
Algorithme Calcul_Surface_Disque;
{ Diamètres D et d existent }
{ Affichage de la surface du disque }

{---- En-tête ----}
Constante
      (Pi : réel) := 3,1416

{---- Partie déclarative ----}
Variable
      D, d : réels       { données }
      S : réel           { résultat }

{---- Partie exécutive ----}
Début
```

```
        { Etape 1 : Préparation du traitement }
        Lire(D, d)

        { Etape 2 : Traitement algorithmique }
        S := Pi * (D² - d²) / 4

        { Etape 3 : Edition des résultats }
        Ecrire("La surface du disque est :", S)
fin
```



Phase 3 : codage, programmation en C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CalculSurfaceDisque
{
    class Program
    {
        static void Main(string[] args)
        {
            // Partie déclarative
            // La constante Pi est définie dans la bibliothèque Math
            Double  D = 0.0, d = 0.0, S = 0.0;

            // Partie Exécutive
            Console.Write("Entrez la valeur de D (en mm) : ");
            D = Single.Parse(Console.ReadLine());
            Console.Write("Entrez la valeur de d (en mm) : ");
            d = Single.Parse(Console.ReadLine());
            S = Math.PI * (Math.Pow(D, 2.0) - Math.Pow(d, 2.0)) / 4;
            Console.WriteLine("L'aire du disque est :" + S + "mm²");
            Console.ReadKey();
        }
    }
}
```