Logique algorithmique

Procédures et fonctions

1 Introduction

Lorsqu'un algorithme devient important, trop long ou trop complexe, on réutilise le vieux principe : diviser pour mieux régner.

Divisons, divisons qu'ils disaient ...

On va donc créer des sous-algorithmes qui, plus petits, seront plus facile à maintenir, à créer et à comprendre.

Faut-il déclarer les fonctions avant ou après l'algorithme principal ?

Certains langages de programmation le demande (soit avant, soit après) mais nous sommes en algo donc peut importe.

Quand utiliser une fonction ?

On utilise une fonction lorsqu'on a l'une ou plusieurs conditions suivantes (liste incomplète):

- une partie de l'algorithme, une série de traitements est répétitive, apparaît plusieurs fois.
- Un algorithme est utilisé dans différents algorithmes d'appel.
- Un algorithme trop complexe sera découpé en fonctions plus petites et donc plus simples à écrire et maintenir.
- L'algorithme est découpé en fonctions élémentaires permettant de séparer la partie affichage de la partie calcul.
- Lorsqu'on en ressent le besoin ou quand on en a envie.

2 Déclaration d'une fonction

Une fonction s'écrit comme un algorithme. Les seuls changements sont dans l'utilisation du mot fonction à la place du mot algorithme.

Exemple :

Algorithme gestionOuvrage

Cet algo sert à gérer les ouvrages d'une bibliothèque

Fonction ajouterOuvrage(oNum : entier, oCode : chaîne, oTitre : chaîne, oEdition : date)

Fonction calculNbTotalOuvrage() : entier

Sont des fonctions utilisées par l'algorithme (ajouter un ouvrage à la bibliothèque, calculer le nombre total d'ouvrages de celle-ci)

Que sont les choses écrites entre les parenthèses ?

Ce sont des paramètres. Ces paramètres sont les données nécessaires à la fonction pour fonctionner.

Ils sont écrits comme dans le paragraphe de déclaration des variables et, pour la fonction, c'est une vraie déclaration de ces variables.

Ces variables paramètre ne seront pas re-déclarées dans le paragraphe 'Variables'.

Qu'est ce qu'il y a d'écrit après la fonction ?

Une fonction peut retourner un résultat (c'est son but).

On indiquera, après la parenthèse fermante, un double point suivi du type du résultat retourné.

UNE FONCTION NE RETOURNE QU'UN SEUL RÉSULTAT !

Une fonction retourne-t-elle toujours un résultat ?

Non, cependant, si une fonction ne retourne aucun résultat, on l'appellera procédure. Les procédures sont toute fois moins courantes que les fonctions. De plus, on aura souvent un résultat qui sera un indicateur de bon fonctionnement, donc une fonction et non une procédure ...

Une fonction possède-t-elle toujours des paramètres ?

Non, mais les parenthèses sont toujours marquées.

Certains professionnels, ouvrages ou professeurs ajoutent même des parenthèses aux algorithmes qui peuvent contenir des paramètres d'appel de l'algorithme.

Comment faire pour retourner le résultat ?

C'est simple, on utilise l'instruction 'retourner' suivie d'une expression. Cette expression est souvent la variable qui contient le résultat.

Exemple :

Dans le cas de notre calcul de volume, le calcul de la surface est effectué plusieurs fois.

On va donc l'extraire de notre algorithme principal pour le mettre dans une fonction.

Cette fonction aura besoin du diamètre du cercle pour calculer la surface. La fonction renverra la surface calculée.

En algo, le prototype (la définition, la première ligne) de la fonction est le suivant :

Fonction calculSurface(diamètre : réel) : réel

-> on met le mot fonction à la place d'algorithme

-> on écrit les paramètres nécessaires entre parenthèses (ça ressemble à une déclaration de variables ...)

-> on écrit le type de la fonction (ça ressemble aussi à une déclaration ...)

Ensuite, on écrit l'algo de la fonction comme un algo normal (ou presque).

Les différences sont les suivantes :

- Les paramètres sont déclarés dans le prototype donc pas besoin de les redéclarer dans le paragraphe des variables (ce serait même une faute)
- Si la fonction a un type (elle retourne une valeur), on utilisera l'instruction retourner pour indiquer quelle valeur est retournée ou quelle variable (ou expression) permet de la connaître. Si la fonction n'a pas de type, c'est une procédure et elle ne retourne aucun résultat.

Fonction calculSurface(diamètre : réel) : réel

Variables

surface : réel // déclaration des variables non paramètres

Début

surface <- Pi*diamètre*diamètre // calcul

Retourner surface // retour du résultat

Fin

Et voilà.

3 Utiliser une fonction

Mais comment utiliser une fonction ?

Rien de plus simple : on utilise une fonction comme une variable

Exemple :

On avait avant : surfaceExt <- Pi*diamètreExt*diamètreExt

On aura maintenant : surfaceExt <- calculSurface(diamètreExt)

L'ordinateur évaluera la fonction en exécutant son algorithme, avec les valeurs des paramètres cités et remplacera l'appel à la fonction par son résultat retourné.

Waou quelle révolution !

