Chapitre 4 : Programmation Fortran

*La programmation est l'art de créer des programmes informatiques, qui sont des ensembles d'instructions exécutées par un ordinateur pour accomplir une tâche spécifique. Elle est au cœur de la technologie moderne et joue un rôle essentiel dans de nombreux domaines, tels que le développement de logiciels, la conception de sites web, l'intelligence artificielle, l'analyse de données et bien d'autres.**

Le processus de programmation implique la création d'un algorithme, qui est une séquence d'étapes logiques pour résoudre un problème. L'algorithme est ensuite traduit dans un langage de programmation compréhensible par l'ordinateur. Les langages de programmation sont des systèmes de notation spécifiques qui permettent aux programmeurs de communiquer avec l'ordinateur en utilisant des instructions structurées.

La programmation offre des possibilités infinies en termes de création et d'innovation. Elle permet de résoudre des problèmes complexes, d'automatiser des tâches répétitives, de manipuler et d'analyser des données, et de créer des applications et des logiciels qui facilitent la vie quotidienne.

1. Pourquoi nous programmons ?

Nous programmons pour plusieurs raisons :

- **Résoudre des problèmes** : La programmation nous permet de résoudre des problèmes de manière efficace et précise. Nous pouvons concevoir des algorithmes et écrire des programmes pour automatiser des tâches, effectuer des calculs complexes, analyser des données, faciliter la communication, etc. La programmation nous donne les moyens de transformer des idées abstraites en solutions concrètes.
- **Automatiser des tâches** : La programmation nous permet d'automatiser des tâches répétitives ou laborieuses. Au lieu d'effectuer manuellement certaines actions, nous pouvons écrire des programmes qui les exécutent pour nous. Cela permet un gain de temps et d'efficacité, nous libérant ainsi pour des tâches plus importantes et créatives.
- **Améliorer l'efficacité et la productivité** : La programmation nous permet d'optimiser des processus et de rendre les systèmes plus efficaces. Nous pouvons écrire des programmes qui automatisent des tâches, améliorent la gestion des données, facilitent la communication et la collaboration, réduisent les erreurs, etc. Cela contribue à une meilleure productivité et à des résultats de meilleure qualité...etc.

1.1. Programmation

Définition

La programmation est le processus de création d'un ensemble d'instructions ou de ***codes informatiques**** qui permettent à un ordinateur d'effectuer une tâche spécifique ou de résoudre un problème donné. C'est un moyen de communiquer avec un ordinateur en utilisant un langage de programmation pour lui dire quoi faire.

Comment programmons ?

Fondamental

Comprendre le problème : Commencez par une compréhension approfondie du problème à résoudre. Identifiez les entrées et les sorties requises, ainsi que les étapes nécessaires pour passer de l'entrée à la sortie.

Concevoir l'algorithme : Un algorithme est une séquence d'étapes logiques pour résoudre un problème. Réfléchissez à la manière dont vous pouvez décomposer le problème en sous-problèmes plus petits et à la manière dont ces sous-problèmes peuvent être résolus. Déterminez l'ordre des étapes, les conditions de bouclage ou de prise de décision, et les opérations nécessaires pour résoudre le problème.

Créer l'organigramme : Un organigramme est une représentation visuelle de l'algorithme, utilisant des symboles et des connexions pour décrire le flux d'exécution. Utilisez des boîtes pour représenter les étapes, des flèches pour représenter le flux de contrôle et des diamants pour représenter les décisions conditionnelles. Dessinez l'organigramme en suivant la structure et la logique de l'algorithme que vous avez conçu.

