UFR MI

Support de cours

INITIATION A L'ALGORITHME

Equipe Pédagogique Informatique

Proposé par M. ASSOHOUN Egomli Stanislas

Plan

- CHI – Généralités et structures de contrôle
- CHII – Types énumérés, Tableaux et enregistrements
- CHIII – Sous programmes
- CHV- Fichiers
- CHVI- Python

Objectifs:

Dans ce cours, nous allons:

- Décrire les sous programme.
- Ecrire des algorithme en utilisant des sous programme de type fonction ou procédure
- Ecrire des algorithme en utilisant des sous programme de type fonction ou procédure recursive

CH III– SOUS PROGRAMMES

CH III – Sous programme

I- Définition et types de sous programme.

I-1- Définition : Un sous programme est un programme particulier, non autonome, qui décrit une action précise, et qui est utilisé dans le contexte d'un autre programme (le programme appelant).

Pour fonctionner, selon l'action qu'il a à réaliser, il peut avoir besoin (ou pas) d'arguments ou paramètres : des données à fournir dès le départ (argument_donnée), ou des variables qui vont contenir des résultats (argument_resultat)

Ex : un sous programme qui calcule l'aire d'un cercle . Argument_donnée = rayon du cercle, argument_resultat = aire du cercle

I-2- Types

Il existe deux types de sous programme : les FONCTION qui est un sous programme qui restitue obligatoirement un résultat. les sous programme qui est tout ce qui n'est pas une fonction, appelé PROCEDURE.

CH III – Sous programme

II- Syntaxe d'un sous programme

Un sous programme est d'abord un programme. Sa syntaxe est pratiquement la même que celle d'un programme.

II-1- Fonction : Puisqu'on est sûr d'obtenir un résultat pour une fonction, la syntaxe et la structure sont:

```
FONCTION <nom_fonction> ( arg1:type1, arg2:type1, ... ) :type_resultat
Var (*Partie déclaration des variables*)
 ……..
 DEBUT (*Corps de la fonction*)
        ……..
        Renvoyer var_resultat
FIN
```

CH III – Sous programme

II- Syntaxe d'un sous programme

II-1- Fonction : (suite)

Appel d'une fonction : L'appel doit être contenu dans une instruction d'affectation.

Exemple 1

```
FONCTION  aire( x : reel) : reel
Var (*Partie déclaration des variables*)
 y: reel
 DEBUT (*Corps de la fonction*)
        y <- 3.14*x*x
        Renvoyer y
FIN
```

Appel de la fonction

```
a <- aire(5)
```

CH III – Sous programme

II- Syntaxe d'un sous programme

Exemple 2

```
FONCTION  factoriel( n : entier) : reel
Var (*Partie déclaration des variables*)
 fact : entier
 DEBUT (*Corps de la fonction*)
        fect <- 1
        TANTQUE (n > 1) FAIRE
                fact  <- fact*n
                n <- n -1
        Renvoyer fact
FIN
```

CH III – Sous programme

II- Syntaxe d'un sous programme

Exemple 2 (suite)

Pour cet exemple : Le nom est factoriel. Il n'y a qu'un unique argument : n. L'appel se fera, par exemple : a<- factoriel(3) et retournera 3 !=6.

Utilisation de la fonction factoriel(n)

Exemple de programme utilisant la fonction factoriel ci-dessus : Calcul du nombre de combinaisons de p éléments parmi n. = n! /p! (n-p)! (voir) TD)

II-2- Procédure

Une procédure peut ne renvoyer aucun résultat au programme appelant ou lui renvoyer un ou plusieurs résultats. Une procédure peut aussi ne pas avoir besoin d'arguments.

CH III – Sous programme

II- Syntaxe d'un sous programme

II-2- Procédure : (suite)

PROCEDURE <nom_procedure> (arg1:type1, arg2:type1, ...)