Mouais, bon, ici une fonction n'a pas beaucoup d'intérêt mais faites un algo qui permet de faire des simulations d'amortissement selon l'amortissement constant, l'annuité constante ou les deux et ça devient déjà plus intéressant.

D'ailleurs, pouvez vous me dire comment est faite l'instruction afficher ? Hein ? Hein ?

On n'en sait rien et on s'en moque. Tant que cette **fonction** fonctionne correctement. Idem pour lire ou saisir.

Ce sont effectivement des fonctions.

Afficher(texte : chaîne) est une fonction qui décompose le texte en une suite de caractères et les écrit un par un, à la queue leu leu à l'écran.
De la même façon, et c'est le cas de certains profs et bouquins, on devrait écrire : '*variable<-lire()*' à la place de la fonction saisir.

Exemple :

```

Tube ...
Fonction calculSurface(diamètre : réel) : réel
Variables
    surface : réel // déclaration des variables non paramètres
Début
    surface <- Pi*diamètre*diamètre // calcul
    Retourner surface // retour du résultat
Fin

Algo volumeTube
Var
    diametreExt, diametreInt, hauteurTube : réel
    surfaceCercleExt, surfaceCercleInt, surfaceAnneau, volumeTube : réel
Début
    // Saisie des informations :
    Afficher ("donnez les diamètres intérieurs puis extérieur et la hauteur du tube :")
    Saisir (diametreExt , diametreInt, hauteurTube)

    // Calculs
    surfaceCercleExt <- calculSurface (diametreExt)
    surfaceCercleInt <- calculSurface (diametreInt )
    surfaceAnneau <- surfaceCercleExt – surfaceCercleInt
    volumeTube <- surfaceAnneau * hauteurTube

    // Restitution des résultats
    Afficher ("le volume du tube est de : ", volumeTube)
Fin
  
```

4 Méthodes et évènements

Les méthodes et évènements sont des fonctions.

Dans le premier cas, une méthode est une fonction attachée à un objet (ou variable objet) qui effectue un traitement sur les valeurs contenues dans cet objet.

En effet, un objet informatique est un regroupement de plusieurs variables élémentaires sous le nom d'une variable générique.

Exemple :

nomFournisseur, numFou, adrFou sont des variables élémentaires.

Grâce à ces variables, on peut construire un type fournisseur complexe (une classe) qui contiendra plusieurs variables élémentaires :

```

Fournisseur (
    numFour : entier
    nomFour, adrFour : chaîne
)
  
```

Fournisseur est une classe.

Lorsque je dit :

```
OrdiPro <- nouveau Fournisseur
```

Je défini un objet nommé OrdiPro qui contient les variables élémentaires.

Une classe (ou définition générique d'objets) permet aussi de définir des petits morceaux d'algorithme qui indiquent des comportements.

Exemples :

```
Fournisseur (
    numFou : entier
    nomFou, adrFou : chaîne
    donnerAdresse(): chaîne
    Début
        Retourner adrFou
    Fin
)
```

On utilise une méthode en l'indiquant juste après le nom d'un objet, derrière un point

Exemples :

```
Afficher ("l'adresse du fournisseur ", OrdiPro.nomFou, " est ", OrdiPro.donnerAdresse())
```

Dans le second cas, une procédure événementielle est un bout d'algorithme associé à un événement associé à la modification de l'état d'un objet dans une application.

Exemple :

Une application contient une fenêtre où on peut cliquer sur un bouton. On pourra attacher au bouton la procédure suivante :

```
surClic()
    début
        afficher (OrdiPro.donnerAdresse())
    fin
```

Lorsqu'on appuiera sur le bouton, le programme ira rechercher l'adresse du fournisseur contenue dans le fournisseur OrdiPro.

Les procédures événementielles sont souvent visibles dans les logiciels access, excel et word mais la liste n'est pas limitative.

5 Résumé des instructions élémentaires disponibles en algorithmique

Groupe	Description banale	Description informatique	Instruction
La remarque	Commentaire pour expliquer l'algo	Ligne de remarque ou de commentaire	// ... ; /* ... */
L'affectation	Remplir une boîte avec une valeur	Affecter une valeur à une variable	... <- ...
Entrées/Sorties	Capter ou restituer des informations	Les entrées/sorties, lecture/écriture	Lire, écrire ; Saisir, afficher
Les tests	Prendre des décisions	Les alternatives	Si, sinon, fins
Les boucles	Faire plusieurs fois le même travail	Les répétitives ou itérations	Répéter, jusqu'à ; tant que, faire, fin tant que ; pour, fin pour
Les fonctions	Sous-traiter le travail	Fonctions et procédures	Fonction nom_de_la_fonction ([liste de paramètres]) : type_du_résultat ...

6 Applications

Dans certains exemples, j'ai utilisé différentes notations des entrées/sorties afin de présenter les différentes façons de les écrire.

