Ministère de l'Enseignement Supérieur, de la Recherche Scientifique et de la Technologie
Direction Générale des Études Technologiques
✯✯✯✯✯

Institut Supérieur des Études Technologiques du Kef
Département INFORMATIQUE
✯✯✯✯✯

Matière : Programmation Evénementielle

Mots-clés du Support

Introduction à la programmation événementielle, les Structures Simples et composé en Visual Basic, la manipulation des Fichiers, Les bases de données etc.

Support de cours adressé aux étudiants du troisième niveau (informatique de gestion) ISETs

Enseignants : Mossaab BOUKHCHIM
Ikbel DAI

Année Universitaire : 2005-2006

Sommaire

Fiche de Présentation

PROGRAMMATION EVENEMENTIELLE

Enseignant Mossaab BOUKHCHIM

Population	**:**	Étudiants du troisième niveau (informatique de gestion) ISETs.
Crédits	**:**	67,5 heures par semestre
Volume Horaire	**:**	4,5 heures par semaine sur 15 semaines
Pré-requis	**:**	Algorithmique
Langue	**:**	Français

Objectifs du cours

- Présenter les principaux caractéristiques de la programmation orienté objet ainsi que les traitements qui leur sont associés.
- Amener les étudiants à construire de petites applications selon le principe de la programmation orienté objet par contrat, c'est-à-dire en s'appuyant sur les spécifications.
- Se familiariser avec l'environnement du développement qui manipule des donnés provenant d'un support de stockage (fichier et base de données).

Évaluations

- Test d'une heure.
- Devoir surveillé d'une heure.
- Examen final écrit de 2 heures sur tout **le programme.**
- Examen TP.
- Trois Mini projets qui englobe tous les aspects de la programmation événementielle.

Moyens Pédagogiques

- Support de cours papier.
- Support de cours numérique.
- Série de travaux dirigés.
- Problèmes industriels : Gestion des stocks, Gestion de la scolarité, Simulation d'un DAB, etc.

But du cours et objectifs généraux

PROGRAMMATION EVENEMENTIELLE

Objectifs Généraux	Conditions de réalisation de la performance	Critères d'évaluation de la performance
Connaître l'environnement de développement de Visual Basic	A partir des notes de cours et des références bibliographiques, l'étudiant devrait identifier les outils de Microsoft Visual Basic.	Aucune erreur n'est permise.
Connaître les types simples et les opérations définies sur ces types et savoir déclarer des variables, des types composés et des constantes.	A partir des notes de cours, des références bibliographiques et les TDs l'étudiant devrait être capable de déclarer des variables, des types et des constantes.	Aucune erreur n'est permise. L'étudiant doit pouvoir identifier avec succès les types, les variables et les constantes.
Etudier, comparer et écrire les structures de contrôle.	A partir des notes de cours et les TDs l'étudiant devrait être capable de connaître les structures itératives et les structures conditionnelles.	L'étudiant doit être capable d'écrire correctement une application en VB utilisant les structures de contrôle adéquates pour la résolution d'un problème donné.
Connaître la définition des procédures et des fonctions et leurs structure générale	A partir des notes de cours et les TDs l'étudiant devrait être capable d'utiliser les procédures et fonctions	L'étudiant doit être capable de décomposer un problème complexe et d'écrire correctement une application utilisant les fonctions et procédures.
Comprendre la notion des fichiers en VB.	A partir des notes de cours et les TDs l'étudiant devrait Comprendre la notion des fichiers.	L'étudiant doit être capable de manipuler les fichiers en VB.
Connaître les outils qui assurent la récupération et la mise à jour des informations des bases de données.	A partir des notes de cours et les TDs l'étudiant devrait Comprendre la notion des Bases de données	L'étudiant doit être capable de manipuler les Bases de données en VB.

Décomposition des objectifs généraux en objectifs spécifiques

Objectif général
(Chapitre 1 : Introduction à la programmation événementielle)

Connaître l'environnement de développement de Visual Basic

Objectifs spécifiques	Eléments du cours	Méthodologies	Durée
Connaître les concepts indispensables pour la programmation événementielle.	I. Introduction II. La programmation événementielle 1. les types de programmation 2 Propriété, méthode et événement III. Les outils 1 Les environnements de développement intégrés 2 Quelques exemples d'IDE	Exposé informel (tableau et Notes de cours)	30 min
Connaître les outils de Visual Basic	IV. Présentation de Microsoft Visual Basic 1 Principe de fonctionnement 2 Organisation d'une application VB 3 Les principaux fichiers d'une application VB	Exposé informel (tableau et Notes de cours)	60 min
Exercices de TD (Présentation de l'environnement de travail)			3h00
Total			**4h30**

Objectif général (Chapitre 2 : Les Structures de Données)

Connaître les types simples et les opérations définies sur ces types et savoir déclarer des variables, des types composés et des constantes.

Objectifs spécifiques	Eléments du cours	Méthodologies	Durée
Connaître la notion des variables et comprendre leurs utilités.	I. Introduction II. Les Types de données simples 1 Définition 2 Les types de données simples 3 Déclaration des variables 3.1 Déclaration explicite des variables 3.2 Déclaration implicite des variables.	Exposé informel (tableau et Notes de cours)	30 min
Connaître la portée des variables déclarés.	4 Portée des variables	Exposé informel (tableau, exemples d'applications et Notes de cours)	30 min
Connaître la notion des constantes et comprendre leurs utilités.	5 Les constantes	Exposé informel (tableau, exemples d'applications et Notes de cours)	30 min
Connaître les tableaux en VB	III. Les tableaux 1 Première méthode 2 Deuxième méthode 3 Troisième méthode 4 Quatrième méthode	Exposé informel (tableau, exemples d'applications et Notes de cours)	45 min
Connaître les types intervalle, enregistrement et énuméré.	IV. Les types utilisateurs 1 Les enregistrements 2 Les énumérations	Exposé informel (tableau, exemples d'applications et Notes de cours)	30 min
Connaître les opérations de base.	V. Les opérateurs 1 Arithmétique et comparaison 2 Les opérateurs logiques 3 Les opérateurs de chaîne de caractères	Exposé informel (tableau, exemples d'applications et Notes de cours)	15 min
Exercices de TD (Manipulation des objets)			6h00
Total			**9h00**

Objectif général (Chapitre 3 : Les Structures de contrôle)

Etudier, comparer et écrire les structures de contrôle.

Objectifs spécifiques	Eléments du cours	Méthodologies	Durée
Connaître les structures conditionnelles et écrire des programmes en utilisant ces notions.	I. Les structures conditionnelles 1 Structure simple et imbriqué 2 Structure à choix multiple	Exposé informel (tableau et Notes de cours)	30 min
Etudier, comparer et écrire les structures itératives.	II. Les structures itératives 1 For … Next 2 For each …. Next 3 Do …. Loop	Exposé informel (tableau, exemples d'applications et Notes de cours)	30 min
Apprendre à choisir correctement une structure bien déterminée.	III. Résumé	Exposé informel (tableau, exemples d'applications et Notes de cours)	30 min
Exercices de TD (Manipulation des objets)			3h00
Total			**4h30**

Objectif général (Chapitre 4 : Les Procédures et les Fonctions)

Connaître la définition des procédures et des fonctions et leurs structure générale

Objectifs spécifiques	Eléments du cours	Méthodologies	Durée
Connaître la notion des procédures	I. Les procédures	Exposé informel (tableau et Notes de cours)	20 min
Connaître la notion des fonctions	II. les fonctions	Exposé informel (tableau, exemples d'applications et Notes de cours)	20 min
Connaître la notion des modules.	III. Module du code	Exposé informel (tableau et Notes de cours)	20 min
Connaître les fonctions prédéfinies	IV. Quelques fonctions prédéfinies 1 Fonction de conversion 2 Fonction de chaîne de caractères	Exposé informel (tableau, exemples d'applications et Notes de cours)	30 min
Exercices de TD (Manipulation des objets)			6h00
Projet N : 1 (Manipulation des objets)			3h00
Total			**10h30**

Objectif général (Chapitre 5 : Les Fichiers en VB)

Connaître la définition des procédures et des fonctions et leurs structure générale

Objectifs spécifiques	Eléments du cours	Méthodologies	Durée
Connaître la notion des Fichiers.	I. Rappel 1. Définition 2. Accès	Exposé informel (tableau et Notes de cours)	20 min
Connaître les fichiers en VB	II. Les fichiers en VB 1. Syntaxe 2. Manipulation des fichiers	Exposé informel (tableau, exemples d'applications et Notes de cours)	50 min
Se familiariser avec les fichiers séquentiels et à accès direct	III. Les fichiers séquentiels. IV. Les fichiers à accès direct.	Exposé informel (tableau, exemples d'applications et Notes de cours)	20 min
Exercices de TD			6h00
Projet N : 2			3h00
Total			**10h30**

Objectif général (Chapitre 6 : Gestion des bases de données)

Connaître la définition des procédures et des fonctions et leurs structure générale