Choisir un langage de programmation : Sélectionnez un langage de programmation approprié pour mettre en œuvre votre algorithme. Le choix du langage dépendra de plusieurs facteurs, tels que les fonctionnalités nécessaires, la plate-forme de déploiement, les préférences personnelles, etc.*

1. **Écrire le code** : Utilisez le langage de programmation choisi pour traduire l'algorithme en code informatique. Écrivez les instructions spécifiques nécessaires pour réaliser chaque étape de l'algorithme. Utilisez des structures de contrôle (boucles, conditions) et des structures de données (tableaux, listes) appropriées pour manipuler les données et réaliser les opérations requises.
2. **Tester et déboguer** : Testez votre programme en fournissant différentes entrées et en vérifiant si les sorties correspondent aux attentes. Identifiez et corrigez les erreurs ou les bogues éventuels en utilisant des techniques de débogage, telles que l'inspection du code, l'affichage de variables, l'utilisation de points d'arrêt, etc.
3. **Optimiser et améliorer** : Une fois que votre programme fonctionne correctement, vous pouvez chercher des moyens d'optimiser les performances, de réduire la consommation de ressources ou d'améliorer la convivialité. Vous pouvez également effectuer des tests de validation supplémentaires pour vous assurer que le programme répond aux besoins initiaux.

Démarche et analyse d'un problème

Fondamental

Un algorithme s'écrit le plus souvent en pseudo-langage de programmation afin de faciliter ultérieurement

sa traduction dans un langage de programmation.

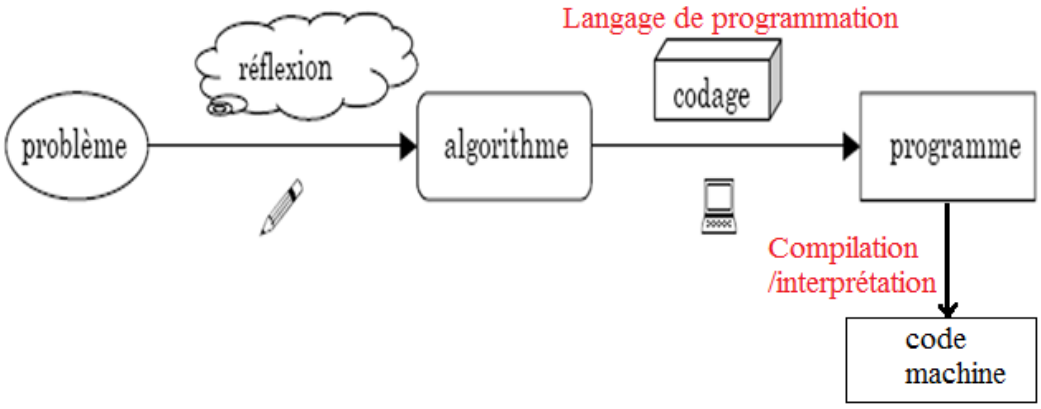
Un programme se construit selon plusieurs étapes, la première consiste en l"analyse du problème posé.la

seconde est l „établissement d"un algorithme et la troisième étape est la traduction de l"algorithme en programme,

en utilisant un langage choisi. En fait L"algorithme est la résolution brute d"un problème informatique, il est

indépendant du langage de programmation, Par exemple, on utilisera le même algorithme pour une implantation en **Java, ou bien en C++ ou Fortran** ou autre langage.[(8)1]

[1] The Pragmatic Programmer" de Andrew Hunt et David Thomas



Démarche et analyse d'un problème

2. Programme Fortran

Fortran est un langage de programmation, développé par ***IBM***∗ vers 1955, et destin?e ?a fournir aux scientifiques un moyen simple pour passer de leurs formules mathématiques jusqu' a un programme effectif (son nom est une abréviation de **FORmula TRANslator**).

Fortran a fait l'objet de plusieurs normalisations : Fortran 77, Fortran 90 et 95, et plus récemment Fortran 2003.∗

2.1. Environnement Fortran

Environnement Fortran ***Définition***

L'environnement de développement pour Fortran est un ensemble d'outils logiciels qui facilite la création, le développement, le ***débogage***∗ et la **compilation de programmes Fortran**. Ces environnements fournissent un ensemble d'outils intégrés qui aident les programmeurs à écrire du code Fortran de manière plus efficace et à améliorer leur productivité.

