Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Centre Universitaire Ahmed Zabana -Relizane

Institut : Des Sciences Exactes et Des Sciences de la Nature et de la Vie

Département : Physique

Support de Cours en Informatique1

(Bureautique et Technologie Web, Algorithmique)

Elaboré Par : Dr. Oufella Sarah

Année Universitaire 2018-2019

Avant-Propos

La notion d'algorithmique est très générale et dépasse le cadre de l'informatique. En effet nous utilisons tous des méthodes algorithmiques plus ou moins consciemment dans notre vie courante.

Ce polycopiéest rédigé à l'intention des étudiants de première année Sciences de la Matière (SM), cependant il s'adresse aussi à toute personne désireuse de maîtriser les bases essentielles de l'algorithmique.

Ce polycopié est le fruit d'une expérience de plusieurs années dans le domaine de l'algorithmique et de la programmation. Ilconstitue un support de cours pour des étudiants n'ayant aucune connaissance en programmation ouayant déjà une première expérience en programmation et qui veulent connaître davantage sur l'art de la programmation.

Le présent polycopié est scindé en deux grandes parties.

Partie1 : Bureautique et Technologie Web

Pour apprendre à programmer, il faut d'abord comprendre ce qu'est vraiment un ordinateur, comment il fonctionne et surtout comment il peut faire fonctionner des programmes, comment il manipule et stocke les données et les instructions, quelle est sa logique. Cette partie du fascicule survole des généralités sur l'informatique. Dans un premier temps l'architecture et le fonctionnement des ordinateurs sont passés en revue par la suite les systèmes d'exploitation ainsi que les réseaux informatiques sont abordés.

Partie2 : Introduction à l'Algorithmique

La seconde partie illustre l'essentiel du cours d'algorithmique, à savoir, le langage algorithmique qui couvre toutes les instructions de base constituant un algorithme,

les structures conditionnelles et itératives, les tableaux à une et à deux dimensions et enfin les fonctions et procédures.

A la fin de chaque chapitre le lecteur trouvera une série d'exercices corrigéspermettant de consolider ses acquis. La solution proposée à chaque exercice n'est pas unique et dans plusieurs cas ellen'est optimale car on a toujours privilégié l'apport pédagogique et la simplicité.

Nous espérons que ce polycopié réponde aux attentes des étudiants et qu'ils puissent constituer une première étape de ce vaste domaine des temps modernes qu'est l'informatique

Partie 1

Bureautique & Technologie Web

Chapitre 1 Généralités sur l'Informatique

1. Introduction

Le terme informatique est un néologisme proposé en 1962 par Philippe Dreyfus pour caractériser le traitement automatique de l'information : il est construit sur la contraction de L'expression « information automatique ». Ce terme a été accepté par l'Académie française en avril 1966, et l'informatique devint alors officiellement la science du traitement automatique de l'information. Le mot informatique n'a pas vraiment d'équivalent aux Etats-Unis où l'on parle de Computing Science (science du calcul) alors que Informatics est admis par les Britanniques Science (science du calcul) .

2. Définition

L'informatique, c'est un mot-valise, formé d'information et d'automatique. L'informatique c'est donc un traitement automatique de l'information (textes, nombres, images, sons, vidéos,...). Apparue au milieu du 20ème siècle, elle a connu une évolution extrêmement rapide. L'informatique est présente de tous les domaines notre vie quotidienne : industrie, gestion, calculs scientifiques et techniques, enseignement, télécommunication, jeux, etc…

3. Historique de l'évolution de l'informatique

Nous allons faire un petit rappel chronologique non exhaustif d'un certain nombre d'évènements majeurs ayant contribué au développement ou carrément à la fondation de l'informatique, telle que nous la connaissons aujourd'hui

- ✓ 780-850: Les premières bases de l'algorithmique établies par Abu Abdullah Muhammad bin Musa Al-Khawarizmi
- ✓ 1623-1662 : Pascal fabrique à 18 ans la première machine à additionner.
- ✓ 1792-1871 : Charles Babbage construit le premier automate comprenant une unité de calcul programmable.
- ✓ 1840 : Ada Lovelace (mathématicienne)nomme le processus logique d'exécution d'un programme :Algorithme, en l'honneur à Al-Khawarizmi

- ✓ 1939 : Von Neumann et ses collègues définissent les fondements mathématiques de l'ordinateur.
- ✓ 1947 : Apparition du transistor (laboratoires Bell Téléphone)
- ✓ 1950 : Invention de l'assembleur à l'université de Cambridge
- ✓ 1954 : IBM (International Business Machine) entre dans le marché des ordinateurs que l'on supposait ne pas devoir dépasser les 100 machines.
- ✓ 1956 : Description du premier langage évolué, le FORTRAN, qui permet aux scientifiques de développer eux-mêmes leurs programmes.
- ✓ 1959 : Passage à la deuxième génération de machines : circuits imprimés et transistors.
- ✓ 1963 : Les mini-ordinateurs apparaissent. Il s'agit d'ordinateurs destinés à assurer des tâches spécifiques et pouvant être incorporés dans des systèmes (avions, chaînes de montages etc...). Les firmes Digital Equipment et Hewlett-Packard occuperont ce marché, négligé par IBM.
- ✓ 1966 : Les circuits intégrés sont utilisés pour construire des ordinateurs de la troisième génération, plus fiables et moins chers .
- ✓ 1974 : Intel construit (presque par hasard) un circuit sans usage bien défini (le premier microprocesseur)
- ✓ 1976 : Le premier micro-ordinateur individuel de la quatrième génération est mis sur le marché par la firme Altair
- ✓ 1984 : Il ne faut plus savoir programmer pour utiliser un ordinateur, la façon dont la machine interagit avec son utilisateur subit une profonde évolution
- ✓ 1995 : Sur le plan des réseaux, IINTERNET devient un enjeu de société.
- ✓ 1997 : Des processeurs de plus en plus puissants voient le jour. Citons la série Pentium de Intel, le PowerPC d'IBM et le ALPHA de Digital.
- ✓ 2001 et après... Le 21ème siècle confirme l'omniprésence de l'ordinateur. L'outil devient incontournable et notre société ne peut plus s'en passer

4. Composants de l'ordinateur

L'ordinateur est une machine à traiter électroniquement, les données. Il permet l'automatisation de tâches et de calculs et leur exécution. L'ordinateur est divisé en deux parties : la partie matérielle « ***hardware*** » et la partie logicielle « ***software*** »:

4.1Partie matérielle

La partie matérielle d'un ordinateur (communément nommée Hardware) correspond aux pièces détachées qui ensemble forment l'ordinateur. Certaines de ces pièces sont internes (carte mère, processeur, carte graphique, etc.) et d'autres externes (clavier, souris, manette de jeu). C'est la partie physique du système informatique : constitués de l'unité centrale et des organes(les périphériques d'entrée/sortie.)

4.1.1 Unité Centrale

C'est le composant le plus important pour le fonctionnement de l'ordinateur, elle est composée généralement d'une carte mère sur laquelle on trouvera la mémoire centrale et le microprocesseur ou CPU (Central Processing Unit).

L'unité centrale a pour rôle d'exécuter les programmes. Elle est composée de l'unité arithmétique et logique(UAL) et de l'unité de contrôle(UC).

– L'unité arithmétique et logique : réalise une opération élémentaire (addition ,soustraction, multiplication, etc.) du processeur à chaque top d'horloge.

– L'unité de commande contrôle les opérations sur la mémoire (lecture/écriture) et les opérations à réaliser par l'UAL selon l'instruction en cours d'exécution. Pour pouvoir effectuer les opérations sur des données et exécuter des programmes l'UC doit disposer d'un espace de travail. Cette espace de travail s'appelle la mémoire centrale.

- **Le microprocesseur**

Le microprocesseur fut (aussi appelé processeur ou CPU pour Central Processing Unit) inventé en 1971 par Ted Hoff pour le compte d'Intel. Un microprocesseur est une puce électronique qui se présente sous la forme d'une boîte, dont la taille et la forme varient selon son degré de perfectionnement et la technologie qui a permis sa fabrication. De nos jours le nouveau microprocesseur d'Intel, l'Ivy Bridge contient1, 4 milliard de transistors concentrés sur un die d'une surface de160 $mm2$.

- **Mémoire Centrale (interne)**

Elle représente l'espace de travail de l'ordinateur. C'est l'organe principal de stockage des informations utilisées par le processeur. Pour exécuter un programme il faut le charger (copier) dans la mémoire centrale. Le temps d'accès à la mémoire centrale et sa capacité sont deux éléments qui influent sur le temps d'exécution d'un programme (performance d'une machine). Il existe différents types de mémoire

- ***Mémoire RAM* (*Random Access Memory*)**

Appelée aussi volatile ou vive, cette mémoire est temporaire, elle permet de stocker les données et les programmes en cour d'exécution des que vous éteignez l'ordinateur, son contenu est effacé. Plus il y a de RAM, plus l'ordinateur est puissant.

- ***Mémoire ROM (Read Only Memory)***

Appelée aussi non volatile ou morte, cette mémoire n'est utilisée que pour en lire des données initialement stockées, elle conserve des données nécessaire pour le démarrage du système et ne s'efface pas lors de sa mise hors tension.

- ***Mémoire Cache (antémémoire)***

Elle sert à stocker les instructions et données fréquemment demandées par le microprocesseur. Au début, les mémoires caches résidaient à l'extérieur du processeur actuellement elles sont intégrées dans la puce du microprocesseur. Elle offre des temps d'accès de l'ordre de 2 à 5 nanosecondes, sa taille est de l'ordre de quelques Mégaoctets (3 à 12 Mo).

- **Disque dur**

Le disque dur est l'organe du PC(Personal Computer) servant à conserver les données de manière permanente, même lorsque le PC est hors tension, contrairement à la mémoire centrale, qui s'efface à chaque redémarrage de l'ordinateur, c'est la raison pour laquelle on parle de **mémoire de masse**, sa capacité exprimée en Go, mais aujourd'hui sa capacité de stockage peut atteindre plusieurs Terra Bytes (Tb).

En plus de tous ces composants s'ajoute les carte d'extension : carte son, carte réseau, carte graphique.

4.1.2 Les périphériques

C'est tout accessoire qu'on peut connecter à un ordinateur. On peut distinguer

- ✓ ***Les périphériques d'entrée*** : Ils permettent de véhiculer les informations du monde extérieur vers la mémoire de l'ordinateur. **Exemple** : le clavier, la souris, le scanner, le microphone, lecteur de codes-barres, crayon à lecture optique..
- ✓ ***Les périphériques de sorties*** : Ils permettent de véhiculer les informations de la mémoire de l'ordinateur vers le monde extérieur. **Exemple** : l'écran, l'imprimante, les enceintes audio…

4.2Partie logicielle (software)

On désigne par software la partie logicielle, qui peut être : des systèmes d'exploitation, des langages de programmation, ou bien de simples logiciels spécialisés.

Un logiciel : est programme, un ensemble d'instruction destiné au fonctionnement de l'ordinateur

On distingue deux types de logiciel

-***Système d'exploitation*** : il est le tout premier programme que peut contenir un ordinateur. Il assure la communication entre le processeur, les périphériques et l'utilisateur. Exemple : MS-DOS (Microsoft Disk Operating System), Windows 95, Windows 98, 2000, XP,Vista,7,8,10 Unix et Linux.

-***Logiciel d'application*** *:* à chaque logiciel d'application correspond une tache précise.

Exemple : MS Word……..traitement de texte

Real one player……..pour lire la musique

Photoshop……..logiciel de retouche photos et d'images

5. Fonctionnement d'un ordinateur

Il est très important de savoir que l'ordinateur n'est pas une machine intelligente, par contre c'est une machine très puissante, capable d'exécuter les instructions sur lesquelles il a été programmé. L'ordinateur est une machine dotée de composants lui permettant :

- ✓ La réception et le stockage des informations ;
- ✓ L'exécution d'opérations complexes ;
- ✓ L'affichage des résultats

L'exécution d'un programme se déroule comme suit :

- ➢ Le chargement du programme et des données en MC ;
- ➢ Les instructions sont ramenées une par une et séquentiellement à l'UCC, qui les décode et déclenche le traitement approprié en donnant des commandes à l'UAL et à la Mémoire centrale (MC) .Cependant l'exécution peut nécessiter des appels aux unités d'E/S.

Le fonctionnement d'un ordinateur est illustré par la figure suivante.

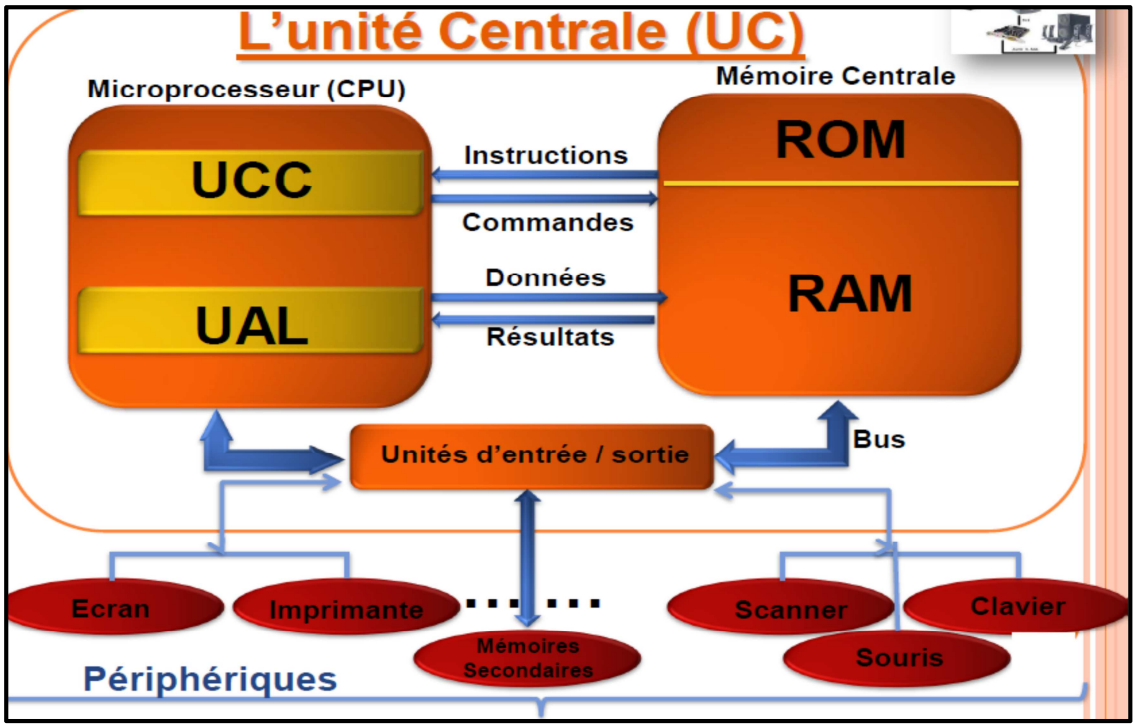


Figure 1.1 Fonctionnement d'un ordinateur

Chapitre2: Les Systèmes d'Exploitation

1. Définition

Le système d'exploitation (SE, en anglais Operating System ou OS) est un ensemble de programmes responsables de la liaison entre les ressources matérielles d'un ordinateur et les applications de l'utilisateur (traitement de texte, jeu vidéo, …). C'est le programme fondamental des programmes systèmes. Il contrôle les ressources de l'ordinateur et fournit la base sur laquelle seront construits les programmes d'application.

