ALGO

LES BASES

CONCEPT IMPORTANTS

Algorithme : mot dérivé du nom du mathématicien al_Khwarizmi qui a vécu au 9ème siécle, était membre d'un académie des sciences à Bagdad .

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

Programme : série d'instructions pouvant s'exécuter en séquence, ou en parallèle (parallélisme matériel) qui réalise (implémente) un algorithm

POURQUOI UN COURS D'ALGO

Pour obtenir de la «machine» qu'elle effectue un travail à notre place

Problème: expliquer à la «machine» comment elle doit s'y prendre

Besoins :

- savoir expliciter son raisonnement
- savoir formaliser son raisonnement
- concevoir (et écrire) des algorithmes:
- séquence d'instructions qui décrit comment résoudre un problème particulier

ALGORITHME

Savoir expliquer comment faire un travail sans la moindre ambiguïté

Langage simple : des instructions (pas élémentaires)

Suite finie d'actions à entreprendre en respectant une chronologie imposée

L'écriture algorithmique : un travail de programmation à visée universelle

Un algorithme ne dépend pas du langage dans lequel il est implanté,

Ni de la machine qui exécutera le programme correspondant.

EXEMPLE COMMUN

On retrouve l'algorithmie dans la vie de tous les jours :

- Une recette de cuisine est un algo
- Une notice de montage de meuble est un algo
- Un tutoriel sur Internet est un algo

Point de départ :

Ingrédients / matériaux (viariables)

Debut de l'algo :

-Mélanger / monter (instructions simples et par étape)

Fin de l'algo :

Un gâteau / un meuble (Un résultat, une variable, une chaine de caractère, un tableau...)

MÉTHODOLOGIE ET OUTILS

Nous allons voir comment réaliser des algorithmes. Il existe 2 façons de les réaliser

- forme littérale avec un langage algorithmique ou pseudo-code (que nous verrons juste après)
- L'organigramme : Représenté graphiquement

Il existe un outil assez basique mais permettant de prendre la main sur les algos

https://www.xm1math.net/algobox/download.html

EXEMPLE LANGAGE ALGORITHMIQUE

```
Algorithme ElèveAuCarré
{Cet algorithme calcule le carré du nombre que lui fournit l'utilisateur}

variables unNombre, sonCarré: entiers          {déclarations: réservation
                                                d'espace-mémoire}
début
                                               {préparation du traitement}
    afficher("Quel nombre voulez-vous élever au carré?")
    saisir(unNombre)
                                               {traitement : calcul du carré}
    sonCarré ← unNombre × unNombre
                                               {présentation du résultat}
    afficher("Le carré de ", unNombre)
    afficher("c'est ", sonCarré)
fin
```

ETAPES ALGO

Préparation du traitement

- données nécessaires à la résolution du problème

Traitement

- résolution pas à pas,
- après décomposition en sous-problèmes si nécessaire

Edition des résultats

- impression à l'écran,
- dans un fichier, etc.

LANGAGE ALGO

```
Algorithme NomAlgorithme
{ ceci est un commentaire}
Début
... Actions
Fin
```

```
Algorithme Bonjour
{il dit juste bonjour mais ... en anglais !
Début
    afficher('Hello world !!!')
Fin
```

- Il faut avoir une écriture rigoureuse
- Il faut avoir une écriture soignée : respecter l'indentation
- Il est nécessaire de commenter les algorithmes
- Il existe plusieurs solutions algorithmiques à un problème posé
- Il faut rechercher l'efficacité de ce que l'on écrit

DÉCLARATION DES DONNÉES

Variable <nom de donnée>: type

- Instruction permettant de réserver de l'espace mémoire pour stocker des données
- Dépendant du type des données : entiers, réels, caractères, etc.)
- Exemples :

Variables val, unNombre: entiers

nom, prénom : chaînes de caractères

A savoir

Dans les langages comme PHP ou javascript (faiblement typé), ce n'est pas obligatoire de typer les variables.

Dans d'autres langages comme Java (fortement typé), c'est obligatoire.