Voici quelques éléments couramment présents dans un environnement de développement pour Fortran :

- **Éditeur de code** : Un éditeur de code spécialement conçu pour la programmation en Fortran est un élément essentiel de l'environnement de développement. Cet éditeur fournit des fonctionnalités telles que la coloration syntaxique, l'indentation automatique, l'auto complétion et la mise en évidence des erreurs de syntaxe, facilitant ainsi la rédaction du code.
- **Débogueur** : Un débogueur permet aux programmeurs de suivre l'exécution de leur programme ligne par ligne, d'inspecter les valeurs des variables et de détecter les erreurs de logique. Il permet de localiser et de corriger les erreurs plus rapidement en facilitant la compréhension du comportement du programme.
- ***Compilateur Fortran : L'environnement de développement doit être compatible avec un compilateur Fortran. Un compilateur Fortran est un logiciel qui traduit le code source Fortran en langage machine exécutable. Il existe plusieurs compilateurs Fortran populaires, tels que GNU Fortran (gfortran), Intel Fortran Compiler (ifort) et IBM XL Fortran.***∗
- **Gestionnaire de projet** : Un gestionnaire de projet permet d'organiser et de gérer les fichiers sources, les bibliothèques et les dépendances d'un projet Fortran. Il facilite la création, la compilation et l'exécution de projets complexes en gérant les dépendances et en automatisant les tâches courantes.

A savoir : ***Remarque***

Un environnement de développement pour Fortran fournit des outils et des fonctionnalités spécifiques pour faciliter la programmation en Fortran, améliorer la productivité et simplifier les tâches de développement. Ces environnements offrent une expérience de développement plus fluide et conviviale pour les programmeurs Fortran.

Les différents environnement ***Remarque***

1. *Il existe plusieurs environnements de développement populaires pour Fortran, tels que* ***Visual Studio Code avec l'extension "Fortran", Intel Parallel Studio, Eclipse avec le plugin Photran, et Code::Blocks avec le plugin*** *Fortran. Chacun de ces environnements offre différentes fonctionnalités et s'adapte aux préférences individuelles des programmeurs..*∗
2. Il est important de noter que même sans un environnement de développement spécifique à Fortran, vous pouvez toujours écrire et compiler des programmes Fortran à l'aide d'un éditeur de texte standard et d'un compilateur Fortran en ligne de commande [13][1]

2.2. Formats d'écritures

Format fixe et format libre ! ***Définition***

En Fortran 90, il existe deux formats de présentation de code : le format fixe et le format libre.

Format fixe :

- Dans le format fixe, les colonnes de la 7 à la 72 sont réservées pour le code Fortran.
- Les colonnes 1 à 5 sont réservées pour les étiquettes de ligne optionnelles.
- Les colonnes 6 et 7 sont utilisées pour les indicateurs de séquence optionnels.
- Les instructions commencent à la colonne 7 et sont limitées à 72 colonnes.
- Les commentaires peuvent être inclus à la fin de la ligne, à partir de la colonne 73.
- Les lignes vides ou ne contenant que des espaces blancs sont ignorées.

Format libre :

- Dans le format libre, il n'y a pas de règles strictes sur les colonnes utilisées pour le code Fortran.
- Les instructions peuvent commencer n'importe où sur la ligne et peuvent occuper plusieurs lignes.
- Les commentaires peuvent être inclus sur la même ligne que le code, après un point d'exclamation (!) ou entre des délimiteurs C (/* ... */).
- Les lignes vides ou ne contenant que des espaces blancs sont ignorées.

