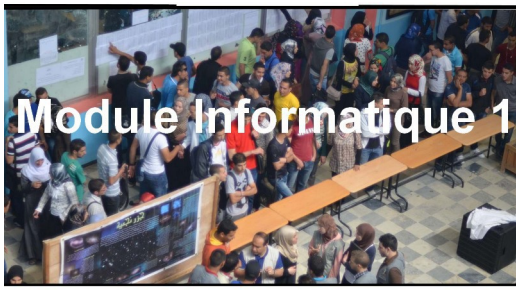
1.1

Informatique 01

Informatique 1

Formation des spécialistes en e-learning
Promotion 2015-2016



année universitaire 2015/2016
Bendiab Keltoum

Table des matières

Objectifs *5*

Introduction *7*

I - Chapitre 4 : Procédures et fonctions *9*

A. 1). Les fonctions..... *10*

B. 2) Les procédures..... *11*

1. 2.1). Définition d'une procédure..... 11

C. 3) Variables globales et de variables locales..... *12*

D. 4). Les paramètres..... *12*

E. 6). Exercices..... *17*

1. Exercice 1..... 17

Bibliographie *19*

Objectifs

La programmation est une activité fondamentale en informatique. La programmation peut être vue comme l'art de déterminer **un algorithme** (**une démarche**) pour résoudre un problème et d'exprimer cet algorithme au moyen d'un langage de programmation.

Ce module permet de savoir transcrire les différentes étapes de résolution d'un problème sous forme d'algorithme, de façon structurée et indépendante de toute contrainte matérielle ou logicielle.

Les **compétences** acquises sont :

- comprendre et analyser un algorithme préexistant ;
- analyser la situation : identifier les données d'entrée, de sortie, le traitement... ;
- mettre au point une solution algorithmique : comment écrire un algorithme en langage courant en respectant un code, identifier les boucles, les tests, des opérations d'écriture, d'affichage... ;
- valider la solution algorithmique par des jeux d'essais simples avec le langage pascale;

Introduction

le cours

Chapitre 4 : Procédures et fonctions

1). Les fonctions	10
2) Les procédures	11
3) Variables globales et de variables locales	12
4). Les paramètres	12
6). Exercices	17

• L'objectif principal de ce chapitre est l'acquisition des notions fondamentales des procédures et des fonctions.

• mettre au point une solution algorithmique en utilisant les procédures (les fonctions).

• valider la solution algorithmique par des jeux d'essais simples ;

• Saisir la différence entre procédures et fonctions

Un algorithme peut prendre une taille et une complexité croissante. De même des séquences d'instructions peuvent se répéter à plusieurs endroits.

Par exemple...

- Résoudre le problème suivant :
  - Écrire un programme qui affiche en ordre croissant les notes d'une promotion suivies de la note la plus faible, de la note la plus élevée et de la moyenne
- Revient à résoudre les sous-problèmes suivants :
  a. Remplir un tableau de naturels avec des notes saisies par l'utilisateur
  a. Afficher un tableau de naturels
  b. Trier un tableau de naturel en ordre croissant
  c. Trouver le plus petit naturel d'un tableau
  d. Trouver le plus grand naturel d'un tableau
  e. Calculer la moyenne d'un tableau de naturels
- Chacun de ces sous-problèmes devient un nouveau problème à résoudre
- Si on considère que l'on sait résoudre ces sous-problèmes, alors on sait "quasiment" résoudre le problème initial, donc écrire un programme qui résout un problème revient toujours à écrire des sous-programmes qui résolvent des sous parties du problème initial.

En algorithmique il existe deux types de sous-programmes

1. Les fonctions

2. Les procédures

A. 1). Les fonctions

Définition : 1.2). Définition

Une fonction est un sous-algorithme qui :

- A un **nom**
- Peut avoir **des paramètres**: Un paramètre est une variable particulière (globale) qui est associée à une variable ou constante (numérique ou définie par le programmeur) du programme appelant :
- retourne une valeur d'un certain **type**
- peut avoir besoin de **variables locale**
- composé **d'instructions**

Syntaxe : 1.3). Déclaration d'une fonction

```
Fonction nom_Fonct (liste de paramètres) : type
Variables identificateur : type
Début
Instruction(s)
Retourner Expression
Fin
```

Conseil

Après le nom de la fonction, il faut donner la liste des **paramètres** (s'il y en a) avec leur type respectif. Ces paramètres sont appelés **paramètres formels**. Leur valeur n'est pas connue lors de la création de la procédure.

