PREMIERE PARTIE : L'ALGORITHMIQUE

DEUXIEME PARTIE : Introduction au Visual Basic

PREMIÈRE PARTIE

L'Algorithmique

Ecrire un programme, ce n'est pas seulement connaître le langage de dialogue entre l'ordinateur et vous. Un programme est un discours adressé à l'ordinateur, et comme tout discours, il présente deux aspects : un contenu et une forme, un sens et une grammaire.

Pour l'ordinateur, il suffit que le discours soit correct au niveau de la forme. A partir de ce moment, il effectue les manipulations qu'on lui demande d'effectuer. Le contenu est pour l'ordinateur purement syntaxique. Si l'on fait une erreur de syntaxe, l'ordinateur affichera un message d'erreur. L'apprentissage de la programmation est donc à ce niveau fortement dépendante du langage que l'on utilise. Mais la cohérence du programme, c'est à dire du contenu, n'est pas évaluée.

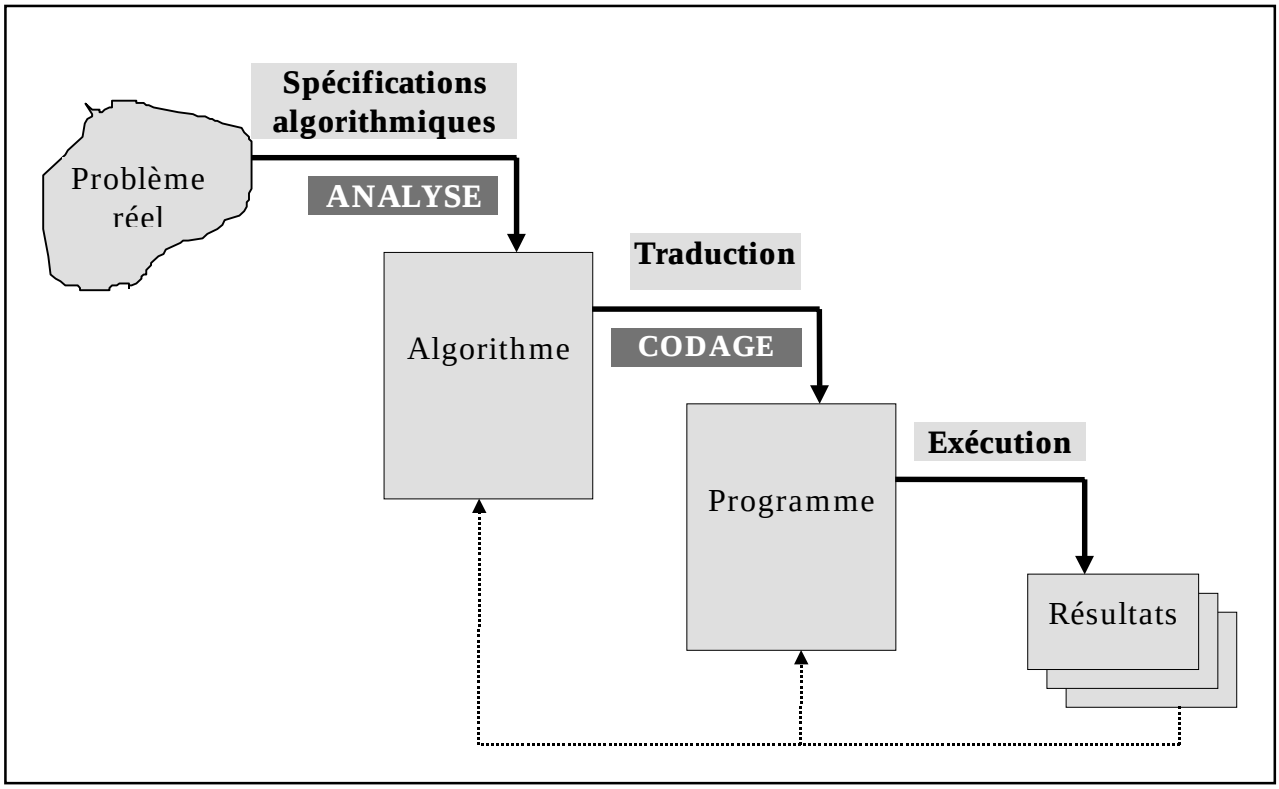
Cette cohérence et cette pertinence de l'analyse du problème à traiter sont donc un préalable à tout exercice de programmation. On doit fixer l'objectif du programme, établir la liste données et des opérations à exécuter, et les ordonner. "La description de la suite des opérations élémentaires ordonnées capables de résoudre le problème posé constitue l'algorithme ".

1. Méthodologie de programmation

1.1. La démarche

L'écriture d'un programme n'est qu'une étape dans le processus de programmation comme le montre le schéma suivant :

- Les différentes étapes du processus de programmation.



L'analyse d'un problème posé consiste à définir les différentes étapes de sa résolution. C'est la partie essentielle dans le processus de programmation. Elle permet de définir le contenu d'un programme en termes de données et d'actions. Une démarche descendante permettrait de décomposer le problème initial en « sous problèmes », plus simples à résoudre. A chacun de ces derniers sera associé une

spécification formelle ayant des conditions d'entrée et le(s) résultat(s) que l'on souhaiterait obtenir. L'ensemble de ces spécifications représente l'algorithme[1] .

La phase suivante consiste à traduire l'algorithme dans un langage de programmation donné. Ce travail, quoiqu'il semble facile, exige le respect strict de la syntaxe du langage.
Lors de l'étape d'exécution, soit des erreurs syntaxiques sont signalées, ce qui entraîne des corrections en général simples à effectuer, soit des erreurs sémantiques plus difficiles à déceler. Dans le cas d'erreur syntaxique, les retours vers le programme peuvent être fréquents. Dans le cas d'erreur sémantique, le programme produit des résultats qui ne correspondent pas à ceux escomptés : les retours vers l'analyse (algorithme) sont alors inévitables.

1.2. Définitions :

◊ Un algorithme est une suite d'actions que devra effectuer un automate (un ordinateur), en un temps fini, pour arriver à un résultat, à partir d'une situation donnée.
◊ Un algorithme est une suite finie d'instructions indiquant de façon précise l'ordre dans lequel doit être effectué un ensemble d'opérations pour obtenir la solution d'un problème.

Avant de présenter les différentes opérations utilisées dans l'écriture d'un algorithme, il faut d'abord découvrir les éléments sur lesquels elles portent, nommés "***données***".

2. Les éléments de base

2.1. Les données

Une donnée peut être considérée comme une boîte, portant une étiquette (***nom***), d'une certaine forme (***type***) et qui contient une information (***valeur***).

Une donnée peut être :
- soit une constante si sa valeur ne change jamais durant l'exécution du programme ;
- soit une variable si sa valeur est susceptible de changer durant l'exécution du programme.

Nous pouvons classer les données selon 3 grandes classes en fonction de la nature des valeurs qu'elles peuvent prendre :

[1] Algorithme : du nom du célèbre mathématicien du IX siècle El-Kharawazmi..

- les données numériques
- les données alphanumériques
- les données logiques.

2.1.1. Les types numériques

Le type numérique caractérise les valeurs entières ou réelles (et parfois complexes),

Entier : De manière générale une variable est caractérisée par un nom appelé ***identificateur*** et un contenu représentant une valeur d'un type donné. Cette dernière peut changer durant l'exécution du programme.
Une variable est dite entière si elle prend ses valeurs dans **Z** (ensembles des nombres entiers relatifs) et qu'elle peut supporter les opérations suivantes :

addition	notée	+
soustraction	notée	-
multiplication	notée	*
division entière	notée	div

n div p = q : la division entière de *n* par *p* donne la partie entière du quotient *q*.

division modulo notée mod

n mod p = r : la division modulo de *n* par *p* donne le reste *r*.

exemples:

12 div 3 = 4
13 div 3 = 4
12 mod 3 = 0
13 mod 3 = 1

Il existe plusieurs types d'entiers représentant chacun un ensemble particulier de valeurs. Cette différenciation de types résulte du mode de stockage des informations selon le langage de programmation utilisé.

Réel : il existe plusieurs types de réels représentant chacun un ensemble particulier de valeurs prises dans **R** (ensembles des nombres réels). Ici encore, cette différenciation se justifie par le mode de stockage des informations.

Il existe deux représentation des réels :

⇒ la *forme usuelle* avec le point comme symbole décimal.

exemples : *0.2467* *345.876* *-12.1* ....

⇒ la *notation scientifique* ayant la forme suivante : **aE+b,** où :

- **a** est la mantisse, qui s'écrit sous une forme usuelle,
- **b** est l'exposant représentant un entier positif.

exemple : *247 = 2.47E2 = 0.247E+3 = 2470E-1 = ...*

Opérations définies sur les rééls :

addition	notée	+
soustraction	notée	-
multiplication	notée	*
division	notée	/

2.1.2. Les types alphanumériques

Le type alphanumérique caractérise les valeurs *caractère* (notées **Car**) ou *chaîne de caractères* (notées **Chaîne**)

Caractère : sa valeur est un caractère quelconque. Un caractère peut appartenir au domaine des chiffres de '0' à '9', des lettres de 'A' à 'Z' (majuscules ou minuscules) et des caractères spéciaux ('+' '-' ',' ';' '.' '(' '{' '[' ']' '}' ')' '$' '%'...). Un caractère sera toujours noté entre des apostrophes. Le caractère blanc (espace) s'écrit ' ', le caractère apostrophe ''''.

Les opérations qu'on définit sur les données de type caractère sont :

égal	notée	=
différent	notée	≠
supérieur ou égal	notée	≥
supérieur	notée	>
inférieur ou égal	notée	≤
inférieur	notée	<

Les quatre dernières représentent un ordre entre les caractères qui est le suivant :

‘ ‘ < ‘0’ < ‘1’ < ... < ‘9’ < ‘A’ < ‘B’ < ... < ‘Z’ < ‘a’ < ‘b’ < ... < ‘z’

Cet ordre est déterminé par la codification ASCII.

Remarque :

- Les minuscules et les majuscules sont considérés comme des caractères différents.

Chaîne : sa valeur est une suite finie de caractères quelconques. Ce type n'est pas toujours pré-défini et doit faire l'objet d'un « paramétrage », en fonction de sa longueur (le nombre de caractères).
Une variable chaîne peut être vide, si elle est de longueur nulle, et sera notée : ' ' . Si cette dernière est égale à 1, la variable est considérée aussi comme ***Car***.

Exemples :

'BONJOUR' *'CECI EST UN EXEMPLE'.*

Les opérations définies sur les variables de type *Chaîne* sont celles des variables de type *Car*.

ChaîneA < *ChaîneB* : si le mot contenu dans *ChaîneA* est 'inférieur' à celui de *ChaîneB* dans le sens du dictionnaire; (inférieur : avant ; supérieur : après).

Exemples:

'BAL' < 'BALLES' < 'BALLON' < 'BAR' < 'Bar' < 'bar'

De plus, il existe une autre opération définie sur les variables chaîne, la ***concaténation*** (notée **| |**). Elle crée une nouvelle en juxtaposant deux ou plusieurs mots.