Var (*Partie déclaration des variables*)

……

DEBUT (*Corps de la procedure*)

……

FIN

Appel d'une procédure: L'appel se fait de la manière suivante : nom_de_la_procédure (liste effectifs)

Nb: à l'ecriture de la procedure les arguments utilisés sont dits formels à l'appel ils sont dits effectifs

CH III – Sous programme

II- Syntaxe d'un sous programme

II-2- Procédure : (suite)

PROCEDURE <nom_procedure> (arg1:type1, arg2:type1, ...)

Var (*Partie déclaration des variables*)

……

DEBUT (*Corps de la procedure*)

……

FIN

Appel d'une procédure: L'appel se fait de la manière suivante : nom_de_la_procédure (liste effectifs)

Nb: à l'ecriture de la procedure les arguments utilisés sont dits formels à l'appel ils sont dits effectifs

CH III – Sous programme

II- Syntaxe d'un sous programme

II-2- Procédure : (suite)

Exemple écrire une procedure pour avoir le perimetre d'un cercle

```
PROCEDURE  perimetre( x : reel)
Var (*Partie déclaration des variables*)
 y : reel
 DEBUT (*Corps de la fonction*)
        y <- 3.14*x
        AFFICHER(" Le perimetre est : ",y)
FIN
```

CH III – Sous programme

III- Variables locales – globales.

Les variables déclarées au sein d'un sous programme sont appelées variables locales. Une variable locale n'existe que le temps de l'exécution du sous programme et n'est pas visible (ne peut être utilisée) à l'extérieur de ce sous programme.

Une variable déclarée à l'extérieur d'un module donné sera en général visible par tout module écrit après cette déclaration. C'est une variable globale pour ces modules. On pourra alors les utiliser pour renvoyer des résultats lors d'un appel de sous-programmes. Cela permettrait ainsi à une fonction de renvoyer plus d'un résultat, à une procédure même sans argument de renvoyer des résultats.

CH III – Sous programme

III- Variables locales – globales.

Exemple: programme principal utilisant la fonction aire(x) et la procedure perimetre(x)

```
ALGO PeriAireCercle
Var (*Partie déclaration des variables globales*)
r a : reel
 y : reel
 DEBUT (*Corps du programme principal*)
        AFFICHER("Entre le rayon du cercle")
        SAISIR(r)
        a <- aire(r)
        AFFICHER("AIRE = ",a)
                perimetre(r)
        FIN
```

CH III – Sous programme

V- Passage des arguments.

Lorsqu'un programme appelle un sous programme donné, il lui donne les valeurs des arguments nécessaires pour son exécution. Ces arguments sont des variables. Le passage se fait **« par valeurs »** si ce sont les copies de ces variables qui sont utilisées par le sous programme. Les variables fournies, qui sont des variables du programme appelant, ne sont donc pas affectées par un quelconque changement de valeur opéré à l'intérieur des sous programmes. Le passage se fait « **par adresses** » si ce sont les adresses de ces variables qui sont données au sous programme. Dans ce cas, toute modification effectuée sur une de ces variables durant l'exécution du sous programme est effective et persiste même en dehors du sous programme.

NB :

Lors de ces passages, les variables fournissant les valeurs requises pour les différents arguments doivent correspondre, en TYPES, en NOMBRE, aux types et au nombre des arguments du sous programme, dans l'ORDRE d'apparition de ceux-ci.

CH III – Sous programme

V- Récursivité .

Définition : Un sous programme qui peut s'appeler lui-même est un sous programme récursif. Particularité : il faut à chaque fois un **critère d'arrêt**. Certains langages permettent la récursivité, comme le C, Pascal,…… Ce n'est pas le cas de Fortran77, par exemple.

Exemple1 :

Programmer en utilisant une fonction récursive l

Si n = 0 fact(n) = 1 sinon fact(n) = n * fact(n-1)

Exemple 2

Programmer en utilisant une fonction récursive le nième terme d'une suite de Fibonacci. Rappel : une suite (Un) de Fibonacci a comme définition par récurrence : U0 = U1 = 1 et pour tout n > 1, Un=Un-2 + Un-1

Merci de votre attention