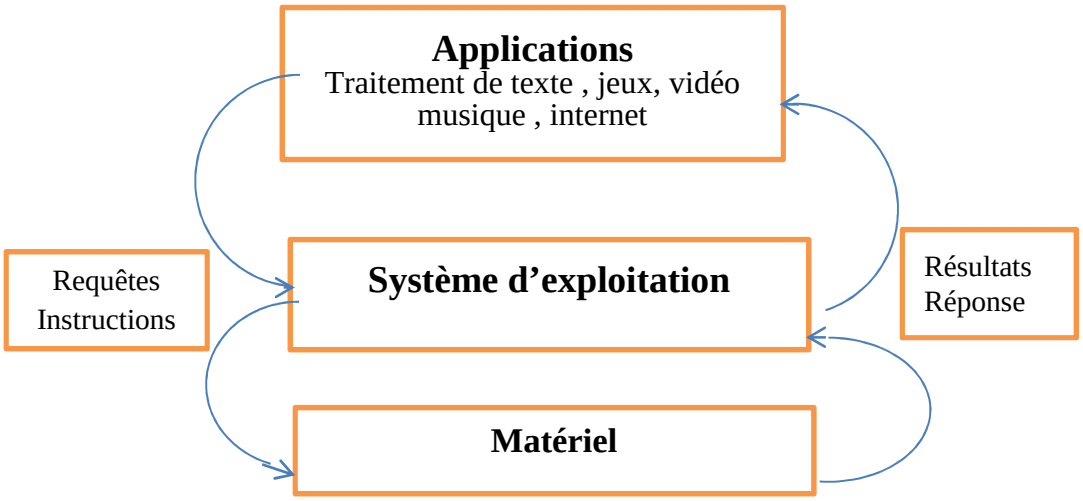


Figure2.1 illustration du rôle d'un système d'exploitation

Trois systèmes d'exploitation se distinguent :

- **Mac OS** (pour Macintosh Operating System) est le nom du système d'exploitation d'Apple pour ses ordinateurs Macintosh
- **Linux** est une version librement diffusable de l'Unix. C'est l'un des systèmes d'exploitation le plus souples
- **Windows** demeure le système d'exploitation le plus utilisé et détient un quasi-monopole sur les ordinateurs personnels avec près de 90 % de part de marché

2. Fonctions d’un système d’exploitation

Un Système d’Exploitation (SE) résout les problèmes relatifs à l’exploitation de l’ordinateur en garantissant :

- La gestion efficace, fiable et économique des ressources physiques de l’ordinateur (notamment les ressources critiques telles que processeur, mémoire...).
- Possibilité de détecter, récupérer les erreurs et en informer l’utilisateur.
- Gestion de l'exécution des applications : le système d'exploitation est chargé de la bonne exécution des applications en leur affectant les ressources nécessaires à leur bon fonctionnement.
- Gestion de l’ensemble des échanges entre le microprocesseur, la mémoire centrale et les divers périphériques (écran, clavier, souris, etc.) ;
- Gestion des droits : le système d'exploitation est chargé de la sécurité liée à l'exécution des programmes en garantissant que les ressources ne soient utilisées que par les programmes et les utilisateurs possédant les droits adéquats.
- Gestion des fichiers : le système d'exploitation gère la lecture et l'écriture dans le système de fichiers et les droits d'accès aux fichiers par les utilisateurs et les applications.

3. Architecture d’un Système d’exploitation

Le système d'exploitation est composé d'un ensemble de logiciels permettant de gérer les interactions avec le matériel. Parmi cet ensemble de logiciels on distingue généralement les éléments suivants

- **Noyau (ou kernel en anglais)** : assure la gestion des processus (programmes exécutés par le microprocesseur). C’est réellement le cœur du système d’exploitation.

  Le noyau assure ces fonctionnalités :

  - gestion des périphériques (au moyen de pilotes).
  - gestion des files d'exécution (aussi nommée processus) .
  - attribution de la mémoire à chaque processus.
  - ordonnancement des processus (répartition du temps d'exécution sur le ou les processeurs).

- gestion des fichiers (au moyen de systèmes de fichiers)
- gestion des protocoles réseau (TCP/IP, IPX, etc.)

✓ **L'interpréteur de commande (en anglais shell,)** : Permet la communication avec le système d'exploitation par l'intermédiaire d'un langage de commandes, afin de permettre à l'utilisateur de piloter les périphériques en ignorant tout des caractéristiques du matériel qu'il utilise.

✓ **Le système de fichiers** (**en anglais «file system», noté FS)** : permet d'enregistrer les fichiers dans une arborescence. Le système de gestion des fichiers assure plusieurs fonctions :

• **Manipulation des fichiers** : créer/détruire des fichiers, insérer, supprimer et modifier un article dans un fichier ;

• **Allocation d'espace** : les fichiers étant de taille différente et cette taille pouvant être dynamique, le SGF(Système gestion de Fichier) alloue à chaque fichier un nombre variable de blocs mémoire.

• **Localisation des fichiers** : chaque fichier possède un ensemble d'informations descriptives (nom, adresse…) qui permet de le retrouver.

• **Sécurité et contrôle des fichiers** : un nom et une clé de protection sont associés à chaque fichier afin de le protéger contre tout accès non autorisé ou mal intentionné lors du partage des fichiers.

4. Types de Systèmes d'exploitation

On distingue plusieurs types de systèmes d'exploitation qui peuvent être classifiés selon les services rendus, leur architecture, leur capacité à évoluer et l'architecture matérielle qui les supporte.

4.1 Selon les services rendus

4.1.1 Systèmes mono-tâches/ multi tâches

✓ ***Mono-tâche* :** Il ne gère qu'une seule tâche à la fois (un seul programme).
Quand le programme est lancé, il utilise seul les ressources de la machine (processeur et mémoire notamment) et ne rend la main à l'O.S. qu'en fin d'exécution ou en cas d'erreur. **Exemple** : MS-DOS (MicroSoft Disk Operating System).

- ✓ ***Multi-tâche* :** La capacité du système à pouvoir exécuter plusieurs tâches « processus » simultanément ; par exemple effectuer une compilation et consulter le fichier source du programme correspondant. **Exemple** UNIX.

4.1.2 mono/multiutilisateurs

- ✓ ***Mono-utilisateur*** : le système ne peut gérer qu'un seul utilisateur. **Exemple** Windows95
- ✓ ***Multi-utilisateurs*** : le système est capable gérer un panel d'utilisateurs utilisant simultanément les mêmes ressources matérielles. **Exemple** Unix

4.2 Selon leur architecture

4.2.1 Système centralisé

Dans ce cas, le système ne gère que les ressources de la machine sur laquelle il est présent. C'est le cas de la majeure partie des systèmes installés**. Exemple :** Linux

4.2.2 Système réparti

C'est un ensemble de machines autonomes connectées par un réseau et équipées d'un logiciel dédié à la coordination des activités du système ainsi qu'au partage de ses ressources. Le système s'exécute sur un ensemble de machines sans mémoire partagée, mais qui apparaissent à l'utilisateur comme une seule et unique machine. **Exemple** les versions d'UNIX sont un peu reparties

4.3 Selon l'architecture matérielle qui les supporte

4.3.1 Systèmes multiprocesseurs

On appelle SMP (**S**ymetric **M**ulti**p**rocessor) une architecture **parallèle** dans laquelle tous les processeurs accèdent à un espace mémoire partagé. Les ordinateurs multiprocesseurs permettent un parallélisme de tâches, où un processus peut être exécuté sur chaque processeur. **Exemple** : le système de sécurité qui détecte les coupures d'énergie

4.3.2 Systèmes embarqués

Sont prévus pour fonctionner sur des machines de petite taille, telles que des PDA ou des appareils électroniques autonomes (sondes spatiales, robot, ordinateur de bord de véhicule, etc.), possédant une autonomie réduite. Ces systèmes électroniques et informatiques, souvent temps réel, sont spécialisés dans une tâche bien précise. **Exemple** Windows Mobile

4.3.3 Systèmes temps réels

Les systèmes temps réel (**R**eal **T**ime **S**ystem), essentiellement utilisés dans l'industrie, sont des systèmes dont l'objectif est de fonctionner dans un environnement contraint temporellement. **Exemple** RTLinux (RealTime Linux)

4.4 Selon leur capacité à évoluer

4.4.1 Systèmes fermés (ou propriétaires)

C'est un Système avec extensibilité réduite *:* Quand on veut rajouter des fonctionnalités à un système fermé, il faut remettre en cause sa conception et refaire une archive (système complet). Il n'y a aucun ou peu d'échange possible avec d'autres systèmes de type différent, voir même avec des types identiques. **Exemple** MS-DOS

4.4.2 Systèmes ouverts

Système avec extensibilité accrue : Il est possible de rajouter des fonctionnalités et des abstractions sans avoir à repenser le système et même sans avoir à l'arrêter sur une machine. **Exemple** LINUX dans toutes ses distributions

Chapitre3 Réseau et Internet

1. Notion de réseau informatique

Les Réseaux informatiques sont nés du besoin de faire communiquer des terminaux distants avec un site central puis des ordinateurs entre eux.
Le réseau informatique est un ensemble d'ordinateurs et des équipements reliés entre eux, afin qu'ils puissent échanger des informations et partager des ressources

1.2. Avantages d'un réseau informatiques

Un réseau informatique peut servir plusieurs buts distincts :
Le partage des ressources logicielles (des fichiers et des applications) ;
Le partage des ressources matérielles (imprimante, scanner, …… ;
La communication entre personnes (courrier électronique, discussion en direct, ...) ;
Accès aux données rapidement.
- La garantie de l'unicité et de l'universalité de l'accès à l'information (bases de données en réseau)

1.3. Les différents types de réseaux

On peut distinguer différent types de réseaux selon plusieurs critère tel que la taille de réseau, sa vitesse de transfert des données et aussi son étendu. Les réseaux informatiques peuvent être classifiés en trois familles

- ✓ **LAN (Local Area Network)** ou **réseau local**

Il s'agit d'un ensemble d'ordinateurs appartenant à une même organisation et reliés entre eux dans une petite aire géographique par un réseau, souvent à l'aide d'une même technologie (la plus répondue étant Ethernet). La vitesse de transfert de données d'un réseau local peut s'échelonner entre 10 Mbit/s et 1Gbit/s. La taille d'un réseau local peut atteindre jusqu'à 100 voire 1000 utilisateurs.

- ✓ **MAN (Metropolitan Area Network) ou réseau métropolitain**

Interconnecte plusieurs LAN géographiquement proches (au maximum quelques dizaines de km) à des débits importants, supérieur à 100 Mbits/s. Ainsi Un MAN est formé de

commutateurs ou de routeurs interconnectés par des liens hauts débits (en général en fibre optique) .

- ✓ **WAN (Wide Area Network) ou réseau étendu**

Interconnecte plusieurs LAN et MAN à travers de grandes distances géographiques. Les WAN fonctionnent grâce à des routeurs qui permettent de choisir le trajet le plus approprié pour atteindre un nœud du réseau. Le plus connu des réseaux étendus est Internet.

2. Architecture des réseaux

On distingue également deux catégories de réseaux dans les entreprises

- ✓ **Les réseaux Post à Post (peer to peer= P2P)**

Sur un réseau post à post, les ordinateurs sont connectés directement l'un à l'autre et il n'existe pas d'ordinateur central. L'avantage majeur d'une telle installation est son faible coût en matériel (les postes de travail et une carte réseau par poste). En revanche, si le réseau commence à comporter plusieurs machines il devient impossible à gérer

- ✓ **Les réseaux client-serveur**

Sur un réseau à architecture client/serveur, tous les ordinateurs (client) sont connectés à un ordinateur central (le serveur du réseau), une machine généralement très puissante en terme de capacité. Elle est utilisée surtout pour le partage de connexion internet et de logiciels centralisés, ce type d'architecture est plus facile à administrer lorsque le réseau et important car l'administration est centralisé mais elle nécessite un logiciel couteux spécialisé pour l'exploitation du réseau.

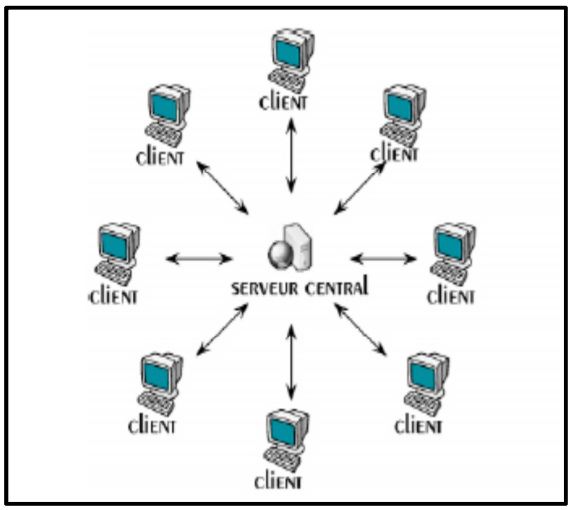


Figure3.1 Architecture client/serveur

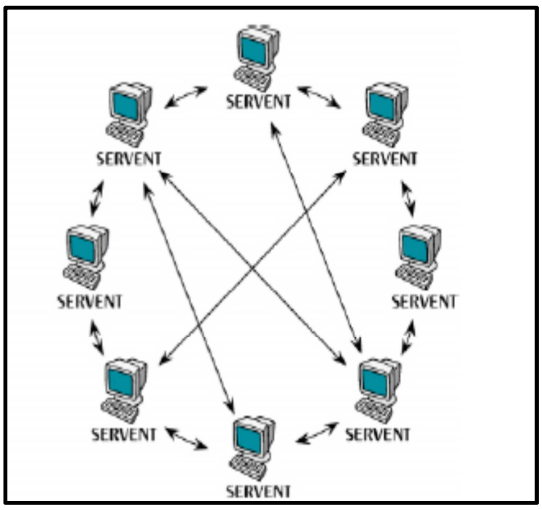


Figure3.2 Architecture peer to peer

3. Topologies des réseaux

La façon de laquelle les ordinateurs sont interconnectés physiquement est appelée topologie physique. Il existe plusieurs topologies physiques parmi les plus connues

- ✓ **Topologie en bus**

Dans cette topologie, les ordinateurs sont disposés et reliés de part et d'autre par un câble principal appelé bus. Le support de transmission utilisé dans ce cas est le câble coaxial. Dans cette topologie, lorsqu'un ordinateur envoie une information, tous les autres ordinateurs du réseau reçoivent l'information mais seule la machine à qui l'information est destinée va l'utiliser**.**

- ✓ **Topologie en anneau**

Dans cette topologie, les ordinateurs sont connectés à une boucle et communiquent chacun à leur tour. Les informations circulent dans une direction unique, d'un ordinateur à un autre.

- ✓ **Topologie en étoile**

Dans cette topologie, les ordinateurs du réseau sont reliés à un équipement central appelé concentrateur (hub) ou un commutateur (Switch), qui a pour rôle d'assurer la communication entre les différents ordinateurs connectés.

- ✓ **Topologie en Maille**

Dans cette topologie, chaque ordinateur est directement relié à tous les autres. Ainsi lorsqu'un ordinateur veut envoyer une information à un autre celui-ci le fait de façon directe sans passer par un équipement spécifique

- ✓ **Topologie en Arbre**

Une topologie arborescente est une combinaison des différentes autres topologies ; elle peut reposer à la fois sur des topologies en bus, en étoile et en anneau.

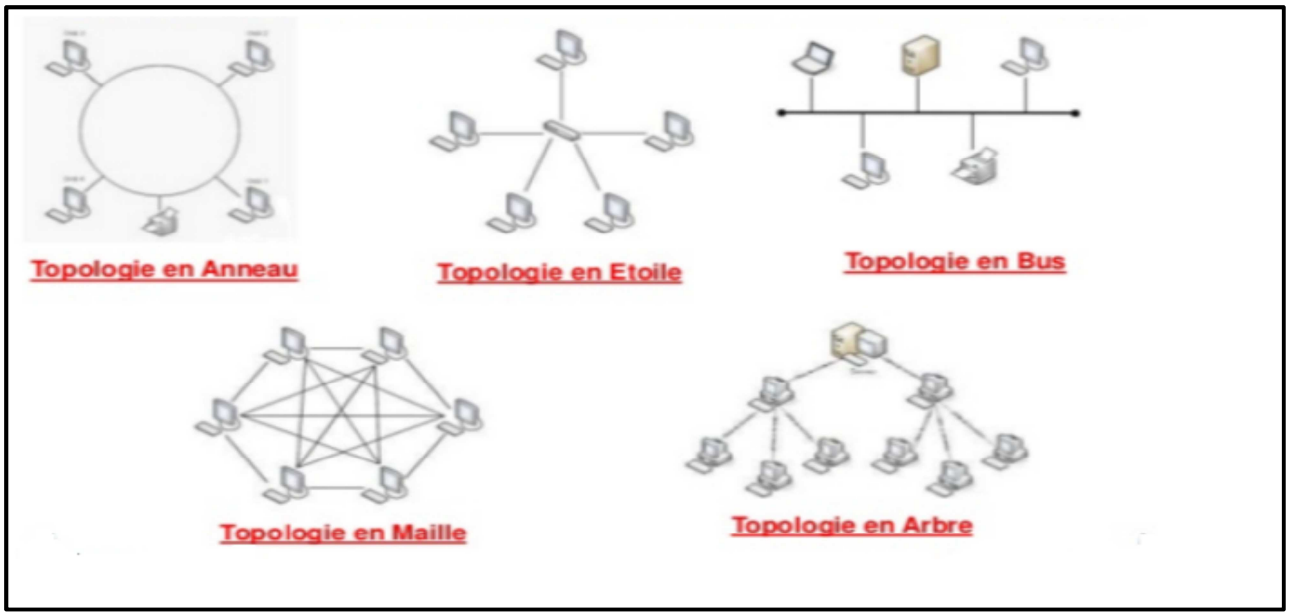


Figure3.3 Topologies des réseaux

4. Réseaux informatiques sans fil

Un réseau sans fil (en anglais Wireless network) est un réseau dans lequel au moins deux terminaux (ordinateur, PDA, téléphone portable...) peuvent communiquer **sans liaison filaire**. On distingue les catégories suivantes :

- ✓ **Réseau personnel sans fil ou WPAN** (Wireless personnel Area Network)

Exemple : Bluetooth

- ✓ **Réseau local sans fil ou WLAN** (Wireless Local Area Network)

Exemple : Wifi

- ✓ **Réseau métropolitain sans fil** ou WMAN (Wireless Metropolitan Wide Area Network)

Exemple : Wimax

- ✓ **Réseau étendu sans fil** ou WWAN ((Wireless Wide Area Network)

Exemple : 3G ;4G…..