Exemple : Une fonction minimum qui retourne le minimum de deux nombres

```
//Déclaration de la fonction Minimum
fonction minimum2 (a,b : Entier) : Entier
début
si a > b alors
retourner b
finsi
retourner a
fin
```

1.4). L'appel d'une fonction

On appelle une fonction par son nom, en lui fournissant les valeurs correspondant à ses paramètres (s'il y en a). Si la fonction renvoie une valeur, on peut la récupérer dans une variable ou l'utiliser directement.

Variable ← nom fonction (liste de paramètres)

//Appel de la fonction Minimum

fonction minimum3 (a,b,c : Entier) : Entier

début

retourner minimum2(a,minimum2(b,c))
fin
//Déclaration de la procédure minmax
procédure calculerMinMax3 (E a,b,c : Entier ; S m,M : Entier)
début
m<--minimum3(a,b,c)
M<-- maximum3(a,b,c)
fin

```
caractère c;
entier A, B;
début
     c ='O';
     tantque (c ≠ 'N') faire
          écrire "voulez vous continuer (O/N)";
          lire c;
          si (c ≠ 'N') alors
               écrire "donner un entier positif";
               lire A;
               écrire "donner un entier positif plus petit que le précédent";
               lire B;
               écrire "le PGCD de A et B est " + pgcd(A,B);
          finsi
     fintantque
fin
```

fun22

B. 2) Les procédures

1. 2.1). Définition d'une procédure

Définition : 2.1). Définition1

Une **procédure** est une série d'instructions regroupées sous un nom, qui permet d'effectuer des actions par un simple appel de la procédure dans un algorithme ou dans un autre sous-algorithme.
Une procédure est un sous-programme qui ne retourne pas de valeur

Syntaxe : 2.2). Déclaration d'une procédure

Procédure nom_proc (liste de paramètres)
Variables
identificateurs : type
Début
Instruction(s)
Fin procédure

Après le nom de la procédure, il faut donner la liste des **paramètres** (s'il y en a) avec leur type respectif. Ces paramètres sont appelés **paramètres formels**. Leur valeur n'est pas connue lors de la création de la procédure.

L'appel d'une procédure

Pour déclencher l'exécution d'une procédure dans un programme, il suffit de l'appeler. L'appel de procédure s'écrit en mettant le nom de la procédure, puis la liste des paramètres, séparés par des virgules. A l'appel d'une procédure, le programme interrompt son déroulement normal, exécute les instructions de la procédure, puis retourne au programme appelant et exécute l'instruction suivante.

Syntaxe

Nom_procédure (**liste de paramètres**)

Les paramètres utilisées lors de l'appel d'une procédure sont appelés **paramètres effectifs**. Ces paramètres donneront leurs valeurs aux paramètres formels.

Remarque : Remarque 1

Pour exécuter un algorithme qui contient des procédures et des fonctions, il faut commencer l'exécution à partir de la partie principale (**algorithme principal**)

Remarque : Remarque 2

Lors de la conception d'un algorithme deux aspects apparaissent :

- **La définition (déclaration)** de la procédure ou fonction.
- **L'appel de la procédure** au sein de l'algorithme principal.

C. 3) Variables globales et de variables locales

- Une **variable globale** est une variable déclarée à l'extérieur du corps de de la procédure (la fonction), et pouvant donc être utilisée n'importe où dans l'algorithme. On parle également de **variable de portée globale**
- Une **variable locale** est une variable qui ne peut être utilisée que dans la procédure (la fonction) ou le bloc où elle est définie. Elle s'oppose à **la variable globale** qui peut être utilisée dans tout l'algorithme.