CONSTANTE

Constante <nom de donnée>: type - valeur ou expression

- Instruction permettant de réserver de l'espace mémoire pour stocker une constante dont la valeur ne varie pas.

Exemples :

```
Constantes  MAX : entier <- 10
            DEUXFOISMAX : entier <- MAX x 2
```

Lors d'une affectation, la variable doit être à gauche et la valeur à affecter à droite

TYPE VARIABLE

Booléen : Le booléen est le plus petit type de variable que vous serez emmené à rencontrer, il est codé sur seulement 2 bits. Son contenu ne peut être que 0 ou 1, soit Vraiou Faux.

L'octet est un type de variable codé sur 8 bits. Son contenu est un nombre pouvant aller de 0 à 255.

L'entier est codé sur 16 bits. Son contenu est un nombre pouvant aller de -2^15 à 2^15 soit de -32768 à 32767.

Le flottant est codé sur 4 octets soit 32 bits. Son contenu est un nombre pouvant aller de 3.4028235E+38 à -3.4028235E+38.

Le caractère est un type de variable codé sur 8 bits. Son contenu est un caractère, codé avec son code décimal ASCII correspondant.

LECTURE ÉCRITURE DE DONNÉES

Saisir <nom de donnée, ...>
Afficher <nom de donnée, ...>

Fonction : Instructions permettant :

- De placer en mémoire les informations fournies par l'utilisateur.
- De visualiser des données placées en mémoire

Saisir et afficher sont considérés comme des fonctions prédéfinies, il n'y a pas besoin d'écrire la logique interne de ces fonctions.

Exemples:

```
Saisir(unNombre)
Afficher (« le nom est « , nom, »et le prénom est » ,prénom )
Saisir(val)
```

PHASE D'ANALYSE

Consiste à extraire de l'énoncé du problème des éléments de modélisation

Technique : Distinguer en soulignant de différentes couleurs :

- Quel est le but du programme (traitement à réaliser)
- Données en entrée du problème :
- Où vont se situer les résultats en sortie

EXEMPLE

On souhaite calculer et afficher , à partir d'un prix hors taxe saisi, la TVA ainsi que le prix TTC

Le montant TTC dépend de :

- Du prix HT
- Du taux de TVA de 21%

Traitement à réaliser

EXEMPLE

On souhaite calculer et afficher , à partir d'un prix hors taxe saisi, la TVA ainsi que le prix TTC

Le montant TTC dépend de :

- Du prix HT
- Du taux de TVA de 21%

Données en entrée

EXEMPLE

On souhaite calculer et afficher , à partir d'un prix hors taxe saisi, la TVA ainsi que le prix TTC

Le montant TTC dépend de :

- Du prix HT
- Du taux de TVA de 21%

Données en sortie

ALGO TVA

```
Algorithme : CalculTVA

{Saisit un prix HT et affiche le prix TTC correspondant}

Constante (TVA : réel) <- 21

Variables prixHT : réel
          prixTTC, montantTVA : réels {déclarations}

Début {préparation du traitement}

   afficher("Donnez-moi le prix hors taxe :")
   saisir(prixHT)
   prixTTC <- prixHT* (1+TVA/100) {calcul du prix TTC}
   montantTVA <- prixTTC - prixHT

   afficher(prixHT, «euros H.T. + TVA ",TVA, « devient » ,prixTTC,«eurosT.T.C.")

Fin
```

EXERCICE

```
Constante (SEUIL : réel) <- 13.25

Variables valA, valB: réels
          compteur : entier
          mot , tom : chaînes
valA <- 0.56
valB <- valA
valA <- valA×(10.5 + SEUIL)
compteur <- 1
compteur <- compteur + 10
mot <- " Bonjour "
tom <- "Au revoir ! "
mot <- 18
```

Quelles sont les différentes valeurs des variables ?

