Algorithme

**Données** : Variable, Constante, Structure itérative, boucle, condition, fonction

**Définition** : Un algorithme est une suite d’instruction permettant de résoudre un ou plusieurs problèmes, qui une fois exécutée correctement conduit à un résultat donné.

Cela implique qu’un algorithme doit comporter uniquement des instructions compréhensibles par la personne qui lit les instructions.

Algorithme = conception, code = implémentation

Opération de base d’un ordinateur : Affectation de variable, test (condition), écriture/lecture, boucle

**Les variables :** une constante est une variable uniquement accessible en lecture, une variable est un emplacement mémoire accessible en lecture écriture

Une variable possède un nom et parfois un type de données

Lil endian : lecture de gauche à droite

Big endian : lecture de droite à gauche

On peut utiliser le dernier bit pour signer (positif/négatif)

1 : négatif 0 : positif

Variable « nom\_de\_variable » : Type = « Valeur » ;

* Number : Tous les nombres
* String : Chaine de caractère
* Array
* List
* Object
* Boolean

Exercice 1 : Déclaration de Variable

Variable toto : String = « Hello World »;

Variable a:Number=10;

Variable b:Number=0,5;

Variable c:Number=a\*b;

Variable classroom:Array=[Marcelo, Régis, Nicolas, Jêrome, Nam]

**Opérateur :**

= Assignation

== Test

+ Addition / Concaténation

-Soustraction

\* Multiplication

/ Division

% Modulo

&&/and et

||/or ou

^/xor ou exclusif

!/not négation

< décalage

-Lire et stocker dans « nom\_de\_la\_variable »

-Ecrire/Afficher valeur/nom\_de\_la\_variable

Si( conditions ) alors

Action

Sinon si (condition) alors

Action

Fin si (optionnel)

Exercice 2 : créer une variable contenant un tableau, ce tableau devra contenir l’ensemble des personnes qui se trouvent dans la salle, si la longueur du tableau est supérieur à 10, alors on écrit « c’est une grande classe » si la longueur du tableau est supérieur à 5 on affiche « c’est une petite classe »

Var classroom :Array=[eleve1, eleve2, eleve3]

Si(classroom.length > 10)

Ecrire “c’est une grande classe”

Sinon si(classroom.length < 5)

Ecrire « C’est une petite classe

Fin si

exercice: "Ecrire un algorithme qui demande un nombre à l’utilisateur, et l’informe ensuite si ce nombre est positif ou négatif (on laisse de côté le cas où le nombre vaut zéro)."

Var nombre :Number = 0

Afficher « veuillez saisir une valeur »

Nombre = Lire valeur entrée

Si(nombre > 0)

Afficher « Ce nombre est positif »

Sinon

Afficher « Ce nombre est négatif »

Condition :

Variable student :array[« eleve1,eleve2,eleve3 »]

Variable i:Number = 0

Constant max:Number=length(student)

Pour (i=0; i<max,i++)

Pour ( initialisation, tant que condition(s), action(s) en fin de de tour de boucle )

jeu d'instructions ...

Fin Pour

Exemple:

// algorithme permettant d'afficher les chiffres

// de 0 à 9

Pour( i = 0; i < 10; i++ )

Afficher i

Fin Pour

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Var nombre :Number = 0

Var i :Number = 0

Afficher « veuillez saisir une valeur »

Nombre = Lire valeur entrée

Pour(i=0 ; i<10 ;i++)

Nombre = Nombre + 1

Afficher Nombre

Fin pour

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :

Var nombre :Number = 0

Var i :Number = 0

Var resultat:Number = 0

Afficher « veuillez saisir une valeur »

nombre = Lire valeur entrée

Afficher Table de « Nombre »

Pour(i=1 ; i <10 ;i++)

Resultat = Nombre \* i

Afficher  Nombre +  « \* » + i + « = » + resultat

Fin pour

Tant que (condition)

Jeu d’instruction

Fin Tant que

Demander un nombre à l'utilisateur Calculer son carré et le stocker quelque part Tant que sa valeur n'excède pas son carré, multiplier le nombre par 2(modifié)

Vous êtes libre d'afficher ou non les valeurs intermédiaires mais il faut à tout prix afficher la valeur finale