D. 4). Les paramètres

Définition : 4.1). Les paramètres

Un **paramètre** est une donnée manipulée par une section de code (voir : sous-algorithme, fonction, procédure) et connue du code appelant cette section. On parle aussi d'**argument**. généralement, on distingue 2 types de paramètres : Les **paramètres formels** et **les paramètres effectifs**.

Définition : 4.2.1). Les paramètres formels

Les paramètres formels qui sont les valeurs que devra recevoir le sous-algorithme (Fonction/ Procédure) pour se mettre en route avec succès. On déclare les paramètres formels pendant la **déclaration du sous-programme**.

Définition : 4.2.2). Les paramètres effectifs

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 sous-algorithme dans l'algorithme principal.

Remarque : Passage de paramètres

On a vu plus haut qu'un **paramètre effectif** pouvait être une constante ou une variable. Lorsqu'il s'agit d'une variable, 2 cas de figures se proposent :

- Utiliser la valeur de la variable et à la sortie du sous-algorithme lui restituer cette valeur malgré les éventuelles modifications subies. On parle de **passage de paramètre par valeur**.passer un paramètre par valeur revient donc à n'utiliser que la valeur de la variable au moment où elle est passée en paramètre. À la fin de l'exécution du sous-programme, la variable conservera sa valeur initiale.
- Utiliser la variable elle-même et lui attribuer dans le bloc principal les modifications rencontrées dans le sous-algorithme. On parle de **passage de paramètre par adresse**. Un tel passage de paramètre se fait par l'utilisation du mot-clé **var** (uniquement sur le paramètre formel, et jamais sur un paramètre effectif).
- **Passage de paramètre par valeur**Syntaxe : PROCEDURE <nom_procédure> (param1 :type1 ; param2, param3 :type2)
- **Passage de paramètre par adresse**/Syntaxe : FONCTION <nom_fonction> (**VAR** param1 :type1 ; param2 :type2) : entier.

Exemple : Passage de paramètres

Écrire un algorithme dans lequel une fonction utilise le résultat d'une procédure 'Produit de 2 entiers' pour en calculer le carré. Calculer ensuite le carré de ce carré.

```
Fonction carre(var racine : réel) : réel
DEBUT
racine<--racine * racine
returne(racine)
FIN FONCTION
```

```
Procedure multiplier(nbre1, nbre2 : entier)
Variables
produit : réel
DEBUT
produit<--nbre1 * nbre2
Ecrire('Le résultat de ce produit est ', produit)
FIN PROCEDURE
```

```
Algorithme produit_carre
Variables
nbre1, nbre2 : entier
produit, carré: réel
DEBUT
Ecrire('entrer les deux nombres qu''il faudra multiplier')
Lire(nbre1, nbre2)
Multiplier(nbre1, nbre2)
carré<--carre(carre(produit))
Ecrire('Le carré de cette multiplication est ',
carre(produit))
```

```
Ecrire('Le carré de ce carré sera alors ', carré)
FIN
```

E. 6). Exercices

1. Exercice 1

Énoncé

Écrire un algorithme permettant à l'utilisateur de saisir un nombre entier positif. En se servant d'une fonction booléenne, l'algorithme principal affichera ensuite un message indiquant si cet entier est premier ou pas.

```
Fonction test(nbr :entier) : booléen
Variables
i : entier
test : booléen
Début
test<--Faux
pour i allant de 1 à (nbr/2) faire
si (nbr mode i = 0) alors
test<-- vrais
Fin si
Fin pour
Fin Fonction
```

```
Algorithme : test_impair
Variables
nmbr : entier
Début
Ecrire('entrer le nombre entier')
Lire (nmbr)
si (test(nmbr)=faux) alors
Ecrire ('ce nombre est impaire')
sinon
Ecrire ('ce nombre est paire')
Fin si
Fin
```

Bibliographie

[01] support du cours "informatique 1" Mezhoud Nassima, MCA, département électronique, université des frères Mentouri constantine

[01] Livre "ALGORITHMIQUE ET PROGRAMMATION POUR NON-MATHEUX", 2008 Christophe Darmangeat.

[02] Initiation `a l'algorithmique", Jaques Tisseau