EXERCICE

Réaliser dans algobox, un algo permettant à partir de deux chiffres saisis afficher le resultat de la
multiplication

(voir : 1-multiplication.alg)

SIMULATION

```
Algorithme CaDoitEchanger?
{Cet algorithme ....................................}

Variables valA, valB: réels {déclarations}

Début {préparation du traitement}

Afficher ("Donnez-moi deux valeurs :")
Saisir (valA, valB)
Afficher ("Vous m'avez donné ", valA, " et ", valB)
{traitement mystère}
valA <- valB
valB <- valA {présentation du résultat}
Afficher("Maintenant , mes données sont : ", valA, " et ", valB)

Fin
```

Que fait cet algorithme ? Pas ce qui est prévu !

CE QU'IL MANQUE

Déclarer une variable supplémentaire

Variables valA, valB, valTemp: entiers

Utiliser cette variable pour stocker provisoirement une des valeurs

Saisir(valA, valB)
valTemp <- valA
valA <- valB
valB <- valTemp

Il s'agit d'une erreur assez courante en programmation. Le but de l'algorithme c'est de poser sa logique en instruction informatique.

En fonction du langage de programmation, la logique restera toujours la même (php, js, java, etc) ce qui changera c'est simplement «l'écriture»

STRUCTURE SI ALORS SINON FSI

```
Algorithme SimpleOuDouble

{Cet algorithme saisit une valeur entière et affiche son double si
cette donnée est inférieure à un seuil donné.}

constante (SEUIL : entier) <- 10
Variable val : entier

début
  Afficher("Donnez-moi un entier : ") { saisie de la valeur entière}
  Saisir(val)
  si val < SEUIL { comparaison avec le seuil}
    alors Afficher ("Voici son double :" , val ×2)
   sinon Afficher ("Voici la valeur inchangée :" , val)
  fsi
fin
```

STRUCTURE SI ALORS SINON FSI

Ou instruction conditionnelle

```
si <expression logique>
    alors instructions
    [sinon instructions]
fsi
```

Si l'expression logique (la condition) prend la valeur vrai, le premier bloc d'instructions est exécuté;

Si elle prend la valeur faux, le second bloc est exécuté (s'il est présent, sinon, rien).

STRUCTURE SI ALORS SINON FSI

Autre écriture de l'exemple :

```
Algorithme SimpleOuDouble
{Cet algorithme saisit une valeur entière et affiche son double si cette
donnée est inférieure à un seuil donné.}

constante (SEUIL : entier) <-10
Variable val : entier

début
    Afficher("Donnez-moi un entier : ") { saisie de la valeur entière}
    Saisir(val)
    si val < SEUIL { comparaison avec le seuil}
        alors val <- val ×2
    Fsi
    Afficher ("Voici la valeur val :" , val)
fin
```

STRUCTURE ALTERNATIVES IMBRIQUÉES

Problème: afficher :

- "Reçu avec mention Assez Bien " si une note est supérieure ou égale à 12,
- " Reçu mention Passable" si elle est supérieure à 10 et inférieure à 12, et
- "Insuffisant" dans tous les autres cas.

```
si note >= 12
    alors afficher( "Reçu avec mention AB" )
    sinon si note >= 10
              alors afficher( « Reçu mention Passable" )
              sinon afficher("Insuffisant" )
          fsi
fsi
```

SELECTION CHOIX MULTIPLES «SELON»

```
selon <identificateur>
      (liste de) valeur(s) : instructions
      (liste de) valeur(s) : instructions
      ...
      [autres: instructions]
```

S'il y a plus de deux choix possibles, l'instruction selon permet une facilité d'écriture

SELECTION CHOIX MULTIPLES «SELON»

```
selon abréviation
      "M" : afficher( " Monsieur " )
      "Mme" : afficher( " Madame " )
      "Mlle" : afficher( " Mademoiselle " )
      autres : afficher( " Monsieur, Madame " )
```

Équivalent avec instruction Conditionnelle (plus longue)

```
si abréviation = "M "
    alors afficher( "Monsieur" )
    sinon si abréviation = « Mlle »
            alors afficher("Mademoiselle")
            sinon si abréviation = "Mme"
                    alors afficher( "Madame" )
                    sinon afficher( "Monsieur,Madame " )
                fsi
        fsi
fsi
```