Objectifs spécifiques	Eléments du cours	Méthodologies	Durée
Connaître les objets nécessaires pour manipuler les données d'une base	I. La hiérarchie des objets du modèle ADO	Exposé informel (tableau et Notes de cours)	30 min
Connaître les traitements assurant la connexion et la déconnexion d'une source de données	II. La connexion à une base de données III. Déconnexion d'une source de données	Exposé informel (tableau, exemples d'applications et Notes de cours)	15 min
Connaître les objets nécessaires pour la récupération des données d'une base	IV. La récupération de données 1 Propriétés de l'objet Recordset 2 Méthodes de l'objet Recordset	Exposé informel (tableau, exemples d'applications et Notes de cours)	45 min
Connaître les scénarios de récupération utilisant les objets déjà décrits.	3 Les différents scénarios de récupération de données	Exposé informel (tableau, exemples d'applications et Notes de cours)	45 min
Connaître les scénarios de la mise à jour des données d'une base.	V. Les scénarios de la mise à jour des données	Exposé informel (tableau, exemples d'applications et Notes de cours)	45 min
Exercices de TD			6h00
Projet N : 3			3h00
Total			**12h00**

Chapitre I

Introduction à la Programmation Evénementielle

Objectif

- ✓ *Connaître les concepts indispensables pour la programmation événementielle.*
- ✓ *Etre sensibilisé des apports de cette programmation dans la conception et la réalisation des systèmes d'information.*
- ✓ *Connaître les outils de Visual Basic*

Eléments de contenu

- ✓ *Les types de programmations*
- ✓ *Les outils*

Chapitre I

Introduction à la programmation événementielle

I. Introduction

Dans les années 80, les micros ordinateurs ont obtenu les capacités nécessaires pour supporter des interfaces graphiques. La complexité des applications utilisant des interfaces graphiques est importante car il faut gérer un nombre important d'actions de bas niveau (par exemple, les fenêtres sur l'écran). Il fallait donc décharger le concepteur d'applications de toutes ces difficultés, l'une raison des raisons qui a favorisé la naissance des outils de développement pour des applications basées sur les interfaces graphiques. La programmation événementielle et les langages de programmation associés sont donc apparus à cette époque.

II. La programmation événementielle

1. Les types de programmations

Il existe un ensemble de langages de programmation, chacun est spécialisé dans un domaine d'application donné et chacun possède un type spécifique. On distingue :

- *Programmation structuré ou modulaire* : le programme est vu comme un ensemble d'unités structurées hiérarchiquement. Il existe une interaction entre les modules et on fait généralement distinction entre les données et les traitements.
- *Programmation orientée objet* : le programme n'est autre qu'une collection d'objets qui communiquent. Un objet et par définition une instance d'une classe dont les composantes peuvent être de deux types : propriétés et méthodes.
- *Programmation événementielle* : Les composants d'une application événementielle c'est à dire les objets interagissent entre eux et avec l'environnement. Ils communiquent en réponse à des événements. Ces événements peuvent correspondre à une action de l'utilisateur : un click sur un bouton de commande, une écriture dans une zone de texte, un choix dans une case d'option ou une case à cocher, le déplacement d'un objet, ... Ils peuvent aussi être déclenchés par le système : chargement d'une feuille, un top déclenché par l'horloge, ...

Les événements sont captés par le système d'exploitation, le traitement consiste en l'exécution des procédures événement associées à celui-ci, s'ils en existent. C'est le programmeur qui doit prévoir la procédure à exécuter en réponse à un événement donné.

Par exemple, le déclenchement de l'événement click sur un bouton quitter doit terminer l'exécution, le choix d'un élément dans un menu doit déclencher certaines opérations, un top d'horloge doit modifier le contenu d'une zone d'image.

2. Propriétés, méthodes et événements

La programmation à objets, à ne pas confondre avec la programmation Orientée Objet (POO) a eu un essor assez important et ce avec l'apparition de Visual Basic. La programmation à objets n'adopte pas tous les principes de la POO. En effet, elle hérite uniquement l'aspect d'encapsulation. Les objets sont alors caractérisés par un ensemble de propriétés et de méthodes. Dans le but de rendre cette programmation par objet exploitable par les interfaces graphiques, on a ajouté les événements. Un événement est encapsulé autour de l'objet. Ceci a donné naissance à la programmation événementielle.

Un objet

Propriétés
Méthodes
Evénements

Désormais un objet est caractérisé par les propriétés qui le décrivent, les méthodes qu'il peut effectuer ainsi que les événements qui peuvent être généré par cet objet.

- Les programmes Visual Basic sont commandés par les événements générés par les objets de l'interface utilisateur.
- Par exemple, le chargement d'une feuille est un événement auquel certaines instructions peuvent être rattachées, comme par exemple l'effacement d'une feuille dans la mémoire. L'événement sera causé par un clic de souris, dans le cas d'un bouton, ou par une modification du contenu d'une zone de texte.

Important : Le programmeur doit toujours avoir à l'esprit le schéma suivant :

Objet : O ***Evénement : E*** ***Traitement : T***
Si l'événement E se produit sur l'objet O alors le traitement T est exécuté.

III. Outils

1. Les environnements de développement intégrés (IDE : Integrated Develompent Environment)

Un environnement de développement est un ensemble d'outils qui, en plus des tâches classiques d'un langage de programmation offre des fonctionnalités étendues permettant de couvrir une plus large partie du cycle de création d'un logiciel.

On peut distinguer des tâches telles que :

- ✓ Compilation et déboguage
- ✓ Déploiement de l'application.
- ✓ Intégrer des outils de test et de vérification.
- ✓ Outil de création graphique (icône).
- ✓ Interface avec les SGBD.
- ✓ Générateur de menus.
- ✓ Un ensemble d'assistants.

Un IDE est basé sur un langage de programmation.

2. Quelques exemples d'IDE

Dans ce qui suit, des exemples d'IDE chez des éditeurs différents et basés sur des langages de programmation différents :

- ✓ Borland Delphi
- ✓ Borland Jbuilder
- ✓ Oracle JDevelopper
- ✓ Microsoft Visual Basic
- ✓ Microsoft Visual C++

IV. Présentation de Microsoft Visual Basic

MS VB est l'un des outils les plus utilisés pour les applications en informatique de gestion. VB fait partie de la suite Microsoft Visual Studio.

VB tourne sur les plates-formes MS Windows. Il est né avec l'interface graphique de Windows 3.0

VB est un langage interprété, pour s'exécuter, une application VB a besoin au moins du fichier VB6FR.DLL (le cas de VB 6 version française).

1. Principe de fonctionnement de VB

- ✓ Conception de l'interface.
- ✓ Mise en place des contrôles.
- ✓ Ecriture du code
- ✓ Test et exécution.

2. Organisation d'une application VB

VB possède une grille d'écran principal (forme principale) qui va contenir des contrôles, un menu.

La forme principale appelle d'autres formes (formes filles) qui sont appelées des boites de dialogue. Les boites de dialogue peuvent être :

- ➢ Boite de dialogue modales : si elle est ouverte, elle recevra tous les événements et aucune autre forme ne peut prendre le focus tant que cette dernière est ouverte.
- ➢ Boite de dialogue non modales : on peut ouvrir plusieurs qui seront chargées en mémoire centrale et on peut passer d'une fenêtre à une autre.

3. Les principaux Fichiers d'une application VB

En VB, une application correspond a un projet, un projet est n'est autre qu'une collection de fichiers :

(.VBP) : le fichier de projet regroupe tous les fichiers et les contrôles associés ainsi que les informations sur la configuration de l'environnement.

(.FRM) : contient une description de la forme ainsi que de son contenu.

(.FRX) : la version exécutable de chaque forme.

(.BAS) : module standard pouvant contenir des déclarations (types, constantes, variables, procédures, fonctions)

(.OCX) : Contrôle ActiveX : ce sont des contrôles optionnels qu'on peut ajouter.

(.RES) : fichiers de ressources pouvant contenir des bitmaps, du texte ou bien autres types de données. Ces fichiers peuvent être modifiés sans refaire la compilation du code.

Chapitre II

Structure de données

Objectif

- ✓ *Connaître la notion des variables et comprendre leurs utilités..*
- ✓ *Connaître la portée des variables déclarés.*
- ✓ *Connaître la notion des constantes et comprendre leurs utilités.*
- ✓ *Connaître les tableaux en VB.*
- ✓ *Connaître les types intervalle, enregistrement et énuméré*
- ✓ *Connaître les opérations de base..*

Eléments de contenu

- ✓ *Types de données simples*
- ✓ *Les tableaux*
- ✓ *Les types utilisateurs*
- ✓ *Les opérateurs*

Chapitre II

Structure des données

I. Introduction

Un programmeur s'il peut mémoriser des données dans les propriétés des contrôles (text1.text, label1.caption) il peut également placer temporairement des informations dans des emplacements précis de la mémoire de l'ordinateur. Ces emplacements sont nommés variables.