Exemple :

‘TELE' | | 'VISION' = 'TELEVISION'

2.1.3. Le type logique

Une valeur logique (ou booléenne) est l'une des deux valeurs ***'vrai'*** ou ***'faux'***. Elle intervient dans l'évaluation d'une condition.

Les opérations définies sur les variables de type logique sont *la négation, l'intersection et l'union*.

- *la négation* (notée : ¬ , **non**)

Soit A une variable booléenne :

A	**¬A**
vrai	faux
faux	vrai

- *l'intersection* (notée : ∧, **et**, **.**)

Soient A et B : deux variables booléennes :

A	**B**	**A et B**
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	faux
faux	faux	faux

- *l'union* (notée : ∨, **ou**, **+**)

Soient A et B : deux variables booléennes :

A	**B**	**A ou B**
vrai	vrai	vrai
vrai	faux	vrai
faux	vrai	vrai
faux	faux	faux

2.2 Les expressions

Ceux sont des combinaisons entre des variables et des constantes à l'aide d'opérateurs. Elles expriment un calcul ou une relation, selon qu'elles sont arithmétiques (algébriques) ou logiques.

2.2.1. expressions arithmétiques

Exemple :

*Var1 * 54.5 /(2+pi)*

L'ordre selon lequel se déroule chaque opération du calcul est important. Afin d'éviter les ambiguïtés dans l'écriture, on se sert des parenthèses et d'une hiérarchie entre les différents opérateurs arithmétiques.
Hiérarchie des opérateurs arithmétiques :

1 -	±	opérateur unaire
2 -	( )	parenthèses
3 -	^	puissance
4 -	* /	multiplication ou division
5 -	+ -	addition ou soustraction

Remarque :

- En cas de conflit entre deux opérateurs de même priorité, on commence par celui situé le plus à gauche.

Exemples :

a+b-c :	on fait d'abord *a + b*, ensuite *- c*
*a/b*c :*	on fait d'abord *a / b*, ensuite * *c*

♦ soit l'expression suivante :

((3 * a) - x^2) - (((c - d) / (a / b)) / d)
1 2 4 5
3 6
7

8

♦ Soit l'expression algébrique suivante :

$$\frac{(3-yx)^2 - 4ac}{2x-z}$$

sa forme algorithmique est la suivante :

((3 - y * x) ^ 2 - 4 * a * c) / (2 * x - z)

2.2.2. expressions logiques

Ceux sont des combinaisons entre des variables et des constantes à l'aide d'opérateurs relationnels (=, <, =<, >, >=, ≠) et / ou des combinaisons entre des variables et des constantes logiques à l'aide d'opérateurs logiques (**¬**, **et**, **ou**).

De la même façon, on utilise les parenthèses et des hiérarchies entres les différents opérateurs pour résoudre les problèmes de conflits.

hiérarchie des opérateurs logiques	**hiérarchies des opérateurs relationnels**
¬	>
et	≥
ou	<
	≤
	=
	≠

2.3. Les actions

Toutes les primitives algorithmiques que nous allons utilisées seront présentées en pseudo-code, un mélange convenable de locutions françaises aussi peu ambiguës que possible et de notations algébriques et assimilées pour autant qu'elles soient nécessaires ou commodes.

2.3.1. l'action de déclaration

Il est nécessaire de déclarer tout donnée utilisée dans un algorithme et de préciser son type.

Une constante ou une variable est désignée par un nom : ***l'identificateur***. Celui-ci est une suite de caractères alphanumériques, dont le premier est nécessairement une lettre (en général, on n'utilise que des lettres et des chiffres).

exemples : *Ident1, Ident2, A10, B,...*
mauvais identificateurs : *5ABC, 35, a+b, ...*

Remarques :

- la suite des caractères n'est pas nécessairement limitée .
- Les majuscules et les minuscules ne sont pas distinctes.
- Donner des identificateurs parlants pour faciliter la relecture de l'algorithme.

Exemple:

VAR1 est identique à Var1 et à var1 .

Syntaxe :

Var liste des identificateurs : Type;

Exemple :

```
Var
    Réponse : booléen;
    Chain1 :  Chaîne[30] ;  (Chain1 est une chaîne de 30 caractères)
    Rep     :  Car ;
    i, j    :  entier ;
    a, b, c :  réel ;
```

2.3.2. L'action d'affectation

syntaxe :

ident := expression

où :

ident : est un identificateur;

expression : valeur ou variable ou expression.

Avant d'utiliser toute variable, la première fois dans un calcul, il faut l'initialiser : lui affecter une valeur de départ, à moins de lui affecter le résultat à obtenir.

Dans la notation d'affectation, l'élément à gauche du symbole est une variable qui reçoit une valeur ou le contenu d'une variable ou le résultat d'une expression (partie de droite).

Autre notation de l'affectation : *ident* ← *expression*

exemples :	*mauvais exemples :*	
A := 10;	*a + 1 := 3;*	
B := A;	*A := 3B;*	*(3B n'est pas un identificateur; n'est pas non plus une expression)*
C := C + 1;		

Exemple

Soit un algorithme qui permet de calculer la moyenne de 3 notes et de l'afficher.

```
Algorithme Moyenne ;

Const   Note1 = 12 ; Note2 := 7.5 ;  Note3 := 14 ;
        Var  Som, Moy   :  réel ;
début
                Som := Note1
        Som := Som  +  Note2
        Som := Som  +  Note3
        Moy := Som / 3
                Afficher ( '' La moyenne est de :  '' , Moy) ;
fin;
```

2.3.3. Lecture/Ecriture des données

Les actions d'Entrée/Sortie (ou de Lecture/Ecriture) permettent d'introduire des données dans un programme ou d'afficher des résultats à partir de celui-ci.

Exemples :

Lire (A, B) :	*Entrer 2 valeurs à stocker dans les variables A et B.*
Afficher('' Le résultat est : '' , Moyenne) :	*le message* Le résultat est *s'affichera suivi de la valeur de la variable Moyenne.*

2.3.4. Les actions simples et composées

⇨ On appelle action simple, toute action de déclaration, d'affectation, d'appel à une procédure, ou de Lecture / Ecriture.

exemples d'actions simples :

```
Var i1,i2 : entier;
Som := Som + 1;
Lire (Age);          ‘ lire une valeur et la stocker dans Age
CALCUL (A,B)         ‘ procédure qui fait un calcul sur A et B
Afficher (A,B)       ’ écrire les valeurs de A et de B.
```

⇨ Une action composée est un ensemble fini d'actions simples (ou composées).

syntaxe :

début :

Action 1;
Action 2;
.....
Action p;

fin;

Exemple :

```
Début
    Var A,B : réel;
    A := B +1;
    Lire (A,B);
    CALCUL (A,B);
    Afficher (A,B)
fin;
```

2.3.5. L'action sélective

Supposons qu'on veut connaître le plus grand de 2 nombres a et b.

```
algorithme  plusgrand(a,b)
        Var  a, b, pgran : entier;
        début
            Lire (a, b);
            pgran := a;
            Si  b > a   alors pgran := b
            finsi
        fin;
```

- Ou encore :

```
algorithme  plugrand(a,b)
    Var  a, b, pgran : entier;
    début
          Lire (a,b);
          Si  a > b   alors pgran := a
                sinon pgran := b
          finsi
    fin;
```

Nous pouvons schématiser les actions de sélection dans ces 2 algorithmes de la manière suivante :

Syntaxe :

1ier cas :

SI Condition ALORS action0 FINSI;

Syntaxe

2ième cas :

```
SI  Condition  ALORS  action1
               SINON  action2
FINSI;
```

1ier cas) - Si la condition est vérifiée, on exécute ***action0***, et on passe à la suite de l'algorithme. Si la condition ne l'est pas, alors on passe à la suite de l'algorithme.

2ième cas) - Si la condition est vérifiée, on exécute ***action1*** et on passe à la suite de l'algorithme; sinon on exécute ***action2*** et on passe à la suite de l'algorithme.

Il existe une autre forme de l'action sélective, qu'on appelle le choix multiple :

Syntaxe

```
SELON     variable
   début
          val 1 : action 1;
          val 2 : action 2;
          ...
          val N :  action N

          Sinon : action
   fin;
```

Si ***variable*** est égale à val_i, on exécute $action_i$, et on passe à la suite de l'algorithme, sinon on exécute ***action*** et on passe à la suite de l'algorithme.

2.3.6. L'action itérative

Lorsqu'on a plusieurs actions qui se répètent, on écrit celles-ci dans une même action composée et l'on réitère plusieurs fois l'exécution. Le nombre d'itérations peut être connu a priori ou pas. Dans ce dernier cas c'est l'exécution de l'action itérative qui déterminera son arrêt.

Structure d'une action itérative :

Boucle Condition	Contrôle de la boucle
...	
Action composée	Corps de la boucle
...	
Fin Boucle	Fin de la boucle

Il existe plusieurs façons d'exprimer une action itérative.

⇒ **La boucle Tant que :**

Syntaxe

Tant que ***Condition*** ***faire***
...
ACTION
...
Fin Tant que;

ACTION : peut être une action simple, composée, sélective ou itérative

Tant que la condition est vérifiée on exécutera le corps de la boucle, on s'arrêtera dès que la condition n'est plus vérifiée.

Exemple :

```
Algorithme Moyenne ;
    Var    Note, Moyenne, Som : réel ;
           nb : entier ;
    début
        Som := 0;   nb := 0  ;
        Lire(' Entrer une note : ', Note) ;
        Tant que   Note  < 0   faire
                nb := nb +1 ;
                Som = Som + Note ;
                Lire(' Entrer une note : ', Note) ;
        Fin Tant que
        Moyenne = Som / nb
        Afficher (' La moyenne des ', nb ' notes est de : ', Moyenne);
    fin.
```

⇒ La boucle Répéter

La boucle Répéter permet de rentrer dans la boucle quelque soit la condition et réitère l'exécution jusqu'à ce que la condition soit vérifiée.

Syntaxe :

```
Répéter
    ...
    ACTION
    ...
Jusqu'à   Condition ;
```

exemple :

```
Algorithme Moyenne ;
    Var Note, Moyenne, Som : réel ;
        nb : entier ;
    début
        Som := 0;   nb := 0 ;
        Répéter
                Lire(' Entrer une note : ', Note) ;
                nb := nb +1 ;
                Som = Som + Note ;
        Jusqu'à   Note  < 0   faire
        Moyenne = Som / nb
        Afficher (' La moyenne des ', nb ' notes est de : ', Moyenne);
    fin.
```

Dans cet algorithme, la boucle Répéter comptabilise tout de même la note négative et arrête ensuite l'action itérative. Pour corriger cette lacune il suffit d'écrire :

```
Si  Note ≥ 0  Alors  Som = Som + Note
Fin Si;
```

Remarques :

- Quelque soit l'état de la condition, dans la 2ième forme, on exécutera au moins une fois le corps de la boucle. Tandis que dans la 1ière forme, si la condition n'étant pas vérifiée au départ on n'exécutera pas la boucle.
- Dans ces deux formes, le nombre d'itérations n'est pas connu à priori. Il dépend de la condition.
- Dans le corps de la boucle, il doit exister une variable - dite de contrôle - qui sera modifiée pour faire évoluer l'état de la condition. En général, cette variable de contrôle doit être initialisée avant l'entrée dans la boucle.
- Le choix de la forme de la boucle doit répondre au problème à résoudre.