Répétition d'un traitement
Boucle « pour »

```
Algorithme FaitLeTotal
{Cet algorithme fait la somme des nbVal données qu'il saisit}

variables nbVal, cpt : entiers
valeur, totalValeurs: réels

début
   afficher("Combien de valeurs voulez-vous saisir ?")
   saisir(nbVal)
   totalValeurs <- 0
   pour cpt <-1 à nbVal faire
       afficher("Donnez une valeur :")
       saisir(valeur)
       totalValeurs <- totalValeurs + valeur {cumul}
   fpour
   afficher("Le total des ", nbVal, "valeurs est " , totalValeurs)
fin
```

BOUCLE «POUR»

pour <var> <- valInit à valfin [par <pas>] faire
traitement {suite d'instructions}
fpour

ValInt (valeur initiale)
valfin (valeur finale)
par <pas> valeur à ajouter à <var> à chaque passage dans la boucle (par défaut 1, si autre valeur il faut l'indiquer)

- Fonction: répéter une suite d'instructions un certain nombre de fois
- Pour : utilisée quand le nombre d'itération est connu

SÉMANTIQUE BOUCLE « POUR »

L'instruction pour:

- initialise une variable de boucle (le compteur)
- incrémente cette variable de la valeur de « pas »
- vérifie que cette variable ne dépasse pas la borne supérieure

Attention :

-le traitement ne doit pas modifier la variable de boucle

```
Pour cpt <- 1 à MAX faire
    si (...) alors
    cpt <- MAX
fpour
```

LE TRAITEMENT CI-DESSUS EST INTERDIT CAR «cpt» EST MODIFIE DANS LA BOUCLE, CE QUI PEUT CAUSER DES BOUCLES INFINIES

TRAITEMENT ITÉRATIONS INCONNU

```
Algorithme FaitLeTotal
{Cet algorithme fait la somme des nbVal données qu'il saisit, arrêt à la lecture de -1 }
constante      (STOP : entier) <- -1
variables      val, totalValeurs: entiers
début
    totalValeurs <- 0
    afficher("Donnez une valeur,", STOP, " pour finir.") {amorçage}
    saisir(val)
    tant que val != STOP faire
        totalValeurs <- totalValeurs+ val {traitement}
        afficher("Donnez une autre valeur,", STOP, " pour finir.")
        saisir(val) {relance}
    ftq
    afficher("La somme des valeurs saisies est", totalValeurs)
fin
```

BOUCLE « TANT QUE ... FAIRE »

amorçage

tant que <expression logique (vraie)> faire
    traitement
    relance
ftq

amorçage : initialisation de la (des) variable(s) de condition
traitement : suite d'instructions à exécuter si condition vraie
relance : ré-affectation de la (des) variable(s) de condition

Fonction: répéter une suite d'instructions un certain nombre de fois

BOUCLE « TANT QUE ... FAIRE »

Structure itérative "universelle"

N'importe quel contrôle d'itération peut se traduire par le "tant que "

Structure itérative irremplaçable dès que la condition d'itération devient complexe

BOUCLE « TANT QUE ... FAIRE »

Saisir des valeurs, les traiter, et s'arrêter à la saisie de la valeur d'arrêt -1 ou après avoir saisi 5 données.

```
Constantes  (STOP : entier) <- -1
            (MAX : entier) <- 5
Variables   nbVal, val : entiers

Début
      nbVal <-0 {compte les saisies traitées}
      saisir(val) {saisie de la 1ère donnée}
      tant que val != STOP et nbVal < MAX faire
            nbVal <- nbVal+ 1
            ...{traitement de la valeur saisie}
            saisir(val) {relance}
      Ftq
      afficher(val, nbVal) {valeurs en sortie de boucle}
```

Remarque : La valeur d'arrêt (-1) n'est jamais traitée (et donc, jamais comptabilisée)

BOUCLE « TANT QUE ... FAIRE »