[1] A. Mayer, Cours de programmation : Fortran 90,

3. Notion d'unité de programme

3.1. Unité de programme

Unité de programme *Fondamental*

*Fortran 90 distingue les unités de programme (**scoping units**) suivantes :*

1. Programme principal;
2. Sous-programmes:
    - sous-routines
    - onctions;
3. Modules
4. Block data

Pangramme principale *Définition*

La structure de programme principale en Fortran suit généralement le schéma suivant :

```
PROGRAM nom_du_programme
  ! Déclarations de variables spécifiques à la section principale

  ! Instructions principales
  ! ...

  ! Fin du programme
END PROGRAM nom_du_programme
```

- **program Nom du Programme** : C'est la déclaration du programme principal. Le mot-clé program est suivi du nom que vous donnez à votre programme. Ce nom peut être choisi librement, mais il est généralement représentatif de la fonctionnalité du programme.
- **! Déclarations de variables** : Cette section est facultative et est utilisée pour déclarer les variables dont vous avez besoin dans votre programme. Les variables peuvent être déclarées avant ou après les instructions, selon les besoins du programme.
- **! Instructions et logique du programme** : C'est la partie principale du programme où vous écrivez les instructions et la logique pour effectuer les tâches souhaitées. Vous pouvez utiliser des boucles, des conditions, des fonctions, des appels à des sous-programmes, etc., pour organiser et exécuter les actions nécessaires.
- **! Fin du programme** : Cette ligne marque la fin du programme principal. Elle est suivie du mot-clé end program suivi du nom du programme pour indiquer la fin du programme principal.

Exemple

Dans cet exemple, nous avons déclaré deux variables (num et result) spécifiques à la section principale. Ensuite, nous avons lu une valeur à partir de l'entrée standard (READ) dans la variable num, effectué un calcul (result = num * 2.0), puis affiché le résultat (WRITE) à l'écran.

```
PROGRAM mon_programme
  ! Déclarations de variables spécifiques à la section principale
  INTEGER :: num
  REAL :: result

  ! Instructions principales
  READ(*, *) num
  result = num * 2.0
```

```
    WRITE(*, *) 'Le résultat est :', result
End
```

3.2. Les procédures

Procédures ? *Définition*

Les procédures sont des éléments fondamentaux de la programmation qui permettent d'organiser et de réutiliser du code de manière modulaire. Elles regroupent un ensemble d'instructions ou d'algorithmes spécifiques sous un même nom, ce qui facilite la compréhension, la maintenance et la réutilisation du code.∗

Il existe deux types principaux de procédures : les procédures internes et les procédures externes.

- **Procédures internes** : Les procédures internes, également appelées procédures intégrées ou prédéfinies, sont des procédures fournies par le langage de programmation lui-même. Elles sont déjà définies et prêtes à être utilisées sans nécessiter de déclarations supplémentaires. Les procédures internes sont souvent utilisées pour effectuer des opérations courantes, telles que les fonctions mathématiques, les opérations sur les chaînes de caractères, les entrées/sorties, etc. Elles permettent de gagner du temps en évitant de réinventer la roue et offrent des fonctionnalités prêtes à l'emploi.
- **Procédures externes** : Les procédures externes, également appelées procédures utilisateur, sont des procédures créées par le programmeur pour répondre à des besoins spécifiques. Elles sont définies par le programmeur et peuvent être utilisées à plusieurs endroits du code. Les procédures externes permettent de regrouper des instructions spécifiques et de les réutiliser à travers le programme. Elles sont particulièrement utiles pour encapsuler des algorithmes complexes, des tâches récurrentes ou des fonctionnalités spécifiques afin de faciliter la maintenance et la lisibilité du code.

Différents types de Procédures *Fondamental*

***Les procédures, les fonctions et les sous-routines sont des éléments fondamentaux de la programmation qui permettent d'organiser et de réutiliser du code de manière modulaire*∗. Voici un résumé de ces concepts** :

- Les **sous-routines**, également appelées **procédures ou subroutines**, sont des séquences d'instructions regroupées sous un même nom. Elles sont utilisées pour exécuter des tâches spécifiques ou des instructions récurrentes dans un programme. Les sous-routines ne renvoient pas de valeur en sortie et peuvent recevoir des paramètres en entrée.
- Les **fonctions** sont des séquences d'instructions similaires aux sous-routines, mais elles renvoient une valeur en sortie. Les fonctions sont utilisées pour effectuer des calculs ou des opérations spécifiques et renvoyer le résultat. Elles peuvent également recevoir des paramètres en entrée[(15)1]