II. Les variables (Les Types de données simples)

1. Définition

Les variables sont des données ou des valeurs qui peuvent changer à l'intérieur d'une application. C'est pourquoi, il est fort utile de les *nommer par un nom*, de déclarer quel genre de variables est-ce (nombre entier, nombre décimal, lettres...) et leur *affecter*, lorsque cela est nécessaire une valeur. La longueur maximale du nom d'une variable est de 255 caractères. Ceux-ci peuvent être des chiffres, des lettres ou autres symboles. Notez que ce nom doit obligatoirement commencer par une lettre. En effet, Visual basic classe les variables en fonction de la valeur affectée à la variable. Ainsi, une variable déclarée comme du type numérique ne peut pas recevoir une valeur chaîne de caractère, ainsi qu'à l'inverse. Notez que si vous ne déclarez pas une variable, Visual Basic se chargera d'affecter par défaut un type de variable (***Variant***) à celle-ci. Une variable du type *Variant* peut aussi bien recevoir des données numériques que des chaînes de caractères. Tout dépend de ce que vous avez affecté à cette variable

2. Type de variables simples

A présent, observons de plus près les différents types de variables :

Type de données	Mot clé	Taille d'enregistrement	Plage
Octet	Byte	1 octet	0 à 255
Logique	Boolean	2 octets	True(-1) ou False(0)
Entier	Integer	2 octets	-32 768 à 32767
Entier long	Long	4 octets	-2 147 483 648 à 2 147 483 647
Décimal simple	Single	4 octets	Nombre réel avec 7 chiffres après la virgule
Décimal double	Double	8 octets	Nombre réel avec 15 chiffres après la virgule
Monétaire	Currency	8 octets	Nombre réel avec 15 chiffres avant la virgule et 4 chiffres après la virgule
Date	Date	8 octets	1er janvier 100 au 31 décembre 9999
Objet	Object	4 octets	Toute référence à des types Object
Chaîne de caractères	String	10 octets + longueur de chaîne	Chaîne de caractère dont la longueur ne doit pas excéder 2^31 caractères
Variant (avec chiffres)	Variant	16 octets	toute valeur numérique jusqu'à l'étendue d'un double
Variant (avec lettres)	Variant	22 octets+longueur de chaîne	Même étendue que pour un String de longueur variable
Défini par l'utilisateur	Type	-	L'étendue de chaque élément est la même que son type de données

Notez que les types de variables les plus utilisées sont : String, Integer, Long, Double.

3. Déclaration de variables

Pour utiliser des variables, il est normalement obligatoire de les prédéfinir, soit dans la ***section Déclarations*** de la liste déroulante Objet, soit en début de procédure ou de fonction. Un programme où les variables sont bien déclarées rend un programme plus facile à comprendre, à lire et surtout à corriger en cas d'erreurs. Certes, il n'est pas obligatoire de les déclarer mais faites-le quand même, c'est un conseil. Si vous êtes prêt à déclarer une variable que vous voulez utilisez mais que vous êtes un étourdi, alors, utilisez simplement l'instruction *Option Explicit* (à placer dans la section Déclarations de la liste déroulante Objet) qui vous oblige à chaque fois à déclarer toutes vos variables avant de pouvoir exécuter l'application.

- ***Déclaration explicite d'une variable***

Pour déclarer une variable, on utilise l'instruction *Dim* suivi du nom de la variable puis du type de la variable.

Exemple :

```
Dim DateNaissance
Dim Message, Titre As String
```

- ✓ Remarquez que la 1ère déclaration ne contient pas d'information sur le type de variable. Si vous déclarez une variable sans donner d'information sur le type de variable que c'est, alors, cette variable (DateNaissance) aura par défaut une variable du type Variant. Si vous avez bien lu l'exemple précédent, vous aurez compris qu'il s'agit ici d'une variable de type Variant (avec chiffres) qui lui est affectée par défaut.
- ✓ La 2ème déclaration est par contre explicite. Vous pouvez aussi mettre 2 variables sur une même ligne à condition que le type de variable est le même pour les 2 et en les séparant par une virgule.

- ***Déclaration implicite d'une variable***

Il existe une autre méthode de déclarer des variables. Pour cela, il suffit d'ajouter juste avant la variable, un symbole spécifique. Voici la liste des symboles avec le type de variable auxquelles ils se rapportent :

Symbole	Type de variable
%	Integer
&	Long
!	Single
#	Double
@	Currency
$	String

Dans l'exemple ci-après, les deux premières instructions sont équivalentes à la troisième :

```
Dim V1 as Integer
V1 = 25
```

```
V1% = 25
```

4. Portée des variables

En général, toute variable déclarée a une portée limitée. Cette même variable a une valeur nulle au départ. De plus, elle ne s'applique pas forcément à toutes les procédures d'une application. Tout dépend de 2 éléments : ***la manière de déclarer*** et ***l'emplacement*** de la variable.

- dans une procédure, si vous déclarez une variable à l'aide de l'instruction *Dim*, sa portée se trouve limitée seulement à cette procédure. On dit que la variable est *locale*. Elle est donc initialisée à chaque appel de la procédure et détruite lorsque celle-ci se termine (à moins que vous remplaciez le mot *Dim* par *Static*). Elle n'est pas accessible en dehors de la procédure. Vous pouvez remplacer l'instruction *Dim* par *Private*, les deux termes étant équivalentes s'ils sont placés à l'intérieur d'une procédure.
- Si vous déclarez une variable dans la section Général/Déclarations d'une feuille ou d'un module en utilisant l'instruction *Dim* (ou *Private*), la variable est dite *locale au module*. Cette variable est disponible pour toutes les procédures de la feuille ou du module, mais pas pour les autres feuilles du projet.
- Enfin, si vous déclarez une variable dans la section Général/Déclarations d'un module (et non d'une feuille) en utilisant l'instruction *Public* au lieu de *Dim*, elle devient accessible par toutes les feuilles et tous les modules de l'application. On dit qu'elle est globale.

5. Les constantes

Contrairement aux variables dont les valeurs diffèrent souvent, les constantes ont des valeurs fixes. Mais tout comme les variables, on affecte aux constantes, un nom et une valeur qui est elle, fixe. De plus, les constantes se définissent de la même façon que les variables. Tout dépend donc de l'endroit où est défini la constante. Le mot *Const* peut être précédé de l'option *Public* pour que toutes les feuilles et modules de l'application puissent y accéder. Attention cependant à ne pas affecter à une constante des noms identiques aux constantes prédéfinies (VbSystemModal, VbOkOnly, VbArrow...) dans Visual Basic ! Prenons l'exemple du taux de TVA :

```
Const TVA = 20.6
TotalTTC=PrixHorsTaxe x (1 + (TVA / 100))
```

III. Les tableaux

1. Première méthode pour définir un tableau

Les tableaux de valeurs sont utiles lorsque l'on manipule des données en tout genre. Ainsi, dans un carnet d'adresses, vous pouvez stocker toutes les adresses dans un tableau structuré. Par exemple :

```
Dim Adresses (1 to 50 ) As String
Adresses(1) = "12, avenue de la Fayette, Tunis"
........
Adresses(50) = "4, rue de la Paix, Kef"
```

Comme vous pouvez le constatez, la définition d'un tableau ressemble beaucoup à celle d'une variable à la seule différence que vous devez donner une limite aux nombres de valeurs contenues dans un tableau. Pour appeler une des variables du tableau, il faut faire précéder de la variable, un numéro qui permet d'identifier la variable spécifique recherchée parmi toutes celles du tableau. Ce tableau est à une dimension, Voici un exemple à 2 dimensions:

```
Dim Adresses (1 to 50, 1 to 50) As String
Adresses(1,0) = "Ali"
Adresses(1,1) = "12, Rue de la paix, Kef"
.....
Adresses(50,0) = "Foazi"
Adresses(50,50) = "7, Boulevard de la Paix"
```

Notez qu'avec les fonctions LBound et UBound, vous pouvez obtenir les limites inférieures et supérieures d'un tableau.

2. Deuxième méthode pour définir un tableau

Il existe une autre façon de définir un tableau :

```
Dim Adresses(50) As String
Adresses(0) = "12, avenue de la Fayette, Tunis"
........
Adresses(49) = "4, rue de la Paix, Kef"
```

Avec cette méthode, les limites du tableau ne sont pas définies. Notez que le nombre d'adresses est de 50 mais la numérotation va de 0 à 49 et non de 1 à 50. Il existe cependant un moyen pour que la numérotation commence à 1 : c'est en utilisant l'instruction *Option Base* (par exemple: Option Base = 1) qu'il faut placer dans la section Déclarations.

3. Troisième méthode pour définir un tableau

Pour construire un tableau, on peut aussi utiliser la fonction *Array* : Exemple :

```
Dim Semaine, Jour As String
Semaine = Array("lundi", "mardi", "mercredi", "jeudi",
"vendredi", "samedi", "dimanche")
Jour = semaine(3) 'retourne jeudi
```

4. Quatrième méthode pour définir un tableau

Une autre méthode consiste en la création de tableaux dynamiques. Pour comprendre cette méthode, prenons un exemple :

```
Dim Adresse() As String
Dim NbAdresses As Integer
'Nouvelle adresse
NbAdresses = nbAdresses + 1
Redim Adresse (NbAdresses)
Adresse (NbAdresses) = "75, Rue bab djédid, Tunis"
```

Ainsi, à chaque fois que l'on lance cette procédure, le nombre total d'adresses (NbAdresses) est incrémenté (augmenté) de 1 et la nouvelle adresse est placé à la fin du tableau. en effet, *Redim* a pour effet de reformater le tableau, et donc, tout ceux qui était dans le tableau est aussi effacé sauf la dernière valeur entrée. Cependant, il existe un moyen de conserver les données déjà présentes dans le tableau. Pour cela, on fait appel à l'instruction *Preserve* :

```
Redim Preserve Adresse (NbAdresses)
```

Remarque :

Enfin, pour terminer, il est possible d'effacer toutes les données d'un tableau. Pour cela, on utilise l'instruction Erase suivi du nom du tableau.