Interpréter l'arrêt des itérations

```
......
nbVal <-0 {compte les saisies traitées}
saisir(val) {saisie de la 1ère donnée}
tant que val != STOP et nbValb< MAX faire
      nbVal <- nbVal+ 1...{traitement de la valeur saisie}
      saisir(val) {relance}
Ftq
si val = STOP
    alors {la dernière valeur testée était la valeur d'arrêt}
        afficher(«Sortie de boucle car saisie de la valeur d'arrêt »)
        {toutes les données significatives ont été traitées.}
    sinon {il y avait plus de 5 valeurs à tester}
        afficher(«Sortie de boucle car nombre maximum de valeurs à traiter atteint »)
        {des données significatives n'ont pas pu été traitées.}
fsi
```

COMPARAISON POUR / TANT QUE

```
pour cpt <- 1 à nbVal faire
      afficher("Donnez une valeur :")
      saisir(valeur)
      totalValeurs <- totalValeurs + valeur {cumul}
fpour
```

Est équivalent à

```
cpt <- 0
tant que cpt < nbVal faire
      afficher("Donnez une valeur :")
      saisir(valeur)
      totalValeurs <- totalValeurs + valeur {cumul}
      cpt <- cpt + 1 {compte le nombre de valeurs traitées}
ftq
```

COMPARAISON POUR / TANT QUE

Implicitement, l'instruction pour:

- initialise un compteur
- incrémente le compteur à chaque pas
- vérifie que le compteur ne dépasse pas la borne supérieure

Explicitement, l'instruction tant que doit

- initialiser un compteur {amorçage}
- incrémenter le compteur à chaque pas {relance}
- vérifier que le compteur ne dépasse pas la borne supérieure {test de boucle}

QUAND CHOISIR POUR / TANT QUE ?

Nombre d'itération connu à l'avance : POUR

- Parcours de tableaux (on connait le nombre d'éléments)
- Test sur un nombre donné de valeurs

Boucle s'arrête sur événement particulier : TANT QUE

- Itération avec arrêt décidé par saisie utilisateur

FAIRE ... TANT QUE

Boucle « répéter ... tant que » : exemple

```
Algorithme Essai
{Cet algorithme a besoin d'une valeur positive paire}
Variables valeur : entier
Début
      Répéter
            afficher("Donnez une valeur positive non nulle : ")
            saisir(valeur)
      tant que valeur <= 0
      afficher("La valeur positive non nulle que vous avez saisie est ")
      afficher( valeur )...{traitement de la valeur saisie}
fin
```

BOUCLE « RÉPÉTER ...TANT QUE »

Répéter
    (ré)affectation de la (des) variable(s) de condition traitement
Tant que <expression logique (vraie)>

- Fonction: exécuter une suite d'instructions au moins une fois et la répéter tant qu'une condition est remplie

- Remarque: le traitement dans l'exemple précédent se limite à la ré-affectation de la variable de condition (saisir(valeur))

COMPARAISON «RÉPÉTER» ET «TANT QUE»

```
Répéter
    afficher("Donnez une valeur positive paire :")
    saisir(valeur)
tant que (valeur < 0 ou(valeur % 2) != 0)
```

Équivaut à

```
afficher("Donnez une valeur positive paire :")
saisir(valeur)
tant que(valeur < 0 ou(valeur % 2) != 0) faire
      afficher("Donnez une valeur positive paire:")
      saisir(valeur)
ftq
```

COMPARAISON «RÉPÉTER» ET «TANT QUE»

boucle tant que
- condition vérifiée avant chaque exécution du traitement
- le traitement peut donc ne pas être exécuté
- de plus : la condition porte surtout sur la saisie de nouvelles variables (relance)

boucle répéter ... tant que
- condition vérifiée après chaque exécution du traitement

=> le traitement est exécuté au moins une fois
- de plus: la condition porte surtout sur le résultat du traitement

Remarque: la boucle répéter est typique pour les saisies avec vérification

DE L'ÉNONCÉ À LA BOUCLE

Saisir des données et s'arrêter dès que leur somme dépasse 500

```
somme <- 0
répéter
    saisir(val)
    somme <- somme + val
tant que somme <= 500
```

```
somme <- 0
saisir(val)
somme <- val
tant que somme <= 500 faire
    saisir(val)
    somme <= somme + val
ftq
```

PLUS LOIN : LES TABLEAUX

On peut regrouper plusieurs variables sous un même nom, chacune étant alors repérée par un numéro.
On peut faire un tableau avec n'importe quel type de variable.