4. Déclarations des données

La déclaration des données, qu'il s'agisse de constantes ou de variables, revêt une grande importance dans la programmation. Elle permet de donner un sens et une structure aux données manipulées dans un programme. Voici quelques points clés sur l'intérêt de la déclaration des données :

- **Organisation et lisibilité du code** : La déclaration explicite des constantes et des variables permet de donner un nom significatif aux données utilisées dans le programme. Cela facilite la compréhension du code pour les développeurs, rend le programme plus lisible et facilite la maintenance à long terme.

[1] N. Dubesset et J. Vignes, Le Fortran – Le langage normalisé (Technip,

- **Utilisation cohérente des données** : En déclarant des constantes, vous pouvez attribuer un nom significatif à une valeur fixe utilisée dans le programme, comme une constante mathématique ou un paramètre de configuration. Cela permet d'utiliser cette valeur de manière cohérente à différents endroits du code, sans risque de confusion ou d'erreur.
- ***Allocation de mémoire**** : La déclaration des variables permet de réserver de l'espace mémoire pour stocker les données. En spécifiant le type de données associé à chaque variable, vous pouvez définir la taille nécessaire en mémoire. Cela permet une utilisation efficace des ressources et évite le gaspillage de mémoire.
- **Validation et détection d'erreurs** : La déclaration des données permet au compilateur de vérifier la cohérence des opérations effectuées sur ces données. Le compilateur peut signaler des erreurs potentielles, telles que des incompatibilités de types ou des erreurs de syntaxe, ce qui facilite la détection précoce des problèmes et améliore la qualité du code.
- **Portabilité du code** : En déclarant explicitement les types de données, vous garantissez que le code est portable entre différentes **plateformes et compilateurs**. Les types de données standards en Fortran ont des tailles spécifiées, ce qui assure une cohérence dans le stockage des données et facilite la portabilité du code.
- **Optimisation des performances** : La déclaration des variables avec les types de données appropriés peut permettre au compilateur d'effectuer des optimisations spécifiques. Par exemple, utiliser un type de données plus petit lorsque vous savez que les valeurs sont limitées peut économiser de la mémoire et améliorer les performances..

4.1. Les constantes et variables

comment déclarons ?

Integer :	**Nombres entiers** **• Valeurs positives ou négatives entre des limites qui dépendent de la machine**
Real :	**Nombres réels** **• Valeurs positives ou négatives, ayant une partie fractionnaire, entre** **des limites qui dépendent de la machine**
Complex :	**Nombres complexes composés d'une partie réelle et d'une partie** **imaginaire, toutes deux de type réel**
Logical	**Valeurs booléennes ou logiques ne prenant que deux valeurs: vrai** **(true) ou faux (false)**
Character :	**• Chaînes d'un ou plusieurs caractères**

```
1 program types_de_donnees
2   implicit none
3
4   ! Déclaration des variables
5   integer :: entier
6   real :: reel
7   character(len=10) :: caractere
8   complex :: complexe
9
10  ! Attribution de valeurs aux variables
```

```
  entier = 42
  reel = 3.14
  caractere = 'Hello'
  complexe = (2.0, 1.0)

  ! Affichage des valeurs
  write(*,*) "Entier :", entier
  write(*,*) "Réel :", reel
  write(*,*) "Caractère :", caractere
  write(*,*) "Complexe :", complexe

end program types_de_donnees
```

Dans ce programme, nous déclarons **quatre variables** de types différents :

- entier est une variable de type **`integer`**, qui stocke des nombres entiers.
- reel est une variable de type **`real`**, qui stocke des nombres réels.
- caractere est une variable de type **`character`**, qui stocke des chaînes de caractères. La longueur de la chaîne est spécifiée avec l***`en=10`***∗, ce qui signifie qu'elle peut contenir jusqu'à 10 caractères.
- complexe est une variable de type **`complex`**, qui stocke des nombres complexes. Dans cet exemple, nous lui attribuons la valeur (2.0, 1.0).