IV. Les types utilisateurs

1. Les enregistrements

```
Type Nom_typ
Champ1 as type1
Champ2 as type2
Champ3 as type3
…
champN as typeN
End type
```

Exemple

```
Type Personne
Nom as string (30)
Prenom as string (30)
Tel as string (8)
End type
Dim Vperso as Personne
```

2. Les énumérations

```
Enum NomType
Menbre1 = Valeur1
Menbre2 = Valeur2
…
Menbren = Valeurn
End enum
```

Exemple

```
Enum DroitAcess
     Tous=1
     Exclusif=2
     Aucun=3
End enum
Dim vardroit as DroitAccess
```

Remarque

L'instruction Enum ne peut apparaître qu'au niveau module.

V. Les opérateurs

1. Arithmétiques et de comparaison

Arithmétiques		
Opérateur	**Signification**	**Exemple**
+	Ajoute deux nombres	A + B
-	Soustrait deux nombres ou rend négatif un nombre	A – B
*	Multiplie deux nombres	A * B
\	Divise entière	A \ B
/	Division	A / B
Mod	Reste de la division	A Mod B

Comparaison		
<	Inférieur	A < B
<=	Inférieur ou égal	A <= B
>	Supérieur	A > B
>=	Supérieur ou égal	A >= B
=	Egal	A = B
<>	Différent	A <> B
Like	Comparaison de caractère	A Like B

2. Logiques

Opérateur	**Signification**
NOT	Opposé
AND	Et
OR	Ou logique
XOR	Ou exclusif
IMP	Implication
EQV	Equivalence

P IMP Q = NOT(P) OR Q

P EQV Q = (NOT(P) OR Q) AND (NOT(Q) OR P)

3. Les opérateurs de chaîne de caractères

&: concaténation

Chapitre III

Structure de contrôle

Objectif

- ✓ *Connaître les structures conditionnelles et écrire des programmes en utilisant ces notions.*
- ✓ *Etudier, comparer et écrire les structures itératives.*
- ✓ *Apprendre à choisir correctement une structure bien déterminée.*

Eléments de contenu

- ✓ *Structures conditionnelles*
- ✓ *Structure itératives*
- ✓ *Résumé*

Chapitre III

Structure de contrôle

I. Structures conditionnelles

1. La structure: If...Then...Else...end If

Voici un exemple de structure simple avec l'instruction *If* :

```
If condition Then
traitement 1
else
traitement 2
End if
```

Interprétation : Si la condition est vérifiée alors les instructions 1 sont exécutées sinon les instructions 2 sont exécutées à la place. (Notez que l'on peut utiliser des opérateurs logiques *And* (et) et *Or* (ou) pour que plusieurs conditions doivent d'abord être vérifiées avant de pouvoir exécuter les instructions suivantes.). Le mot *Else* et les instructions qui suivent ne sont pas obligatoires.

Voici un autre exemple de structure avec *If* mais un peu plus complexe:

```
If Condition 1 Then
traitement 1
ElseIf Condition 2 Then
traitement 2
Else
traitement x
End If
```

Interprétation : Si la condition 1 est vérifiée alors les instructions 1 sont exécutées. Sinon si la condition 2 est vérifiée alors les instructions 2 sont exécutées. Sinon, si aucune de ces deux conditions ne sont vérifiée alors les instructions X sont exécutées.

2. La structure Select Case...End Select

Lorsque l'on doit effectuer toute une série de tests, il est préférable d'utiliser la structure Select Case...End Select au lieu de If...Then...End if.

Ainsi, les deux exemples suivants sont équivalents :

```
Select Case Semaine
Case 1
Jour = "Lundi"
Case 2
Jour = "Mardi"
Case 3
Jour = "Mercredi"
Case 4
Jour = "Jeudi"
Case 5
Jour = "Vendredi"
Case 6
Jour = "Samedi"
Case 7
Jour = "Dimanche"
Else
MsgBox "erreur", vbOKOnly , "Note"
End Select
```

```
If Semaine = 1 Then
Jour = "Lundi"
ElseIf Semaine = 2 Then
Jour = "Mardi"
ElseIf Semaine = 3 Then
Jour = "Mercredi"
ElseIf Semaine = 4 Then
Jour = "Jeudi"
ElseIf Semaine = 5 Then
Jour = "Vendredi"
ElseIf Semaine = 6 Then
Jour = "Samedi"
ElseIf Semaine = 7 Then
Jour = "Dimanche"
Else
MsgBox "erreur", vbOKOnly , "Note"
End If
```

Interprétation : Si la variable Semaine vaut 1, alors la variable Jour reçoit la valeur Lundi. Si la variable Semaine vaut 1, alors la variable Jour reçoit la valeur Mardi. (Etc...). Si la variable Semaine ne reçoit aucune valeur comprise entre 1 et 7, alors un message indiquant une erreur est affiché.

II. Structure itératives

1. La structure For..Next

La structure For...Next est utile lorsqu'on doit répéter plusieurs fois la même instruction. Exemple :

```
For J = 1 To 10
T(J) = J
Next
```

Interprétation : Dans cette boucle, l'instruction "T(J)=J" est répétée 10 fois c'est à dire jusqu'à ce que J soit égale à 10. A chaque passage, l'instruction "T(J)=J" affecte à T(J) les valeurs 1 à 10.

Pour que l'instruction soit exécutée une fois sur deux, vous pouvez utiliser l'instruction *Step* :

```
For J = 1 To 10 Step 2
T(J) = J
Next
```

Interprétation : Dans cette boucle, l'instruction "T(J)=J" affecte à T(J) les valeurs 1,3,5,7,9.

2. La structure For Each...Next

Cette structure est moins utilisée que la précédente. Elle sert surtout à exécuter des instructions portant sur un tableau. Exemple :

```
Dim Semaine(1 to 7) As String
Dim Jour As Variant
Semaine(1) = "Lundi"
'Etc...
For Each Jour In Semaine()
Combo1.AddItem Jour
Next
```

Interprétation : Pour chaque Jour de la Semaine, on ajoute à la liste combinée, la valeur de la variable Jour jusqu'à ce que l'indice de la semaine soit égale à 7.

3. Les structures Do...Loop et While...Wend

Avec ces boucles, le nombre de fois où sont répétées les instructions est indéfini. Les deux exemples qui suivent sont équivalents :

```
i = 1
Do
T(i) = i
i = i + 1
Loop Until i > 10
```

```
i = 1
While i < 10
T(i) = i
i = i + 1
Wend
```

Interprétation : La variable i est initialisée à 1 au départ. Les instructions qui suivent sont exécutées jusqu'à ce que i soit supérieure à 10. A chaque passage, on affecte à la variable T d'indice i la valeur de i.

Il existe d'autres variantes de ces boucles et qui sont chacune équivalentes :

```
Do
Traitement
Loop While Condition
```

```
Do Until Condition
Traitement
Loop
```

```
Do While Condition
Traitement
Loop
```

III. Résumé

Structures conditionnelles		
Structure	**Syntaxe**	**Exemple**

If... Then	Exécution d'un traitement suite à l'évaluation booléenne d'une condition
	
	If condition **Then**
	[traitement]]
	End If
		If a = b Then
		MsgBox "Pareil"
		End If

If... Then... Else	Évaluations imbriquées de conditions booléennes
	
	If condition1 **Then**
	[traitement]]
	Elself condition2 **Then**
	[traitement]
	Else
	[traitement]]
	End If
		If a = 1 Then
		MsgBox "Un"
		ElseIf a >= 2
		MsgBox "Plusieurs"
		Else
		MsgBox "Inconnu"
		End If

Select Case	Exécution d'un traitement suite à l'évaluation d'une condition à valeur multiple **Select Case** variable **Case** condition1 [traitement] **Case** condition2 [traitement] **Case Else** [traitement] **End Select**	Select Case a Case 1 MsgBox "Un" Case 2 To 10 MsgBox "Plusieurs" Case Else MsgBox "Inconnu" End Select

Structures de boucle		
Boucle	**Syntaxe**	**Exemple**
For... Next	Exécution d'un traitement un nombre fini de fois **For** counter = start **To** end [**Step** step] [traitement] [**Exit For**] [traitement] **Next** [counter]	For i = 1 To 5 Step 1 MsgBox i Next i
While... Wend	Exécution d'un traitement jusqu'à invalidation d'une condition booléenne située avant le traitement **While** condition [traitement] **Wend**	While (i < 10) MsgBox i i = i + 1 Wend
Do... Loop	Exécution d'un traitement jusqu'à invalidation d'une condition booléenne située avant ou après le traitement **Do** [{**While**|**Until**} condition] [traitement] [**Exit Do**] [traitement] **Loop** **Do** [traitement] [**Exit Do**] [traitement] **Loop**[{**While**|**Until**} condition]	Do While (i < 10) MsgBox i i = i + 1 Loop Do MsgBox i i = i + 1 Loop Until (i = 10)

Chapitre IV

Les procédures et les fonctions

Objectif

- ✓ *Connaître la notion des procédures*
- ✓ *Connaître la notion des fonctions*
- ✓ *Connaître la notion des modules*
- ✓ *Connaître les fonctions prédéfinies*

Eléments de contenu

- ✓ *Les procédures*
- ✓ *Les fonctions*
- ✓ *Module de code*
- ✓ Quelques Fonctions prédéfinies

Chapitre IV

Les procédures et les fonctions

I. Les procédures

Une application est constituée essentiellement de l'*interface utilisateur*, formée elle même de plusieurs *contrôles*, et de *procédures* qui génère des actions. A présent, analysons la *structure d'une procédure*.