⇒ **La boucle Pour**

Syntaxe :

3ième forme

```
Pour  i := val_initiale  à  val_finale, Val_pas  faire
        ...
        ACTION
        ...
    Fin Pour ;
```

Du fait que le nombre d'itérations est connu, on se sert d'un compteur (ici ***i***) qui sera initialisé automatiquement à la valeur initiale et sera incrémenté de la valeur du pas, jusqu'à la valeur finale.
Dans le corps de la boucle il est interdit de modifier la valeur du compteur ***i***, même si on peut s'en servir.

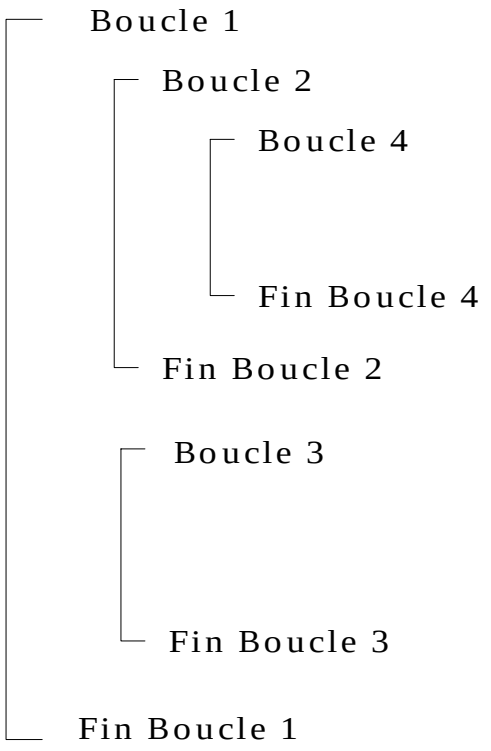
Exemple:

```
Algorithme Salaires ;
    Var
        Sal_annuel, Sal_mensuel : réel ;
        i : entier ;
    Début
        Sal_annuel := 0 ;
        Pour   i := 1  à  12   faire
                Lire (‘Entrer le salaire mensuel : ’ , Sal_mensuel)
                Sal_annuel:= Sal_annuel + Sam_mensuel ;
        Fin Pour ;
        Afficher (‘Le salaire annuel est de : ‘, Sal_annuel) ;
    Fin ;
```

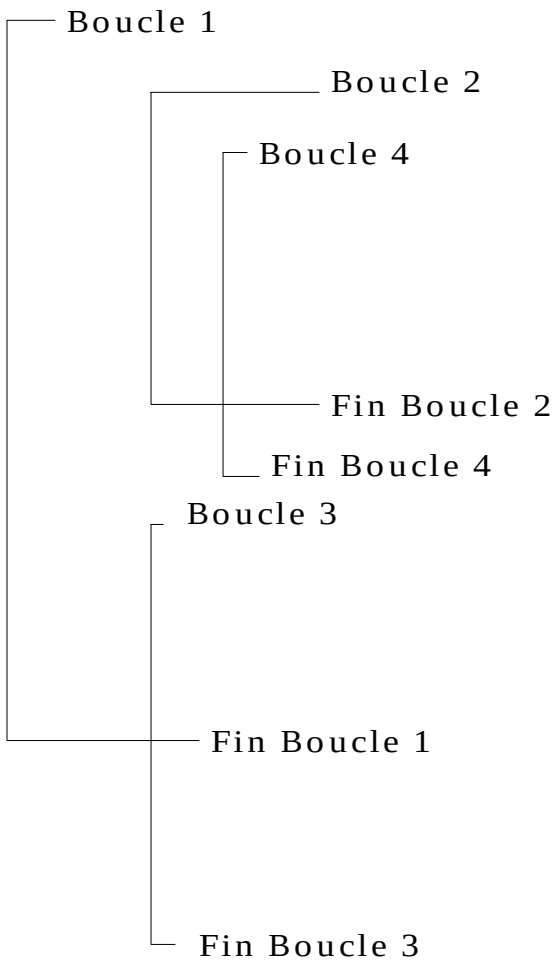
Remarque :

- Dans le corps d'une boucle, on peut avoir une ou plusieurs autres boucles. L'imbrication des boucles doit suivre les règles suivantes :

Imbrications autorisées



Imbrications interdites



2.4. Les autres types de données

Certaines données peuvent avoir un nombre limité de valeurs. Leurs types sont définis à partir d'autres types élémentaires.

2.4.1 Les types construits

2.4.1.1. Le type énuméré

On définit en extension toutes les valeurs que peuvent prendre les variables de ce type. Les valeurs doivent être ordonnées. Elles sont en nombre fini.

Exemple :

Jour = (lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche)

On peut écrire :

```
Var j1, j2 : jour ;
j1 := mardi ;
j2 := dimanche ;
```

Les opérateurs relationnels ($<, \leq, >, \geq, =, \neq$) s'appliquent sur les différentes valeurs de ce type.

Exemple :

lundi < mardi < mercredi < jeudi < vendredi < samedi < dimanche

2.4.1.2. Le type intervalle

On définit en compréhension toutes les valeurs que peuvent prendre les variables de ce type. Les valeurs doivent être ordonnées et on ne précisera que le minimum et le maximum.

Exemple :

Chiffre = 0 .. 9
Lettre = A .. Z
Jour_ouvré = lundi .. vendredi

Remarque :

- Les valeurs appartenant à un intervalle sont nécessairement de type déjà défini. Bien que les types énuméré et intervalle sont peu utilisés, ils permettent néanmoins d'accroître la lisibilité des algorithmes et facilitent les contrôles.

2.4.2. Les types structurés

Il peut y avoir à traiter des lots de données qui peuvent être toutes de même type ou de types différents. Elles sont stockées alors dans une variable multiple à laquelle on donne un nom (identificateur) qui désignera l'ensemble des données et une autre information permettra de désigner individuellement chacune d'elles.

Lorsque leur type est le même, les données sont rangées dans un *tableau*, sinon elles sont représentées dans un *enregistrement*.

2.4.2.1. Les tableaux

C'est un type structuré constitué d'un nombre fini d'éléments obligatoirement de même type. Celui-ci peut être simple ou structué.

Exemple :

Supposons que l'on dispose des chiffre d'inflation pour chaque mois. Pour calculer l'inflation annuelle, il suffit de calculer la somme des données mensuelles.

On stockera alors les données dans un tableau de 12 éléments : *Tab_Inflation* .

Tab_Inflation :

0.05	*0.1*	*0.6*	*0.7*	*-0.4*	*-0.2*	*0.2*	*0.8*	*0.6*	*0.4*	*0.3*	*-0.2*
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*

Pour définir un tableau, il faut préciser un nom commun pour toutes ces données (ici *Tab_Inflation*) et un indice - variable entière pouvant prendre des valeurs entre 1 et

le nombre d'éléments du tableau - (par exemple, *i* entre *1* et *12*). Cet indice indiquera le rang de l'élément dans le tableau.

Exemple :

Tab_Inflation(1) : ‘ le chiffre de l'inflation du mois de janvier.
Tab_Inflation(6) : ‘ le chiffre de l'inflation du mois de juin.
Tab_Inflation(10) : ‘ le chiffre de l'inflation du mois de octobre.

Pour déclarer un tableau, il faut :

- déclarer le type commun aux éléments du tableau
- le type de l'indice, généralement un intervalle.

Exemple :

Type Tab = TABLEAU(1..12) de réel ;

Var Tab_Infaltion : Tab ;

Ou alors :

Var Tab_Infaltion = TABLEAU(1..12) de réel ;

Si on utilise une variable entière *i* comme indice, celle-ci est de type 1..12

Un tableau est caractérisé par sa taille (nombre d'éléments qu'il peut contenir). En général, lorsque ce nombre d'éléments n'est pas connu, on prévoit une taille suffisamment grande pour contenir tous les éléments (même si ceux-là sont en nombre inférieur à la taille déclarée).

Si le tableau contient une seule série de données, on dira que sa *dimension* est égale à 1 (il s'agit d'un vecteur ou tableau-colonne ou tableau-ligne). S'il contient 2 séries, sa *dimension* est égale à 2, c'est une matrice.

Exemple :

Considérons les chiffres mensuels de l'inflation , du chômage et des prix à la consommation. Nous stockerons ces 3 séries de données dans une matrice (tableau à 2 dimensions) :

0.05	*0.1*	*0.6*	*0.7*	*-0.4*	*-0.2*	*0.2*	*0.8*	*0.6*	*0.4*	*0.3*	*-0.2*
0.03	*0.2*	*0.5*	*0.3*	*0.2*	*0.1*	*-0.3*	*0.1*	*0.1*	*0.2*	*0.1*	*0.3*
0.1	*0.005*	*0.03*	*0.2*	*0.1*	*0.07*	*0.03*	*0.3*	*0.2*	*0.4*	*0.3*	*0.3*

Pour exploiter ce tableau, on aura besoin d'un premier indice *i* pour parcourir les lignes du tableaux (chaque série de données) et d'un deuxième indice *j* pour les colonnes (chiffre du mois).

Soit le tableau suivant :

```
Type Tablo = TABLEAU(1..3 , 1..12) de réels ;
Var   Tab_chiffres : Tablo ;
```

On écrira :

```
Tab_Chiffres(1,9)      ‘ pour le taux d'inflation du mois de septembre ;
Tab_Chiffres(2,9)      ‘ pour le taux de chômage du mois de septembre ;
Tab_Chiffres(3,9)      ‘ pour le taux de la hausse des prix à la consommation du mois de septembre ;
```

2.4.2.2. Les enregistrements

Contrairement aux tableaux, ce type structuré permet de regrouper des données de types différents.

Exemple :

On identifie un ouvrage par un code, un titre, un ou plusieurs auteurs, un éditeur et éventuellement la date de parution :

Ouvrage :

- *Code*
- *Titre*
- *Auteur*
- *Editeur*
- *Date*

Ouvrage est une variable de type ***enregistrement*** ; chacune des ses 5 données, est un ***champ***, pouvant être simple ou structuré.

Syntaxe :

Type Nom_Enreg = Enregistrement
Champ 1 : type ;
Champ 2 : type ;
Champ 3 : type ;
...
Champ n : type ;
fin ;

La variable *Ouvrage* se déclare ainsi :

```
Type   Livre = Enregistrement
                    Code :  entier ;
                    Titre :  Chaîne[20] ;
                    Auteur :  Chaîne[40] ;
                    Editeur :  Chaîne[25] ;
                    Date  =  Enregistrement
                                Mois  =  1..12 ;
                                Année = 1900..1999 ;
                             fin ;
                fin ;

Var   Ouvrage : Livre
```

Remarque :

- Le champ *Date* est lui-même un enregistrement.