Dans tous les cas, le ième élément d'un tableau appelé TAB sera adressé par TAB(i).
Généralement on fait des tableaux à une dimension, mais il existe également des tableaux à deux dimensions , dans ce cas TAB(i,j) représente la Jème colonne et la ième ligne.

PLUS LOIN : LES TABLEAUX

```
Tableau Note(11) : réel
Variables Moy, Som : réel
Début
Pour i <- 0 à 11
      afficher ("Entrez la note n°", i)
      saisir (Note(i))
fpour
Som <- 0
Pour i <- 0 à 11
      Som <- Som + Note(i)
fpour
Moy <- Som / 12
Fin
```

Le premier indice d'un tableau est 0 donc en définissant note(11) notre tableau aura 12 cases.
En algo si on ne connait pas le nombre de case du tableau on écrit simplement
Tableau Note : réel

PLUS LOIN : LES TABLEAUX

NB : On a fait deux boucles successives pour plus de lisibilité, mais on aurait tout aussi bien pu n'en écrire qu'une seule dans laquelle on aurait tout fait d'un seul coup.
Remarque générale : l'indice qui sert à désigner les éléments d'un tableau peut être exprimé directement comme un nombre en clair, mais il peut être aussi une variable, ou une expression calculée.

Dans un tableau, la valeur d'un indice doit toujours :

être égale au moins à 0 (dans quelques rares langages, le premier élément d'un tableau porte l'indice 1). Mais comme je l'ai déjà écrit plus haut, nous avons choisi ici de commencer la numérotation des indices à zéro, comme c'est le cas en langage C et en Visual Basic. Donc attention, Truc(6) est le septième élément du tableau Truc !

être un nombre entier Quel que soit le langage, l'élément Truc(3,1416) n'existe jamais.

être inférieure ou égale au nombre d'éléments du tableau (moins 1, si l'on commence la numérotation à zéro). Si le tableau Bidule a été déclaré comme ayant 25 éléments, la présence dans une ligne, sous une forme ou sous une autre, de Bidule(32) déclenchera automatiquement une erreur.

PROCEDURE ET FONCTION

Un sous-algorithme (procédure ou fonction) est un bloc faisant partie d'un algorithme. Il est déclaré dans la partie entête (avant le début de l'algorithme) puis appelé dans le corps de l'algorithme.

Étant donné qu'il s'agit d'un bloc à part entière, il possède éventuellement un en-tête, une série de traitements, et une gestion des résultats tout comme l'algorithme qui le contient.
Les fonctions et procédures sont utilisées pour éviter les comportements redondants.

PROCEDURE ET FONCTION

Un sous-algorithme utilise les variables déclarées dans l'algorithme (appelées variables globales).

Il peut aussi avoir ses propres variables (dites locales) déclarées dans l'espace qui lui est réservé ; mais qui ne peuvent être utilisées que dans ce sous-algorithme et nulle part ailleurs car sa portée (visibilité) est limitée au bloc qui la contient.

L'espace de ces variables locales n'est réservé que lorsque le sous-algorithme est appelé et est libéré dès la fin de l'exécution.

PROCEDURE ET FONCTION

Un sous-algorithme est déclaré de manière générale c.-à-d qu'il peut être appelé plusieurs fois avec différentes valeurs grâce à des arguments.

Un paramètre est une valeur du bloc principal dont le sous-algorithme a besoin pour exécuter avec des données réelles l'enchaînement d'actions qu'il est chargé d'effectuer.

On distingue deux types de paramètres :

- Les paramètres formels sont la définition du nombre et du type de valeurs que devra recevoir le sous algorithme pour se mettre en route avec succès. On déclare les paramètres formels pendant la déclaration du sous-algorithme.