La syntaxe d'écriture d'une procédure est la suivante :

```
[Public / Private] [Static] Sub Nom_proc (arguments)
   [Déclarations]
   [Instructions]
   [exit Sub]
   [Instructions]
End Sub
```

Le mot **Sub** peut ou non être précédé des options suivantes :

1. Les mots ***Public*** ou ***Private*** définissent les limites de la procédure.
   - Avec ***Public***, la procédure peut être appelée depuis n'importe quelle instruction de l'application.
   - Avec ***Private,*** la procédure ne peut être appelée qu'à l'intérieur de la même feuille ou du même module.
2. Vous devez déclarer en début de procédures vos *variables* et *constantes* si vous ne l'avez pas fait dans la partie Général de la liste déroulante *Objet* qui se trouve en haut, à gauche de l'éditeur de code. Remarquez que si vous déclarez vos variables à l'intérieur de la procédure, sa portée sera limitée qu'à cette procédure;
3. ***Static*** signifie que toutes les variables locales déclarées dans la procédure doivent conserver leur valeur entre 2 appels.
4. ***Exit Sub*** permet de quitter la procédure avant la fin normale de celle-ci;
5. ***End Sub*** ferme la procédure.

Pour cela, prenons un simple exemple :

```
Private Sub Form_Load()
Dim DateNaissance
Dim Message, Titre As String
Message = "Date de naissance ?"
Titre = "Entrez votre date de naissance : "
DateNaissance = InputBox(Message, Titre)
If DateNaissance <> "" Then
DateNaissance = Format(DateNaissance, "Long Date")
MsgBox DateNaissance, vbOKOnly + vbInformation, "Vous êtes
né(e) le"
End
Else
While DateNaissance = ""
MsgBox "Entrez une date", vbOKOnly + vbExclamation,
"Attention!"
DateNaissance = InputBox(Message, Titre)
Wend
DateNaissance = Format(DateNaissance, "Long Date")
MsgBox DateNaissance, vbOKOnly + vbInformation, "Vous êtes
né(e) le"
End
End If
End Sub
```

A présent, passons à une analyse simplifiée de cette procédure :

- La 1ère ligne contient des infos sur le nom de la feuille principale (Form1) du projet, du type d'événement qui permet de lancer cette procédure. Ici L'événement en question est Load, ce qui veut dire que cette procédure sera exécuté au chargement de la feuille "Form1".
- dans les 2èmes et 3èmes lignes, sont définis des variables limitées à cette procédure uniquement.
- Les lignes suivantes sont formées d'instructions servant à définir le comportement de la feuille dès son chargement. (Nous verrons de plus près ces instructions dans les cours suivants.)
- La dernière ligne End Sub clôt la procédure.

Remarque :

L'appel de la procédure peut être fait de deux manières différentes : on précède le nom de la procédure par le mot clé CALL ou bien le nom directement. En précédant le nom de la procédure du mot clé call, on est obligé de mettre des parenthèses pour les paramètres.

Exemple :

Ecrire une procédure DivReste qui prend deux entiers et retourne la division et le reste.

```
Private Sub divReste(x1 as integer, X2 as integer, D as integer,
R as integer )
D = X1 mod X2
R = X1 div X2
End Sub
'Appel de la procédure
Call divReste (9,2,a,b)
'Ou encore
divReste 9,2,a,b
```

II. Les fonctions

La syntaxe d'une fonction est la suivante :

```
[Public / Private] [Static] Function Nom (arguments) [As type]
   [Déclarations]
   [Instructions]
End Function
```

Contrairement à une procédure, une Fonction renvoie obligatoirement une valeur. Pour qu'une valeur soit renvoyée, vous devez l'affecter au nom de la fonction.

```
Public Function calcul_impot(salaire as curency)as Currency
if Salaire > 3000 then
Calcul_impot = salaire * 0.33
Else
 Calcul_impot = salaire * 0.1
End if
End Function
'Appel de la foction
label1.caption = calcul_impot(x)
```

III. Module de code

Dans Visual Basic le code est stocké dans des modules. Ils existent trois types de modules : module de feuille, module standard et module de classe.

Le module de feuille contient les feuilles de l'application vb, si l'application est simple et constitué d'une seule feuille alors on peut déclarer les fonctions dans le module de feuille en revanche si on a plusieurs feuilles avec du code en commun, alors il est préférable de les déclarer ailleurs.

Le module standard (dont le nom de fichier possède l'extension .bas) est un conteneur de fonctions, de procédures et de déclarations, il est généralement utilisé par d'autres modules internes à l'application.

Appel de procédure dans d'autres modules

Procédure dans les feuilles

Tous les appels qui ne sont pas dans le module de feuille doivent designer le nom dun module de feuille

Exemple

Form1.nomProcedure (args)

Procédures dans les modules standards

- Si le nom de procédure est unique alors il n'est pas nécessaire d'insérer le nom du module dans l'appel
- S'il y a deux modules ou plus contenant une procédure portant le même nom alors lors de l'appel, il faut qualifier le nom de procédure par le nom du module.

NomModule.NomProcédure ().

IV. Quelques Fonctions prédéfinies

1. Fonction de conversion

Fonction	Type Renvoyé
CBool	Boolean
CByte	Byte
CCur	Currency
CDate	Date
CDbl	Double
CDec	Decimal
CInt	Integer
Csng	Single
Cstr	String
Cvar	Variant
ASC	integer

2. Fonctions de chaîne de caractères

- ***Trim, LTrim, RTrim :***

Renvoie une valeur de type Varient (String) contenant une copie d'une chaîne de caractères en supprimant les espaces de gauche (LTrim), les espaces de droite (RTrim) ou les deux (Trim).

Syntaxe

```
LTrim(string)
RTrim(string)
Trim(string)
```

Exemples

```
Ltrim ("     Test"): donne la  chaine "Test"
Rtrim("Test     "): donne la  chaine "Test"
```

- ***UCase, LCase***

Renvoie une valeur de type Variant(String) contenant la chaîne indiquée, convertie en majuscules, minuscules.

Syntaxe

```
UCase (string)
LCase (string)
```

- ***InStr***

Renvoie une valeur de type Variant (Long) indiquant la position de la première occurrence d'une chaîne à l'intérieur d'une autre chaîne.

Syntaxe

```
Instr ([PositionDébut ,] chaîne 1, chaîne 2)
```

Exemple

```
Instr (1, « Bonjour », « jour ») : retourne 4.
```

- ***Mid***

Renvoie une valeur de type Variant (String) contenant un nombre indiqué de caractères extraits d'une chaîne de caractères

Syntaxe

```
Mid (string, start  [,lengh])
```

Exemple

```
Mid("bonjour",4,4) : retourne "jour"
```

- ***Len***

Renvoie une valeur de type long contenant l nombre de caractères d'une chaîne de caractères ou le nombre d'octets requis pour stocker une variable.

Syntaxe

```
Len (string | VarName)
```

Chapitre V

Les fichiers En VB

Objectif

- ✓ *Connaître la notion des Fichiers*
- ✓ *Connaître les fichiers en VB*
- ✓ *Se familiariser avec les fichiers séquentiels et à accès direct*

Eléments de contenu

- ✓ *Rappel*
- ✓ *Les fichiers en VB*
- ✓ *Les fichiers séquentiels.*
- ✓ *Les fichiers à accès direct.*
- ✓ *Les fichiers binaires.*

Chapitre V

Les Fichiers En VB

I. Rappel algorithmique

1. Définition

Un fichier est une suite d'octets enregistrés sous un même nom sur un support de données (Disque, Disquette, CD-ROM…).

2. Accès

Il existe traditionnellement deux modes d'accès aux fichiers :

- l'accès séquentiel : traitement séquentiel des informations.
- L'accès direct : se placer immédiatement sur l'information souhaitée.

II. Les fichiers en VB

1. Syntaxe

```
Open Nom_Du_Fichier For mode [Acess accès]  [lock] As numéro
[Len = longueur]
```

- **Open :** Mot clé qui indique l'action d'ouverture d'un fichier.
- Nom_Du_Fichier est un String qui contient le nom du fichier.
- **For :** mot clé qui précède le mode d'accès.
- **Mode :** Mot clé qui représente le mode d'accès dans le fichier. Il peut être l'un des modes suivants **: Append, Binary, Input, Output ou Random.**
- **Access :** mot clé qui précède les actions permises dans le fichier. Les actions peuvent être : Read, Write ou Read –Write
- **Lock :** Mot clé qui indique l'accés des autres usagers dans le même fichier. Ceci peut être: **Shared , Lock Read, Lock Write ou Lock Read Write.**
- **As:** Mot clé qui précède le numéro de fichier ouvert.
- **Len :** Mot clé qui précède la longueur d'un enregistrement.

2. Manipulation des fichiers :

Les instructions qui servent au traitement des fichiers sont les suivantes : **Print, Line Input, Read, Write, Get et Put.** Les modes qui demandent un peu plus d'attention sont les modes binary et random. Le mode random demande que la longueur d'enregistrement soit connue. Dans ce mode on peut en même temps écrire et lire un fichier. Les instructions get et put seront utilisées pour les deux actions.