Exemple :

pour exprimer une date sous la forme suivante : 'lundi 23 Septembre 1996', on déclare la structure d'enregistrement suivante :

```
Type  Date  =  Enregistrement
```

```
    Jour = (lundi, mardi, mercredi, jeudi, vendredi, samedi,
            dimanche) ;
    Quantième = 1..31 ;
    Mois  = (janvier, février, mars, avril, mai, juin, juillet, août,
             septembre, octobre, novembre, décembre) ;
    Année = 1900..1999 ;
fin ;
```

Pour référencer un champ, on préfixe le nom de celui-ci avec le nom de l'enregistrement auquel il appartient :

```
Var  Dat1 : Date ;

    Dat1.jour := lundi
    Dat1.quantième := 27
    Dat1.Mois := septembre
    Dat1.année := 1996
```

autre exemple :

```
Var Ouvrage : Livre ;

Ouvrage.Auteur := 'V. Hugo '
Ouvrage.Date.Mois := 10
Ouvrage.Date.Année :=  1995
```

2.4.3. Les Fichiers

Un fichier est un ensemble de données. Il peut servir soit à la lecture, pour rentrer des informations dans un programme , soit à l'écriture pour sauvegarder les résultats obtenus.

Les fichiers sont caractérisés par deux notions :

- le mode d'organisation : comment sont organisées les données dans le fichier (séquentiel, indexé,...) ;
- le mode d'accès : comment sont accédées les données dans le fichier (séquentiel, direct, binaire,...).

Ces caractéristiques sont étroitement liées aux langages de programmation utilisés. Chacun de ces derniers offre différents types de fichiers.

En algorithmique, nous nous limiterons par souci de facilité aux fichiers texte et aux fichiers d'enregistrements.

L'utilisation d'un fichier se fait selon les phases suivantes :

- ♦ Ouverture du fichier.
- ♦ Traitement du fichier.
- ♦ Fermeture du fichier.

⇨ Les fichiers texte

Les fichiers de type *Texte* sont des fichiers séquentiels (mode d'organisation séquentielle). Les informations sont disposées de façon séquentielle, les unes à la suite des autres. Elles ne sont ni en ligne ni en colonne ! Elles sont repérées par un pointeur. Leur organisation est séquentielle et leur accès ne peut être que séquentiel .

Trois opérations sont définies sur ce type de fichiers :

- *La lecture* : Lors de l'ouverture du fichier, le pointeur *pointe* sur la 1° information, quelque soit son type. A chaque accès (***Lire***), le pointeur se déplace sur l'information suivante, (mode d'accès séquentiel). Si on veut lire une information en amont du pointeur, il faut fermer le fichier, le rouvrir et lire jusqu'à l'information désirée.

- *L'écriture* : un fichier non vide ouvert en écriture perd tout ce qu'il possède. En effet, dès son ouverture le pointeur est positionné sur la première ligne. Seules les opérations d'écriture sont autorisées.

- *L'ajout* : cette opération permet de rajouter de nouvelles données à la fin du fichier sans détruire ce qu'il y avait auparavant. Le pointeur est positionné sur la marque de fin de fichier qui est décalée d'une position après chaque rajout.

Exemple :

```
Var Fich1 : FICHIER texte

Pour i := 1  à  100 faire
     Lire (Fich1, Tab(i))
fin Pour
```

Les données du fichier *Fich1* sont lues et stockées dans la tableau *Tab.*

Ecrire (Fich1, ‘’Le salaire annuel est de : ‘‘, *Sal_Annuel) ;*

Ces informations sont stockées dans *Fich1* à l’endroit où se trouve le pointeur lors de cette action.

Ajout (Fich1, ‘‘Le salaire annuel est de : ‘‘, *Sal_Annuel) ;*

Ces informations sont rajoutées à la fin de *Fich1* et le pointeur reste sur la marque de fin de fichier.

Il n’y a que la fin du fichier qui est marquée par un symbole repéré par la fonction *EOF*(*Nomfichier*), qui rend la valeur *vraie* si elle le rencontre, la valeur *faux* sinon.

Un fichier séquentiel ne peut être ouvert qu’on lecture ou en écriture. Après l’ouverture d’un fichier, la première opération (***Lire, Ecrire*** ou ***Ajout***) indique si le fichier est accessible en lecture ou en écriture.

Ecrire et ***Ajout*** sont des opérations d’écriture, leur seule différence et due au fait que pour ***Ecrire***, le pointeur se place en début du fichier, alors que pour ***Ajout***, il se place en fin du fichier.

Toute manipulation d’un fichier nécessite 3 phases :

- ☞ Ouverture du fichier :

  OUVRIR (Nomfichier)

- ☞ Traitement du fichier : Lecture ou Ecriture :

  LIRE(Nomfichier, ……..) ;
  ECRIRE(Nomfichier, ……..) ;
  AJOUT(Nomfichier, ……..) ;

- ☞ Fermeture du fichier :

  FERMER(Nomfichier)

Remarques :

- La fonction ***EOF(Fich1)*** permet de tester si le pointeur est sur la fin du fichier *Fich1*.
- L’utilité des fichiers est la sauvegarde les données.

- Il est préférable d'utiliser ***Ecrire*** à la place d'***Afficher***, quand on utilise les fichiers.

⇨ Les fichiers d'enregistrements

C'est un ensemble d'enregistrements (ou d'articles) de type structuré que l'on déjà défini.

Exemple :

```
Type Etudiant = Enregistrement
                    Numéro :  entier ;
                    NomPrénom : Chaîne[30]
                    Discipline : Chaîne[25]
                    Année_Inscrip :  1950..2000
                fin ;

Var E1 : Etudiant
Fich_Etudiant :  FICHIER de Etudiant ;

E1.Numéro := 134561 ;
E1.NomPrénom := ''Dupont Lionel'' ;
E1.Discipline := ''Sciences économiques'' ;
E1.Année_Inscrip := 1996
Afficher(Fich_Etudiant, E1) ;
```

On peut traiter un fichier d'enregistrements de manière séquentielle, mais son intérêt et de permettre un accès direct aux données.

Lors de l'ouverture d'un tel fichier, le pointeur est positionné sur le premier enregistrement. On peut se déplacer directement sur n'importe quel enregistrement avant une opération de lecture ou d'écriture à l'aide de l'action :

Positionner(Fichier, N°enregistrement)

Remarques :

- Contrairement aux fichiers séquentiels, un fichier d'enregistrements peut être ouvert en lecture et en écriture.
- La taille d'un fichier de ce type est le nombre de ses enregistrements.

2.5. La notion de sous programmes

Dans la résolution d'un problème, on peut constater qu'une suite d'actions revient plusieurs fois. Dans ce cas il serait judicieux de l'écrire une seule fois, et de l'utiliser autant de fois que c'est nécessaire, en effectuant des calculs avec des données différentes.
Cette suite d'actions sera définie dans un sous programme, qui peut prendre soit la forme d'une *procédure*, soit la forme d'une *fonction*.

D'autre part, on peut observer que certains groupes d'actions se rapportent à des traitements précis et différents. Il est souhaitable alors de représenter chacun d'eux dans un sous programme, ce qui permettra d'améliorer la conception du programme et sa lisibilité. On perçoit alors un programme comme un ensemble de procédures / fonctions. La structuration d'un programme par morceaux (modules) est la base de la programmation structurée et modulaire.

2.5.1. La procédure

Une procédure est un sous programme qui peut retourner 0, 1 ou plusieurs résultats.

⇒ **définition de la procédure :**

```
Procédure nom (listes d'arguments :type);
    début

        corps de la procédure

    fin;
```

La première ligne s'appelle l'en-tête (ou la signature) de la procédure. La liste d'arguments est une suite de données à échanger avec d'autres programmes.

⇒ **appel de la procédure :**

```
nom-de-la-procéduer (liste d'arguments);
```

Remarques :

- Si une liste d'arguments apparaît dans la définition d'un sous-programme, lors de l'appel à ce dernier, elle doit également apparaître dans l'action d'appel.
- Les listes d'arguments dans la définition d'un sous-programme et dans son appel doivent se correspondre. C'est à dire, il doit y avoir le même nombre d'arguments dans les deux listes. L'ordre d'apparition des arguments dans les deux listes doit être le même. Chaque argument d'une liste doit être de même type que son homologue de l'autre liste.
- La liste d'arguments dans la définition d'un sous-programme contient le type de chaque argument. Ce n'est pas le cas de celle de l'appel.

exemple :

On veut écrire un algorithme qui calcule le salaire des commerciaux d'une entreprise. Celui-ci est composé d'un fixe différent d'un employé à un autre et d'une prime d'intéressement de 10% du chiffre d'affaire réalisé si celui-ci dépasse les 50 000 F, de 3% sinon.

On isolera la suite d'actions qui permet de rentrer les salaires fixes de chacun ainsi que leur chiffre d'affaire.

```
Procédure Saisie( Var  Sal , CA : entier) ;

début
        Lire (''Entrer le salaire du commercial : '', Sal) ;
        Lire (''Entrer son chiffre d'affaire : '', CA) ;
fin.
```

2.5.2. La fonction

Une fonction est un sous programme qui retourne obligatoirement une valeur. Cette dernière sera stockée dans une variable qui porte le même nom que la fonction.

⇒ **définition de la fonction :**

```
Fonction nom (liste d'arguments : type) : type;
        début

                corps de la fonction

        fin;
```

Le nom de la fonction est utilisé comme identificateur d'une variable, on déclare alors le type de cette dernière.

⇒ **appel de la fonction :**

var := expression (... fonction(liste d'arguments.) ...)

Exemple (suite) :

De la même manière, on isole les actions permettant de calculer la commission de chaque commercial.

```
Fonction Commission(Montant : réel) : réel ;
   Const    plafond = 50000 ;
   Var   taux  :  réel ;
   début
         Si  Montant ≥ plafond  Alors  taux := 0,1
                                Sinon  taux := 0,03
            Fin Si ;
         Commission := Montant * taux ;
         fin.
```

On peut construire un troisième sous-programme qui calculera le salaire de chacun.

```
Procédure Calcul_Salaire ;
   Var  Salaire, Sal_fixe, Chif_Aff :  réel ;
   début
         Saisie(Sal_fixe, Chif_Aff) ;
         Salaire := Sal_fixe + Commission(Chif_Aff )
         Afficher(' Le salaire est de : Salaire) ;
   fin.
```

Commentaires :

- Les listes d'arguments sont optionnelles.
- La procédure *Calcul_Salaire* est le programme appelant. La procédure *Saisie* et la fonction *Commission* sont les programmes appelés.

- ♦ N'importe quel sous-programme peut être appelant ou appelé (il faut éviter les appels circulaires).
- ♦ Les arguments des Procédures / Fonctions appelées sont les paramètres formels (ou fictifs). Ils peuvent porter les mêmes noms (ou des noms différents) que leurs correspondants des programmes appelant qualifiés de paramètres réels (ou effectifs).
- ♦ Dans la procédure *Calcul_Salaire,* à l'appel de la procédure *Saisie,* les paramètres réels *Sal_fixe, Chif_Aff* sont vides (ou contiennent plutôt n'importe quoi). Au retour du sous programme, ils posséderont respectivement le salaire et le chiffre d'affaire réalisé.
- ♦ Dans l'appel à la fonction *Commission* (action qui calcule le salaire), le paramètre réel *Chif_Aff* possède déjà la valeur du chiffre d'affaire retournée par *Saisie.* Au retour de cette fonction la variable *commission* contiendra la valeur de la prime.