- Les paramètres effectifs sont des valeurs réelles (constantes ou variables) reçues par le sous-algorithme au cours de l'exécution du bloc principal. On les définit indépendamment à chaque appel du sousalgorithme dans l'algorithme principal

PROCEDURE ET FONCTION

L'exécution d'un sous-algorithme (procédure ou fonction) se fait par une instruction d'appel.

L'application de cette instruction génère un saut vers le sous-algorithme appelé.

La terminaison de ce sous-algorithme redémarre la suite d'instruction interrompue par l'appel.

PROCEDURE ET FONCTION

Un sous-algorithme peut se présenter sous forme de fonction ou de procédure.

Une fonction est un sous-algorithme qui, à partir de donnée(s), calcul et rend à l'algorithme Un et Un seul résultat à travers une fonctionnalité «return»

Une procédure affiche à l'écran le(s) résultat(s) demandé(s).

PROCEDURE

Une procédure est un bloc d'instructions nommé et déclaré dans l'entête de l'algorithme et appelé dans son corps à chaque fois que le programmeur en a besoin.

Déclaration d'une procédure :

```
Procédure Nom_Procédure (Nom_Paramètre : Type_praramètre;.....) ;
Déclaration
        Nom_variable : Type_variable ;   }   Variables locales
        ...
Début
        ...
        Instructions ;                   }   Corps de la procédure
        ...
Fin ;
```

L'appel d'une procédure peut être effectué en spécifiant, au moment souhaité, son nom et éventuellement ses paramètres

Cela déclenche l'exécution des instructions de la procédure.

PROCEDURE

```
Algorithme essai;

Variables
    I, S : entier;

Procedure Somme ;
  Debut /*Début de la Procédure*/
        S <- 0 ;
        Pour I <- 1 a 100 Faire
            S <- S + i
        FPour ;
  Afficher ('La somme des 100 premiers nombres est', S) ;
Fin /*Fin de la Procédure*/

Debut /*Début de l'algorithme*/
    Somme
Fin. /*Fin de l'algorithme*/
```

FONCTION

Une fonction est un bloc d'instructions qui retourne obligatoirement une et une seule valeur résultat à l'algorithme appelant. Une fonction n'affiche jamais la réponse à l'écran car elle la renvoie simplement à l'algorithme appelant.

Déclaration d'une fonction :

Fonction Nom_Fonction (Nom_Paramètre : Type_praramètre;.....) : type_Fonction ;

Déclaration

Nom_variable : Type_variable ; } Variables locales

...

Début

...

Instructions ; } Corps de la procédure

...

Nom_Fonction ← Résultat

Fin ;

Etant donné qu'une fonction a pour but principal de renvoyer une valeur, il est donc nécessaire de préciser le type de la fonction qui est en réalité le type de cette valeur.

Un appel de fonction est une expression d'affectation de manière à ce que le résultat soit récupéré dans une variable globale

Nom_variable-globale <- Nom_Fonction (paramètres) ;

FONCTION

```
Algorithme essai;
Variables
   I, Som : entier;
Fonction Somme: entier ;
Variable
   S : entier ;
Debut /*Début de la fonction*/
S <- 0 ;
Pour I <- 1 a 100 Faire
S <- S + I
FPour ;
Somme <- S
Fin /*Fin de la Fonction */

Debut /*Début de l'algorithme*/
   Som <- Somme ;
   Afficher ('La somme des ', N, 'premiers nombres est', Som) ;
Fin. /*Fin de l'algorithme*/
```

De même qu'une procédure, une fonction peut appeler d'autres sous-algorithmes à condition qu'ils soient définis avant elle ou qu'ils soient déclarés dans son entête.