'Déclaration de type

```
Type personne
     NCIN as integer
     Nom as string
     Prénom as string
End type
```

```
Dim lenPer as integer

Dim FileNum as integer

Dim emp as Personne

Din pos as integer

LenPer = len(emp)

FileNum = FreeFile

Open « c:\liostemp.dat » For Random As FileNum Len = LenPer

Pos = 1

Get #FileNum, Pos, Emp

Emp.Nom = "ALI"

Put #FileNum, Pos, Emp
```

III. Les fichiers à accès séquentiel

Il existe trois manières d'ouvrir un fichier texte :

1. La 1ère manière est l'ouverture d'un fichier uniquement en lecture à l'aide de l'instruction *Input*.
2. La 2ème manière est l'ouverture d'un fichier uniquement en écriture à l'aide de l'instruction *Output*.
3. La dernière manière est l'ouverture d'un fichier en écriture aussi mais à la seule différence de la précédente manière, les nouvelles données sont entrées à la suite de l'enregistrement précédent. Pour cela, on utilise l'instruction *Append*.

Pour comprendre son fonctionnement, prenons les exemples suivants.

Exemple 1:

```
Private Sub ouvrir_Click()
On Error Resume Next
Open "C:\Windows\Bureau\note.TXT" For Input As #1
retour = Chr$(13) + Chr$(10)
Line Input #1, texte
tout = texte
If Len(tout) <> 0 Then
While Not EOF(1)
Line Input #1, texte
tout = tout + retour + texte
Wend
End If
Close #1
End Sub
```

Explication :

- L'instruction *On error Resume Next* permet d'ignorer toute erreur que peut entraîner cette procédure.
- L'instruction *Open "C:\Windows\bureau\note.TXT" For Input As #1* permet d'ouvrir le fichier "note" situé sur le bureau. Lorsque l'application est exécutée pour la première fois, une erreur 53 devrait apparaître car le fichier "note" n'existe pas encore. Mais grâce à l'instruction *On Error Resume Next*, cette erreur sera tout simplement ignorée.

- L'instruction *retour = Chr$(13) + Chr$(10)* permet d'aller à la ligne suivante. Le code ASCII 13 correspond au "retour chariot", c'est-à-dire de revenir à gauche du texte et 10 au "passage à la ligne", c'est-à-dire d'aller à la ligne suivante. Vous pouvez aussi utiliser le code *vbCrLf* à la place et c'est plus court et plus facile à retenir. Vous aurez ainsi: "tout = tout + vbCrlf + texte"
- L'instruction *Line Input* permet de placer dans la variable "texte" le contenu de la première ligne du fichier "note.TXT".
- Ensuite, le contenu de la variable "texte" est à son tour, affecté à la variable "tout".
- Le test avec l'instruction *If* permet de vérifier que le fichier "note.TXT" n'est pas vide à l'aide de la fonction *Len* qui compte le nombre de lettres ou tout autre symbole contenu dans la variable "tout". Si le nombre de lettres est différent de zéro alors, les instructions à l'intérieur du test sont exécutées.
- La boucle avec *While* permet de parcourir tout le fichier à la recherche d'éventuelles autres lignes que la première. La fonction *EOF* permet de savoir si l'on arrive à la fin du fichier ouvert. Cette boucle est donc exécutée jusqu'à ce que tout le contenu du fichier soit placé dans la variable tout. Les chaînes de caractères affectées à la variable "tout" sont concaténées (concaténées = ajoutées) aux précédentes contenues dans la variable. La variable "retour" provoque un retour à la ligne à chaque fois que cela est nécessaire.
- Lorsque tout le contenu du fichier sera affecté à la variable "tout", cette dernière sera à son tour affectée à la propriété *Text* du contrôle *TextBox*. Ce qui provoquera l'affichage du contenu du fichier dans le contrôle TextBox.
- L'instruction *Close* ferme le fichier ouvert en lecture.

Exemple 2:

```
Private Sub Sauvegarder_Click()
Open "C:\Windows\Bureau\note.TXT" For Output As #1
Print #1, Text1.Text
Close #1
End Sub
```

Explication :

- L'instruction *Open "C:\Windows\Bureau\note.TXT" For Output As #1* ouvre le fichier "note.TXT" en écriture.
- Le contenu de la zone de texte "Text1" est ensuite copié dans le fichier à l'aide de l'instruction *Print*.
- Enfin, le fichier est fermé à l'aide de l'instruction *Close*.

IV. Les fichiers à accès direct

Les fichiers à accès direct contiennent des données contenues dans plusieurs enregistrements fixes. Pour déclarer un enregistrement, on utilise l'instruction *Type* pour créer un nouvel type de variable. Ensuite, On déclare une variable de ce type. Pour ouvrir un fichier à accès direct, on utilise l'instruction *Open* en indiquant la longueur de chaque enregistrement. Ainsi, un fichier à accès direct s'ouvre de la manière suivante : *Open "C:\Windows\Bureau\fichier.adr" For Random As #1 Len = Len(Adr)* où adr est le nom d'une variable de type prédéfini. La lecture d'un enregistrement s'effectue de la manière suivante: *Get #1, Numéro, adr*. L'écriture dans un fichier à accès direct s'effectue de la manière suivante: *Put #1, Numéro, adr*.

Chapitre VI

Gestion des Bases de Données

Objectif

- ✓ *Connaître les objets nécessaires pour manipuler les données d'une base*
- ✓ *Connaître les traitements assurant la connexion et la déconnexion d'une source de données*
- ✓ *Connaître les objets nécessaires pour la récupération des données d'une base*
- ✓ *Connaître les scénarios de récupération utilisant les objets déjà décrits.*
- ✓ *Connaître les scénarios de la mise à jour des données d'une base.*

Eléments de contenu

- ✓ *La hiérarchie des objets du modèle ADO*
- ✓ *La connexion à une base de données*
- ✓ *Déconnexion d'une source de données*
- ✓ *La récupération de données*
  - o *Propriétés de l'objet Recordset*
  - o *Méthodes de l'objet Recordset*
  - o *Les différents scénarios de récupération de données*
- ✓ *Les scénarios de la mise à jour des données*

Chapitre VI

Gestion des Bases de Données

VB s'interface avec pratiquement tous les types de bases de données (BD), notamment à travers son support ODBC, tout en optimisant les accès à MS Access ou SQL Serveur. Afin de faciliter l'accès aux données, VB fournit un ensemble d'objets permettant de gérer la structure d'une BD ainsi que l'accès aux données qui y sont stockées. Ces objets sont appelés DAO (Database Access Object) et s'appuient sur le moteur Jet BD de Microsoft.

I. La hiérarchie des objets du modèle ADO

Les objets ADO vous offrent un accès simple et rapide aux données de tous types. Le modèle objet ADO comporte trois principaux composants : l'objet connection, l'objet Command et l'objet Recordset.

- **Objet Connection :**

  Dans le modèle objet ADO, l'objet Connection est celui qui occupe le niveau le plus élevé. Il est utilisé pour établir une connexion entre votre application et une source de données externe, telle que Microsoft SQL Server.

- **Objet Command :**

  L'objet Command sert à créer des requêtes, y compris des paramètres spécifiques à l'utilisateur, pour accéder aux enregistrements d'une source de données. En règle générale, ces enregistrements sont renvoyés au sein d'un objet Recordset.

- **Objet Recordset :**

  L'objet RecordSet permet d'accéder aux enregistrements renvoyés par une requête SQL. Avec cet objet, vous pouvez naviguer dans les enregistrements renvoyés, ajouter des enregistrements ou en supprimer.

- **Objet ADO :**

ADO prend en charge trois collections, Errors, Parameters et Fiels. Si elles peuvent apporter des fonctionnalités supplémentaires à une application, ces collections ne sont pas indispensables à la création de solution ADO.

- ✓ *Collection Errors :*

  La collection Errors permet de renvoyer des informations détaillées sur des erreurs survenues pendant l'exécution, ainsi que d'autres messages émis par une source de données.

- ✓ *Collections Parameters :*

  La collection Parameters permet de passer certaines données spécifiques à une requête paramétrée ou à des procédures stockées d'une base de données SQL server.

- ✓ *Collection Fields :*

  La collection Fields permet d'accéder à certains champs d'un objet Recordset..

II. La connexion à une base de données

L'objet Connection établit une connexion avec une source de données. Il permet à votre application de passer des information provenant du client (nom d'utilisateur et mot de passe par exemple) à la base de données pour validation.

Pour utiliser ADO pour l'établissement d'une connexion à la base données

1. Créez une référence à la bibliothèque d'objets ADO

Avant de pouvoir utiliser ADO dans votre application Visual Basic, vous devez créer une référence à la bibliothèque **Microsoft ActiveX Data Objets 2.0**

Pour créer une référence à la bibliothèque d'objets ADO

- ✓ Dans le menu Projet, cliquer sur Références
- ✓ Sélectionnez Microsoft ActiveX Data Objets 2.0 Library, puis sur OK.

2. Déclarez un objet connection

Une fois que vous avez créé une référence à la bibliothèque d'objets ADO, vous pouvez déclarer un objet Connection dans votre application. En vous servant de l'objet Connection, vous pouvez alors créer l'objet Command et on crée une instance :

Dim cn as connection

Set cn = New Connection

3. Spécifiez un fournisseur de données OLE DB

Une fois que vous avez créé une instance d'un objet Connection, vous devez spécifier un fournisseur de source de données OLE DB ; Pour ce faire, définissez la propriété Provider.