5. Internet

5.1Définition Internet

Internet est un ensemble de réseaux interconnectés utilisant une suite protocolaire appelée TCP/IP (Transmission Control Protocol/Internet Protocol) pour échanger des informations à travers le monde. En termes simples : Internet est un immense réseau d'ordinateurs qui peuvent communiquer entre eux en utilisant TCP/IP.

Internet permet échanger des informations en toute liberté et rend accessible au public des services comme le courrier électronique et le web. Chaque ordinateur connecté directement sur internet possède un numéro d'identification unique (appelée adresse IP) et peut envoyer et recevoir des informations avec n'importe quel autre ordinateur ou machine possédant une adresse IP .

5. 2 Definition WEB (WWW: World Wide Web)

Le web n'est pas Internet. Internet comme nous l'avons vu est un vaste ensemble de réseaux interconnectés, « le réseau mondial » ou « le réseau des réseaux ».

Le web n'est qu'une application parmi les applications mises en œuvre sur Internet.

World Wide Web (ou WWW ou Web) est constitué de milliards de documents électroniques liés les uns aux autres, comme les fils d'une toile d'araignée. Ces documents sont stockés sur des serveurs, répartis dans le monde entier. Le web est associé au protocole **http,** qui permet de lire des pages web d'un serveur et les afficher sur le navigateur du client.

5.3 Les pages web

Une page web est un document électronique écrit dans un langage informatique appelé **HTML**. Une page peut contenir du texte, des graphiques, de la vidéo, des animations, du son et des éléments interactifs tels que des formulaires à remplir directement sur l'ordinateur.

- ✓ Chaque page possède une adresse unique, appelée URL (Uniform Ressource Locator) pour identifier son emplacement sur le serveur (ex : **http://www.google.fr/**)
- ✓ Les pages web contiennent souvent des liens hypertextes (textes ou images) qui renvoient sur d'autres pages web.

5.4 Les Sites Web

Un site web est un ensemble de pages web reliées entre elles et traitant d'un même thème (personne, entreprise, organisation ...). La première page d'un site est la page d'accueil (*Home page*), elle sert le plus souvent de sommaire et indique le contenu du site, elle contient des liens hypertexte sur lesquels il suffit de cliquer pour accéder aux autres pages.

Un site web est hébergé sur un serveur web appelé aussi serveur **http** parce qu'il utilise le protocole (programme) **http** pour fournir des pages web en réponse à des requêtes fournis par des clients web.

5.5 Langage HTML

HTML (Hyper Text Markup Langage) est un des langages d'écriture utilisé pour créer des pages web.

Forme des fichiers HTML

Toutes les pages html ont la structure suivante :

```
<html>
<head>
<title>
</title>
</head>
<body>
</body>
</html>
```

-La balise <**html**> et </**html**> délimite l'ensemble de la page htm.

-La balise <**head**> et </**head**> l'en-tête du document, qui contient les informations permettant aux moteurs de recherche de trouver votre page

-La balise <**title**> et </**title**> : définit le titre de la page

-La balise <**body**> et </**body**> délimite le texte principale de la page,

Partie 2

Introduction à l'Algorithmique

Chapitre 4 : Algorithme : Notions de Base

1. Introduction

L' origine du mot « algorithme » tire son nom du mathématicien Mohammed Al-Khwarizmi (780 - 850) géographe, astrologue et astronome musulman arabe dont les écrits ont permis l'introduction de l'algèbre en Europe.

Al-Khwarizmi a écrit en langue arabe le plus ancien traité d'algèbre baptisé « Abrégé de calcul par la complétion et la simplification » dans lequel il décrivait des procédés de calcul à suivre étape par étape pour résoudre des problèmes ramenés à des équations.

2. Définitions

2.1 Algorithme

Un algorithme peut être défini comme

- Un ensemble de règles opératoires dont l'enchaînement permet de résoudre un problème au moyen d'un nombre fini d'opérations.
- Une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes.

-Comment résoudre un problème ?

Pour résoudre un problème il faut passer par les étapes suivantes

- ✓ Comprendre l'énoncé du problème
- ✓ Préciser les données d'entrées (données nécessaires à la résolution du problème)
- ✓ Préciser les données de sorties (résultats)
- ✓ Déterminer les étapes de transformation des données en résultats (traitement)
- ✓ Elaboration d'un algorithme

-Exemple d'un algorithme :

Algorithme préparation d'un gâteau

Entrées : œufs, farine levure, lait, noix de coco, chocolat

Traitement :

- Mettre les ingrédients liquides dans un récipient
- Ajouter les ingrédients solides
- Battre le mélange
- Faire chauffer le four
- Cuire le gâteau pendant 30 minutes
- Napper le gâteau avec du chocolat fondu

Sortie : gâteau au chocolat

2.1.1 Caractéristiques d'un algorithme

L'algorithme est un moyen pour résoudre un problème, il doit être :

- Lisible : compréhensible même par un non informaticien
- Précis : il décrit un traitement sur un ensemble fini de données, avec un ensemble fini de traitement.
- Structuré : composé de différentes parties facilement identifiables.
- Complet : il doit tenir compte de tous les cas possibles.
- Efficace : idéalement, un algorithme doit être conçu de telle sorte qu'il se déroule en un temps minimal et qu'il consomme un minimum de ressources.

2.2 Algorithmique

L'algorithmique est la science des algorithmes. C'est l'art de construire des algorithmes de caractériser leur validité, leur robustesse, leur réutilisabilité, leur complexité ou leur efficacité. L'algorithmique est la logique d'écrire des algorithmes via un langage algorithmique appelé pseudo-code. Ce langage est un compromis entre un langage naturel et un langage de programmation, il utilise un ensemble de mots clés et permet de décrire de manière complète et claire les objets manipulés par l'algorithme ainsi que l'ensemble des actions à exécuter sur ces objets.

2.3 Programme

Un programme informatique est une séquence d'instructions écrites dans un langage de programmation, en d'autre terme c'est la traduction de l'algorithme dans un langage de programmation.

2.4 Langage de programmation

Un langage de programmation est un code de communication permettant à un être humain de dialoguer avec la machine, il donne à l'utilisateur (programmeur) la possibilité de rédiger un programme pour arriver à un résultat. Il existe plusieurs langages de programmation **Exemple**: *Pascal, Fortran, C, C++, Java….*

3. Les objets manipulés dans un algorithme

Dans un algorithme on manipule des objets simples et structurés. On distingue deux types d'objets :

- Des objets qui peuvent varier durant le déroulement d'un algorithme**(Variables)**
- Des objets qui ne subissent aucune variation lors du déroulement d'un algorithme**(Constantes)**

3.1 Caractéristiques des objets

Un objet possède trois caractéristiques : le nom (identificateurs), le type et la valeur.

- **L'identificateur (Nom)**

C'est le nom (identificateur) de l'objet, il sert à le désigner dans l'algorithme, il est représenté par une suite de caractères alphanumériques [0.9]+[a, z]+[A,Z] . Il ne doit pas contenir des signes de ponctuation ni d'espaces, et il commence par une lettre.

- **La valeur**

C'est le contenu de l'objet. Cette valeur peut changer ou non pendant l'exécution de l'algorithme.

- **Le type** :

Caractérise l'ensemble des valeurs permises pour un objet et les opérations qui lui sont autorisées. On distingue 5 types standards.

- ***Le type Entier*** : Une variable est dite entière si elle prend ses valeurs dans l'ensemble Z , **exemple** : âge, numéro d'inscription…..
- ***Le type Réel*** ou (décimal) : Une variable est dite réelle si elle prend ses valeurs dans l'ensemble IR .**Exemple** : poids, taille, masse…..
- ***Le type Booléen*** : Une variable (booléenne) logique ne peut prendre que deux valeurs "Vrai" ou "Faux". Elle intervient surtout dans l'évaluation d'une condition. **Exemple** : marié ou non….
- ***Le type Caractère*** : Regroupe les lettres, chiffres, caractères spéciaux, opérateurs exemple : section d'études….'A', 'b', '1', '?', '<'. Un caractère sera toujours noté entre des guillemets.
- ***Le type Chaîne de caractères*** : Représente toute suite de caractères. Le contenu de ces variables est noté entre guillemets. **Exemple** : adresse, nom, code postale.

3.2 Les variables

Une variable est une entité qui contient une information. Les variables jouent le rôle de "tiroirs" dans lesquels on place une valeur durant l'exécution de l'algorithme, elles possèdent :
-un nom, on parle d'identifiant
-une valeur qui change
-un type qui caractérise l'ensemble des valeurs que peut prendre la variable.

3.2.1 Déclaration des variables

Toute variable utilisée dans un algorithme doit faire l'objet d'une déclaration préalable. La déclaration de variables est effectuée par la forme suivante

Syntaxe

Variables

Nom variable1 : type

Remarque

Les variables du même type peuvent être déclarées dans une même ligne et séparées par des virgules.

Exemple

Variables

Age, Compteur : entier {Age et Compteur sont deux variables de type entier }

Note, Moy : réel { Note et Moy sont deux variables de type réel}

OK: booléen {OK est une variable de type booléen }

Nom: chaîne de caractères { Nom est une variable de type chaîne de caractères}

3.3 Les constantes

Une constante est un objet qui a un nom fixe, un type fixe et une valeur fixe. Une constante est invariable durant l'exécution d'un algorithme contrairement aux variables.

3.3.1 Déclaration des constantes

Toute constante utilisée dans un algorithme doit faire l'objet d'une déclaration préalable.

Une constante doit toujours recevoir une valeur dès sa déclaration. La déclaration de constantes est effectuée par la forme suivante

Syntaxe

Constantes

Nom- constante1 = valeur1

Exemple :

Constantes

x=2 { x est une constante entière sa valeur est égale à 2}

Pi =3.14 {Pi est une constante réelle sa valeur est égale à 3.14}

4. Expressions sur les objets

Une expression est une combinaison d'opérateur(s) et d'opérande(s) qui sont évalués pour donner un résultat durant l'exécution de l'algorithme.

4.1Opérandes

Une opérande est une entité (variable, constante ou expression) utilisée par un opérateur .

4.2 Opérateurs

Un opérateur est un symbole d'opération qui permet d'agir sur des variables ou de faire des calculs .Un opérateur est associé à un type de donnée et ne peut être utilisé qu'avec des variables, des constantes, ou des expressions de ce type.

Exemple

Dans «8+3 », «+» désigne l'opérateur ; «8 » et «3 » sont les opérandes.

4.2.1 Type operateurs

Il existe plusieurs types d'opérateurs résumés dans le tableau suivant

Nature	**Variables utilisées**	**Notation**	**Signification**
Opérateurs arithmétiques	Entier Réel	+	addition
		-	soustraction
		*	multiplication
		/	division
		div	Division entière
		mod	Reste de la division
Opérateurs logiques (booléens):	Booléen et Entiers	et	fonction et
		ou	fonction ou
		non	fonction non
		non et	fonction non et
		non ou	fonction non ou
Operateur de concaténation	Chaîne de caractères	+	concaténation
Opérateurs de comparaison	Entier Réel Booléen Caractère Chaîne de caractères	>	supérieur
		<	inférieur
		>=	supérieur ou égal
		<=	inférieur ou égal
		=	égal
		≠	différent

Table4.1 Type Operateurs

4.2.2 Priorité des operateurs

A chaque opérateur est associée une priorité. Lors de l'évaluation d'une expression, la priorité de chaque opérateur permet de définir l'ordre d'exécution des différentes opérations Dans la majorité des langages de programmation on utilise les règles suivantes :

1) Les parenthèses et on commence par les plus internes.

2) Le NON logique, et le – unitaire

3) *, /, MOD , DIV , ET logique

4) + , - , OU logique

5) Les opérateurs de relations ($\neq \leq < \geq > =$)

6) En cas d'égalité des priorités, on commence par l'opérateur le plus à gauche

Exemple

10+(4*4) div 2 = 10 + 16 div 2 = 10 + 8 = 18

17 mod (5 div 2)= 17 mod 2 = 1

Remarques

- Un opérateur peut être unaire ou binaire :
  - Unaire s'il n'admet qu'un seul opérande, par exemple l'opérateur non
  - Binaire s'il admet deux opérandes, par exemple l'opérateur +
- Une expression est évaluée de gauche à droite mais en tenant compte des priorités des opérateurs.
- S'il y a des parenthèses on commence par évaluer les plus internes.
- Un algorithme peut utiliser des fonctions standards telles que SIN( ), COS() ou encore SQRT() qui permet de calculer la racine carré

5. Les actions élémentaires

Nous distinguons trois types d'instructions élémentaires, l'affectation ainsi que la lecture et l'écriture qui permettent à la machine de communiquer avec l'utilisateur. Dans ce qui suit chacune de ces actions est définie.

5.1 L'affectation

L'affectation est l'action élémentaire dont l'effet consiste à attribuer une valeur à une variable (c'est-à-dire remplir ou modifier le contenu d'une zone mémoire). Cette valeur peut elle-

même être une variable, le résultat d'une expression arithmétique ou logique ou une constante.

L'affectation est réalisée au moyen de l'opérateur ←, elle est représentée par une flèche.

Syntaxe

Nom variable← Expression

« Expression » peut être :

- ✓ Une constante.
- ✓ Une autre variable.
- ✓ Le résultat d'une fonction.
- ✓ Le résultat d'un calcul

Exemple :

X←3 {Signifie mettre la valeur 3 dans la case identifiée par X. A l'exécution de cette instruction, la valeur 3 est rangée en X (nom de la variable)}.

A←B { Signifie mettre le contenu de la variable B dans la case identifiée par A.

R ← SQRT(nombre) {Signifie mettre le résultat de la racine carré de la variable nombre dans la case identifiée par R}

C ← ((A + B) *3) { Lors d'une affectation, l'expression de droite est évaluée et la valeur trouvée est affectée à la variable de gauche C.

Remarques

-La valeur ou le résultat de l'expression à droite de l'affectation doit être de même type ou de type compatible avec celui de la variable à gauche.

-Dans la plupart des langages de programmation on peut affecter une valeur de type entier à une variable de type réel, mais le contraire n'est pas vérifié.

5.2 La lecture

C'est l'action par laquelle nous pouvons introduire des données en utilisant un périphérique d'entrée (généralement le clavier). Une action de lecture est représentée par le terme 'Lire' ; elle peut concerner une variable ou plusieurs variables.

a)Pour une seule variable, le format suivant est adapté

Syntaxe

Lire (Nom variable)

b) Pour plusieurs variables, deux formats de lecture peuvent être appliqués soit

- ✓ ***Une lecture de toutes les variables ensemble***

  Syntaxe

  Lire(Nom variable1, Nom variable2, ……., Nom variableN).

- ✓ ***Une lecture pour chaque variable***

  Syntaxe

  Lire(Nom variable1)

  Lire(Nomvariable2)

  ……..

  Lire(Nom variablen)

  Exemple

  Lire(X) {On demande à l'utilisateur d'introduire une valeur pour X }

  Lire(Y,Z) { On demande à l'utilisateur d'introduire une valeur pour la variable Y et la variable Z}

Remarques

-La saisie se fait uniquement dans des variables. Ce sont les cases (cellules) qui pourront accueillir les données correspondantes.

-La donnée à introduire lors de la lecture doit être de même type que la variable réceptrice.

5.3 L'écriture

C'est l'action par laquelle nous pouvons communiquer un résultat ou un message à l'utilisateur par l'intermédiaire de l'écran généralement .

Une action d'écriture, parfois dite une primitive d'écriture peut se faire :

-sur une ou plusieurs variables,

-sur des constantes,

-sur des expressions arithmétiques et logiques

-sur des messages

a) Écriture sur une ou plusieurs variables

-Pour une variable l'action d'écriture ou encore la primitive d'écriture doit se présenter Comme suit :

Syntaxe

Écrire(Nom variable) :consiste à afficher le contenu de la variable à l'écran.

-Pour plusieurs variables, deux formats d'écriture peuvent être appliqués : soit

- ✓ ***une écriture de toutes les variables ensemble***

Syntaxe

Écrire (Nom variable1, Nom variable2, ……., Nom variablen).