Var nombre :Number = 0

Var resultatcarre:Number = 0

Afficher « veuillez saisir une valeur »

nombre = Lire valeur entrée

resultatcarre = nombre \* nombre

Tant que (nombre<=resultatcarre)

nombre = nombre \* 2

Fin Tant que

Afficher nombre

**Les tableaux :**

Les tableaux sont « 0-based », le premier index d’un tableau est 0

Tableau à 2 dimensions

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 |

Notation :

Var Tab : Array = [ ]

Fonction tableau :

Push ( 1 parametre / Valeur )

Tab.push(10) -> Tab[10]

Tab.Length -> Renvoie la longueur du tableau

Exercice : - Créer un tableau possédant une série de 10 notes notées sur 20 en demandant chacune des notes à l'utilisateur - Une fois le tableau crée, calculer la moyenne et l'afficher

Var Tab : Array = [ ]

Var i :Number = 0

Var moyenne :Number = 0

Var valeursaisie:Number = 0

Afficher « veuillez rentrer les 10 notes »

For (i=0, i < 10, i++)

Valeursaisie = Lire valeur entrée

Tant que(valeursaisie > 20 || valeur saisie < 0)

Afficher « veuillez saisir une note comprise entre 0 et 20 »

Valeursaisie = Lire valeur entrée

Fin Tant que

Tab.push(valeursaisie)

Fin For

For (i = 0 , i < 10, i++)

Moyenne = Moyenne + Tab[i]

Fin For

Moyenne = Moyenne / 10

Afficher Moyenne

Exercice Ecrire un algorithme permettant de trier les valeurs dans le tableau suivant par ordre croissant ( du plus petit au plus grand ) : [15,10,12,13,80,100,60]

Var Tab : Array = [ 15,10,12,13,80,100,60]

Var i:Number = 0

Var j:Number=0

Var tampon:Number= 0

Tant que (i != 1)

i = 1

For(j=0; j < tab.length - 1; j++)

si (tab[j] > tab[j+1])

tampon = tab[j]

tab[j] = tab[j + 1]

tab[j + 1] = tampon

i = 0

fin si

fin for

fin tant que

Afficher « tableau trié » tab

Exercice - Soit le tableau à 2 dimensions suivant: [ [20,15,7,8,5], [10,7,8,12,3], [18,2,3,17,10] ] - Chaque élément du tableau représente un tableau de notes ( / 20 ) - Créer un algorithme permettant de : - Calculer l'ensemble des moyennes - Stocker chaque moyenne dans un tableau - Ordonner le tableau - Afficher les moyennes supérieures à 10

Var Tab : Array = [ [20,15,7,8,5], [10,7,8,12,3], [18,2,3,17,10]]

Var TabMoyenne:Array = []

Var Moyenne:Number = 0

Var i :Number = 0

Var j : Number = 0

Var tampon:Number= 0

For (i = 0 , i < 3, i++)

For(j=0, j <5, j++)

Moyenne = Moyenne + Tab[i][j]

Fin for

Moyenne = Moyenne / 5

TabMoyenne.push(Moyenne)

Moyenne = 0

Fin For

i = 0

Tant que (i != 1)

i = 1

For(j=0; j < tabMoyenne.length - 1; j++)

si (tabMoyenne [j] < tabMoyenne [j+1])

tampon = tabMoyenne [j]

tabMoyenne [j] = tabMoyenne [j + 1]

tabMoyenne [j + 1] = tampon

i = 0

fin si

fin for

fin tant que

For(j=0 ; j < tabMoyenne.length, j++)

Si (TabMoyenne[j] > 10)

Afficher tabmoyenne[j]

Fin si

Fin for

\*\* Le jeu du + ou - **\***

Objectif: Créer un algorithme permettant de faire tourner un jeu selon les règles suivantes:

- On demande à un utilisateur ( nommé Yoda par exemple ) de choisir un nombre entre 0 et 1000 compris ( ensemble de type -> [0-1000] ).

- Puis faire jouer un second utilisateur ( nommé Luke ) en lui demandant de taper un nombre entre 0 et 1000.