L'exemple de code suivant spécifie le fournisseur de données Microsoft Jet (Access)

Cn.Provider= « Microsoft.jet.OLEDB.4.0 »

Le tableau suivant présente quelques-uns des fournisseurs OLE DB actuellement disponibles :

Fournisseur De données OLE DB	Description
SQLOLEDB	Pour Microsoft SQL Server
MSDASQL	Fournisseur OLE DB pour ODBC
MSDAORA	Fournisseur OLE DB pour Oracle

4. Passer les informations sur la connexion.

La dernière étape, avant l'établissement d'une connexion avec une source de données, consiste à spécifier les informations de connexion. Pour se faire, définissez les propriétés de la chaîne ConnectionString de l'objet Connection. Ces propriétés sont propres à un fournisseur, auquel elles sont passées directement, et ne sont pas traités par ADO.

Une fois ces opérations accomplies, vous pouvez établir la connexion au moyen de la méthode Open.

- ***Exemple 1 Connexion à une base de données Microsoft SQL Server***

Une connexion à une base de données Microsoft SQL Server appelée IG sur le serveur MSDB par le biais du fournisseur de données SQL Server OLE DB :

```
With cn
        .provider = « SQLOLEDB »
        .ConnectionString = « User ID = sa ; Password= ; » & _
        « Data Source = msdb ; »&_
        « Initial Catalog = IG »
End with
```

- ***Exemple 2 Connexion à une base de données Microsoft Access***

```
Set db = new connection
Db.open "PROVIDER = Microsoft.Jet.OLEDB.4.0; data source = c:\IG31.mdb;"
```

III. Déconnexion d'une source de données

Quand vous n'avez plus besoin d'une connexion à une source de données, utilisez la méthode Close pour supprimer cette connexion. L'exemple de code suivant ferme une connexion active à une source de données.

```
Cn.close
Set cn = nothing
```

IV. La récupération de données

La récupération des données à partir d'une source de données se fait par le biais de l'objet Recordset déjà définie. L'objet Recordset contient les colonnes et les lignes retournées à l'issue d'une opération spécifique (requête SQL). En vous servant de l'objet Recordset, on peut naviguer ou bien mettre à jour des enregistrements.

1. Propriétés de l'objet Recordset

Le tableau suivant décrit les propriétés de l'objet Recordset les plus utilisées pour créer un jeu d'enregistrements.

Propriété	Description
ActiveCommand	Renvoie la commande active pour le jeu d'enregistrements.
ActiveConnection	Renvoie la connexion active pour le jeu d'enregistrement.
CursorLocation	Renvoie ou définit l'emplacement ou est géré le curseur. La valeur par défaut est adUseServer
CursorType	Renvoie ou définit le type du curseur. La valeur par défaut est adOpenStatic.
LockType	Renvoie ou définit le type du verrouillage d'enregistrement. La valeur par défaut est adLockReadOnly
MaxRecords	Renvoie ou définit le nombre maximal d'enregistrements à renvoyer.
RecordCount	Renvoie le nombre d'enregistrements contenus dans le jeu d'enregistrements.
State	Renvoie l'état en cours du jeu d'enregistrements.

a) La propriété CursorLocation

Définit ou renvoie une valeur de type Long pouvant être l'une des valeurs suivantes :

Constante	Description
adUseClient	Le curseur coté client crée un recordset déconnecté qui contient tous les enregistrements satisfaisant à la requête et qui sont stockées dans l'ordinateur du client. Ceci permet une gestion statique et instantanée des donnés.
adUseServer	Valeur utilisée par défaut. Avec un curseur côté serveur, le client n'a plus besoin de placer dans un cache de gros volumes de données ni de gérer des informations concernant la position du curseur ; c'est le serveur qui se charge de toutes ces fonctionnalités.

Remarque :

Avec un curseur côté client, la totalité des enregistrements de l'objet recordset seront copiés dans l'ordinateur du client, ce qui peut surcharger le réseau.

b) La propriété CursorType

Permet de définir la façon avec laquelle vous allez récupérer l'information lorsque plusieurs personnes accèdent simultanément aux mêmes données. Il existe quatre types de curseur définis dans ADO :

- **Curseur à défilement en avant (adOpenForwadOnly)** : il s'agit du type de curseur par défaut, il permet de visualiser les ajouts, les modifications et les suppressions effectués par d'autres utilisateurs. Le parcours de données se fait que vers l'avant, mais vous ne pouvez pas définir un signet sur l'enregistrement courant. Si le type du curseur permet les signets, vous pouvez enregistrer un signet vers l'enregistrement courant afin d'y revenir ultérieurement. Ce type de curseur permet d'obtenir de meilleures performances lorsque vous devez parcourir une seule fois un objet Recorset.
- **Curseur statique (adOpenStatic)** : fournit une copie statique d'un jeu d'enregistrements que vous pouvez utiliser pour rechercher des données ou créer des rapports. Lorsque le client a reçu tous les enregistrements, le curseur peut défiler à travers les données sans qu'il y ait besoin d'autres interactions avec le serveur. (Mais il ne permet pas de voir les modifications faites sur la base de données pendant que le recordset est ouvert).
  Il permet l'utilisation de signets et rend donc possible tout type de déplacement dans l'objet Recordset. Les ajouts, modifications et suppressions effectués par d'autres utilisateurs ne sont pas visibles. Ce type de curseur est le seul autorisé si vous ouvrez un objet Recordset côté client.
- **Curseur dynamique (adOpenDynamic)** : permet de visualiser les ajouts, les modifications et les suppressions effectués par d'autres utilisateurs. Il permet également tout type de déplacement ne nécessitant pas l'utilisation de signets dans l'objet Recordset. Ce type de curseur permet l'utilisation de signets si le fournisseur est en mesure de les prendre en charge.
- **Curseur à jeu de clés (adOpenKeyset)** : son fonctionnement est identique à celui d'un curseur dynamique mais il ne permet pas de visualiser les enregistrements ajoutés par d'autres utilisateurs ou d'accéder aux enregistrements qu'ils ont supprimés. Les modifications effectuées sur les données par les autres utilisateurs sont néanmoins visibles. Ce type de curseur prend toujours en charge les signets et il permet donc tout type de déplacement dans l'objet Recordset. Avec un curseur Keyset seules les clés des enregistrements sont envoyées ce qui accélère les performances. Les données associées aux clés ne sont lues que lorsque vous parcourez le Recordset sauf pour la première requête qui retourne les clés et les données.

Remarques :

- ✓ Choisissez le type de curseur en définissant la propriété CursorType avant d'ouvrir l'objet Recordset.
- ✓ Access ne reconnaît pas les curseurs dynamiques, si vous choisissez comme même cette valeur, le type de curseur sera automatiquement converti en type Keyset.
- ✓ Lorsque cous définissez la propriété CursorLocation sur adUseClient, le moteur du curseur client ne prend en charge que les types de curseur « statique », quelle que soit la valeur de la propriété CursorType.

c) La propriété LockType

Indique le type de verrouillage des enregistrements lors des modifications.

Constante	Description
adLockReadOnly	Valeur utilisée par défaut. Lecture seule, vous ne pouvez pas modifier les données.
adLockPessimistic	Verrouillage pessimiste, enregistrement par enregistrement, la base de données verrouille les enregistrements dés le début de la modification, les enregistrements sont verrouillés après achèvement de toutes les modifications. Deux utilisateurs différents ne peuvent pas accéder simultanément aux mêmes enregistrements.
adLockOptimistic	Verrouillage optimiste, enregistrement par enregistrement ; le fournisseur utilise le verrouillage optimiste et ne verrouille les enregistrements que lorsque vous appeler la méthode **Update**. La base de données ne verrouille les enregistrements en cours de modification qu'au moment de la validation des modifications. Deux utilisateurs différents peuvent accéder simultanément au même enregistrement, et la base de données doit être en mesure d'harmoniser les conflits.
adLockBatchOptimistic	Mise à jour par lots optimiste.

Remarques :

- ✓ Vous définissez la propriété LockType avant d'ouvrir un Recordset pour indiquer le type de verrouillage que le fournisseur doit appliquer à l'ouverture.
- ✓ Certains fournisseurs ne prennent pas en charge tous les types de verrouillage.
- ✓ La valeur adLockPessimistic n'est pas acceptée si la propriété CursorLocation est adUseClient.