- ✓ ***une écriture pour chaque variable*** :

Syntaxe

Ecrire (Nom variable1)

Ecrire (Nom variable2)

……

Ecrire (Nom variablen)

Exemples

Variables

Ecrire (X) afficher le contenu de la variable X

Ecrire(Y,Z) afficher le contenu des variables Y et Z

b) Écriture d'une constante

C'est une opération qui consiste à afficher la valeur d'une constante

Syntaxe

Écrire (Val constante)

Ou

Écrire (Nom constante)

Exemples

Constante PI=3.14

Ecrire (PI) {afficher la valeur de la constante pi c'est adire 3.14}

Ecrire (5) {afficher la valeur de la valeur 5}

c) Écriture d'une expression arithmétique ou logique

Dans certains algorithmes, et en vue d'optimiser le nombre de variables ainsi que le nombre d'instructions, on réalise directement une primitive d'écriture sur une expression.

Syntaxe

Écrire (expression)

Exemples

Écrire(a+b*c) { afficher le résultat de l'expression arithmétique après évaluation

Écrire(a ou b et c) { afficher le résultat de l'expression booléenne après évaluation

d) Écriture d'un message

L'utilisation des messages dans un algorithme ne fait que faciliter son utilisation ; il devient de plus en plus convivial. Un message est une suite de caractère ayant un sens et délimité par des guillemets.

Syntaxe

Écrire('message')

Exemple :

Écrire('Bonjour') : cette opération affichera à l'écran le mot Bonjour.

Remarque

Une action d'écriture peut être mixte, c'est à dire qu'elle regroupe des variables, des constantes, des expressions et des messages.

6. Structure d'un Algorithme

Pour écrire un algorithme on utilise un ensemble de mots clés permettant de décrire de manière complète et claire l'ensemble des opérations à exécuter sur les données. Dans un algorithme, on trouve trois parties essentielles :

-Partie entête : comprend le nom de l'algorithme. Généralement, on choisit un nom évoquant le rôle de l'algorithme

Partie déclarative : contenant tous les objets qui seront impliqués par les différentes actions de l'algorithme (constantes, types, variables, etc.).

-Partie réservée aux actions (en programmation, on dit les instructions) : elle est délimitée par les deux mots-clés Début et Fin.

Entête	**Algorithme** nom de l'algorithme
Partie Déclarative	Déclaration **Constantes**
	Déclaration **Variables**
Corps Du programme	**Début**
	Instruction1
	Instruction2
	Instruction3
	Instruction4
	Fin

Exemple

Ecrire un algorithme qui calcule la surface d'un carré

Algorithme surface

Variables

Long, Surf : Réel

Début

Lire (Long)

Surf ← Long*Long

Ecrire(Surf)

FIN

7. Représentation graphique d'un algorithme (organigramme)

L'organigramme est une façon de montrer un algorithme (ou un programme) sous forme d'actions schématisées et leurs enchaînement (flèches).

La représentation graphique permet une lecture aisée des algorithmes mais présente toutefois l'inconvénient de consommer une place importante. Les opérations dans un organigramme sont représentées par les symboles dont les formes sont normalisées. Ces symboles sont reliés entre eux par des lignes fléchées qui indiquent le chemin.

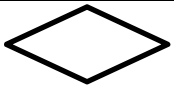
Les différentes figures d'organigramme	
Figure	**Sémantique**
	Représente le Début et la Fin Del 'organigramme
	Entrées/Sorties Lecture des données et écriture des résultats
	Calculs, Traitements
	Test et décision : on écrit le test a l'intérieur de losange
	Ordre d'exécution des opérations (enchainement)
	Connecteur

Table4.1. Représentation graphique d'un algorithme

Récapitulatif

- ✓ Un algorithme est une suite d'actions, qui une fois exécutée correctement conduit à un résultat
- ✓ Un algorithme manipule des constantes et des variables
- ✓ Une constante est emplacement réserve en mémoire, qui contient la valeur d'une donnée non modifiable. Tandis qu'une variable est emplacement réservé en mémoire, qui contient la valeur d'une donnée modifiable.
- ✓ Toute variable ou constante utilisée dans l'algorithme doit être préalablement déclarée
- ✓ Les variables et les constantes sont caractérisées par

Objet	**Nom**	**Type**	**Valeur**
Constante	fixe	fixe	fixe
Variable	fixe	Fixe	variable

- ✓ Il est possible de déclarer plusieurs variables, sans répéter le mot clé variables. Il est possible de déclarer plusieurs constantes, sans répéter le mot clé constante.

- ✓ Pour des variables de même type, on les écrit dans la même ligne séparées par des virgules.
- ✓ L'opération de lecture permet d'entrer des donnés à partir du clavier (la saisie), l'opération d'écriture permet d'afficher des résultats tandis que l'affectation permet d'attribuer une valeur ou le résultat d'un calcul à une variable .
- ✓ Après affectation la valeur d'une variable peut varier, l'ancienne valeur est tout simplement écrasée et remplacée par la nouvelle.
- ✓ L'évaluation d'une expression arithmétique ou logique repose sur les règles de priorité entre opérateurs. Afin d'éviter les ambiguïtés dans l'écriture des expressions, on se sert des parenthèses.

8. Exercices d'application

Exercice1

Classez les objets suivants en constantes ou variables, s'il s'agit de variable donner le type approprié : Marque d'un ordinateur, Poids, Nom d'étudiant, nombre d'enfants, Potentiel hydrogène(Ph) groupe sanguin, Pi, Coefficient d'un module, Surface d'un carré, numéro de groupe, l'élément potentiel Hydrogène, ?, Taille

Solution

Variables	**Type**	**Constantes**
Marque d'un ordinateur	Chain de caractères	Gravité de la terre
Nom d'étudiant		pi
groupe sanguin		
Potentiel hydrogène	Réel	
Taille		
Poids		
Surface d'un carré		
Coefficient d'un module	Entier	
numéro de groupe		
Nombre enfants		
*	Caractères	

Exercice2

Evaluer les expressions suivantes

1) 10+(4*4) div 2

2) 17 mod (5 div 2)

3) Not(12 ≠ (3*15/5))and True

4) 5 + 2 * 6 – 4 + (8 + 2 *2*2) / (2 – 4 + 5 * 2)

Solution

1) 10+(4*4) div 2 = 10 + 16 div 2 = 10 + 8 = 18

2) 17 mod (5 div 2)= 17 mod 2 = 1

3) Not(12 ≠ (3*15/5))and True = Not(True) and True=False

4) 5 + 2 * 6 – 4 + (8 + 2 *2*2) / (2 – 4 + 5 * 2)= 15

Exercie3

Donnez la trace d'exécution des algorithmes suivants ?

Algorithme TEST1

Variables

A, B, C : entier

Début

A ← 7

B ← 4

C← A + B

A ← 5

C ← B – A

Fin

Algorithme Test2

Variables

A,B,C : Booléen

Début

A←Vrai

Ecrire(A)

B←A

Ecrire(B)

A←Faux

B←Non A

Ecrire(B)

C←A et B

Ecrire(C)

C←Non (A ou B)

Ecrire(C)

Fin

Solution

La trace d'exécution est l'état des variables après exécution de chaque instruction

1. Déroulement Algorithme Test1

Instructions	**Contenu des variables**		
	A	**B**	**C**
A ← 7	**7**	**-**	**-**
B ← 4	**7**	**4**	**-**
C ← A + B	**7**	**4**	**11**
A ← 5	~~**7**~~ **5**	**4**	**11**
C ← B – A	**5**	**4**	~~**11**~~ **-1**

-Avant de se lancer dans le déroulement de l'algorithme Test 2 un petit rappel s'impose

a	**b**	**Non a**	**a et b**	**a ou b**
Vrai	Vrai	Faux	Vrai	Vrai
Vrai	Faux	Faux	Faux	Vrai
Faux	Vrai	Vrai	Faux	Vrai
Faux	Faux	Vrai	Faux	Faux

2. Déroulement Algorithme Test 2

Instructions	**Contenu des variables**		
	A	**B**	**C**
A←Vrai	**Vrai**	**-**	**-**
B←A	**Vrai**	**Vrai**	**-**
A←Faux	~~**Vrai**~~ **faux**	**Vrai**	**-**
B←Non A	**Vrai**	**Vrai**	**-**
C←A et B	**Vrai**	**Vrai**	**Vrai**
C←Non (A ou B)	**Vrai**	**Vrai**	~~**Vrai**~~ **Faux**

Exercice 4

Ecrire un algorithme qui calcule et affiche la résistance d'un composant électrique en utilisant la loi d'Ohm U=R*I

Solution

Algorithme Résistance

Variables

U, I, R : Réel

Début

Lire (U)

Lire(I)

R← U/I *(* on suppose toujours I # 0 *)*

Ecrire (R)

Fin

Conseil:

Avant de lire une variable, il est fortement conseillé d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit taper (sinon longue attente).

Exercice 5

-Ecrire un algorithme qui affiche le produit, la somme et la différence de deux nombres réels.

Solution

Algorithme nombres

Variables

X,Y, Prod , Diff , Som, : **Réel**

Début

Ecrire (‘Entrez SVP les deux nombres réels ’)

Lire(X, Y)

Som ←X+Y

Prod ←X*Y

Diff←X-Y

Ecrire (Som, Diff,Prod)

Fin

Exercice 6

- Ecrire un algorithme qui permet de saisir cinq notes d’un étudiant, et de calculer sa moyenne.

-Représenter cet algorithme sous forme d’un organigramme.

Solution

Algorithme moyenne

Var

N1,N2, N3,N4,N5, Moy : réel

Début

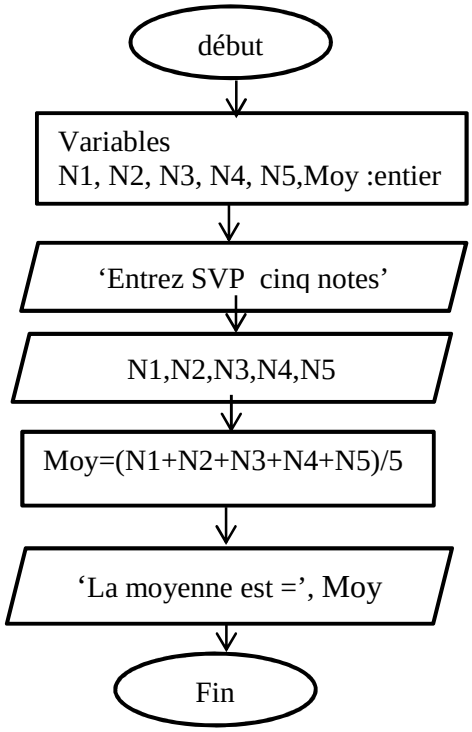
Ecrire (‘Entrez SVP cinq notes’)

Lire(N1,N2,N3,N4,N5)

Moy←(N1+N2+N3+N4+N5)/5

Ecrire(‘La moyenne est =’, Moy)

Fin



Exercice 7

Ecrire un algorithme qui lit le rayon d'un cercle et qui affiche ensuite son périmètre et sa surface.

Solution

Algorithme Cercle

Constantes

pi = 3.14

Variables

r, P, S : Réel

Début

Ecrire('Entrer le rayon du cercle ')

Lire(r)

p←2 * pi * r

S ←pi * r *r

Ecrire ('Périmètre = ', p)

Ecrire ('Surface = ', S)

Fin

Remarque

Une action d'écriture peut être mixte, c'est à dire qu'elle regroupe des variables et des messages.

Exercice 8

Ecrire un algorithme qui permet d'échanger les valeurs de deux nombres entiers

Exemple

a=7 et b=15 après échange : a=15 et b=7

Solution

Algorithme permutation

Variables

a,b,c : réel

Début

Ecrire (‘Donnez la valeur de deux réels’)

Lire (a,b)

c ← a

a ← b

b ← c

Ecrire (a,b)

Fin

Remarque

-La variable **C** est une variable intermédiaire (temporaire) qu’on est obligé de l’utiliser, elle doit être déclarée de même type que a et b.

-Les trois variables peuvent être déclarées d’un autre type (entier, booléen,…)

Chapitre 5 : les Structures Conditionnelles

1. Introduction

Un algorithme doit résoudre des problèmes très divers. La résolution de certains problèmes ne peut se faire que sous condition et pour chaque condition un traitement spécifique sera déclenché. On doit alors trouver une structure algorithmique capable de prendre en charge les différents traitements relatifs aux différentes conditions et d'en déclencher exclusivement le traitement qui respecte une certaine condition. Une telle structure est appelée en algorithmique **Structure Conditionnelle**.

2. Définition Structure Conditionnelle

Une Structure conditionnelle permet à l'algorithme de modifier son traitement en fonction d'une condition. Elle permet d'exécuter ou non un bloc d'instructions selon le résultat d'un test.

Qu'est-ce qu'un bloc d'instructions ?

Des instructions forment une séquence, quand la fin d'une instruction déclenche l'exécution de la suivante ainsi de suite jusqu'à la dernière instruction constituant ainsi un bloc d'instructions

Qu'est-ce qu'une condition ?

Une condition est une expression booléenne qui peut être vraie ou fausse, elle est composée de trois éléments :

- ✓ Une valeur,
- ✓ Un opérateur de comparaison
- ✓ Une autre valeur

L'ensemble des trois éléments de la condition constitue donc une assertion qui à un moment donné est VRAIE ou FAUSSE.

3. Types Structures Conditionnelles

Les structures conditionnelles peuvent être classifiées comme suit :

- Structure Conditionnelle à un seul choix(Simple).
- Structure Conditionnelle à deux choix (Alternative).
- Structure Conditionnelle imbriquée(Générale).
- Structure Conditionnelle à choix multiples

3.1 Structure Conditionnelle Simple (à un seul choix)

Une structure de contrôle conditionnelle est dite à forme simple lorsque le traitement dépend d'une condition. Si la condition est évaluée à « vrai », le traitement est exécuté dans le cas contraire rien ne devrait se passer.

Syntaxe

Si (Cond)**Alors**

<Séquence d'instructions 1>

FinSi

3.1.1 Fonctionnement

1- Evaluer la condition

2- Si Cond **vraie,** toutes les instructions qui se trouvent dans le bloc seront exécutées

3- Si Cond **fausse** aucune instruction n'est exécutée.

Exemple

Ecrire un algorithme qui permet de calculer l'inverse d'un entier s'il est non nul

Algorithme inverse
Variables
X : entier
Y : Réel
Début
Lire (X)
Si (X≠0) **Alors**
Y ← 1/X
Ecrire(Y)
Finsi
Fin

3.2 Structure Conditionnelle Alternative (à deux choix)

Une structure de contrôle conditionnelle est dite à forme alternative lorsqu'elle offre un choix entre deux possibilités, selon le résultat du test de la condition.

Syntaxe

Si (Cond) **Alors**

<Séquence d'instructions 1> } **Bloc1**

Sinon

<Séquence d'instructions 2> } **Bloc2**

FinSi

3.2.1 Fonctionnement

1- Evaluer la condition

2- Si **COND vraie,** toutes les instructions du Bloc 1 seront exécutées

3- Si **COND fausse** toutes les instructions du Bloc2 seront exécutées

Exemple

Ecrire un algorithme qui affiche la mention selon la moyenne de l'étudiant

MOY>=10 admis MOY <10 ajourné

Algorithme Moyenne

Variables

moy : Réel

Début

Ecrire('donner la moyenne de ')

Lire (moy)

Si (moy>=10**) Alors**

Ecrire('admis')

Sinon

Ecrire('ajourné')

Finsi

FIN

3.3 Structure Conditionnelle Imbriquée(Générale)

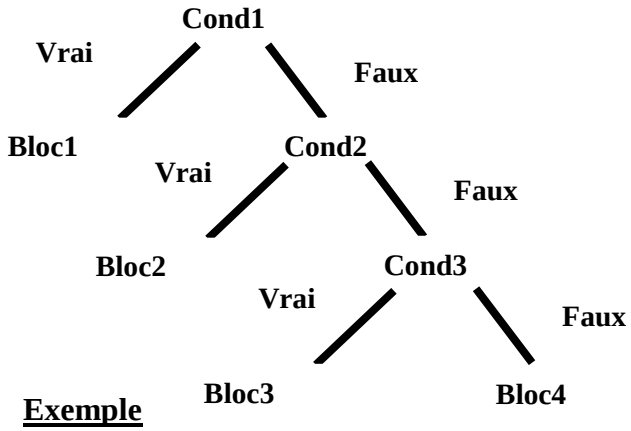
Le traitement de certains problèmes qui présentent plusieurs possibilités (choix multiples) implique l'ouverture de plusieurs voies, correspondant à des tests imbriqués les uns dans les autres

Syntaxe

Si (Cond1) **Alors**
<Séquence d'instructions 1> **Bloc1**
Sinon
Si (Cond2) **Alors**
<Séquence d'instructions 2> **Bloc2**
Sinon
Si (Cond3) **Alors**
<Séquence d'instructions 3> **Bloc3**
Sinon
<Séquence d'instructions 4> **Bloc4**
FinSi
FinSi
FinSi

3.3.1 Fonctionnement

1- Evaluer la condition Cond1
2-Si **Cond 1** est **vraie**, le bloc d'instructions1 sera exécuté.
3-Si **Cond 1** est **fausse**, on évalue **Cond 2**.
4-Si **Cond 2** est **vraie**, le bloc d'instructions2 sera exécuté.
5-Si **Cond 2** est **fausse** on évalue **Cond 3**.
6-Si **Cond 3** est **vraie**, le bloc d'instructions3 sera exécuté.
7-Si **Cond 3** est **fausse**, le bloc d'instructions4 sera exécuté



Exemple

Ecrire un algorithme qui lit la moyenne d'un étudiant puis affiche sa mention selon les cas suivants :

Moyenne≥10 admis

8<Moyenne<10 passe le rattrapage

Moyenne <8 ajourné

Algorithme Moyenne

Variables

moy : Réel

Début

Ecrire('donner la moyenne de ')

Lire (moy)

Si (moy>=10) **Alors**

Ecrire('admis')

Sinon

Si(Moyenne <8) **Alors**

Ecrire('ajourné')

Sinon

Ecrire('passe le rattrapage ')

Finsi

Finsi

FIN

Remarques

L'utilisation de structures imbriquées présente certains avantages :

- Economie de l'édition du programme, au lieu de devoir taper trois conditions, dont une est composée, nous n'avons plus que deux conditions simples. Donc, un programme plus simple et plus lisible.
- Economie de temps d'exécution du programme, si le premier test est vrai, le programme passe directement à la fin, sans tester le reste qui est forcément faux. Donc, un programme plus performant à l'exécution.
-Attention cependant au nombre de niveau d'imbrications, préjudiciable à l'intelligibilité des algorithmes

3.4 Structure de Sélection Multiple

La structure conditionnelle à choix multiple est une structure algorithmique utilisée de préférence dans des structures à plusieurs traitements selon des conditions.