6.1 Calcul de la surface d'un cercle, fonction à utiliser dans un algorithme.

```
Fonction surfaceCirculaire(rayon : réel) : réel
/*Description : ... */
Var // déclaration des variables
    Surface, rayon : réel, variables : surface et rayon du cercle
    PI : réel <- 3.1415926535897932384626433832795, constante Pi
Début // corps
    surface <- 2*PI*rayon^2
    retourner(S)
Fin
```

6.2 Procédure afficher()

```

Procédure afficherEntête (nom : chaîne, prénom : chaîne)
/* Description : Cette procédure affiche le nom et prénom de l'utilisateur dans un cadre. */
/* elle ne retourne rien, forcément, c'est une procédure ... */
Var // déclaration des variables: Aucune variable locale nécessaire, elles sont toutes transmises en paramètre
Début // corps
Ecrire ("=====")
Ecrire (> " , nom, " " , prénom)
Ecrire ("=====")
Fin

```

6.3 Fonctions calcul de quelque chose

```

Fonction calculerPrime (somme : réel) : réel
/* Description : Calcul de la prime.
Renvoie la valeur de la prime selon la valeur de la somme : si la somme est inférieure à 1000, alors la prime est de
1% de la somme, sinon la prime est de 2% de la somme.
*/
Var // déclaration des variables
Prime : réel, montant de la prime calculée
Début // corps
Si (somme<1000)
Alors
prime ← somme*0,01
Sinon
prime ← somme*0,02
FinSi
Retourner (prime)
Fin

```

6.4 Utilisation de fonctions et procédures

```

calculPrime()
/* Description : Calcul de la prime d'une personne en fonction de son chiffre d'affaire */
Var // déclaration des variables
nom, prénom : chaîne, nom et prénoms de la personne
chiffreAffaire : réel, chiffre d'affaire de la personne
Début // corps
Lire("nom ? ", nom)
Lire("prénom ? ", prénom)
Lire("chiffre d'affaire ? ", chiffreAffaire)
prime <- calculerPrime(chiffreAffaire) // appel à la fonction de calcul de la prime
afficherEntête(nom, prénom) // appel à la procédure d'affichage des données sur la personne
Ecrire("prime sur chiffre d'affaire : ", prime)
Fin

```

6.5 Amortissements

```

Algo CalculAmortissement
Variables
mode, continuer : caractère
tauxPourcent, taux : réel
capitalInit, capitalRestant, intérêt, remboursement, annuité : réel
totalIntérêt, totalAnnuité : réel
n, durée : entier
Début
// Initialisation globale des variables
continuer <- 'o'
Répéter
// Initialisation locale des variables
tauxPourcent <- 0
// Saisie des données
Afficher ("Choisissez un type d'amortissement")
Répéter

```

```

Afficher ("Annuité constante=A ; Remboursement constant=R")
Saisir (mode)
Jusqu'à (mode = "A" ou mode = "R")
Afficher ("Donnez le capital initial")
Saisir (capitalInit)
Afficher ("Donnez la durée de l'emprunt")
Saisir (durée)
Afficher ("Donnez le taux (en %) pour un amortissement par annuités constantes")
Saisir (tauxPourcent)

// Calculs préliminaires : taux en valeur absolue, annuité (cste)
taux<-tauxPourcent/100
Si (mode = "A") alors
    annuité<- calculAnnuité(capitalInit, taux, durée)
    remboursement <-0
Sinon
    annuité<- 0
    remboursement <- capitalInitial/durée
FinSi

// Préparation de l'affichage des résultats : affichage de l'entête des colonnes
Si (mode = "A") alors
    Afficher ("Amortissement par annuités constantes")
Sinon
    Afficher("Amortissement par remboursements constants")
FinSi
Afficher ("Année \tCapital \tIntérêt \tRemboursement \tAnnuité")

// Calculs et affichage des résultat au fur et à mesure ; n est l'année en cours
capitalRestant <- capitalInitial
Pour (n=1 à durée)
    intérêt <- capitalRestant * taux
    Si (mode = "A") Alors
        // anuité = constante
        remboursement <- annuité – intérêt
    Sinon
        annuité <- intérêt +remboursement
        // remboursement = constante
    FinSi
    Afficher (n, "\t", capitalRestant, "\t", intérêt, "\t", remboursement, "\t",
annuité)

// Calcul des totaux de récapitulation en fin d'algo
totalAnnuité <- totalAnnuité+annuité
totalIntérêt <- totalIntérêt <- intérêt

// Préparation du calcul de la ligne suivante
capitalRestant <- capitalRestant - remboursement
FinPour
// Affichage de fin de calcul
Afficher ("Le total des intérêts se monte à : ", totalIntérêt)
Afficher ("Le total des annuités s'élève à : ", totalAnnuité)

```

```
    // Recommencer ?
    Afficher ("Voulez-vous effectuer une nouvelle simulation ? (o/n) ")
    Saisir (continuer)
    Jusqu'à (continuer <> 'o' et continuer <>'O')

    // Traitements de fin du programme
    Afficher ("À bientôt")
Fin

Fonction calculAnnuité(capitalInit : réel, tauxAbs : réel, durée : entier) : réel
Début
    Retourner capitalInit*tauxAbs/( 1-(1+tauxAbs/100) ^(-durée) )
Fin
```