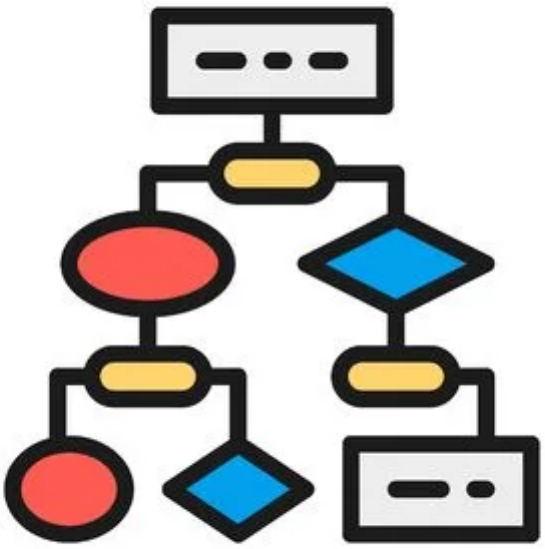
Algorithmique

Version : 1.0



Introduction

Algorithme est un mot dérivé du nom du mathématicien **Al Khwarizmi**, membre de l'académie des sciences de Bagdad, au 9ème siecle.

Un algorithme prend des **données en entrée**, **exprime un traitement** particulier et fournit des **données de sortie**.

Exemple

Des algorithme dans la vie de tous les jours

- Recette de cuisine
- Notice de montage

Notion de programme

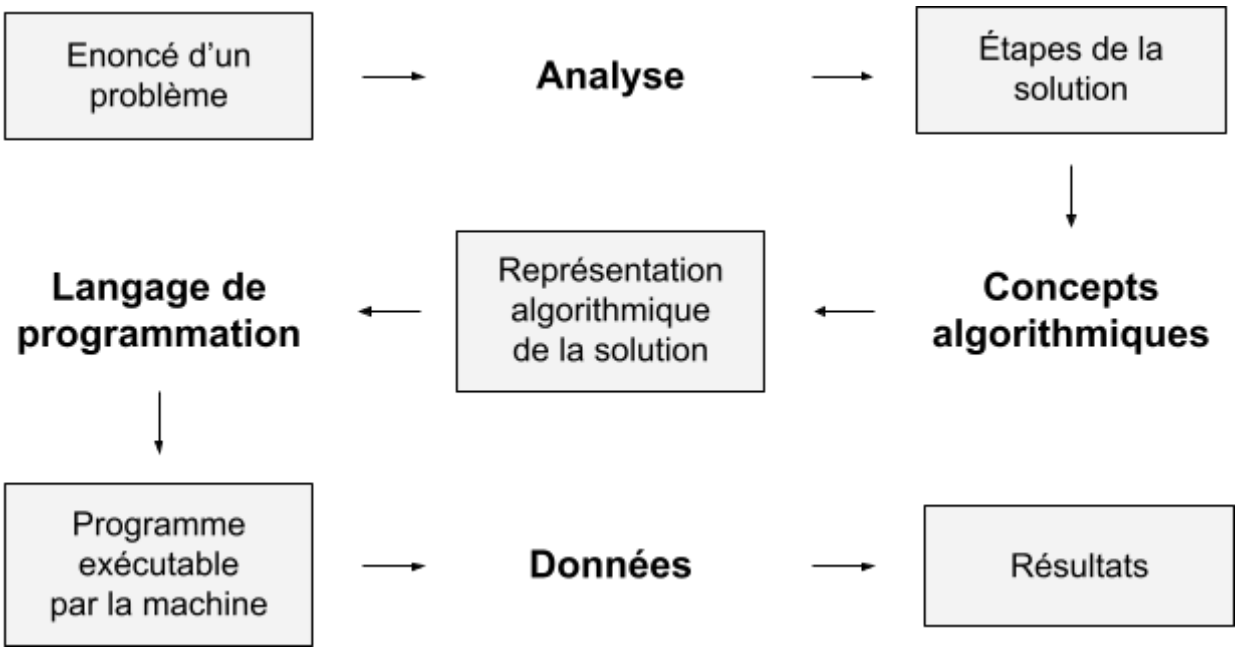
Qu'est ce que c'est ?

Un programme est une série d'instructions pouvant s'exécuter en séquence ou en parallèle qui réalise (implémente) un algorithme.

Etapes de programmation

Pour écrire un programme, il est nécessaire de toujours commencer par établir une description détaillée d'un problème donné.