Le choix d'un traitement se fait suivant la valeur d'un sélecteur. Cette structure permet d'éviter le recours à une structure conditionnelle généralisée lorsque le nombre de conditions est élevé et offre une meilleure lisibilité de la solution.

Syntaxe

Selon Sélecteur **Faire**

Val 1 : < Séquence d'instructions 1> } **Bloc1**

Val 2: < Séquence d'instructions 2> } **Bloc2**

Val n : < Séquence d'instructions n> } **Blocn**

Sinon

< traitement n>

FinSelon

3.4.1 Fonctionnement

Le choix du traitement à effectuer dépend de la valeur que prendra le sélecteur"(expression ordinale dont le type est un entier, un caractère, un booléen, mais pas un réel ni une chaîne de caractères).

1- Ce sélecteur est comparé à une série de valeurs

2-En cas d'égalité l'instruction qui lui est associée sera exécutée, les autres ne seront pas exécutées

3-Si la valeur du sélecteur est différente des valeurs proposées alors c'est le traitement qui suit la clause sinon qui sera exécuté.

Remarque

Le sélecteur doit avoir le même type que les valeurs devant les cas.

Exemple

Ecrire un algorithme qui permet de lire un numéro compris entre 1 et 7 et d'afficher le jour de semaine correspondant. Si le numéro entré est en dehors de cet intervalle, un message d'erreur doit être affiché.

Algorithme jour

Variable Num : entier ;
Début
Ecrire ('Donner le numéro du jour')
Lire **(**Num **)**
Selon Num **faire**
Cas 1 : Ecrire ('Dimanche')
Cas 2 : Ecrire ('Lundi')
Cas 3 : Ecrire (' Mardi')
Cas 4 : Ecrire (' Mercredi')
Cas 6 : Ecrire (' Jeudi')
Cas 6 : Ecrire (' Vendredi')
Cas 7 : Ecrire ('Samedi')
Sinon
Ecrire ('erreur')
Finselon
Fin

Récapitulatif

- ✓ Une structure est dite conditionnelle si l'exécution de son action dépend d'une ou de plusieurs conditions.
- ✓ Il existe plusieurs formes de structures conditionnelles la formes simple, alternative imbriquée, et à choix multiples.
- ✓ Dans une condition les valeurs à comparer peuvent être a priori de n'importe quel type (numériques, caractères…). Mais si l'on veut que la comparaison ait un sens, il faut que les deux valeurs de la comparaison soient du même type.
- ✓ La condition peut être une expression booléenne simple ou une suite composée d'expressions booléennes

4. Exercices d'application

Exercice1

Ecrire un algorithme qui calcule et affiche la valeur absolue d'un entier quelconque lu au clavier.

Solution

Algorithme Val_Abs
Variables
x, val : Entier
Début
Ecrire('Entrer un entier ')
Lire(x)
Si (x >= 0) **Alors**
val←x
Sinon
val← -x
FinSi
Ecrire ('la valeur absolue est =', val)
Fin

Exercice2

Ecrire un algorithme qui lit un entier et affiche ensuite s'il est pair ou impair.

Solution

Algorithme pair_impair

Variables

x : Entier

Début

Ecrire('Entrer la valeur un entier : ')

Lire(x)

Si (x Mod 2 = 0) **Alors**

Ecrire('ce nombre est pair')

Sinon

Ecrire(' ce nombre est impair')

FinSi

Fin

Exercice3

Ecrire un Algorithme qui lit la température de l'eau et affiche son état selon les cas suivants :

- Température ≥100 afficher état gazeux
- Température ≤0 afficher état solide
- 0<Température <100 afficher état liquide

Solution

Algorithme temperature

Variables

temp: reel

Debut

Ecrire('donner la valeur de temperature')

Lire (temp)

Si(temp>=100) **Alors**

Ecrire('état gazeux ')

Sinon

Si(temp<=0) **Alors**

Ecrire(‘état solide’)

Sinon

Ecrire(‘état liquide’)

Finsi

Finsi

Fin

Remarque

En formulant une condition dans un algorithme, il faut faire attention à certaines notations qui sont valides en mathématiques, mais qui mènent à des non-sens informatiques. Prenons par exemple la condition « température est compris entre 0 et 100 ». En fait, cette expression contient deux conditions simples reliées par l’opérateur logique ET. Elle sera formulée ainsi (Température>0) et (Température<100)

Exercice 4

Soit l’algorithme suivant

Algorithme ABC

Variables

a,b,c : entier

Début

Lire (a,b,c)

Si ((a<5) ou (b>2) et (c>3)) **Alors**

a←1

  Si (a-b>0) **Alors**

  c←0

  Finsi

b←b+c

Sinon

a←c

FinSi

c←b+c

Ecrire (a,b,c)

Fin

-Dérouler l'algorithme selon les cas suivants

	a	**b**	**c**	**affichage**
Cas1	2	0	3	
Cas2	8	3	7	
Cas3	7	1	4	

Solution

	a	**b**	**c**	**Affichage**
Cas1	2	0	3	a=1 b=0 c=0
Cas2	8	3	7	a=1 b=10 c=17
Cas3	7	1	4	a=4 b=1 c=5

Exercice 5

Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif ou positif (on inclut le traitement du cas où le produit peut être nul).

Solution

Algorithme produit

Variables

m, n : Entier

Début

Ecrire (‘Entrez deux nombres :’)

Lire (m, n)

Si ((m = 0) OU (n = 0)) **Alors**

Ecrire (‘Le produit est nul’)

Sinon

Si ((m < 0 ET n < 0) OU (m > 0 ET n > 0)) **Alors**

Ecrire **(**‘Le produit est positif ’)

Sinon

Ecrire (‘Le produit est négatif ’)

Finsi

Finsi

Fin

Remarque

Une condition est dite composée (ou complexe) lorsqu’elle est formée de plusieurs conditions simples reliées par des opérateurs logiques ET, OU, OU exclusif ….

Exercice 6

Ecrire un algorithme permettant de résoudre dans R une équation du second degré de la forme **$ax^2+bx+c = 0$(**avec la valeur de a non nulle)

Solution

Algorithme equation2

Variables

a, b, c, delta :,X1,X2,X0 : Réel

Début

Ecrire(‘entrez la valeur de a (non nulle)’)

Lire(a)

Ecrire(‘entrez la valeur de b’)

Lire(b)

Ecrire(‘entrez la valeur de c’)

Lire(c)

delta ←((b*b)– 4*a*c)

Si (delta < 0) **Alors**

Ecrire(‘pas de solution dans R’)

Sinon

Si (delta = 0) **Alors**

X0← - b/(2*a)

Ecrire(‘solution est =’, X0)

Sinon

X1 ← (-b-sqrt(delta))/(2*a))

X2 ← (-b+sqrt(delta))/(2*a))

Ecrire(‘solution1 est =’, X1)

Ecrire(‘solution2 est =’, X2)

FinSi

FinSi

Fin

Exercice7

Ecrire un algorithme qui permet de lire un numéro compris entre 1 et 12 et d’afficher le nom du mois correspondant. Si le numéro entré est en dehors de cet intervalle, un message d’erreur doit être affiché

Solution

Algorithme mois

Variables

n : Entier

Début

Ecrire(‘Entrer le numéro du mois : ’)

Lire(n)

Selon n **Faire**

1 : Ecrire(‘janvier’)

2 : Ecrire(‘février’)

…

12 : Ecrire(‘décembre’)

Sinon

Ecrire(”numéro de mois erroné... ”)

FinSelon

Fin

Chapitre 6 : les Structures Itératives

1. Introduction

Il arrive souvent dans un algorithme qu'une même action soit répétée plusieurs fois. Il est alors fastidieux d'écrire un algorithme qui contient de nombreuses fois la même instruction. Pour gérer ce cas de figure, on fait appel à des structures appelées itératives ou boucles qui ont pour effet de répéter plusieurs fois une même instruction.

2. Les Structure Itératives (les Boucles)

La notion d'itération (boucle) est une des notions importantes de l'algorithmique. Une structure itérative répète l'exécution d'un traitement, dans un ordre précis, un nombre déterminé ou indéterminé de fois. Une structure itérative est aussi appelée boucle ou structure répétitive.

3. Types de boucles

Les boucles permettent une grande puissance de calcul en exécutant, un grand nombre de fois, un nombre d'instructions données.

Deux cas sont cependant à envisager

- ✓ le nombre de répétitions est connu à l'avance : c'est le cas des boucles itératives
- ✓ le nombre de répétitions n'est pas connu ou est variable : c'est le cas des boucles conditionnelles

Généralement on distingue trois sortes de boucles en langages de programmation

3.1 La boucle « Pour…Faire »

La boucle **Pour** est une boucle itérative ; elle consiste à répéter un traitement un nombre de fois fixé à l'avance.

Cette boucle utilise une variable de contrôle (appelé **compteur** ou **indice**) d'itérations caractérisée par

- ✓ sa valeur initiale,
- ✓ sa valeur finale,
- ✓ son pas de variation.

Si la valeur initiale de l'indice est inférieure à sa valeur finale, le pas de variation est positif et la structure est dite « croissante ». Dans le cas contraire, le pas est négatif et la structure est dite « décroissante ».

La structure répétitive **Pour** est formulée ainsi.

Syntaxe:

Pour compteur ← valeur initiale **à** valeur finale **Faire**
<Séquence d'instructions>
FinPour

Principe de fonctionnement

- ✓ Le compteur (variable de contrôle) prend la valeur initiale au moment d'accès à la boucle.
- ✓ Le compteur passe automatiquement à la valeur suivante dans son domaine, à chaque parcours.
- ✓ La boucle s'arrête lorsque la valeur finale est atteinte.

Propriétés

- La boucle Pour est utilisée lorsque le nombre de répétition est connu à l'avance
- Le bloc est répété (valeur finale - valeur initiale+1) fois
- La sortie de cette boucle s'effectuer lorsque le nombre souhaité de répétition est atteint
- La variable indice (compteur) peut être utilisée dans le bloc d'instructions
- Dans cette boucle l'incrémentation est faite automatiquement.
- Une boucle « pour » peut être exécutée 0, 1 ou n fois

Exemple

Ecrire un algorithme qui permet d'afficher le message bonjour cinq fois via la boucle Pour

Algorithme Affichage
Variables
i : entier

Début

Pour i ← 1 **à** 5 **Faire**

Ecrire(‘bonjour’)

FinPour

Fin

-Déroulement de l’algorithme

Compteur(i)	**Affichage (Ecran)**
1	bonjour
2	bonjour
3	bonjour
4	bonjour
5	bonjour

Remarque

Dans une boucle « Pour », l’évolution du compteur peut se faire dans le sens **décroissant** comme dans l’exemple suivant :

Exemple

Pour i **de** 5 **à** 1 **Faire**

Ecrire (i*10)

FinPour

Dans ce cas, le compteur i sera **décrémenté** après chaque parcours. Cette boucle affiche respectivement

-Déroulement de l’algorithme

Compteur(i)	**Affichage (Ecran)**
5	50
4	40
3	30
2	20
1	10

3.2 La boucle « Tant que… Faire »

Dans cette boucle conditionnelle, le même traitement est effectué tant qu'une condition reste valide ; la boucle s'arrête quand celle-ci n'est plus remplie. Cette structure répétitive est ainsi formulée.

Syntaxe:

Tant Que (COND) **Faire**

<Séquence d'instructions>

FinTQ

Principe de fonctionnement

- ✓ Au début de chaque itération, la condition est évaluée.
- ✓ Si la condition est vraie, on exécute les instructions (corps de la boucle), puis on retourne tester la condition. Si elle est encore vraie, on répète l'exécution.
- ✓ Si la condition est fausse, on sort de la boucle et on exécute l'instruction qui est après FinTQ

Propriétés

- Le nombre de répétitions n'est pas connu à l'avance il dépend de la valeur de la condition COND
- La condition à vérifier à chaque fois est considérée comme une condition d'entrée, car elle nous permet d'accéder au corps de la boucle. quand elle n'est plus vérifiée, on sort de cette boucle.
- Si la condition n'est pas vérifiée au début, alors la boucle ne sera pas exécutée du tout.
- Il est indispensable d'initialiser correctement les variables de la condition d'exécution et de les mettre à jour à la fin de chaque itération, cela est obligatoire afin de pouvoir reboucler.
- La structure Tant que est conseillée surtout pour les problèmes indiquant dans leur énoncé une condition d'exécution.
- Une boucle Tant que peut s'exécuter 0, 1 ou n fois

Exemple

Ecrire un algorithme qui calcule le produit des nombres lus au clavier tant qu'ils sont non nuls.

Algorithme boucle

Variables

n, p : entier

Début

Tant que (n ≠ 0) **Faire**

Ecrire (‘Donner un nombre’)

Lire (n)

p ← n*n

Ecrire(p)

FinTQ

Ecrire (‘Fin de l’algorithme’)

Fin

-Les instructions encadrées par **Tant que** et **FinTQ** constitue le bloc de la boucle qu’il faut répéter tant que la condition n≠0 est vérifiée. Donc le nombre de répétitions de cette boucle dépend des données fournies par l’utilisateur.

3.3 La boucle « Répéter… jusqu’à »

Tout comme la boucle Tant que, la boucle Répéter est une boucle conditionnelle. Dans ce type de boucle le test est effectué à la fin, si bien que le traitement est exécuté au moins une fois, que la condition soit ou non vérifiée.

Syntaxe:

Répéter

<Séquence d’instructions>

Jusqu’à (COND)

Principe de fonctionnement

- ✓ La séquence d’instructions est exécutée une première fois,
- ✓ On teste la condition d’arrêt
- ✓ Si elle n’est pas atteinte, le corps de la boucle est exécuté à nouveau puis la condition est réévaluée encore une fois
- ✓ Si la condition de sortie est vérifiée, cela signifie l’arrêt de la boucle

Propriétés :

- Dans une boucle Répéter, le traitement est exécuté au moins une fois quelle que soit la valeur de la condition à vérifier, donc cette structure permet de répéter un traitement un ou plusieurs fois.
- La condition à vérifier à chaque fois est appelée condition de sortie(arrêt) car elle nous permet de quitter la boucle.
- Il est indispensable d'initialiser correctement les variables de la condition d'arrêt et de les mettre à jour à la fin de chaque itération cela est nécessaire pour pouvoir reboucler.
- La structure Répéter est conseillée surtout pour les problèmes indiquant dans leur énoncé une condition d'arrêt.

Exemple

Algorithme boucle
Variables
n, p : entier
Début
Répéter
Ecrire ('Donner un nombre')
Lire (n)
p ← n*n
Ecrire(p)
Jusqu'à (n=0)
Ecrire ('Fin de l'algorithme')
Fin

Les instructions encadrées par les mots clé « Répéter et Jusqu'à » constitue le bloc de la boucle qu'il faut répéter jusqu'à ce que la condition n=0 soit vérifiée. Donc le nombre de répétitions de cette boucle dépend des données fournies par l'utilisateur.

Remarques

1. Dans les boucles conditionnelles Tant que et Répéter, si la condition d'arrêt reste inchangée (non mise à jour), on risque de reboucler indéfiniment et par conséquent le programme se bloque.

