**Algorithmique 2ème partie**

Table des matières

[4) Procédures et fonctions 2](#_Toc91540817)

[1. Fonctions. 2](#_Toc91540818)

[2. Procédure 3](#_Toc91540819)

[5) Structure des données 4](#_Toc91540820)

[1. Les tableaux 4](#_Toc91540821)

[2. Les structures 5](#_Toc91540822)

[3. Tableau et structure 6](#_Toc91540823)

# 

# 4) Procédures et fonctions

* On peut regroupe des opérations qui risquent de se répéter.
* Pour rendre l’algorithme plus lisible on attaque le problème bout par bout.
* On utilise des blocs responsables d'une chose à la fois.

## 1. Fonctions.

* C’est une section d’algorithme qui a un objectif et un nom bien défini.
* Communique avec l’algorithme extérieur par des paramètres avec des types défini nommé variables locales et non visibles à l’extérieur.
* À la fin de la fonction il faut qu’il retourne le résultat via return

**Déclaration**

Fonction NomDeFonction ( ListeParamètres) : TypeRésultat ; *// si type de résultat est vide il s’agit* *d’une procédure.*

Début

*// bloc instruction et appel à retourne*

Fin

FinFonction.

**Exemple :**

Fonction somme (x : entier , y : Entier) : Entier ( type de retour)

Var : somme : Entier

Début

Ecrire ( ‘’Calcul de la somme en cours… ‘’)

Somme= x + y ;

Retourner somme (le type a été déclaré au début);

Finfonction

Cette fonction calcul la somme et retourne le résultat à l’algorithme qui l’appelle.

## 2. Procédure

Procedure somme (x : Entier, y : Entier) : vide ;

Var : somme : Entier

Début

Ecrire (‘’calcul de la somme en cours…’’)

Somme = x + y

Ecrire (‘’la somme des deux nombres est : ‘’, somme)

Fin procédure

Ici on ne fait qu’afficher le résultat. La différence entre ces deux exemples est le raisonnement à la fin : fonction (retourne le résultat) et la procédure lit.

**Appel de fonction.**

* Il faut appeler une fonction dans un algorithme pour que son code puisse s’exécuter.
* Lors de l’appelle fonction, le nom de la variable associé aux différents paramètres peuvent changer. Seul leur type doivent être les mêmes.

Exemple d'appel fonction :

Var :x, y : Entiers

Début

Lire (x,y)

Result=somme (x,y)

Print (‘’ la somme de x et y est :’’ + result)

Fin

* Une fonction ou une procédure peuvent être appelées de plusieurs manières.
  + Comme dans l’exemple ci-dessus
  + a=a\*5

Ecrire (‘’la somme des deux nombres est, '' somme (a,10 )

Ici on a remplacé le deuxième paramètre par un entier et ça marche !

* + Ecrire(‘’entrer la valeur de a'')

Lire (a)

a=10\*somme (a,a)

* + Ecrire (‘’la somme des 5 premiers nombres entier est :’’ + somme(5, somme(4, somme(3, somme(2,1)))))

# 5) Structure des données

* On a besoin de structure de données complexes pour traiter une grande quantité de donnée liée entre elle.
* Les données complexes c’est un ensemble d’éléments avec différentes caractéristiques chacun. Ex : une transaction financière : montant, date, destinataire, expéditeur, … Donc on traite un ensemble d’entités.

## 1. Les tableaux

* On regroupe les données dans un tableau lorsqu’elles sont nombreuses et de même type afin d’éviter de multiplier le nombre de variable.
* Une structure de type tableau précise le type commun de tous les éléments.
* Syntaxe : tabName : TYPE[TAILLE]
* Le tableau est de type quadruplet puisqu’il a : un nom, un type, une valeur et une taille pour chaque élément.
* Il faut établir à l’avance la taille du tableau en fonction du nombre d’éléments (de préférence on choisit une taille plus élevée par précaution).
* Les tableaux peuvent avoir une ou plusieurs dimensions.
* On fait la distinction entre les différents éléments du tableau par leur position nommée indice (ou index). C’est un nombre entier de 0 à N-1.
* Il suffit de préciser l'indice de la case [] pour accéder à un élément du tableau. Ex : Tab[4].
* Les instructions (lecture, écriture et affectation) peuvent aussi s’appliquer aux tableaux. Lorsqu’on l’affecte à une variable, celle doit être du même type que l’élément du tableau associé. Ex : x=Tab[0]
* On peut même modifier le contenu des cases avec une affectation en précisant son indice comme suit : Tab[6]=43.

-Exemple d'initialisation des éléments d’un tableau avec une boucle : (avec n entier et taille du tableau, i entier et indice d'itération pour passer d’une case à l'autre)

AlgoInitTableau

Début

Pour i de 0 à n-1 faire

Tab[i]=0

FinPour

Fin

* Pour les tableaux à 2 dimensions ou matrice, les données sont de même nature mais dépendent de deux critères différents.
* Syntaxe : Tab[L, C]. Ex : Tab[1,4]=38

## 2. Les structures

* Les types de variable sont extensibles à l’infini. On peut créer un type de variable (hors types de base).
* Une structure ou enregistrement est un produit nommé de plusieurs types. En définissant un nouveau type, on a une variable ayant comme type ce nouveau type.

**Syntaxe**

Struct Person { *// « person » est le nom du nouveau type créé*

Nom : car (20) *// « nom » est une caractéristique et « car(20) » est un type de caractéristique*

Prenom : car(20) *// idem*

Age : Entier

Déclaration de variable :

p1=Person ;

p2=Person ;

Utilisation :

p1.nom=’’MOUCHENE’’ *// le nom de la variable de type person*

p1.prenom=’’Djamel’’

p1.age=25

Ecrire (‘’le nom de la personne est :’’ + p1.nom)

## 3. Tableau et structure

Struct Person {

Nom : car (20)

Prenom : car(20)

Age : Entier }

Persons : Person[20] ; *// Tableau de type Person de taille 20*

p1=Person ;

p1.nom=’’MOUCHENE’’

p1.prenom=’’Djamel’’

p1.age=25

persons[0]=p1 // remplissage du tableau avec les 3 infos à l’indice 0 (1ère case)

On peut même faire appel à une fonction comme suit :

Persons : Person[20]

p : Person

Persons = chargerLesPersonnesDepuisLaListeBD() // appel de fonction

Ecrire(‘’voici la liste des personnes’’)

Pour index allant de 0 à 19 Faire //boucle avec le compteur index

p=persons[index]

écrire(‘’Nom :’’ p.nom)

écrire(‘’Prenom :’’ p.prenom)

écrire(‘’Age :’’ p.age) //affichage automatique des 3 types pour les 20

FinPour

Ce programme affiche le nom, le prénom et l’âge des 20 première personne de la liste dans la base de données