Paramètre d'entrée

```
Algorithme exemple1 ;

Variables  <-global au programme
   a, b : Entier ;

Fonction abs (unEntier : Entier) : Entier ;
   Variable <- porté de la fonction
      valeurAbsolue : Entier ;
   Debut
      si unEntier ≥ 0 alors
         valeurAbsolue <- unEntier
      sinon
         valeurAbsolue <- - unEntier
      fsi ;
      abs <- valeurAbsolue
   fin ;

Debut
   Afficher ('Entrez un entier : ') ;
   saisir (a) ;
   b <- abs (a) ;
   Afficher ('la valeur absolue de ', a, ' est ', b)
Fin.
```

Lors de l'exécution de la fonction abs, la variable a et le paramètre unEntier sont associés par un passage de paramètre en entrée : La valeur de a est copiée dans unEntier.

STRUCTURES (OBJET)

Les structures sont des types de variable que le développeur va lui-même définir.
Elles permettent de regrouper des données se référant au même concept dans une seule variable, ce qui aura pour effet d'améliorer la lisibilité du code et sa maintenance/évolutivité. Avec ce type de variables, on commence à se rapprocher d'un développement objet.

```
Variables
   nom <- « toto »
   email <- « toto@pouet.fr »
   age <- 25
   nom2 <- « abdel »
   email2 <- « abdel@mail.fr »
   age2 <- 32
debut
   afficher nom
   isMajeur(age2)
Fin
```

```
Personne : structure {
nom : chaine
email : chaine
age : entier
}
Variables
   personne1, personne2 : Personne
Debut
   personne1.nom = « Toto »
   personne1.email = « toto@pouet.fr »
   personne1.age = 25
   personne2.nom = « abdel »
   personne2.email = « abdel@mail.fr »
   personne2.age = 32
   afficher personne1.nom
   isMajeur(personne2.age)
Fin
```

ALGO CONNEXION BDD

Voici un algo assez simplifié d'une connexion à une base de données

```
Player : structure {
nom : chaine,
points : entier
}
 Tableau : Joueurs : Player

Debut
   {récupération des données en bdd}
   Connexion_bdd(« localhost », « root », « 1234 »)
   {on considere qu'il s'agit d'une fonction prédéfini}
   choixbase(« formation_m2i »)
   joueurs <- Requête_SQL (« SELECT name, points FROM player ORDER BY points DESC »)
   {affichage des joueurs dans la page}
   POUR i <- 0 à LENGTH(players)-1
            Afficher (players[i].name , « : » , players[i].points , « <br> »);
   FPOUR
Fin
```

ALGORIGRAMME

Alternative au pseudo-code, il permet d'afficher des algos sous forme de schéma.

- LES PRINCIPAUX SYMBOLES :

SYMBOLE	DÉSIGNATION	SYMBOLE	DÉSIGNATION
	début ou **fin** d'un algorithme	*ou* *non* *oui*	**Test ou Branchement conditionnel** décision d'un choix parmi d'autres en fonction des conditions
	symbole général de « **traitement** » opération sur des données, instructions, … ou opération pour laquelle il n'existe aucun symbole normalisé		**sous-programme** appel d'un sous-programme
	entrée / **sortie**		**Liaison** Les différents symboles sont reliés entre eux par des lignes de liaison. Le cheminement va de haut en bas et de gauche à droite. Un cheminement différent est indiqué à l'aide d'une flèche
	commentaire		

ALGORIGRAMME

- LES DIFFÉRENTES STRUCTURES :

Structure linéaire	Structure alternative
On exécute successivement une suite d'action dans l'ordre de leur énoncé.	Cette structure offre le choix entre deux séquences s'excluant mutuellement.
Algorithme	
Début Action 1 Action 2 **Fin**	**Début** **Si** Condition **Alors** Action 1 **Sinon** Action 2

Structure itérative (répétitive)	
On teste d'abord la condition, la séquence est exécutée tant que la condition est vraie	L'action est exécutée au moins une fois, elle est répétée tant qu'elle est fausse
Algorithme	
Tant que Condition vraie **Faire** Action	Action **Répéter** Action **Jusqu'à** Condition vrai

Plus d'infos : https://rascolsin.fr/psti2d/algo/01-Algo-Orgas/#2-lorganigramme