- Si Luke trouve le nombre donné par Yoda, alors il gagne - Si Luke donne un nombre supérieur à celui donnée par Yoda alors, on indique au joueur que le nombre recherché est "inférieur" et inversement.

- Luke possède un nombre de tentatives limité ( on va dire 12 ) - Indiquer à la fin du jeu qui a gagné ( Yoda ou Luke )

Var ValeurSolution : Number = 0

Var NombreEssai :Nombre = 12

Var ValeurProposee :Number = 0

Var i :Number = 0

Afficher Le joueur 1 doit rentrer une valeur compris entre 0 et 1000

ValeurSolution = Valeur saisie

Tant Que (ValeurSolution < 0 || ValeurSolution > 1000)

Afficher Veuillez saisir une valeur compris entre 0 et 1000

ValeurSolution = Valeur saisie

Fin Tant Que

Afficher Le joueur 2 doit trouver le nombre saisi par le joueur 1, il a pour cela 12 essais

For (i=0 ; i < 12 ; i++)

Afficher « Veuillez saisir votre proposition » + (i+1)

ValeurProposee = Lire Valeur

Si (ValeurProposee == ValeurSolution)

Afficher Bravo Joueur 2 Tu as trouvé c’est gagné, c’est gagné

i = 42

Fin Si

Si((i+1) == 12 && ValeurProposee != ValeurSolution)

Afficher Perdu Joueur 2, t’es pas ouf, retente ta chance

Fin Si

Si((i+1) < 12 && ValeurProposee > ValeurSolution)

Afficher C’est moins mon coco !

Fin Si

Si((i+1) < 12 && ValeurProposee < ValeurSolution)

Afficher C’est plus mon coco !

Fin Si

Fin For

Exercice des grenouilles

Haskell est un fermier, toute l'année il s'occupe de ses champs et des propriétés terriennes qui comportement notamment 8 mares.

Haskell entretien de bonnes relations avec un groupe de 53 grenouilles. Il les aime tellement, que tous les étés, il les emmènent passer des vacances dans ses mares.

Mais les grenouilles sont un peu capricieuses, primo elles refusent de voyager accompagnées, Haskell fait donc un aller / retour pour déposer chaque grenouille.

Mais les exigences batraciennes ne s'arrêtent pas là, en effet, chaque grenouille demandera systématiquement à être déposée dans la mare la moins peuplée, la plus proche, la priorité étant donnée au critère de population.

Exemple: entre une mare à 50km et 8 grenouilles et une mare à 60km et 5 grenouilles, la grenouille choisira en priorité la moins peuplée.

Les mares sont situées à distances respectives de la ferme à: - 60km, 40km, 10km, 20km, 30km, 80km, 60km, 17km.

- Créer un algorithme permettant de répartir les grenouilles de façon à respecter leurs critères. Puis afficher le nombre de km total parcouru par Haskell.

Var listegrenouille :Array[Grenouille1, … ,Grenouille53]

Var ListeMare:Array[[60],[40],[10],[20],[30],[80],[60],[17]]

Var DistanceParcourue:Number = 0

Var i:Number=0

Var j:Number=0

Var tampon:Number=0

Tant que (j != 1)

j = 1

For(i=0; i < ListeMare.length - 1; i++)

si (ListeMare [i][0] < ListeMare [i+1][0])

tampon = ListeMare [j][0]

ListeMare [j][0] = ListeMare [j + 1][0]

ListeMare [j + 1][0] = tampon

j = 0

fin si

fin for

fin tant que

For(i=0; i < listegrenouille.length; i++)

For (j=0; j < 8 ; j++)

ListeMare[j].push(listegrenouille[i])

DistanceParcourue = DistanceParcourue + (ListeMare[j][0] \* 2)

si (i >= 53 )

break;

Fin si

Fin For

Fin For

Afficher DistanceParcourue

Complexité

A \* : Ne renvoie pas systementiquement le chemin le plus court

djikstra : Renvoie systematiquement le chemin le plus court

Complexité : nombre d’opération nécessaire à l’algorithme, plus il est elevé, plus le programme est complexe

Efficacité : L’efficacité d’un algorithme se mesure à sa capacité à résoudre plus ou moins bien le probleme donné