2.5.3. La portée des données

Les sous programmes communiquent entre eux par des paramètres. Une Procédure / Fonction peut avoir des variables internes qui ne sont pas visibles par les autres. Il s'agit de variables locales. Elles ne sont accessibles que par le sous-programme où elles sont définies. Par conséquent, différents sous-programmes peuvent avoir des variables locales portant le même nom éventuellement. Celles-ci n'auront pas la même signification pour chacun d'eux. On dira également que la portée d'une variable locale est le sous-programme où elle a été définie.

Exemple :

Sal_fixe et *Chif_Aff* sont des variables locales à *Calcul_Salaire.*

Si une variable doit être accessible par tous les sous-programmes, il faut la définir comme une variable globale. Ce qui veut dire qu'elle est visible par tout sous-programme et que sa valeur peut être utilisée ou modifiée n'importe où. La portée d'une variable globale est l'ensembles des sous-programmes pouvant l'utiliser.

Remarques :

- La déclaration des variables globales dépend du langage de programmation utilisé. Certains les définissent dans le programme principal (programme d'appel aux Procédures / Fonctions) ; d'autres dans une section spéciale.
- La notion de portée s'appliquent également aux constantes et aux types.

Exemple :

Dans la fonction *Commission, plafond* est une constante locale.

2.5.4. Le passage des paramètres entre sous programmes

Lors de l'échange de données entre sous-programmes, on peut soit autoriser, soit interdire la modification des valeurs.

Il existe deux modes de passage de paramètres :

* ***passage par valeur*** : dans les sous-programmes appelés, les valeurs des paramètres transmis sont utilisées sans qu'il soit possible de les modifier. Les paramètres formels contiennent une copie des valeurs des paramètres réels.

Exemple :

Dans la fonction *Commission*, la valeur initiale du paramètre *montant* est conservée.

* ***passage par adresse :*** les sous-programmes appelés peuvent modifier les valeurs des paramètres réels transmis. Les paramètres formels contiennent l'adresse des paramètres réels. Les modifications de valeurs effectuées dans le programme appelé seront effectives au retour dans le programme appelant.

Exemple :

Dans la procédure *Saisie*, les valeurs affectées aux paramètres formels *Sal* et *CA* seront disponibles dans le programme appelant *Calcule_Salaire.*

DEUXIÈME PARTIE

Introduction au Visual Basic

1. Généralités

1.1. Programmation objet, visuelle, événementielle

La programmation d'aujourd'hui est par objet, visuelle, événementielle... Mais que représentent tous ces concepts ? quelle est la démarche à suivre pour construire une application ?
Nous allons tenter de répondre simplement à ces questions. Quoiqu'il existe toute une théorie sur la conception des programmes orientés objet, nous nous contentons de donner une définition simple d'un objet afin de le situer dans ce contexte de travail, sans verser dans les détails précis des notions telles *l'héritage, « l'encapsulation », le polymorphisme....*
Un objet est un ensemble de données : qu'on appelle ***attributs*** ou ***propriétés*** - permettant de le caractériser ; et de programmes : qu'on appelle ***méthodes -***, servant ente autres à modifier les propriétés. Construire un programme-objet revient à définir un ensemble d'objets. Souvent l'utilisateur fait appel à des objets déjà pré définis dans le langage utilisé, en les personnalisant par modification des valeurs des propriétés et / ou en adaptant les méthodes à la logique de son application.
Un programme *visuel* est fondamentalement constitué d'une partie communication conséquente. Celle-ci représente une interface conviviale facilitant le discours avec l'utilisateur même non initié à l'outil utilisé. Elle est formée d'éléments graphiques simples à créer et faciles à lire, auxquels sont associés des actions, programmées par l'utilisateur. Ce dernier déclenche ces traitements selon sa propre logique de résolution (comme, lorsqu'il utilise un logiciel de bureautique). Ces déclenchements - ou *événements* - sont provoqués en appuyant sur une touche du clavier ou en manipulant la souris ou tout simplement lorsqu'une variable est dans un état donné.

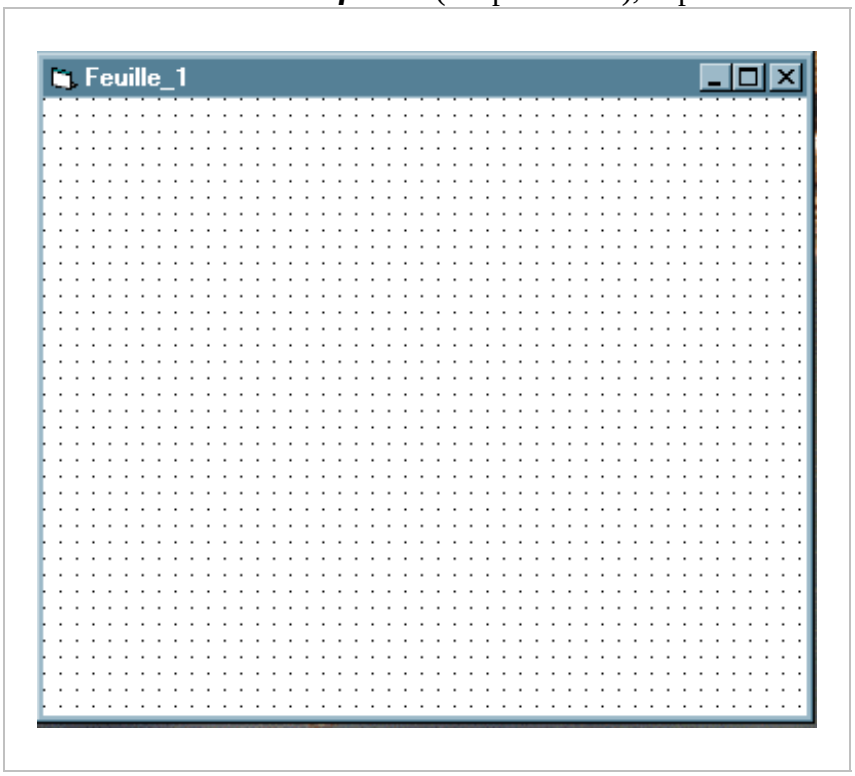
Un programme constitué d'objets visuels et exécuté à l'aide d'événements est un *programme visuel et événementiel*.
Visual Basic est un langage de programmation visuelle et événementielle. Il permet de construire des applications en trois phases :

- Conception de l'interface graphique. Cela consiste à construire des fenêtres et d'autres objets graphiques à l'aide d'outils de dessin.
- Affectation des valeurs à certaines propriétés des ces objets graphiques.
- Ecriture du code des actions à associer aux objets graphiques.

1.2. Composition d'une application VB

1.2.1. La forme visuelle

Une application est formée d'une ***feuille*** (ou plusieurs), représentant une fenêtre



- Fig.1 : Une Feuille VB

dans laquelle on dispose des ***contrôles*** : étiquettes, zones de saisie, listes, boutons de commandes ou d'options, cases à cocher, etc ...
La feuille et les contrôles sont des objets possédant des propriétés ayant déjà des valeurs et des méthodes, qui souvent sont des cadres vides que l'utilisateur peut remplir par des « bouts de programmes ».
Tout objet (feuille ou contrôle) a une liste de propriétés ayant une valeur lors de sa construction. L'utilisateur peut modifier certaines d'entre elles pour changer la taille, la couleur, le titre, le contenu, l'emplacement... afin de personnaliser l'objet. Ces modifications peuvent avoir lieu lors da la construction de la forme visuelle. C'est à

dire de façon statique, avant l'exécution du programme, ou alors en cours de celui-ci de manière dynamique.
Avant de développer le code des différentes actions, il faut d'ores et déjà penser aux différents événements qui vont associer les objets visuels aux programmes. Leur rôle consiste à permettre le déclenchement des différents traitements.
La conception de la forme visuelle : création de la feuille et des contrôles, valorisation des propriétés, détermination des événements, nécessite une analyse portant sur la partie communication entre l'utilisateur et l'application. C'est une réflexion qui se distingue de l'algorithmique classique.

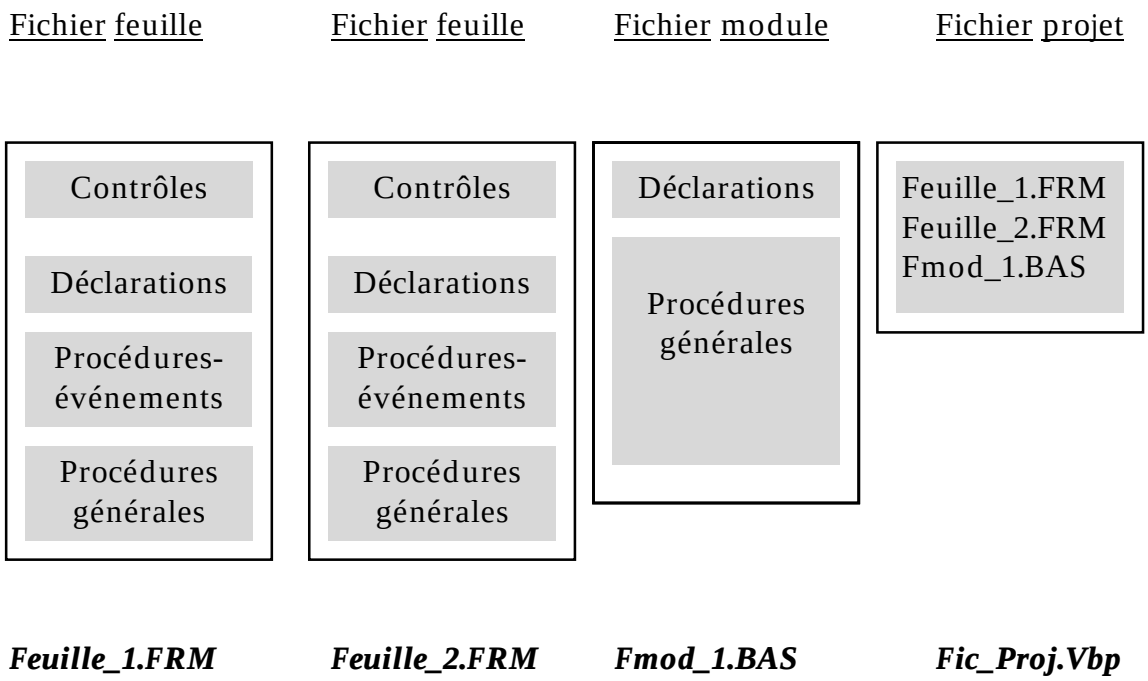
1.2.2. Les programmes VB

Le code dans une application VB est représenté par des sous-programmes écrits dans des procédures ou des fonctions. Il existe plusieurs types de procédures.
Les procédures-événements : ceux sont des procédures associées aux différents contrôles ou feuilles par les événements, d'où leur nom. Elles sont toujours stockées dans un fichier feuille. Chaque contrôle ou feuille possède une liste d'événements pré définies auxquels correspondent des procédures-événements. En fait, ces dernières sont des cadres vides dans lesquels l'utilisateur écrit du code pour programmer une action donnée. Rappelons qu'une action est une suite d'instructions.