Exemple

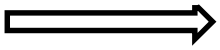
i ← 1

TantQue (i > 0) **Faire**

i ← i+1

FinTQ

Correction



i ← 1

TantQue (i <100) **Faire**

i ← i+1

FinTQ

Exemple

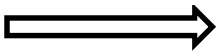
i ← 1

Répéter

i ← i+1

Jusqu'a(i<=0**)**

Correction



i ← 1

Répéter

i ← i+1

Jusqu'a(i<=100)

2. Lorsque le nombre d'itérations est connu à l' avance, contrairement à la boucle « Pour », dans les boucles « Répéter » et « Tant que », l'initialisation et l'avancement du compteur doivent être gérés manuellement par le programmeur.

Exemple

Écrire un algorithme permettant d'afficher le message bonjour 5 fois via les boucles Tant que et Répéter

Algorithme Affichage

Variable

i : entier

début

i←1

Tant que (i<=5) **Faire**

Ecrire('bonjour')

i←i+1

FinTQ

Fin

Initialisation indispensable Avant la boucle

Incrémentation

Algorithme Affichage

Variable

i : entier

Début

i←1

Répeter

Ecrire('bonjour')

i←i+1

Jusqu'à (i>5)

FIN

-Déroulement de l’algorithme

Compteur	Affichage
1	-
2	bonjour
3	bonjour
4	bonjour
5	bonjour
6	bonjour

- Déroulement de l’algorithme

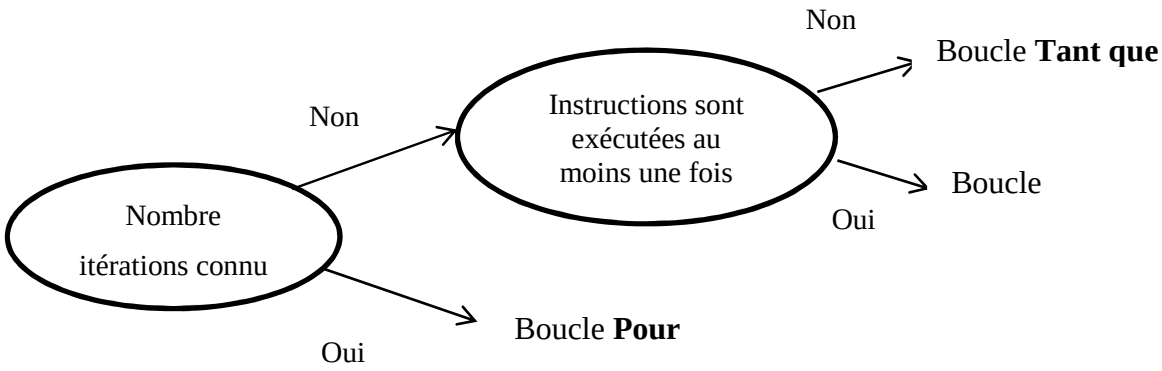
Compteur	Affichage
1	-
2	bonjour
3	bonjour
4	bonjour
5	bonjour
6	bonjour

4. Méthodologie pour l'écriture d'une boucle

- ✓ Repérer une ou plusieurs actions répétitives
- ✓ Vérifier si on doit prévoir/déterminer le nombre d'itérations ?
  - si oui, boucle avec compteur : la boucle « Pour…. Faire »
  - si non, boucle sans compteur.
- ✓ Est ce qu’il faut commencer l'action avant de tester ou l'inverse ?
  - si tester d'abord, alors boucle « Tant Que… Faire »
  - si action puis tester, alors « Répéter … jusqu’à »
- ✓ Ecrire l'action(s) répétitive(s) et l'instruction de boucle choisie.
- ✓ Initialiser les variables utilisées (si nécessaires)
- ✓ Ecrire les conditions d'arrêt, voire l'incrémentation de la variable de contrôle.

5. Choix de la structure itérative adéquate

Comment choisir la structure itérative le plus adéquate pour la résolution d’un problème ?



Récapitulatif

Les instructions itératives, appelées aussi instructions ou structures répétitives, permettent d'exécuter plusieurs fois une instruction.

Critère	**Répéter**	**Tant que**	**Pour**
Traitement	Au moins 1 fois	De 0 à N fois	(valeur finale - valeur initiale+1)
Initialisation compteur	Oui avant Répéter	Oui avant Tant que	Inclus dans la boucle
Incrémentation/décrémentation compteur	Nécessaire	Nécessaire	Automatique en fonction du Pas
Sortie de la boucle	Condition d'arrêt atteinte	Condition d'exécution non vérifiée	Compteur dépassant les bornes

Tableau6. 1 : Tableau comparatif entre les trois boucles.

6. Exercices d'application

Exercice 1

Tracer l'exécution des morceaux des algorithmes suivants

(1)
I←0
S←0
Tant que (I <=5) **Faire**
I ←I+1
S← S+ I+1
Ecrire (S)
Ecrire (I)
FinTQ

(2)
I←1
Tantque (I <=5) **Faire**
Ecrire (i)
FinTQ

(3)
S←0
Pour I ← de 1 à 6 **Faire**
S← S+ I
Ecrire (I)
Ecrire (S)
Finpour

(4)
I←0
S←1
Répéter
I←I+2
S← S* I
Ecrire (I)
Ecrire (S)
Jusqu'à (I >7)

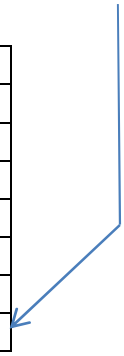
(5)
Pour I ← de 6 à 1
Ecrire (I)
Finpour

(1)

I	S
0	0
1	2
2	5
3	9
4	14
5	20
6	27

(2) **Boucle infinie**

I
1
1
1
1
.
.
1



(3)

I	S
-	0
1	1
2	3
3	6
4	10
5	15
6	21

4)

I	S
0	1
2	2
4	8
6	48
8	348

5)

I
6
5
4
3
2
1

Remarque

Dans les boucle répéter et tant que, il faut toujours s'assurer que la condition ne reste pas inchangée pour ne pas tomber dans le cas de boucle infinie

Exercice 2

Ecrire un algorithme qui lit un entier positif n puis affiche tous ses diviseurs. (via la boucle Pour, Tant que, Répéter)

Solution

1) Avec la boucle **Pour**

Algorithme Diviseurs **Variables**

Variables

n, i : Entier

Début

Ecrire('Entrer un entier positif ')

Lire(n)

Pour i ← 1 à n **Faire**

Si (n Mod i = 0) **Alors**

Ecrire(i)

FinSi

FinPour

Fin

2) Avec la boucle **Répéter**

Algorithme Diviseurs

Variables

n, i : Entier

Début

Ecrire('Entrer un entier positif ')

Lire(n)

i←1

Répéter

Si (n Mod i = 0) **Alors**

Ecrire(i)

FinSi

i←i + 1

Jusqu'à (i > n)

Fin

3) Avec la boucle Tant que

Algorithme Diviseurs

Variables

n, i : Entier

Début

Ecrire('Entrer un entier positif ')

Lire(n)

i←1

Tant que (i<=n) **Faire**

Si (n Mod i = 0) Alors

Ecrire(i)

FinSi

i←i + 1

FinTQ

Fin

Remarques

-Dans les boucle **Tant que** et **Répéter** il est indispensable d'initialiser le compteur avant la boucle et de l'incrémenter à l'intérieur de la boucle

-L'instruction i←i+1 permet de modifier la valeur de la variable i qui compose la condition si on oublie cette instruction, le programme s'exécutera infiniment

Exercice 3

Soit l'algorithme suivant

Algorithme somme

Variables

i,SP,SI,N : Entier

Début

Ecrire('donner la valeur de N')

Lire(N)

SI ← 0

SP ← 0

Pour i←1 **à** N **faire**

Si (i mod 2=0) **alors**

SP ←SP+i

Ecrire(SP)

Sinon

SI←SI+i

Ecrire(SI)

Finsi

Finpour

Fin

1. Dérouler l'algorithme pour N=10

2. Que calcule cet algorithme.

Solution

1. Déroulement de l'algorithme

i	SP	SI
-	0	0
1	0	0+1
2	0+2	
3	0+2	1+3
4	2+4	1+3
5	2+4	1+3+5
6	2+4+6	1+3+5
7	2+4+6	1+3+7
8	2+4+6+8	1+3+7
9	2+4+6+8	1+3+7+9
10	2+4+6+8+10	1+3+5+7+9

2. Cet algorithme calcule deux somme SP (la somme des nombres paires) et SI(la somme des nombres impaires) pour les nombres qui se trouvent dans l'intervalle [1.. n].

Exercice 4

Ecrire un algorithme qui calcule la N^{ieme} puissance d'un entier positif X. Tel que X^N = X**X*X(N fois)

Solution

1) Avec la boucle **Pour**

Algorithme Puissance

Variables

i,N,X,Puiss : **Entier**

Début

Ecrire ('entrez SVP deux entiers positifs ')

Lire(X,N)

Puiss ← 1

Pour i←1 **à** N **faire**

Puiss←Puiss*X

FinPour

Ecrire (‘X a la puissance N = ’, Puiss)

Fin

-Déroulement de l’algorithme

Supposons que N=4 et X=3

i	**Puiss**
1	3
2	3*3=9
3	9*3=27
4	27*3=**81**

2) Avec la boucle **Tant que**

Algorithme Puissance

Variables

i,N,X,Puiss : **Entier**

Début

Ecrire (‘entrez SVP deux entiers positifs ’)

Lire(X,N)

Puiss ← 1

i←1

Tant que (i<=N) faire

Puiss←Puiss*X

i←i+1

FinTQ

Ecrire (‘X a la puissance N = ’, Puiss)

Fin

-Déroulement de l'algorithme

Supposons que N=4 et X=3

i	**Puiss**	**condition**
1	1	True
2	1*3=3	True
3	3*3=9	True
4	9*3=27	True
5	27*3=**81**	False(sortie de la boucle)

3) Avec la boucle **Répéter**

Algorithme Puissance

Variables

i,N,X,Puiss : **Entier**

Début

Ecrire ('entrez SVP deux entiers positifs ')

Lire(X,N)

Puiss ← 1

i←1

Répéter

Puiss←Puiss*X

i←i+1

Jusqu'à (i>N)

Ecrire ('X a la puissance N = ', Puiss)

Fin

-Déroulement de l'algorithme

Supposons que N=4 et X=3

i	**Puiss**	**Condition**
1	1	False
2	1*3=3	False
3	3*3=9	False

4	9*3=27	False
5	27*3=**81**	True (sortie de la boucle)

Remarque

La condition d'arrêt de la boucle répéter est l'inverse de la condition d'exécution de la boucle tant que

Exercice 5

1. Ecrire un algorithme permettant de calculer l'expression suivantes via la boucle tant que

$$S = 1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots + \frac{1}{n^2}$$

2. Dérouler l'algorithme pour N=4

Solution

Algorithme somme

Variables

n, i :entier

s : reel

Début

Ecrire('Entrer la valeur de n :')

Lire(n)

s ←0

i←1

Tant que(i<= n) **Faire**

s ← s + (1/(i*i))

i←i+1

FinTQ

Ecrire ('la somme est = ',s)

Fin

-Déroulement de l'algorithme

Pour N=4

I	**S**	**Condition**
1	0	True
2	0+1/1	True
3	1/1+1/4	True
4	1/1+1/4+1/9	True
5	**1+1/4+1/9+1/16**	False(sortie de la boucle)

Exercice 6

Ecrire un algorithme qui lit deux entiers A et B puis calcule et affiche leur PGCD en utilisant la méthode suivante :

· Si A = B ; PGCD(A,B) = A

· Si A > B ; PGCD(A,B) = PGCD(A–B,B)

· Si B > A ; PGCD(A,B) = PGCD(A,B-A)

Exemple : PGCD(18,45)=PGCD(18,27)=PGCD(18,9)=PGCD(9,9)=9

Solution

Algorithme PGCD

Variables

a, b : Entier

Début

Ecrire('Entrer la valeur de a : ')

Lire(a)

Ecrire('Entrer la valeur de b : ')

Lire(b)

Répéter

Si (a > b) **Alors**

a ←a – b

FinSi

Si (b > a) **Alors**

b ←b - a

FinSi

Jusqu'à (a = b)

Ecrire('Le PGCD = ', a)

Fin.

-Déroulement de l'algorithme

Pour a=14 et b=45

a	**b**	**Condition**
18	~~45~~ 27	False
18	~~27~~ 9	False
~~18~~ **9**	**9**	True(sortie de la boucle)

.

Chapitre 7 : les Tableaux

1. Introduction

Un algorithme peut utiliser des variables de type simple (entier, réel, caractère, chaîne et booléen) où chaque variable est associée à un seul espace mémoire, ne pouvant loger (ou bien contenir) qu'une seule valeur à instant de données. Cependant dans un algorithme, il est possible qu'une variable puisse contenir à un moment donné, non pas une valeur, mais plusieurs valeurs à la fois. C'est pourquoi, les langages de programmation offrent la possibilité de rassembler toutes ces variables dans une seule structure de donnée est appelée tableau.

Quelle est l'utilité des tableaux dans un algorithme ?

Supposons que nous avons à déterminer à partir de 30 notes fournies en entrée, le nombre d'étudiants qui ont une note supérieure à la moyenne de la classe. Pour parvenir à un tel résultat, nous devons :

1. Lire les 30 notes
2. Déterminer la moyenne de la classe
3. Compter combien parmi les 30 notes sont supérieures à la moyenne

Il faut donc conserver les notes en mémoire afin qu'elles soient accessibles durant l'exécution du programme.

Solution 1 : utiliser 30 variables réelles nommées x1, x2, …, x30

Vous pouvez remarquer que c'est un peu lourd de manipuler une trentaine de variables (avec 30 fois de lecture et d'écriture…). Imaginons maintenant le cas pour une promotion de 1000 étudiants le problème traité se complexifie encore plus.

Solution 2 : Lorsque les données sont nombreuses et de même type, afin d'éviter de multiplier le nombre des variables, on les regroupe dans une même structure de donnée appelé **tableau.**

2. Définition

Un tableau est une structure de données regroupant un ensemble de variables de même type, repérés au moyen d'indices. Les éléments d'un tableau sont rangés selon un ou plusieurs axes appelés dimensions du tableau.

3. Tableaux unidimensionnels (Vecteurs)

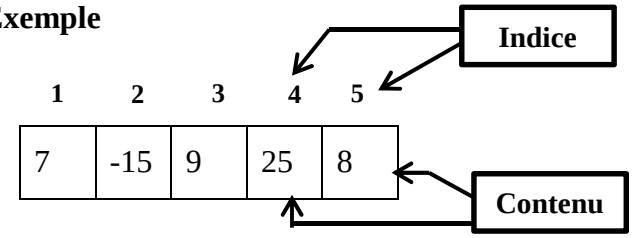
Un tableau à une dimension, appelé aussi **vecteur** est une structure de données linéaires constituée d'un nombre fini d'éléments de même type. Pour accéder à une composante (élément) du vecteur on a besoin d'une variable entière nommée indice (index ou rang) permettant d'indiquer la position d'un élément donné au sein du tableau pour déterminer sa valeur

3.1 Accès à un élément du tableau

Un vecteur est donc caractérisé par :

- Un nom
- Un type particulier de données
- Une dimension : correspond au nombre maximum de cases composant le tableau
- Un indice : les cases du tableau sont numérotées successivement. Ce numéro est appelé indice chaque élément de ce tableau est repéré par son indice. La première case du vecteur peut porter le numéro 0 ou 1 cela dépend du langage de programmation.

Exemple



-Ce vecteur se nomme T, il contient des éléments de type entier, il est de longueur 5, car il contient 5 emplacements.

-Chaque fois qu'on doit désigner un élément du tableau, on utilise le nom de ce tableau, suivi de l'indice de l'élément entre parenthèses.

-Ainsi, **T[3]** désigne le troisième élément du tableau T c'est-à-dire -15

-T[i] désigne le ième élément du tableau T.

3.2Déclaration d'un tableau

La déclaration d'un tableau s'effectue en précisant son nom le type de ses éléments et sa dimension (le nombre de ses éléments)

Syntaxe :

Nom **Tableau** : Tableau [Borne_inf.. Borne_sup] de **type éléments**

Ou

Borne_inf : est le numéro de la première case du vecteur

Borne_sup : est le numéro de la dernière case du vecteur

Remarque

Il est nécessaire de rajouter, dans la partie déclaration, la définition de l'indice de parcours (de la borne inférieure du vecteur à la borne supérieure) qui est une variable simple de type entier

Exemple

Variables

T : Tableau [1..100] de réel

i : entier

T est un vecteur de100 cases destinées à contenir des éléments de type réél, indicé par une variable entière i.