4.2. Les types de données paramétrés

5. Structures de contrôles

Les structures de contrôle sont des éléments essentiels de la programmation. Elles permettent de contrôler le flux d'exécution d'un programme en fonction de certaines conditions ou de répéter des blocs de code plusieurs fois.∗

Les types de sont :

1. **Les tests**
    - **Le bloc IF**
    - **Le bloc SELECT-CASE**
2. **Les itération**
    - **L'instruction GOTO**
    - **Les bloucles DO**
    - **Boucle Do while**

5.1. Les tests

Fondamental

*Il existe plusieurs types de **structures de tests** pour la prise de décision dans la programmation. Voici les principaux types de structures de tests :*∗

Structure conditionnelle simple (if) :

La structure conditionnelle simple est utilisée pour exécuter un bloc de code uniquement si une condition est vraie. Si la condition est fausse, le bloc de code est ignoré. La syntaxe générale en Fortran est la suivante :[16][1]

[1] Kerrigan James F., Migrating to Fortran 90,

```
if (condition) then
    ! Code à exécuter si la condition est vraie
end if
```

Structure conditionnelle avec alternative (if-else) :

La structure conditionnelle avec alternative permet d'exécuter un bloc de code si une condition est vraie, et un autre bloc de code si la condition est fausse. La syntaxe générale en Fortran est la suivante :

```
if (condition) then
    ! Code à exécuter si la condition est vraie
else
    ! Code à exécuter si la condition est fausse
end if
```

Structure conditionnelle avec alternatives multiples (if-else if-else) :

La structure conditionnelle avec alternatives multiples permet d'exécuter différents blocs de code en fonction de différentes conditions. On peut spécifier plusieurs conditions avec **else** if pour tester différentes alternatives. La syntaxe générale en Fortran est la suivante :

```
if (condition1) then
    ! Code à exécuter si condition1 est vraie
else if (condition2) then
    ! Code à exécuter si condition1 est fausse et condition2 est vraie
else
    ! Code à exécuter si toutes les conditions sont fausses
end if
```

Structure conditionnelle en cascade (select case) :

La structure conditionnelle en cascade permet d'exécuter différents blocs de code en fonction de la valeur d'une expression. Elle est utile lorsque vous avez un grand nombre de conditions à tester. La syntaxe générale en Fortran est la suivante :[17][1]

```
select case (expression)
    case (valeur1)
        ! Code à exécuter si l'expression est égale à valeur1
    case (valeur2)
        ! Code à exécuter si l'expression est égale à valeur2
    case default
        ! Code à exécuter si aucune des valeurs précédentes ne correspond
end select
```

5.2. Les itération

Fondamental

Les itérations, également appelées **boucles**, sont des structures de contrôle utilisées pour répéter un bloc de code plusieurs fois. Elles permettent d'exécuter une série d'instructions de manière répétée jusqu'à ce qu'une condition spécifiée soit satisfaite. Voici les principaux types d'itérations :

Boucle for :

La boucle for est utilisée pour répéter un bloc de code un nombre fixe de fois. La syntaxe générale en Fortran est la suivante :

```
do variable = début, fin, pas
    ! Code à exécuter à chaque itération
end do
```

[1] Kerrigan James F., Migrating to Fortran 90,

Boucle do while :

La boucle **do while** permet de répéter un bloc de code tant qu'une **condition donnée** est vraie. La syntaxe générale en Fortran est la suivante :

```
do while (condition)
    ! Code à exécuter tant que la condition est vraie
end do
```

6. Les opérateurs

6.1. Opérateurs arithmétiques

Fondamental

En Fortran, vous disposez d'un ensemble ***d'opérateurs arithmétiques*** *pour effectuer des calculs mathématiques*∗. Voici les opérateurs arithmétiques couramment utilisés : 18[1]