Le schéma ci-dessous présente les différentes étapes nécessaires pour la réalisation d'un programme informatique exécutable par une machine en vue d'obtenir des résultats automatiques.



- **L'analyse** sert à déterminer les différentes étapes à suivre pour obtenir une solution « manuelle » qui peut être décrite par un schéma, un paragraphe, une liste des grandes lignes à suivre... Ces différentes étapes constituent les éléments de base du programme.
- Les **concepts algorithmiques** servent à donner une autre forme aux étapes de la solution obtenue dans l'étape précédente, plus proche de la forme d'un programme tout en travaillant en langage naturel.

- Le **langage de programmation** sert à traduire l'algorithme obtenu dans l'étape précédente à un programme compréhensible et exécutable par la machine.
- **L'exécution** en dernier lieu, et pour avoir des résultats, il faudrait passer les données au programme.

Si le programmeur estime que les résultats du programme sont faux, il doit revoir toutes les étapes du schéma pour détecter l'erreur..

Les concepts algorithmiques

Structure général d'un algorithme

Nom de l'algorithme	*Entête de l'algorithme*
Déclaration des constantes et variables	*Utilisées dans le bloc d'instruction*
Début	*Début du corps de l'algorithme*
Suite d'instructions	
Fin	*Fin du corps de l'algorithme*

Entête de l'algorithme

L'entête de l'algorithme se compose d'un mot clé permettant de déclarer l'algorithme, suivi du nom de l'algorithme.

Selon le langage et le contexte, le mot clé peut être **Algo**, **function**, **class** ….

Le nom de l'algorithme est une chaîne de caractères attribuée par le programmeur, celle-ci peut être significative ou non.

Exemple

```
class Employe;
function addition;
```

Note

Il est préférable que le nom de l'algorithme soit le plus significatif possible, c'est à dire qu'il reflète l'objectif de l'algorithme, cela permettra une meilleure distinction entre les différents algorithmes.

Déclaration des variables et constantes

Dans un algorithme nous manipulons des données stockées dans des variables ou des constantes.

La différence entre les deux est que les variables peuvent changer de valeur pendant l'exécution du programme alors que les constantes conservent la même valeur tout au long de l'exécution.

Déclarer des variables ou des constantes sous-entend de leurs identificateur - leurs noms - dans le bloc d'instructions.

Lors de la déclaration, on réserve des espaces mémoire sur la machine qui exécute le programme. Ces espaces mémoires serviront à stocker les valeurs des variables et constantes déclarés.

Chaque espace mémoire est identifié par le nom de la variable ou de la constante.

Déclarer une variable

Selon le langage, la déclaration d'une variable commence par un mot clé ou un symbole, suivi du nom permettant d'identifier la variable.

Toujours selon le langage, nous devons ajouter le typage, parfois facultatif, avant ou après le nom de la variable.

Exemple JavaScript

```
var variableA;
let variableB;
```

Exemple PHP

```
$variableA;
$variableB;
```

Exemple TypeScript

```
variableA:Number;
variableB:String;
```

Déclarer une constante

La déclaration d'une constante se fait de la même façon que la déclaration d'une variable mais en utilisant la mot clé **const**.

Exemple

```
const PI = 3.141592653;
const EUR = "Euro";
```

Affectation de valeur

Selon le langage, nous affectons une valeur à une variable ou à une constante avec le symbole **égale "="** ou **deux points ":"**.

Exemple JavaScript

```
var variableA = 42;
var variableB = "Hello There !";
```

Exemple PHP

```
$variableA = 42;
$variableB = "Hello There !";
```

Exemple TypeScript

```
variableA:Number = 42;
variableB:String = "Hello There !";
```

Note

La déclaration d'une variable ou d'une constante est une instruction du programme, il est donc nécessaire de terminer chaque déclaration par un point-virgule.

La déclaration sans affectation de valeur revient à affecter la valeur NULL à la variable.

Il est préférable de nommer les variables et constantes avec des noms significatifs.

Nommer toujours vos constantes en majuscule.

Bloc d'instructions

Le bloc d'instructions constitue le corps de l'algorithme. Il contient les instructions nécessaire à la résolution du problème.

L'ordre des instructions doit être bien étudié pour que l'algorithme donne le résultat attendu.

Les instructions sont séparées par un point-virgule.