Remarque

Il est également possible de définir un type tableau comme suit

Exemple

Constantes

n = 10

Types

Tab = Tableau [1..n] de Entier

Variables

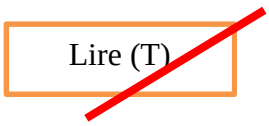
T : Tab

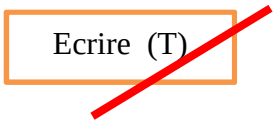
3.3Manipulation d'un tableau

Une fois déclaré, un tableau peut être utilisé comme un ensemble de variables simples. Les instructions de lecture, écriture et affectation s'appliquent aux tableaux comme aux variables

3.3.1 Lecture et écriture d'un tableau

Les instructions de lecture, écriture peuvent s'appliquer à chacun des éléments mais pas au tableau tout entier.





Exemple

Pour lire les éléments d'un tableau T composé de 5 éléments. On peut procéder comme suit

Lire (T[1]))

Lire (T[2])

Lire (T[3])

Lire (T[4])

Lire (T[5])

L'instruction lire est répétée 5 fois. Afin d'éviter la répétition de cette instruction, nous pouvons utiliser une des structures itératives que nous avons vu précédemment. Le nombre d'itérations étant connu (5), nous pouvons ainsi utiliser la boucle Pour

Pour I ←1 à 5 **Faire**

Lire (T[i])

FinPour

i	**T[i]**	**Cases**
1	T[1]	Numéro 1
2	T[2]	Numéro 2
3	T[3]	Numéro 3
4	T[4]	Numéro 4
5	T[5]	Numéro 5

- ✓ **Saisie(Lecture) d'un tableau :**

Elle consiste en un remplissage des différentes cases qui constituent le vecteur. On doit utiliser une boucle qui permet de saisir à chaque entrée dans la boucle la iième case.
Le nombre d'éléments à saisir ne doit pas dépasser la taille du tableau pour ne pas déborder sa capacité.

Syntaxe

Pour i ← 1 à N **Faire** {Avec N taille du tableau à utiliser}
Lire (T[i]) {l'indice i parcourt les case numérotées de 1 jusqu'à N}
Finpour

Fin

- ✓ **Affichage(Ecriture) d'un tableau :**

Il s'agit d'afficher les valeurs des cases du tableau

Syntaxe

Pour i ← 1 à N **Faire** **{**Avec N taille du tableau à utiliser}
Écrire (T[i]) {l'indice i parcourt les case numérotées de 1 jusqu'à N}
Finpour
Fin

3.3.2 Affectation

L'opération d'affectation au niveau du vecteur se fait comme suit

Syntaxe

Nom vecteur [Indice] ← < expression >

Exemple :

Soit T un vecteur de 5éléments entiers
T[1] ←5 {affecter au premier élément du tableau T la valeur 20}
T[2] ← 2
T[3] ← T[1]+T[2]

T[4] ← T[2] *2

T[5] ←T[3] + 6

Le déroulement de cet exemple nous donne le vecteur suivant

5	2	7	4	13

Pour affecter la même valeur à tous les éléments d'un tableau T de type numérique et de dimension 100, on utilise une boucle.

Exemple

Pour i ←1 à 100 **Faire**

T[i] ← 10

FinPour

Remarques

- ✓ Il n'est pas obligatoire d'utiliser une constante pour la taille maximale de tableau ; cependant, c'est une bonne pratique dans la programmation.
- ✓ Il n'est obligatoire d'utiliser toutes les composantes du tableau, pour cela on déclare une variable entière n qui représente la taille du tableau à utiliser.

Exemple

Ecrire un algorithme qui permet de remplir un tableau T de type réel puis de mettre tous ses éléments à zéro

Algorithme Vecteur

Variables

i,n: entier

T : Tableau [1..100] de réels

Début

Ecrire ('donner la dimension du vecteur ')

Lire(n)

Pour i ← 1 à n **Faire**

Lire (T[i])

FinPour

Pour i ← 1 à n **Faire**

Lire (T[i])←0

FinPour

Pour i ← 1 à n **Faire**

Ecrire (T[i])

FinPour

Fin

4. Tableau à deux dimensions(Matrice)

Dans d'autres applications, on a besoin d'utiliser des tableaux à deux dimensions, qui sont capables de manipuler des structures plus complexes.

Un tableau à deux dimensions, comme celui à une dimension, est un ensemble de variables de même type (numériques, caractères,…) ayant toutes le même nom. Il est dit à deux dimensions car il se présente sous forme d'un ensemble de ligne et de colonnes.

4.1 Accès à un élément d'une matrice

Une matrice est donc caractérisée par :

- Un nom
- Un type particulier de données
- Une dimension : correspond au nombre maximum de lignes et colonnes composant le tableau
- Indice : les lignes et les colonnes d'une matrice sont numérotées par des entiers successifs. En programmation, cela dépend du langage utilisé si la première case porte le numéro 0 ou 1 (ou autre)

Exemple

	1	2	3	4	5
1	5	55	3	-9	7
2	42	7	3	8	5
3	8	-4	99	23	11

Colonnes

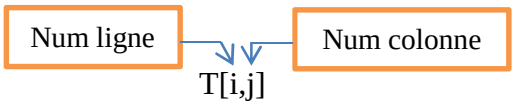
Lignes

- ✓ La matrice ci-dessous possède 3 lignes et 5 colonnes.
- ✓ La matrice est caractérisée donc par l'utilisation de deux indices : indice de la ligne et indice de la colonne.
- ✓ Une valeur dans la matrice : est désignée par le nom de la matrice et entre parenthèses l'indice de la ligne suivi de celui de la colonne

  Par exemple, T[2,1] désigne l'élément de la ligne 2 et la colonne 1 sa valeur est = 42
- ✓ T[i ,j] : désigne le contenu de l' élément situé à l'intersection de ligne i et la colonne j de la matrice T (i, j sont des variables entières).

Attention :

Toujours le premier indice est le numéro de ligne et le second est le numéro de colonne.



4.2 Déclaration d'un tableau à deux dimensions(Matrice)

La déclaration d'une matrice s'effectue en précisant son nom, le type de ses éléments et sa dimension (le nombre de lignes et de colonnes)

Syntaxe

Nom-de-matrice : Tableau [B_inf_l .. B_sup_l , B_inf_c .. B_sup_c] : Type

B_inf_l : est le numéro de la première ligne

B_sup_l : est le numéro de la dernière ligne

B_inf_c : est le numéro de la première colonne

B_sup_c : est le numéro de la dernière colonne

Type : est le type des éléments de la matrice

Exemple

Variables

M : Tableau [1..10 ,1..4] de réels

i,j : entiers

M est une matrice de 10 lignes et de 4 colonnes, ses cases sont destinées à contenir des éléments de type réel

M[i, j] est le contenu de la case qui se trouve à la ligne i et la colonne j.

Remarques

-Si on a plusieurs matrices de même ordre et de même type, on peut les déclarer à la fois en les séparant par des virgules.

-De même que pour les vecteurs, on doit déclarer les indices de parcours des lignes et des colonnes.

4.3 Manipulation d'une matrice

Une fois déclaré, la matrice peut être utilisée comme un ensemble de variables simples. Les instructions de lecture, écriture et affectation s'appliquent aux tableaux bidimensionnels comme aux variables

4.3.1 Lecture et écriture d'une matrice

La lecture et l'écriture d'un tableau bidimensionnel à n lignes et m colonnes se fait à peu près de la même façon qu'un tableau unidimensionnel. Seulement, il est nécessaire d'utiliser deux boucles imbriquées. Où la première boucle englobe la seconde. L'indice *i* est utilisé pour le parcours des lignes et *j* pour les colonnes

- ✓ **Saisie(Lecture) d'une matrice**

La saisie d'une matrice se fait, ligne par ligne ou colonne par colonne à l'aide de deux boucles imbriquées .

Syntaxe

Pour i ← 1 à n **Faire** { n :nombre de lignes}
Pour j ← 1 à m **Faire** { m :nombre de colonnes}
Lire (X [i, j])
FinPour
FinPour

Exemple

Nous avons à remplir les cases d'une matrice X constituée de 3 lignes et 5, pour cela nous aurons à utiliser deux boucles imbriquées

Pour i ← 1 à 3 **Faire**

Pour j ← 1 à 5 **Faire**

Lire (X [i, j])

FinPour

FinPour

i	**j**	**X(I,J)**	**Case**
1	1	X(1, 1)	Ligne 1 colonne 1
1	2	X(1, 2)	Ligne 1 colonne 2
1	3	X(1, 3)	Ligne 1 colonne 3
1	4	X(1, 4)	Ligne 1 colonne 4
1	5	X(1, 5)	Ligne 1 colonne 5
2	1	X(2, 1)	Ligne 2 colonne 1
2	2	X(2, 2)	Ligne 2 colonne 2
2	3	X(2, 3)	Ligne 2 colonne 3
2	4	X(2, 4)	Ligne 2 colonne 4
2	5	X(2, 5)	Ligne 2 colonne 5
3	1	X(3, 1)	Ligne 3 colonne 1
3	2	X(3, 2)	Ligne 3 colonne 2
3	3	X(3, 3)	Ligne 3 colonne 3
3	4	X(3, 4)	Ligne 3 colonne 4
3	5	X(3, 5)	Ligne 3 colonne 5

- ✓ **Affichage(Ecriture) d'une matrice :**

De même que pour le remplissage d'une matrice, l'affichage de son contenu se fait à l'aide de deux boucles imbriquées

Syntaxe

Pour i ← 1 à n **Faire** { n :nombre de lignes}

Pour j ← 1 à m **Faire** { m :nombre de colonnes}

Ecrire (**X** [i, j])

FinPour

FinPour

4.3.2 L'affectation

Comme dans le cas d'un vecteur, l'initialisation d'un élément d'une matrice se fait par l'affectation comme suit

Syntaxe

Nom Matrice [Indice- ligne, indice-colonne] ← expression

Exemple

Soit M une matriced'ordre2 * 3

M[1, 1] ←-5

M[1, 2] ← 2

M[1, 3] ← 10

M[2, 1] ← M[1, 2] + 6

M[2, 2]← M[1, 1]*M[2, 1]

M[2, 3] ← M[1, 2] *5

Le déroulement de cet exemple nous donne la matrice suivante:

M(1,1)=-5	M(1,2)=2	M(1,3)=10
M(2,1)=8	M(2,2)=-40	M(2,3)=10

Remarque

Il n'est obligatoire d'utiliser toutes les composantes d'une matrice , pour cela on déclare deux variables entières n et m qui représentent le nombre de lignes et colonnes à utiliser

Exemple

Ecrire un algorithme qui permet de remplir une matrice de type réel puis de mettre tous ses éléments à zéro

Algorithme Matrice

Variables

i,n,j,m: entier

T : Tableau [1..100,1..50] de réels

Début

Ecrire ('donner la dimension de la matrice')

Lire(n,m)

Pour i ← 1 à n **Faire**
Pour j ← 1 à m **Faire**
Lire (T[i,j])
FinPour
FinPour
Pour i ← 1 à n **Faire**
Pour j ← 1 à m **Faire**
T[i,j]←0
FinPour
FinPour
Pour i ← 1 à n **Faire**
Pour j ← 1 à m **Faire**
Ecrire (T[i,j])
FinPour
FinPour
Fin

Récapitulatif

- ✓ Les éléments d'un tableau sont des variables comme les autres. C.à.d. on peut créer un tableau de numériques comme on peut créer un tableau de caractères, de booléens …etc.
- ✓ Les instructions de lecture, écriture et affectation s'appliquent aux tableaux comme aux variables.
- ✓ Les boucles sont extrêmement utiles pour les algorithmes associés aux tableaux.
- ✓ Pour la lecture et l'écriture des éléments d'un vecteur il est indispensable d'utiliser une boucle.
- ✓ Pour la lecture et l'écriture des éléments d'une matrice il est indispensable d'utiliser deux boucles imbriquées.
- ✓ Le traitement de chacun des éléments d'un vecteur ou d'une matrice étant souvent le même, seule la valeur de l'indice ou des indices dans le cas d'un matrice est amenée à changer. Les boucles sont donc parfaitement adaptées à ce genre de traitements.

5. Exercice d’application

Exercice1

Ecrire un algorithme qui permet calculer la somme et la moyenne des éléments d'un vecteur V de type réel.

Solution

Algorithme Moyenne
Variables
V : **Tableau** [1..100] de Réel
i,n : entier
Som, Moy : Réel
Début
Ecrire(‘donner la dimension du vecteur’)
Lire(n)
Pour i←1 **à** n **faire**
Lire(V[i])
FinPour
Som ← 0
Pour i←1 **à** n **faire**
Som ← Som + V[i]
FinPour
Moy ← Som / n
Ecrire(‘La somme est =’,Som)
Ecrire(‘La moyenne est =’,Moy)
Fin

-Déroulement de l’algorithme

Soit le vecteur V constitué de cinq éléments réels (n=5)

-5	2	-7	10	25

i	V[i]	Som
1	-5	-5
2	2	-3
3	-7	-10
4	10	0
5	25	**25**

Moy=5

Exercice2

Ecrire un algorithme qui permet d'inverser les éléments d'un vecteur T de type entier dans un autre vecteur V.

Solution

Algorithme inverse
Variables
T,V : Tableau [1..100] d'entiers
i,n : entier
Début
Ecrire('donner la dimension des vecteurs')
Lire(n)
Pour i←1 à n **Faire**
Ecrire('donner les éléments du vecteur ')
Lire(T[i])
FinPour
Pour i←1 à n faire
V[i] ← T[n-i+1]
FinPour
Pour i←1 à n faire
Écrire(V[i])
FinPour
Fin

Exercice 3

Ecrire un algorithme qui permet de rechercher le plus petit élément dans un vecteur réel V ainsi que sa position.

Algorithme Recherche

Variables

V : **Tableau** [1..100] **de** Réel

i,n, Pos : entier

Min:Réel

Début

Ecrire (‘donner la dimension du vecteur’)

Lire(n)

Pour i←1 **à** n **Faire**

Ecrire(‘donner les éléments du vecteur ’)

Lire(V[i])

FinPour

Min ← V[1]

Pos ← 1

Pour i←2 **à** n **Faire**

Si V[i] < Min **alors**

Min ← V[i]

Pos ← i

FinSi

FinPour

Ecrire (‘La plus petite valeur est = ', Min, 'Elle se trouve à la position ',Pos)

Fin

Remarque

La recherche d’un minimum des éléments d’un vecteur se fait comme suit:

-On fait entrer les éléments du vecteur

-On suppose que la première case contient la valeur minimale

- On compare le contenu de la deuxième case avec le minimum précédent.

-Si celui-ci (l'élément de la deuxième case) est inférieur, il devient le maximum;

-On continue la même opération avec les cases suivantes

-**Déroulement de l'algorithme**

Soit le vecteur V constitué de quatre éléments réels (n=4)

-5	21	30	-11

i	**V[i]**	**Min**	**Pos**
-	V [1]=-5	-5	1
2	V [2]=21	-5	1
3	V [3]=30	-5	1
4	V [4]=-11	~~-5~~ **-11**	**4**

Exercice4

Ecrire un algorithme qui calcule la somme des éléments de la matrice A de type entier.

Algorithme Somme

Variables

A:Tableau [1..10,1..20] d'entiers

i,j, n, m : entier

Som, : réel

Début

Ecrire ('donner la dimension de la matrice')

Lire(n, m)

Pour i←1 à n **Faire**

Pour j←1 à m **Faire**

Lire(A[i,j])

FinPour

FinPour

Som ← 0

Pour i←1 à n **Faire**

Pour j←1 à m **Faire**

Som ← Som + A[i, j]

FinPour

FinPour

Écrire (‘La somme est =’,Som)

Fin

Remarque

La somme des éléments d’une matrice se fait comme suit:

- Entrée des éléments de la matrice.

- Initialisation de la variable Som.

-Addition des éléments de la matrice ligne par ligne.

-Déroulement :

Soit A une matriced’ordre2 × 3(n=2 et m=3)

2	4	-5
8	11	7

i	**j**	**A(i,j)**	**Som**
1	1	2	0+2
1	2	4	4+2=6
1	3	-5	6-5=1
2	1	8	1+8=9
2	2	11	9+11=20
2	3	7	20+7=**27**

Exercice 5

Ecrire un algorithme qui permet de calculer la transposée d'une matrice A de type réel

Solution

Algorithme transposee_matrice

Variables

A, T:Tableau [1..20,1..30] de réels

i, j,n,m:entier

Debut

Ecrire (‘donner la dimension de la matrice’)

Lire(n, m)

Pour i←1 à n **Faire**

Pour j←1 à m **Faire**

Ecrire (‘donner la dimension de la matrice’)

Lire(A[i,j])

FinPour

FinPour

Pour i←1 à n **Faire**

Pour j←1 à m **Faire**

T[j,i]←A[i,j]

FinPour

FinPour

Pour i←1 à n **Faire**

Pour j←1 à m **Faire**

Ecrire(T[I,j])

FinPour

FinPour

Fin

Exercie6

Ecrire un algorithme qui initialise à zéro la diagonale d'une matrice carré de type entier.