Opérateurs de base :

- **+ : Addition**
- **- : Soustraction**
- *** : Multiplication**
- **/ : Division**
- **** : Élévation à la puissance. Par exemple, x ** y élève x à la puissance y.**

Opérateurs de division modulo :

- **// : Division entière. Par exemple, x // y effectue une division entière de x par y, en renvoyant le quotient entier.**
- **mod : Modulo. Par exemple, mod(x, y) renvoie le reste de la division entière de x par y.**

Opérateurs d'incrémentation et de décrémentation :

- **= : Affectation. Par exemple, x = y affecte la valeur de y à x.**
- **+= : Incrémentation. Par exemple, x += y équivaut à x = x + y.**
- **-= : Décrémentation. Par exemple, x -= y équivaut à x = x - y**

6.2. Opérateurs relationnels

opérateurs relationnel

Fondamental

les **opérateurs relationnel**s sont utilisés pour effectuer des comparaisons entre des valeurs, renvoyant une valeur booléenne (vrai ou faux). Voici les opérateurs relationnels couramment utilisés :

Opérateurs de comparaison :

Opérateur	Opération
`== ou .EQ.`	**x == y renvoie vrai si x est égal à y.**
`/=`	**x /= y renvoie vrai si x est différent de y.**
`< ou .LT.`	**x < y renvoie vrai si x est strictement inférieur à y.**

[1] Lignelet P., Manuel complet du langage Fortran

`>ou .GT.`	**x > y renvoie vrai si x est strictement supérieur à y.**
`<=ou .LE.`	**x <= y renvoie vrai si x est inférieur ou égal à y.**
`>= ou .GE.`	**x >= y renvoie vrai si x est supérieur ou égal à y**

Opérateurs logiques :

- **.and. : ET logique. Par exemple, x > 0 .and. y < 10 renvoie vrai si x est strictement positif et y est strictement inférieur à 10.**
- **.or. : OU logique. Par exemple, x > 0 .or. y < 10 renvoie vrai si x est strictement positif ou y est strictement inférieur à 10.**
- **.not. : NON logique. Par exemple, .not. (x > 0) renvoie vrai si x est inférieur ou égal à zéro.**

6.3. Fonctions Mathématiques

Les differnts Fonctions Mathématiques *Fondamental*

Le langage Fortran offre un large éventail de fonctions mathématiques intégrées pour effectuer des opérations mathématiques courantes. Voici quelques-unes des fonctions mathématiques les plus utilisées en Fortran :

Il existe de nombreuses autres fonctions mathématiques disponibles en Fortran. Vous pouvez consulter la documentation de votre compilateur Fortran spécifique pour obtenir une liste complète des fonctions mathématiques prises en charge et des détails sur leur utilisation spécifique

ABS(X)	Valeur absolue d'un nombre entier, réel ou complexe
ASIN(X)	Arc sinus de x
ACOS(X)	Arc cosinus de x
ATAN(X)	Arc tangente de x dans l'intervalle de -π/2 a π/2
ATAN2(Y, X)	Arc tangent de y/x dans l'intervalle de -π a π
COS(X)	Cosinus d'un nombre réel ou complexe, x
COSH(X)	Cosinus hyperbolique de x.
COT(X)	Cotangent de x.
EXP(X)	Valeur de la fonction exponentielle de x
INT(X)	Conversion d'un entier, réel ou complexe, x en entier : exemple : INT(3.9) donne 3, INT(- 3.9) donne - 3.