- *Les procédures générales* : ceux sont des procédures classiques que l'utilisateur pourra écrire pour isoler un traitement ou éviter de le répéter plusieurs fois. Elles sont appelées soit par des procédures-événements, soit par d'autres procédures générales. Elles peuvent prendre forme d'une fonction si elles retournent toujours un résultat. Elles sont toujours stockées dans un fichier feuille ou dans un fichier module.
- *Les fonctions VB* : VB met à la disposition de l'utilisateur une librairie de fonctions ou instruction pré définies pouvant servir dans les différents traitements. Nous pouvons citer :
    - les fonctions mathématiques
    - les fonctions financières
    - les fonctions de conversions des données
    - les fonctions de traitements de chaînes de caractères
    - les fonctions de dates / heures
    - les fonctions d'accès aux fichiers, aux répertoires et disques
    - ....

Contrairement à un programme classique, l'utilisateur conçoit une application VB en une forme visuelle et un ensemble de programmes dont une partie est affectée aux différents objets (contrôles ou feuilles).

1.2.3. Les fichiers d'une application VB



- Fig.2 : Les fichiers d'une application VB.

Une application VB peut être composée d'un ou plusieurs fichiers feuille et / ou d'un ou plusieurs modules et toujours d'un fichier projet.

❶ *Le fichier feuille* : On y stocke tous les objets graphiques (feuille, contrôles) avec toutes leurs propriétés et les procédures-événements qui sont définies. On peut également y mettre des procédures générales. Cependant, celle-ci sont « locales » aux procédures du fichier. Elles ne sont pas visibles des autres fichiers de l'application. Toutes les variables, constantes et types présents dans la section *Déclarations* sont également accessibles à l'ensembles des procédures de ce fichier.

❷ *Les fichiers module* : On y trouve que les procédures générales qui de plus sont globales à l'ensemble de l'application. Il en est de même pour les données (variables, constantes et types) déclarées dans la section *Déclarations.* Les types spécifiques (enregistrement, par exemple) ne sont définis que dans les déclarations d'un module.

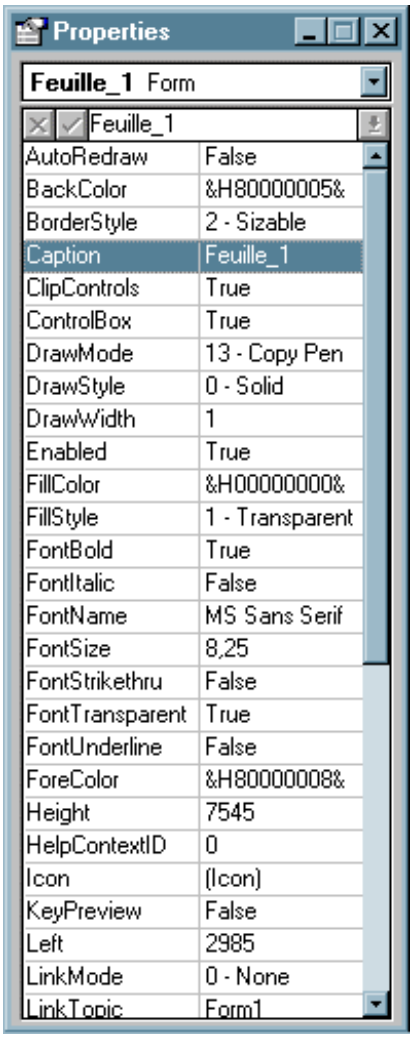
❸ *Les fichiers projet* : C'est un fichier répertoire. Il ne contient que les noms des fichiers feuilles et / ou modules qui composent une application. Lors de l'ouverture d'une application, il suffit de charger le fichier projet.

Remarque :

- Lorsqu'on sauvegarde une application, il faut sauvegarder chacun de ses fichiers.

1.2.3. Les fenêtres de VB

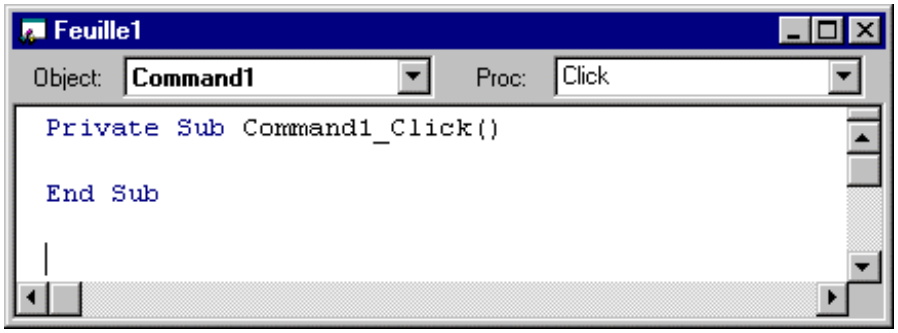
VB dispose de plusieurs fenêtres de travail pour créer une application :



- Fig.3 : Fenêtre des propriétés.

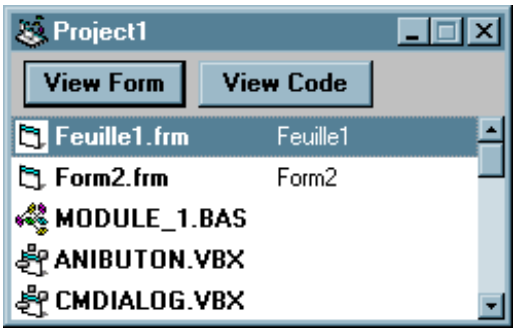
- Fig.4 : Fenêtre des outils de dessin.

La création d'une forme visuelle se fait interactivement. L'utilisateur sélectionne des contrôles dans la boîte à outils (fig.4) et les dépose sur la feuille (fig.1) . Il les personnalise en modifiant les valeurs des propriétés à l'aide de la fenêtre des propriétés(fig. 3).
Quant à l'écriture des programmes, elle se fait dans la fenêtre code d'un fichier feuille :



- Fig.5 : Fenêtre de code d'un fichier feuille.

La *fenêtre objet* : contient la liste de tous les objets (feuille et contrôles) de l'application (ici l'objet est le contrôle bouton de commande *Command1*).
La *fenêtre Proc* contient la liste de toutes les procédures-événements des contrôles et de la feuille (ici la procédure-événement *Click* du contrôle *Command1* nommée *Command1_Click* .
Il existe un objet *Général* contenant la section *Déclarations* et toutes les procédures générales créées par l'utilisateur.
La fenêtre code d'un module a la même apparence que celle d'une feuille, sauf qu'elle ne possède que l'objet *Général*.
Enfin la fenêtre projet permet de gérer tous les fichiers feuille et / ou modules qui composent l'application :



- Fig.6 : Fenêtre de projet d'une application.

2. Eléments de base du VB

2.1. Les variables

2.1.1. Les identificateurs

Ils servent à référencer les variables (les constantes et les types aussi). Ils sont constitués de chaînes de caractères (mélange de lettres et de chiffres et le symbole « _ ») dont le premier est une lettre et de longueur inférieure ou égale à 40. Les majuscules et les minuscules sont identiques. Les caractères accentués sont autorisés.

Exemples :

Le_résultat
M1
Var1
...

2.1.2. Les types

VB possède plusieurs types

- ☞ Numériques
  - ♦ Entiers
    - INTEGER : stockés sur 2 octets (-32768 à +32767) ; entiers positifs ou négatifs.

      Suffixe : ***%***
    - LONG : stockés sur 4 octets (-2,147 à +2,147 milliards)

      Suffixe : ***&***
  - ♦ Réels
    - SINGLE : stockés sur 4 octets ($-3,402^{E}38$ à $-1,40^{E}-45$ pour les valeurs négatives et $1,40^{E}-45$ à $3,40^{E}38$ pour les valeurs positives) ; nombres avec partie décimale.

      Suffixe : ***!***
    - DOUBLE : stockés sur 8 octets ($-1,797^{D}308$ à $-4,94^{D}-324$ pour les valeurs négatives et $4,94^{D}-324$ à $1,797^{D}308$ pour les valeurs positives) .

      Suffixe : ***#***
    - CURRENCY : réels à virgule fixe ayant 4 décimales, pour les chiffres financiers.

      Suffixe : ***@***
- ☞ Chaînes de caractères
  - ♦ STRING : pour stocker du texte. Elles peuvent avoir une taille jusqu'à 65 000 caractères. La longueur d'une chaîne est soit variable : elle s'adapte à son contenu ; soit fixe déclarée initialement.

    Suffixe : ***$***

Exemples :

```
Ch1   As   String * 20
Ch2   As   String
```

☞ Le type Variant

Le type d'une variable Variant peut changer selon le contexte. Il peut s'agir d'un nombre, d'une chaîne ou une date / heure. VB effectue automatiquement les conversions nécessaires.

Suffixe : pas de suffixe.

☞ Les types spécifiques

L'instruction TYPE sert à définir les enregistrements.

Exemple :

```
TYPE  Personne
      Nom   As   String * 40
      Numéro   As Integer
      Adresse   As  String
End Type
```

Remarque :

- Il existe d'autres types dans VB pour déclarer les variables objet ou autres...
- Par défaut, le type est *Variant*.

Exemple :

```
Dim Variable_1
```

Selon ce qu'on écrit :

```
Varaible_1  =  28                    ' Variable_1 est Integer ;
Varaible_1  =  « OUI »                    ' Variable_1 est String ;
```

- La fonction *VarType* permet de connaître la représentation interne d'une variable *Variant*.
- Initialement une variable *Variant* est *Empty* (vide), on peut lui affecter la valeur *NULL* (valeur nulle).
- Les fonctions *IsEmty* ou *IsNull* permettent de tester ces conditions.

2.1.3. Les déclarations

On peut déclarer explicitement ou implicitement toutes les données utilisées (variables, constantes ou types).

2.1.3.1. Les déclarations explicites

C'est à l'aide des instructions ***Dim, Static*** ou ***Global*** qu'on fait une déclaration.

Exemples :

```
Dim   A   As Long
Dim   Ch_1 As String
Static   X   As Integer
Global   V   As   Single
```

Remarques :

- Lors de la première exécution de l'une de ces instructions, les variables sont initialisées à 0 , à vide ou *Empty*, selon le cas. Cependant s'il y a d'autres exécutions (par exemple, lors des appels de sous programmes) :
  - Si l'instruction est *Dim*, les variables sont ré initialisées à 0 , à vide ou *Empty*.
  - Si l'instruction est *Static*, les variables gardent leur valeur avant l'exécution de celle-ci.

2.1.3.2. Les déclarations implicites

1ier cas :

L'instruction *DefType* indique que les variables commençant par une lettre définie à l'aide de celle-ci sont du type déclaré.

Exemples :

DefInt A-E	*Les variables commençant par A ou B ou C ou D ou E sont Integer.*
DefSng X, Y	*Les variables commençant par X ou Y sont Single.*
DefStr H, L-O	*Les variables commençant par H ou une lettre de L à O sont String.*

2ième cas :

On peut aussi utiliser le suffixe d'un type pour utiliser une variable sans la déclarer.