Problèmes fondamentaux en algorithmie

Complexité

En combien de temps un algorithme va-t-il atteindre le résultat escompté ?

De quel espace mémoire a-t-il besoin ?

Calculabilité

Existe t'il des tâches pour lesquelles il n'existe aucun algorithme ?

Etant donnée une tâche, peut-on dire s'il existe un algorithme qui la résolve ?

Correction

Peut on être sûr qu'un algorithme réponde au problème pour lequel il à été conçu ?

Ecrire un algorithme

Phase d'analyse

La phase d'analyse consiste à extraire de l'énoncé du problème tous les éléments permettant la modélisation de l'algorithme.

Ont peut identifier ces éléments avec les questions suivante :

1. **Quel est le but du programme ?**

   Il est important de bien comprendre ce que doit faire notre algorithme.

   Afin de le rendre le plus simple et compréhensible au possible.

2. **Quelles sont les données d'entrée du problème ?**

   Pour atteindre son objectif, l'algorithme a-t-il besoin d'informations externes ? Si oui, identifier s'il s'agit de données **constantes** ou **variables**.

3. **Que doit-on obtenir comme résultat en sortie ?**

   Il est important de bien comprendre ce que notre algorithme doit retourner comme résultat.

Enoncé d'un problème

Enoncé du problème

Prenons en considération l'énoncé du problème suivant :

> On souhaite calculer et afficher le montant de la TVA et le prix TTC à partir d'un prix hors taxe et d'un taux de TVA de 20%.

Analyse du problème

1. **Quel est le but du programme ?**

> On souhaite **calculer et afficher le montant de la TVA et le prix TTC** à partir d'un prix hors taxe et d'un taux de TVA de 20%

2. **Quelles sont les données d'entrée du problème ?**

Le montant TTC dépend :

On souhaite calculer et afficher le montant de la TVA et le prix TTC à partir d'un **prix hors taxe** et d'un **taux de TVA de 20%**

- Du prix Hors taxe
- Du taux de TVA de 20%

3. **Que doit-on obtenir comme résultat en sortie ?**

On souhaite calculer et afficher le **montant de la TVA** et le **prix TTC** à partir d'un prix hors taxe et d'un taux de TVA de 20%

Ecriture de l'algorithme

Nom	*CalculTVA*
Déclaration	*TVA ← 20;* *montantTVA;* *prixHT;* *prixTTC;*
Début	
Instructions	***Affiche*** *Saisir le prix Hors Taxe ?;* ***Saisir*** *prixHT;* *montantTVA ← prixHT x (TVA / 100);* *prixTTC ← prixHT + montantTVA;* ***Affiche*** *Montant TVA : montantTVA;* ***Affiche*** *Prix TTC : prixTTC;*
Fin	

Simulation d'algorithme

Voici un algorithme dont le but est d'échanger les valeurs de ses deux variables.

Nom	*Echange*
Déclaration	*varA;* *varB;*
Début	
Instructions	***Affiche*** *Saisir la valeur de A;* ***Saisir*** *varA;* ***Affiche*** *Saisir la valeur de B;* ***Saisir*** *varB;* ***Affiche*** *Vous m'avez donné varA et varB;* *{... ici, vous inversez les valeurs de varA et varB …}* ***Affiche*** *Maintenant, les données sont varA et varB;*
Fin	

- Inversez les valeurs de `varA` et `varB`.

Solution (1)

Nom	*Echange*
Déclaration	*varA;* *varB;*
Début	
Instructions	***Affiche*** *Saisir la valeur de A;* ***Saisir*** *varA;* ***Affiche*** *Saisir la valeur de B;* ***Saisir*** *varB;* ***Affiche*** *Vous m'avez donné varA et varB;* ***varA*** ← ***varB*** ***varB*** ← ***varA*** ***Affiche*** *Maintenant, les données sont varA et varB;*
Fin	

- Que fait l'algorithme ?

Solution (2)

Pour que l'algorithme fonctionne comme prévu, il faut ajouter une variable supplémentaire et utiliser cette variable pour y stocker provisoirement l'une de nos entrée.