LOG(X)	Logarithme népérien d'un nombre réel ou complexe, x. Notez qu'un argument entier posera de problème avec Fortran.
LOG10(X)	Logarithme népérien à base de 10 de x.
MAX(X1, X2)	Maximum de deux nombres entiers ou réels
MIN(X1, X2)	Minimum de deux nombres entiers ou réels
MOD(K, L)	Reste de la division de K sur L. les arguments doivent être entiers tous les deux ou réels tous les deux.
NINT(X)	Plus proche de x, e.g. NINT(3.9) donne 4, tandis que NINT(-3.9) donne -4.
REAL(X)	La fonction REAL convertis un entier, réel ou complexe x en réel, e.g. REAL(2)/4 donne 0.5, tandis que REAL(2/4) donne 0.0.
SIN(X)	Sinus d'un nombre réel ou complexe, x
SINH(X)	Sinus hyperbolique de X.
SQRT(X)	Racine carrée d'un nombre réel ou complexe x.
TAN(X)	Tangente de x.
TANH(X)	Tangente hyperbolique de x.
DEG(X) :	Conversion d'un angle en radians x en degrés.
RED(X)	Conversion d'un angle en degrés x en radians.

6.4. Exemple de programmation

Écrire un programme Fortran calculant la somme S : ? ***Exemple***

A savoir que F est un nombre réel et X et un entier

$$F(x) = x! - \ln(x) \; et \; S = \sum_{1}^{1000} F$$

Le programme utilise une fonction procédurale appelée **calculerSomme** pour calculer la somme **S = x! - ln(x) de 1 à 1000**. La fonction **factorial** est utilisée pour calculer la factorielle de chaque valeur de i dans la boucle do. La fonction log est utilisée pour calculer le logarithme népérien de chaque valeur de i. La somme est accumulée dans la variable **Sum** et renvoyée comme résultat de la fonction **calculerSomme**. Enfin, le résultat est affiché à l'aide de l'instruction **write**.

```
1 program somme_factorielle_ln
2   implicit none
3
4   ! Déclaration des variables
5   integer :: i, x
```

```
  real :: S

  ! Appel de la fonction pour calculer la somme
  S = calculerSomme()

  ! Affichage du résultat
  write(*,*) "La somme S = x! - ln(x) de 1 à 1000 est :", S

contains

  ! Définition de la fonction calculerSomme
  function calculerSomme() result(Sum)
    implicit none
    integer :: i
    real :: Sum

    Sum = 0.0

    do i = 1, 1000
      Sum = Sum + factorial(i) - log(i)
    end do

  end function calculerSomme

  ! Définition de la fonction factorielle
  recursive function factorial(n) result(result)
    integer, intent(in) :: n
    integer :: result

    if (n == 0) then
      result = 1
    else
      result = n * factorial(n - 1)
    end if

  end function factorial

end program somme_factorielle_ln
```

7. Exercices

7.1. Exercice

[solution n°15 p. 50]

Fortran

Quel est le mot clé utilisé en Fortran pour déclarer une variable entière ?

- ☐ Integer
- ☐ Real
- ☐ Complx

Fortran

Quelle est la syntaxe correcte pour une boucle "do" en Fortran ?

- ☐ do (i = 1, 10)
- ☐ do (i = 1 to 10)
- ☐ do i = 1, 10

Fortran

Exécuter le programme

```
program somme_deux_valeurs
    implicit none
    integer :: valeur1, valeur2, somme

    ! Saisie des deux valeurs
    write(*,*) "Entrez la première valeur : "
    read(*,*) valeur1

    write(*,*) "Entrez la deuxième valeur : "
    read(*,*) valeur2

    ! Calcul de la somme
    somme = valeur1 + valeur2

    ! Affichage du résultat
    write(*,*) "La somme de", valeur1, "et", valeur2, "est", somme

end program somme_deux_valeurs
```

Exercice programmation

Calculer l'aire du cercle pour le cas de rayon=2

```
program calcul_aire_cercle
    implicit none
    real :: rayon, aire
    real, parameter :: pi = 3.14159

    ! Saisie du rayon
    write(*,*) "Entrez le rayon du cercle : "
    read(*,*) rayon

    ! Calcul de l'aire
    aire = pi * rayon**2

    ! Affichage du résultat
    write(*,*) "L'aire du cercle de rayon", rayon, "est", aire

end program calcul_aire_cercle
```