Exemple :

```
Compteur% = Compteur% +1
Ch1$ = "Exemple"
```

Remarque :

- Il est recommandé d'utiliser des déclarations explicites, en les forçant par ***Option Explicit***.

2.1.3.3. La déclaration des tableaux

Exemple :

```
Dim Tjours(1 To 31) As Integer
Dim Tsemaine(6) As String
```

On pourra écrire :

```
Tsemaine(0) = "Lundi"
Tsemaine(1) = "Mardi"
...
Tsemaine(6) = "Dimanche"
```

On peut également utiliser un tableau à 2 dimensions pour stocker des températures par exemple.

Exemple :

```
Dim Tdate(1..31 , 1..12) As Single
```

Si on écrit :

```
TDate(10,9) = 20
```

Cela veut dire que le 10 septembre il a fait 20°.

Remarque :

- La borne inférieure d'un tableau est par défaut égale à 0.
- On peut avoir jusqu'à 60 dimensions .

2.1.4. La portée des variables

⇒ Si une variable est déclarée à l'aide de *Dim* ou *Static* dans une Procédure / Fonction, elle est locale.

⇒ Si elle est déclarée à l'aide de *Dim* dans la section *Déclarations* d'un module (ou d'une feuille), elle est utilisable par tous les sous programmes du module (ou de

la feuille). Elle est « globale » pour les Procédures / Fonctions du module ou de la feuille.

⇒ Si elle est déclarée dans la section *Déclarations* d'un module à l'aide de *Global*, tous les feuilles et modules de l'application peuvent l'utiliser.

2.1.5. Les constantes

Les constantes sont déclarées à l'aide de l'instruction *Const* dans une feuille ou un module, pour être utilisée localement ; ou globalement à l'aide de *Global Const* .

2.2. Les actions

2.1.1. Les actions simples

✍ *La déclaration*

toutes les instructions de déclaration du chapitre précédent sont considérées comme des actions simples.

✍ *Le commentaire*

C'est une instruction non exécutable et sert à faciliter la lisibilité du programme.
Il s'écrit sous plusieurs formes (à l'aide de *Rem* ou du symbole ').

Exemples :

```
Rem      Ceci est un exemple de commentaire.
'  Ceci est un exemple de commentaire.
i = i + 1          'incrémentation de la variable i
```

✍ *L'affectation*

Syntaxe :

Var = Valeur

où ***Var*** est une variable
Valeur est une valeur ou variable ou expression

Les 2 membres d'une affectation doivent être de même type.
Si le membre gauche est Variant, celui de droite peut être String, numérique ou Variant.
Si le membre de droite est Variant alors celui de gauche peut être String ou numérique ou Variant.
Il existe d'autres instructions simples telles l'appel aux sous programmes.

2.1.2. Les actions sélectives

2.1.2.1. Les tests simples

Syntaxe :

```
If Condition Then Action_1
           Else Action_2
End If
```

Exemples:

```
If Chif_Aff >= 10000 Then Taux = 0.10
                     Else Taux = 0.05
End If

If x > y Then Rep$ = "x est plus grand que y "
         Else Rep$ = "y est plus grand que x "
End If

If a <> 0 Then x = y /a
End If
```

2.1.2.2. Les tests en cascade

Syntaxe :

```
If Condition_1 Then Action_1
Else If Condition_2 Then Action_2
Else If Condition_3 Then Action_3
...
Else If Condition_n Then Action_n
Else Condition_n+1 Then Action_n+1
End If
```

Exemple :

```
If jour = "lundi" Then Temp% = 20
Else If jour = "mardi" Then Temp% = 25
Else If jour = "mercredi" Then Temp% = 21
Else If jour = "jeudi" Then Temp% = 19
Else If jour = "Vendredi" Then Temp% = 20
Else If jour = "Samedi" Then Temp% = 21
Else Temp% = 20
End If
```

2.1.2.3. Le test IIf

Syntaxe :
IIf (Condition , Si_Vrai , Si_Faux)

Exemples:

IIf (Chif_Aff >= 10000 , Taux = 0.10 , Taux = 0.05)

2.1.2.4. Les choix multiples

Syntaxe :
Select variable_1
Case Liste_1 Action_1
Case Liste_2 Action_2
...
Case Liste_n Action_n
Case Action
End Select

Exemples:

```
Select moyenne
   case 16 To 20
        Mention = ″ T. Bien ″
   case 14 To 15.99
        Mention = ″ Bien ″
   case 12 To 13.99
        Mention = ″ A. Bien ″
   case 10.5 , 11 , 11.5
        Mention = ″ Passable ″
   case Else
        Mention = ″ Sans ″
End Select
```

2.1.2.3. L'instruction Choose

Syntaxe :
Choose (Index , Exp_1 , Exp_2, ...)

Index prend les valeurs 1, 2, 3, ... jusqu'au nombre d'expressions désignées (13 au maximum).

Exemples:

```
taux = Choose(Ind_1, 0.05 , 0.07 , 0.1 , 0.15)
Si  Ind_1 = 1  alors  Taux = 5%
Si  Ind_1 = 2  alors  Taux = 7%
Si  Ind_1 = 3  alors  Taux = 10%
Si  Ind_1 = 4  alors  Taux = 15%
```

2.1.3. Les actions répétitives

☝ Le nombre d'itérations peut être inconnu au départ.

2.1.3.1. La boucle While... Wend

Syntaxe :
While Condition
ACTION
Wend

Dans cette forme, si la condition est vraie au départ, on entre dans la boucle et on exécute son corps (ACTION) tant qu'elle est vérifiée. Dans le cas contraire, on saute simplement toute l'action *While.*
Dans ACTION, il doit y avoir une instruction qui modifie la condition pour qu'elle puisse changer d'état et permettre la sortie de la répétition.

Exemple :

```
i = 1
Sal_Annuel = 0
While i <= 12
   Sal_annuel = Sal_Annuel + Salaire
   i = i +1
Wend
```

2.1.3.2. Les boucles Do ... Loop

✍ **1ier cas :**

Syntaxe :	
Do While Condition ***ACTION*** ***Loop***	***Do Until Condition*** ***ACTION*** ***Loop***

Dans ces 2 formes, Si la condition est vraie dès le départ, on entre dans le corps de la boucle et on répète l'exécution tant que la condition est vérifiée ou jusqu'à ce qu'elle ne le soit plus. Dans le cas contraire, on saute simplement l'action de répétition.
Ces deux formes du *Do ... Loop* sont identiques à l'instruction *While ... Wend.*

✍ **2ième cas :**

Syntaxe :	
Do ***ACTION*** ***Loop While Condition***	***Do*** ***ACTION*** ***Loop Until Condition***

Dans ces 2 formes, on entre systématiquement dans le corps de la boucle quelque soit l'état de la condition et on répète l'exécution tant que la condition est vérifiée ou jusqu'à ce qu'elle ne le soit plus. Si la condition est fausse, on exécute une fois ACTION est on sort de la boucle.

Exemples :

i = 1 *Sal_Annuel = 0* *Do While i <= 12* *Sal_annuel=Sal_Annuel+Salaire* *i = i +1* *Loop*	*i = 1* *Sal_Annuel = 0* *Do Until i <= 12* *Sal_annuel=Sal_Annuel+Salaire* *i = i +1* *Loop*
i = 1 *Sal_Annuel = 0* *Do* *Sal_annuel=Sal_Annuel+Salaire* *i = i +1* *Loop While i <= 12*	*i = 1* *Sal_Annuel = 0* *Do* *Sal_annuel=Sal_Annuel+Salaire* *i = i +1* *Loop Until i <= 12*

Remarque :

- On peut sortir à tout instant de ces boucles à l'aide de l'instruction *Exit Do*.

2.1.3.3. La boucle For

☝ Le nombre d'itérations peut être connu au départ.

Syntaxe :

```
For Compt = Val_début To Val_fin [Step pas ]
        ACTION
Next [ Compt ]
```

On compte de la valeur *Val_début* à la valeur *Val_fin* avec un pas égal à 1 (par défaut) ou à la valeur *pas* définie par *Step*. A chaque valeur du compteur *Compt*, on exécute une itération de ACTION. La variable de contrôle (ici le compteur) est incrémentée automatiquement et vérifiée qu'elle ne déborde pas la valeur finale. Les clauses entre crochets sont optionnelles.

Exemple :

```
Total_Notes = 0
For i = 1 To 10
      Total_Notes = Total_Notes + Note
Next
Moy = Total_Notes / 10
```

Remarque :

- On peut sortir à tout moment d'une boucle *For* par l'instruction *Exit For*.

3. Les sous programmes

Outre les procédures-événements déjà vues, il est possible d'écrire des procédures / fonctions dites générales - c'est à dire, qu'elles ne sont associées à aucun contrôle ou feuille -, qu'on stockerait dans un fichier feuille ou module.

3.1. Définition d'une Procédure/Fonction

⇒ Pour la procédure :

```
Syntaxe :
[Static] [Private] Sub  nom_Sub (Liste d'arguments)
        ...
        Corps de la procédure
        ...
End Sub
```

⇒ Pour la fonction :

```
Syntaxe :
[Static] [Private] Function  nom_Sub (Liste d'arguments) : As type
        ...
        Corps de la procédure
        ...
End Sub
```

```
Syntaxe :
Liste d'arguments : [ByVal] variable_1 [As type] [,[ByVal] variable_1 [As type]]...
```

3.2. Appel d'une Procédure/Fonction

⇒ Pour la procédure :

```
Syntaxe :
nom_procédure [Arg_1, Arg_2...]
ou
Call nom_procédure[(Arg_1,Arg_2,...)]
```

Remarque :

- La liste d'arguments peut-être vide, auquel cas dans l'appel, il n'y a que le nom de la procédure qui apparaît.

⇒ Pour la fonction :

Syntaxe :
var_1 = nom_fonction [Arg_1,Arg2,...]

Remarque :

- Si la liste d'arguments est vide, les parenthèses sont quand même obligatoires.
- Il est tout à fait possible de quitter la procédure / fonction à tout moment par l'instruction :

Exit Sub	pour la procédure.
nom_F=résultat *Exit function*	pour la fonction, après avoir affecter à la variable nom_F le résultat qu'elle doit retourner.

Les procédures / fonctions définies dans une feuille sont « locales » seulement à celle-ci. Seules les procédures / fonctions de ce fichier peuvent les appeler.
Les procédures / fonctions définies dans un module sont globales à l'ensemble de l'application. Sauf si une procédure / fonction est définie avec *Private.*
VB dispose de fonctions et procédures qui lui sont propres et qu'on peut appeler dans toute application.

4. Les Fichiers

Lorsqu'on veut sauvegarder des données produites par une application ou rentrer des informations déjà existantes : on utilise des fichiers.
VB permet de pratiquer 3 types de fichiers :

- ✓ Les fichiers séquentiels
- ✓ Les fichiers d'accès direct
- ✓ Les fichiers binaires.