Solution

Algorithme diagonale

Variables

T:Tableau [1..20,1..20] d'entiers

i, j,n: entier

Debut

Ecrire ('donner la dimension de la matrice carré')

Lire(n)

Pour i←1 à n **Faire**

Pour j←1 à n **Faire**

Lire(T[i,j])

FinPour

FinPour

Pour i←1 à n **Faire**

T[i,i]←0

FinPour

Pour i←1 à n **Faire**

Pour j←1 à n **Faire**

Ecrire(T[i,j])

FinPour

FinPour

Fin

Remarque

- La diagonale est constituée d'un élément par ligne. (1, 1) ; (2, 2) ; (3, 3) ;…

-Une seule boucle (pour passer d'une ligne à une autre et traiter ainsi toutes les lignes) est nécessaire au traitement de la diagonale.

-Déroulement :

Soit T une matrice carré d'ordre 3 × 3(n=3)

2	4	-15
8	11	73
3	7	22

i	**T(i,i)**
1	~~2~~ 0
2	~~11~~ 0
3	~~22~~ 0

0	4	-15
8	0	73
3	7	0

Chapitre 8 : Procédures et Fonctions

1. Introduction

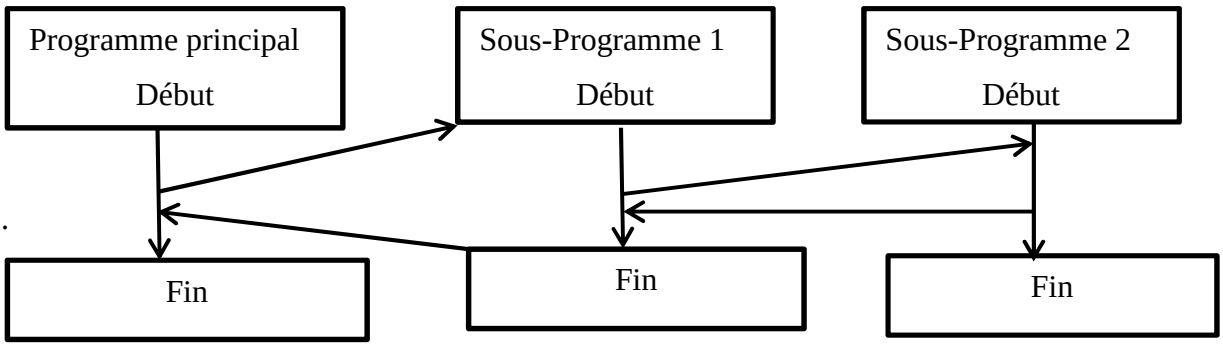
Lorsque l'algorithme devient trop compliqué, on aura envie de le découper, de manière à ce que chaque partie soit plus simple et donc plus lisible.

Pour maîtriser la complexité d'un problème, il est préférable de procéder à une conception modulaire qui consiste à le décomposer en plusieurs sous-problèmes. Ces sous-problèmes peuvent à leurs tours être décomposés jusqu'à aboutir à des traitements élémentaires simples. Ensuite, chaque sous-problème sera résolu par un sous algorithme (programme) qui peut-être une procédure ou une fonction.

2. Définition d'un sous algorithme

Un sous-algorithme 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. Le schéma suivant illustre le mécanisme d'appel et retour des sous algorithme



L'utilisation des sous algorithmes apportent plusieurs avantages :

- Eviter de réécrire un même traitement plusieurs fois.
- Organiser le code et améliorent la lisibilité des programmes.
- Faciliter la maintenance du code (il suffit de modifier une seule fois).

- Les sous algorithmes (procédures et fonctions) peuvent éventuellement être réutilisées dans d'autres programmes.

3. Les Paramètres d'un sous algorithme

Les paramètres servent à échanger des données entre l'algorithme principal et le sous algorithme, de ce fait avant de présenter les notions de fonctions et procédures il est indispensable de passer en revue les notions suivantes.

3.1 Variable globale

C'est une variable définie dans l'en-tête de l'algorithme principal. Elle est utilisable dans n'importe quel sous algorithme (programme) sans nécessité de redéfinition. Toutefois, si dans un sous-bloc il existe une variable qui porte le même nom que la variable globale, alors c'est cette variable qui sera considérée à l'intérieur du sous-bloc.

3.2 Variable locale

C'est une variable définie à l'intérieur d'un sous-algorithme (programme). Sa portée (visibilité) est limitée au bloc qui la contient. Il serait donc erroné de l'utiliser dans le bloc principal ou dans un autre sous-bloc appartenant à l'algorithme.

3.3 Paramètres

Considéré comme une variable locale, 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 2 types de paramètres :

3.3.1 Les paramètres formels

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

3.3.2 Les paramètres effectifs

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

4. Types de sous-algorithme

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

4.1 Procédure

Une procédure est un sous-algorithme (mini-programme) qu'on déclare en général dans la partie réservée aux variables. Étant donné qu'il s'agit d'un bloc à part entière, elle possèdera éventuellement un en-tête, une série de traitements, et une gestion des résultats tout comme l'algorithme qui la contient.

4.1.1 structure procédure

Procédure **<Nom_de_la_procedure>** **[** (liste des paramètres : type) **]**
Variables : liste des variables **{***Variables locales***}**
Debut **{***corps de la procédure***}**
<LISTE DES ACTIONS> {*la liste ne doit pas être vide*}
FinProcedure

4.1.2 Appel d'une procédure

Pour appeler une procédure dans un algorithme principale ou dans une autre procédure, il suffit d'écrire une instruction indiquant le nom de la procédure. La syntaxe de l'appel d'une procédure est comme suit:

Syntaxe

<Nom de procédure> (liste de paramètres)

4.1.3 Caractéristiques d'une procédure

- ✓ Une procédure peut appeler d'autres sous-algorithmes définis avant elle.
- ✓ La liste des paramètres dans une procédure est facultative. Mais quand elle existe, ces paramètres sont déclarés de la même façon qu'on déclare une série de variables de différents types.
- ✓ Les variables déclarées à l'intérieur de la procédure sont inutilisables à l'extérieur du bloc. Si leur type est prédéfini, alors ce type sera déclaré dans l'en-tête du bloc principal. Idem pour les paramètres de la procédure au cas où il en existe.

Exemple

Écrire un algorithme qui utilise une procédure qui permet de faire la somme de 100 nombres.

Algorithme essai

Variables

i, S : entier

Procedure Somme

Debut

S ←0

Pour i ←1 A 100 **Faire**

S←S + i

FinPour

Ecrire ('La somme est=', S)

FinProcedure

Debut

Somme

Fin

4.2 Fonctions

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.

4.2.1 Structure d'une fonction

Fonction **<Nom_de_la_fonction> [** (liste des paramètres : type) **] : Type_Fonction**

Variables : liste des variables **{***Variables locales***}**

Debut **{***corps de la fonction***}**

<LISTE DES ACTIONS> {*la liste ne doit pas être vide*}

<Nom_de_la_fonction> ←résultat_des_calculs

FinFonction

4.2.2 Appel d'une fonction

La fonction est appelée dans l'algorithme principal directement dans une instruction : en général, elle apparaît dans la partie droite d'une affectation. Lors de son appel, la fonction est évaluée à partir d'arguments qui lui sont fournis, le résultat vient se substituer au nom de la fonction dans l'expression appelante. La syntaxe de l'appel d'une fonction est comme suit:

Syntaxe

Nom_variable-globale ← Nom_Fonction (Paramètres)

4.2.3 Caractéristiques d'une fonction

- Étant 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.
- Une fonction peut appeler d'autres sous-algorithmes définis avant elle.
- On peut utiliser le nom d'une fonction presque comme une variable globale, puisqu'elle renferme une valeur (affectation, affichage, mais pas lecture)
- La liste des paramètres est facultative. Mais quand elle existe, ces paramètres sont déclarés de la même façon qu'on déclare une série de variables de différents types.
- Les variables déclarées à l'intérieur de la fonction sont inutilisables à l'extérieur du bloc. Si leur type est prédéfini, alors ce type sera déclaré dans l'en-tête du bloc principal. Idem pour les paramètres de la fonction au cas où il en existe.
- À l'intérieur du sous-algorithme le nom de la fonction ne doit figurer qu'en recevant le résultat final

Exemple

Écrire un algorithme qui en utilisant une fonction permet de faire la somme de N nombres.

Algorithme essai
Variables
i,Som : entier
Fonction Somme : entier
Variables
S : entier
Debut

S ←0

Pour i← 1 à 100 **Faire**

S ←S + i

FinPour

Somme ←S

FinFonction

Debut

Som ←Somme

Ecrire(‘La somme est=‘, somme)

Fin

5. Mode de passages de paramètres

Un sous algorithme avec paramètres est très utile parce qu’il permet de répéter une série d’opérations complexes pour des valeurs qu’on ne connaît pas à l’avance. Il existe deux types de passage de paramètres : par valeur et par variable (dite aussi par référence ou encore par adresse).

5.1 Passage paramètres par valeur

C’est le mode de transmission par défaut, il y a **copie** de la valeur des paramètres effectifs dans les variables locales issues des paramètres formels de la procédure ou de la fonction appelée. Dans ce mode, le contenu des paramètres effectifs ne peut pas être modifié par les instructions de la fonction ou de la procédure ; car nous ne travaillons pas directement avec la variable mais sur une copie. À la fin de l’exécution du sous-algorithme la variable conservera sa valeur initiale. Les paramètres dans ce cas sont utilisés comme données.

Syntaxe :

Procedure **Nom_procédure** (**param1 :**type1 **; param2, param3 :**type2)

Fonction **<nom_fonction>** (**param1 :**type1 **; param2 :**type2) : Type_fonction

Exemple

Algorithme Valeur

Variables

M : entier

Procedure Calcul (N : entier)

Debut

Si (N < 0) Alors

N←N*(-1)

FinSi

Ecrire (N)

FinProcedure

Debut

Lire (M)

Calcul (M)

Ecrire (M)

Fin

-Exécutons cet algorithme pour la valeur (-5)

- ✓ Avant l'appel de procédure : la seule variable déclarée est la variable globale (M)

M	**Ecran**
-5	

- ✓ Après l'appel de procédure : la variable-paramètre N est déclarée et reçoit en copie la valeur de M.

M	**N**	**Ecran**
-5	-5	
-5	+5	
-5	5	5

- ✓ Au retour à l'algorithme (au niveau de l'appel) il ne reste que la variable globale avec sa valeur initiale

M	**Ecran**
-5	
-5	-5

5.2 Passage paramètres par variable

Ici, il s'agit non plus d'utiliser simplement la valeur de la variable mais également son emplacement dans la mémoire (d'où l'expression « par adresse »). En fait, le paramètre formel se substitue au paramètre effectif durant le temps d'exécution du sous-programme et à la sortie il lui transmet sa nouvelle valeur. Un tel passage de paramètre se fait par l'utilisation du mot-clé Var.

Syntaxe

Procedure nom_procédure (Var param1 :type1 ; param2, param3 :type2)

Fonction <nom_fonction> (Var param1 : type1 ; param2 :type2) : Type_fonction

Dans ce cas param1 est passé par référence(par variable) alors que les deux autres paramétres sont passés par valeur

Exemple

Algorithme valeur

Variables

M : entier

Procedure Calcul (Var N : entier)

Debut

Si (N < 0) Alors

N←N*(-1)

FinSi

Ecrire (N)

FinProcedure

Debut

Lire (M)

Calcul (M)

Ecrire (M)

Fin

-Exécutons cet algorithme toujours pour la valeur (-5)

- ✓ Avant l'appel de procédure : la seule variable déclarée est la variable globale (M)

M	**Ecran**
-5	

- ✓ Après l'appel de procédure : la variable-paramètre nombre se substitue à la variable M

M **N**	**Ecran**
-5	
5	
5	5

- ✓ Au retour à l'algorithme il ne reste que la variable globale avec sa nouvelle valeur.

M	**Ecran**
-5	
5	5

Récapitulatif

- ✓ La notion de sous-algorithme représente toute la puissance du langage algorithmique, un sous algorithme peut être sous forme d'une procédure ou d'une fonction.
- ✓ Les procédures peuvent communiquer de 0 à plusieurs résultats au programme appelant à travers des paramètres résultats ou données/résultats.
- ✓ Les fonctions ne peuvent communiquer qu'un seul résultat au programme appelant à travers une valeur de retour (et non à travers un paramètre)
- ✓ La différence entre une procédure et une fonction et le type de retour. Une procédure ne possède pas de type de retour et une fonction possède un type de retour.
- ✓ Toute fonction doit avoir dans son corps comme dernière instruction :
  <Nom_fonction> ← <resultat>

- ✓ Contrairement à l'appel d'une fonction, on ne peut pas affecter la procédure appelée ou l'utiliser dans une expression.
- ✓ Toute procédure peut être convertie en une fonction et toute fonction peut être convertie en une procédure.
- ✓ Les paramètres placés dans la déclaration d'un sous algorithme sont appelés paramètres formels. Ce sont des variables locales.
- ✓ Les paramètres placés dans l'appel d'une sous algorithme sont appelés paramètres effectifs. Ils contiennent les valeurs pour effectuer le traitement.
- ✓ Le nombre de paramètres effectifs doit être égal au nombre de paramètres formels. L'ordre et le type des paramètres doivent correspondre.
- ✓ Les paramètres passés par valeur et par adresse peuvent cohabiter à l'intérieur d'un même sous-algorithme.

6. Exercices d'application

Exercie1

Soit l'algorithme suivant

Algorithme Test

Variables

X :entier

Procedure B

Variables

Y :entier

Début

Y← X*2

X←Y+1

Ecrire(Y)

FinProcedure

Debut

X←3

B

Ecrire(X)

B

Ecrire(x)

Fin

-Donner le déroulement détaillé de l'algorithme test en précisant l'état des variables après l'exécution de chaque instruction

Solution

X variable globale	**Y variable locale**
3	?
3	6
7	6
7	14
15	14

Exercice 2

En utilisant une action paramétrée (fonction ou procédure). Ecrire un algorithme qui permet de faire la somme de deux entiers

Solution

1) Solution avec Procédure

Algorithme somme

Variables

A,B,S :Entier

Procedure Som(X,Y :Entier,Var Z :Entier)

Debut

Z←X+Y

FinProcedure

Debut

Ecrire('Donner la valeur de deux entiers')

Lire(A,B)

Som(A,B,S)

Ecrire(‘La somme est =’,S)

Fin

2) Solution avec Fonction

Algorithme somme

Variables

A,B,S :Entier

Fonction Z (X,Y :Entier) :Entier

Debut

Z←X+Y

FinFonction

Debut

Ecrire(‘Donner la valeur de deux entiers’)

Lire(A,B)

S←Z(A,B)

Ecrire(‘La somme est =’,S)

Fin

Exercice3

En utilisant une action paramétrée (fonction ou procédure). Ecrire un algorithme qui permet de trouver le maximum entre deux nombres entiers.

Solution

1)Solution avec Procédure

Algorithme Nombre

Variables

A,B,Max :Entier

Procedure maximum (X,Y :Entier,Var M :Entier)

Debut

Si(X> Y)alors

M← X

Sinon

M←Y

FinProcedure

Debut

Ecrire(‘Donner la valeur de deux entiers’)

Lire(A,B)

maximum (A,B,Max)

Ecrire(‘Le maximum somme est =’,Max)

Fin

2) Solution avec Fonction

Algorithme Nombre

Variables

A,B,Max :Entier

Fonction M (X,Y :Entier) :Entier

Debut

Si(X> Y)alors

M← X

Sion

M←Y

FinFonction

Début

Ecrire(‘Donner la valeur de deux entiers’)

Lire(A,B)

Max←M(A,B)

Ecrire(‘Le maximum somme est =**’,** Max**)**

Fin

Exercie4

En utilisant une action paramétrée (fonction ou procédure). Ecrire un algorithme qui permet de calculer le factoriel d’un nombre entier

Solution

1) Solution avec Procédure

Algorithme Factoriel

Variables

A,F :Entier

Procedure Factor(X :Entier,Var Fact :Entier)

Variables

i:Entier

Debut

Fact←1

Pour i ←1 à X **Faire**

Fact← Fact*i

FinProcedure

Debut

Ecrire(‘Donner une valeur de type entier)

Lire(A)

Factor (A,F)

Ecrire(‘Factoriel est =’,F)

Fin

2) Solution avec Fonction

Algorithme factoriel

Variables

A,F :Entier

Fonction Fact (X :Entier):Entier

Variables

i:Entier

Debut

Fact←1

Pour i ←1 à X **Faire**

Fact← Fact*i

FinFonction

Debut

Ecrire(‘Donner une valeur de type entier)

Lire(A)

F←Factor (A)

Ecrire(‘Factoriel est =’,F)

Fin