- ✓ Si vous choisissez une valeur non prise en charge, aucun message d'erreur n'est généré, elle sera automatiquement remplacée

2. Méthodes de l'objet Recordset

Le tableau suivant décrit les méthodes de l'objet Recordset les plus utilisées pour créer un jeu d'enregistrement

Méthode	Description
Open	Ecécute une commande Sql et ouvre un curseur.
Close	Ferme le jeu d'enregistrements.
Requery	Réexécute la commande SQL et recrée le jeu d'enregistrement

3. Les différents scénarios de récupération de données

Un objet recordset peut être créé de différentes manières :

- ➢ Par le biais de la méthode Execute d'un objet Command

```
Sub connect_click()

     Dim cd as command
     Dim cn as connection
     Dim rs as Recordset

     Set cn = New connection
     Set cd = New command
     Set rs = New Recordset

     With cn
               .Provider = "SQLOLEDB"
               .ConnectionString = "User ID= sa;"&_
                                   "Initial catalog = IG"
               .open
     End with

     With cd
               .ActiveConnection = cn
               .CommandText = "Select * from etudiant"
     End with

     Set rs = cd.execute
End sub
```

- Par le biais de la méthode Execute de l'objet Connection

```
Sub connect_click()


     Dim cn as connection
     Dim rs as Recordset

     Set cn = New connection
     Set rs = New Recordset

     With cn
               .Provider = "SQLOLEDB"
               .ConnectionString = "User ID= sa;"&_
                                   "Initial catalog = IG"
               .open
     End with


     Set rs = cn.execute ("Select * from etudiant")
End sub
```

- Par le biais de la méthode Open de l'objet Recordset

```
Sub connect_click()


     Dim cn as connection
     Dim rs as Recordset

     Set cn = New connection
     Set rs = New Recordset

     With cn
               .Provider = "SQLOLEDB"
               .ConnectionString = "User ID= sa;"&_
                                   "Initial catalog = IG"
               .open
     End with


     rs.open ("Select * from etudiant"),cn
End sub
```

- Création d'un jeu d'enregistrements autonome

```
Sub connect_click()

     Dim rs as Recordset
     Set rs = New Recordset

     rs.open ("Select * from etudiant"),_
     "Provider = SQLOLEDB; User ID= sa;"&_
     "Initial catalog = IG"

End sub
```

V. Les scénarios de la mise à jour des données

Lorsque votre application a besoin de mettre à jour des données dans une source de données externe, deux techniques sont offertes :

- Utilisation de la méthode Execute

```
Sub connect_click()
          Dim sSQL as string
          Dim cn as connection
          Set cn = New connection
          With cn
                    .Provider = "SQLOLEDB"
                    .ConnectionString = "User ID= sa;"&_
                                        "Initial catalog = IG"
                    .open
          End with
sSQL="Insert into etudiant (nce,nom,pernom) values (40,'x','y')"
cn.execute sSQL
End sub
```

```
Sub connect_click()
          Dim sSQL as string
          Dim cn as connection
          Set cn = New connection
          With cn
                    .Provider = "SQLOLEDB"
                    .ConnectionString = "User ID= sa;"&_
                                        "Initial catalog = IG"
                    .open
          End with
sSQL = "UPDATE etudiant set nom ='a' where NCE = 40"
cn.execute sSQL
End sub
```

```
Sub connect_click()
          Dim sSQL as string
          Dim cn as connection
          Set cn = New connection
          With cn
                    .Provider = “SQLOLEDB”
                    .ConnectionString = “User ID= sa;”&_
                                        “Initial catalog = IG”
                    .open
          End with
sSQL = “DELETE FROM etudiant where NCE = 40”
cn.execute sSQL
End sub
```

- Utilisation des méthodes de l’objet recordset

Passons à présent, aux principales méthodes utilisées pour gérer la récupération et la mise à jour des données via Recordset

AddNew	Crée un nouvel enregistrement.
Delete	Supprime un enregistrement donné.
Movefirst, MoveLast, MovePrevious, MoveNext	Elles servent respectivement à revenir au premier enregistrement, à aller au dernier enregistrement, à revenir à l'enregistrement précédent et à aller à l'enregistrement suivant.
Refresh	Rafraîchit la base de donnée.

```
     Sub connect_click()
          Dim cn as connection
          Dim rs as Recordset
          Set cn = New connection
          Set rs = New Recordset
          With cn
                    .Provider = “SQLOLEDB”
                    .ConnectionString = “User ID= sa;”&_
                                        “Initial catalog = IG”
                    .open
          End with
          rs.open (“Select * from etudiant”),cn
          rs.addnew
     End sub
```

Annexe Chapitre VI

Le langage de requête structuré SQL

Objectif

- ✓ *Se familiariser avec l'environnement SQL.*

Eléments de contenu

- ✓ *Recherche de données*
- ✓ *Manipulation des données*

Chapitre V

Le langage de requête structuré SQL

ATTENTION:

Veuillez noter que le langage SQL peut varier d'un constructeur à l'autre. Dans ce cas, il vous faut consulter la documentation qui lui est propre. Le langage SQL abordé ici, est le langage standard et peut ne pas être compatible avec votre base de données.

I. Recherche de données

1. La syntaxe

L'instruction utilisée pour effectuer une recherche dans une base de données est *SELECT*. Elle sert à interroger une base et de retourner si elles existent, les données que vous recherchez.

La syntaxe de l'instruction *SELECT* est la suivante:

```
SELECT [ *|ALL|DISTINCT ] [ TOP X [PERCENT] ] Colonne1,Colonne2
FROM Tables
WHERE (Critère de recherche)|(Critère de jointure)
AND|OR [ (Critère de recherche) ]
GROUP BY [ ALL ] Colonne1, Colonne2
HAVING (Critère de recherche)
ORDER BY Colonnes [ ASC|DESC ]
```

- ✓ L'instruction *SELECT* permet de sélectionner les données à partir d'une ou plusieurs colonnes d'une table.
  - ➢ L'argument * permet de faire une recherche dans toutes les colonnes de la table donnée.

- L'argument *ALL* sert à indiquer de retourner toutes les valeurs recherchées même ceux qui sont en double. Par exemple, dans une base de données contenant une liste de tous les clients d'une société, il est parfaitement possible qu'il y ait 2 clients ayant le même nom de famille ou le même prénom. Eh bien, cet argument indique de retourner les données de ces 2 clients. Cet argument est celui par défaut. Il est facultatif.

- L'argument *DISTINCT* est le contraire de *ALL*. Il sert à indiquer de ne retourner que les données uniques. Il est facultatif.
- L'argument *TOP* permet de préciser le nombre d'enregistrements que vous souhaitez recevoir en réponse à votre requête à partir du premier enregistrement. *PERCENT* indique le pourcentage X d'enregistrements que vous souhaitez retournez. Il est facultatif.

- Les mots *Colonne1,Colonne2* Indiquent dans quel(les) colonne(s) de la table ou des tables vous souhaitez effectuez votre recherche. N'oubliez pas la virgule si vous effectuez votre recherche dans plusieurs colonnes.

✓ La clause *FROM* sert à indiquer à partir de quel (les) table(s) les données doivent être extraites. Cette clause est indispensable pour toute requête

✓ La clause *WHERE* permet d'insérer des critères de recherche dans votre requête. Il est facultatif.

- Le mot *Critères de recherche* représente les contraintes de sélection. Par exemple: "WHERE nom= 'x'" ou bien "WHERE NCE = 40".
- Le mot *Critère de jointure* permet d'effectuer une recherche dans une colonne présente dans 2 tables différentes.

✓ Les opérateurs *AND* et *OR* permettent d'insérer un autre critère de recherche. Ils sont facultatifs.

✓ La clause *GROUP BY* permet de regrouper des données. Il est facultatif.

- ✓ La clause *HAVING* permet d'insérer un critère de recherche pour les données regroupées avec la clause *GROUP BY*. Il est facultatif.

- ✓ Enfin, la clause *ORDER BY* permet de trier les données par colonnes. Il est facultatif.

    - Le mot *Colonnes* représente le nom des colonnes dans lesquelles sont triées les données.
    - L'argument *ASC*Permet de trier les données par ordre croissant. C'est l'ordre par défaut. Il est facultatif.
    - L'argument *DESC* permet de trier les données par ordre décroissant. Il est facultatif.

Voici un exemple simple sur la façon d'utiliser les requêtes avec SQL en s'appuyant sur le contrôle *Data* qui vous permet de visualiser les données:

```
Data.RecordSource="SELECT * FROM etudiant WHERE NCE<'15' AND
NCE>'3' ORDER BY NCE"
```

2. Les opérateurs

Vous pouvez aussi faire appel à des opérateurs pour spécifier des conditions dans une instruction SQL ou servir de conjonction à plusieurs conditions. Il existe 5 types d'opérateur en tout:

- Les opérateurs de comparaisons
- Les opérateurs conjonctifs permettent d'effectuer plusieurs conditions à la fois (AND et OR).
- Les opérateurs logiques servent à effectuer des comparaisons (IN, NOT IN, BETWEEN, NOT BETWEEN, LIKE, NOT LIKE, IS NULL, IS NOT NULL, EXISTS, NOT EXISTS, ALL et ANY).
- Les opérateurs arithmétiques

3. Les fonctions mathématiques

Il est aussi possible avec le langage SQL d'effectuer des calculs directement à partir des données. Cela se fait à l'aide des fonctions mathématiques suivantes:

Fonction	Utilisation
COUNT()	Comptabilise le nombre d'enregistrements retourné
SUM()	Calcule la somme des valeurs retournées
AVG()	Calcule la moyenne des valeurs retournées
MAX()	Retourne la plus haute des valeurs trouvées
MIN()	Retourne la plus petite des valeurs trouvées

II. Manipulation de données

- Ajout de données

Pour ajouter des données dans une base de données, on utilise l'instruction *INSERT*. La syntaxe pour ajouter de nouvelles données avec SQL est la suivante:

```
INSERT INTO Table (Colonnes)
VALUES (valeurs1,valeurs2)
```

- Suppression de données

La suppression de données se fait à l'aide de l'instruction *DELETE*. La syntaxe pour supprimer des données à l'aide de SQL est la suivante:

```
DELETE FROM table WHERE (Critère de recherche)
```

- Mise à jour de données

La mise à jour de vos données se fait à l'aide de l'instruction *UPDATE*. La syntaxe est la suivante:

```
UPDATE table SET Colonne='valeur'
WHERE (Critère de sélection)
```

Remarque :

Le bout de code suivant permet d'effectuer une recherche à partir d'un mot tapé dans un champ de texte

```
valeur = "SELECT * FROM Liste WHERE Produit LIKE " + " ' " +
Text1.Text + " ' "
Data.RecordSource = valeur
Data.refresh
```