Nom	*Echange*
Déclaration	*varA;* *varB;*
Début	
Instructions	***Affiche*** *Saisir la valeur de A;* ***Saisir*** *varA;* ***Affiche*** *Saisir la valeur de B;* ***Saisir*** *varB;* ***Affiche*** *Vous m'avez donné varA et varB;* ***varTemp*** ←***varA*** ***varA*** ← ***varB*** ***varB*** ← ***varTemp*** ***Affiche*** *Maintenant, les données sont varA et varB;*
Fin	

Structures algorithmiques

Structures conditionnelles

Les structures conditionnelles sont des instructions qui conditionnent l'exécution d'une portion de code au résultat d'une expression logique.

if

La structure `if` exécute un bloc de code lorsque le résultat de l'expression logique est vrai.

Syntaxe

```
si expression logique
        alors instruction;
```

Nom	*Superieur10*
Déclaration	*val;*
Début	
Instructions	***Affiche*** *Saisir une valeur :;* ***Saisir*** *val;* ***si*** *val > 10;* ***alors affiche*** *val est supérieur à 10;*
Fin	

if ... else

La structure `if ... else` exécute un bloc de code lorsque le résultat de l'expression logique est vrai.

Si le résultat de l'expression logique est faux, alors la structure `if ... else` exécutera du code alternatif.

Syntaxe

```
si expression logique
        alors instruction;
sinon instruction alternatives;
```

Nom	*Superieur10*
Déclaration	*val;*
Début	
Instructions	***Affiche*** *Saisir une valeur :;* ***Saisir*** *val;* ***si*** *val > 10;* ***alors affiche*** *val est supérieur à 10;* ***sinon affiche*** *val n'est pas supérieur à 10;*
Fin	

if ... else if ...

La structure `if ... else if ...` exécute un bloc de code lorsque le résultat d'une première expression logique est **vrai**.

Si le résultat de la première expression logique est **faux**, alors la structure `if ... else if ...` test une seconde expression logique. Si le résultat de cette seconde expression logique est **vrai**, la structure exécute un autre bloc de code.

Si aucune expression logique n'est vrai, aucun code ne sera exécuté.

Syntaxe

```
si expression logique
        alors instruction;
sinon si expression logique
        alors instruction;
```

Nom	*SuperieurOuEgal10*
Déclaration	*val;*
Début	
Instructions	***Affiche*** *Saisir une valeur :;* ***Saisir*** *val;* ***si*** *val > 10;* ***alors affiche*** *val est supérieur à 10;* ***sinon si*** *val == 10;* ***alors affiche*** *val vaux exactement 10;*
Fin	

if ... else if ... else

La structure `if ... else if ... else` fonctionne de la meme façon que la structure précédente, mais peut exécuter un bloc de s-code dans le cas ou aucune des expression logique n'est satisfaite.

Syntaxe

```
si expression logique
        alors instruction;
sinon si expression logique
        alors instruction;
sinon instruction alternatives;
```

Nom	*SuperieurOuEgal10*
Déclaration	*val;*
Début	
Instructions	***Affiche*** *Saisir une valeur :;* ***Saisir*** *val;* ***si*** *val > 10;* ***alors affiche*** *val est supérieur à 10;* ***sinon si*** *val == 10;* ***alors affiche*** *val vaux exactement 10;* ***sinon affiche*** *val est inférieur à 10;*
Fin	

Commutateur

switch

Le commutateur `switch` peut être une structure conditionnelle alternative de la structure `if ... else if ... (else)`, à la différence que l'exécution du commutateur sera plus rapide.

Syntaxe

```
selon identificateur
        cas valeur: instruction;
        cas valeur: instruction;
        cas valeur: instruction;
        autre instruction;
```

Nom	*Civilité*
Déclaration	*abreviation;*
Début	
Instructions	***selon*** *abreviation;* ***cas*** *'M': **affiche** "Monsieur";* ***cas*** *'Mme': **affiche** "Madame";* ***cas*** *'Mlle': **affiche** "Mademoiselle";* ***autre****: **affiche** "Madame, Monsieur";*
Fin	

Boucles

Les boucles permettent d'automatiser la répétition de l'exécution d'instructions.

for

On utilise une boucle de type for pour répéter une suite d'instruction un certain nombre de fois.

Syntaxe

```
pour var ← valeur initiale; valeur de fin; var++
        instructions
```

Nom	*CompteJusque10*
Déclaration	
Début	
Instructions	**pour** *i ← 0*; *i < 10*; *i++* **affiche** *instructions;*
Fin	

while

Syntaxe

```
tant que expression logique
        faire instructions
```

Nom	
Déclaration	
Début	
Instructions	
Fin	

do... while

Syntaxe

faire *instructions*
tant que *expression logique*