4.1. Les fichiers séquentiels

Ce type est utilisé pour les fichiers texte. Chaque ligne est limitée par les symbole CR-LF (Retour Chariot et Fin de Ligne). A chaque accès, on traite la ligne désignée par un pointeur et on passe à la ligne suivante. Les lignes contiennent des caractères alphanumériques.
Comme en algorithmique, un fichier est d'abord *ouvert*, ensuite il est *traité* puis enfin *fermé*.
C'est l'instruction *Open* qui permet d'ouvrir un fichier. Les fichiers séquentiels sont ouverts soit en *Input* : en lecture seule ; soit en *Output* : en écriture à partir du début, le fichier est alors réinitialisé, il perd son contenu s'il en avait ; Soit en *Append* : on rajoute des informations à partir de la fin sans perdre ce qu'il contient déjà.

Syntaxe :
Open "Nom_Fichier" For Input As #1
Open "Nom_Fichier" For Output As #1
Open "Nom_Fichier" For Append As #1

Les noms de fichiers obéissent aux règles du DOS ; La longueur est limitée à 8 caractères maximum et une extension de 3 caractères au maximum.
VB numérote les fichiers ouverts. ***As #1*** affecte le n° 1 au fichier. C'est par ce numéro qu'on désigne le fichier dans les instructions.
La fonction *FreeFile* retourne un numéro disponible pour l'ouverture d'un fichier.

Exemple :

```
num = FreeFile
Open Fich_1.txt For Input As #num
```

L'instruction *Line Input #* permet de lire une ligne d'un fichier séquentiel.

Exemple :

```
Dim Ch_1 As String
Line Input #num, Ch_1        ' On lit dans le fichier num (c.à.d. Fich_1.txt) une ligne qu'on stocke dans la chaîne Ch_1.
```

Input$ sert aussi d'instruction de lecture.
Lorsque des fichiers séquentiels sont ouverts en *Output* ou *Append*, l'instruction d'écriture à utiliser est *Print #* .

Exemple :

```
Dim Ch_2 As String
Ch_2 = '' Cette information est à sauvegarder ''
num = FreeFile
Open Fich_2.txt For Output As #num
Print #num , Ch_2
Close
```

Lorsqu'on finit de travailler avec un fichier il faut le fermer avec l'instruction *Close*.

Syntaxe :
Close [#] [num]

Exemple :

```
Close
```

Ou

```
Close #num
```

Ou

```
Close num
```

4.2. Les fichiers d'accès direct

Ceux sont des fichiers d'enregistrements. On peut les ouvrir on écriture et en lecture à l'aide de l'instruction *Open*. Ils sont par conséquent en mode *Random*. On doit préciser la longueur des enregistrements.

Syntaxe :
Open ''nom_fichier'' For Random As [#] num Len =Len(enregistrement)

On peut écrire ou lire n'importe quel enregistrement qu'on accède directement à l'aide de l'instruction *Seek*.

Syntaxe :
Seek [#] num, num_enregistrement

Exemple :

```
Seek num, num_erg
```

positionne le pointeur dans le fichier numéro *num* à l'enregistrement *num_erg*.
Les instruction *Get* et *Put* permettent respectivement la lecture et l'écriture de ce type de fichiers.

Exemple :

```
Type Etudiant
        Numéro As intger
        NomPrénom As  String * 30
        Discipline As  String * 25
        Année_Inscrip  As  Integer
End Type
```

L'enregistrement Etudiant doit être déclaré dans un module.

```
Dim   E_1  As  Etudiant
Dim   E_1  As  Etudiant

Open ''Fic_Etud ''  For random  As 1  Len =Len(E_1)
...
E1.Numéro = 134561 ;
E1.NomPRénom = Dupont Lionel ;
E1.Discipline = Sciences économiques ;
E1.Année_Inscrip = 1996
...
Put  1, E_1
...
Get 1, E_2
...
```

4.3. Les fichiers binaires

Les fichiers binaires n'ont pas de structure propre. Ils contiennent des suites d'octets. Ils sont utiles lorsque les autres types ne conviennent pas au problème traité. Ils sont à accès direct et s'ouvrent à l'aide de *Open*.

Syntaxe :
Open ''nom_fichier'' For Binary As [#] num

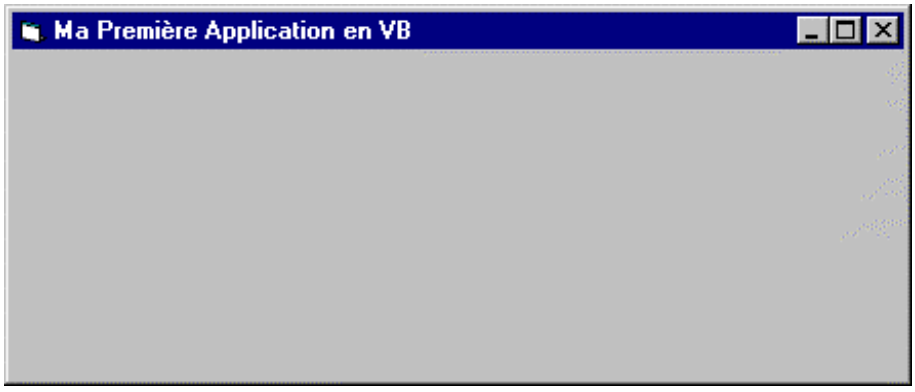
On lit et on écrit à l'aide des instructions respectives *Get* et *Put*. Ils sont ouverts en lecture et en écriture.

5. Eléments visuels

5.1. Les feuilles

5.1.1. La feuille

Une feuille est le cadre dans lequel tous les autres éléments d'une application VB sont placés : contrôles (boutons de commandes, zones de texte, listes, ...), affichages de texte ou de dessin d'images ou de figures géométriques.
La feuille, tout comme les contrôles, dispose de propriétés et d'événements. Les propriétés peuvent être modifiées lors de la conception de la forme visuelle. Les événements peuvent servir pour modifier la forme de la feuille ou contenir des instructions traduisant une partie de la logique du programme.



- Fig.7 : Feuille de travail..

Les éléments systèmes d'une feuille sont gérés par des propriétés. Ils comprennent:

- Titre (***Caption***)
- Menu système (***ControlBox***)
- Case de réduction en icône (***MinButton***)
- Case d'extension plein écran (***MaxButton***)
- Bordure (***BorderStyle***)

La propriété ***Name*** permet d'identifier la feuille. Les propriétés ***Left, Top, Width et Height*** permettent de positionner ou de dimensionner une feuille. La propriété ***WindowState*** détermine l'état de la fenêtre lors de son affichage. Elle peut prendre l'une des 3 valeurs :

0 : Normal (par défaut)
1 : Réduit en icône.
2 : Agrandit en plein écran.

Lors du lancement de VB, une feuille est créée par défaut. Elle sera considérée comme feuille principale de l'application. Il est possible de rajouter d'autres feuilles en utilisant la commande ***Nouvelle feuille*** du menu ***Fichier***. L'application sera alors multifeuille. Les différents contrôles seront alors répartis sur les différentes feuilles. Il

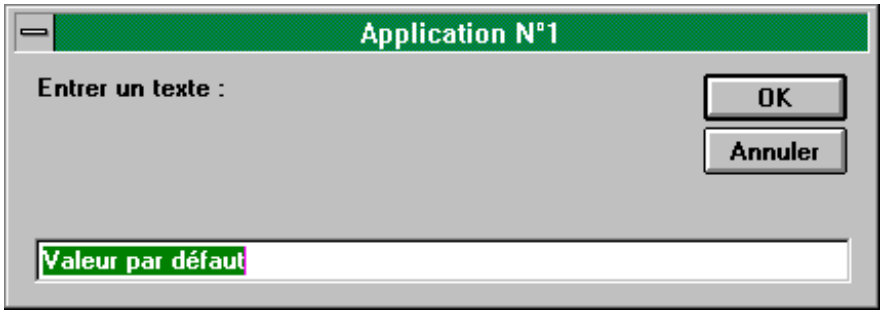
faudrait cependant indiquer à VB la feuille de démarrage (c'est à dire celle qui sera affichée lors de l'exécution du programme) dans l'onglet ***Projet*** de la commande ***Options*** du menu ***Outils***.
Il existe des méthodes (***Show, Hide,...***) et des instructions (***Load***, ***Unload***,...) permettant de manipuler les feuilles d'une application.
Des événements (***Load***, ***Resize***, ***Activate***, ***Paint***,...) peuvent servir pour exprimer la logique du programme. ***Load*** est souvent utilisé pour initialiser des variables dans le programme.

5.1.2. Les fenêtres prédéfinies

Il est possible de faire appel à des fenêtres prédéfinies pour saisir ou afficher du texte.

a) Saisie de texte :
C'est la fonction ***InputBox*** qui affiche une boîte de saisie et retourne une chaîne de caractère :

Syntaxe :
Var1 = InputBox(*"Entrer un texte : "* **,** *"Application N° 1"* **,** *"Valeur par défaut"****)***



- Fig.8 : Exemple d'une boîte de saisie.

La valeur retournée par ***InputBox*** dans Var1 est de type Variant.

b) Affichage de message :
C'est la fonction ***MsgBox*** qui affiche une boîte avec un texte comme message, un ou plusieurs boutons de commande et éventuellement une icône.

Syntaxe :
Var2 = MsgBox (*"Voulez-vous arrêter le programme"* **, MB_OKCANCEL ,**
*"Application N° 2"****)***

- Fig.9 : Exemple d'une boîte de message.

L'argument ***MB_OKCANCEL*** permet d'afficher le bouton de commande **OK**.
On peut se servir de cet outil pour afficher le résultat d'un programme.

5.2. Les contrôles

Les contrôles sont des objets d'interaction grâce auxquels l'utilisateur construit un dialogue. Ce dernier s'articule sur des affichages et / ou des saisies de données, des ordres de calculs... Comme pour les feuilles, des propriétés et des événements sont associés aux contrôles. La valorisation des propriétés peut se faire statiquement lors de la conception de la forme visuelle ou dynamiquement dans les procédures.
Il existe différents types de contrôles :

- ☞ de texte pour l'affichage ou la saisie de données : zone de texte, étiquette.
- ☞ les boutons de commande pour le déclenchement d'actions, de radio pour les choix d'options, les cases à cocher pour les réponses oui / non.
- ☞ les listes, sous différentes formes : simples, modifiables (ou combinées), déroulantes, d'unités de disques, de répertoires, de fichiers.
- ☞ les barres de défilement horizontales et verticales.
- ☞ de dessin : image ou forme et ligne.
- ☞ le cadre pour regrouper d'autres contrôles.
- ☞ de données pour l'accès aux bases de données.
- ☞ personnalisés : grille, boîte de dialogue, OLE....

En général, l'ensemble des ces contrôles sont présents dans la boîte à outils.

Remarques :

- l'ensemble de ces contrôles ont quelques propriétés communes, telles que :
- ***Name*** : pour définir leur identificateur ;
- ***BackColor*** et ***ForeColor*** : pour déterminer la couleur de fond et de caractère ou de dessin.;
- ***Caption*** ou ***Text*** : pour définir le titre ou le contenu pour la zone de texte;
- ***Visible*** : pour rendre le contrôle visible ou masqué (*True* ou *False*).

5.2.1. La zone de texte

La zone de texte est un champ de saisie. Elle peut être monoligne (maximum 2048 caractères), la propriété ***MultiLine*** est à *False* , ou multiligne (maximum 32 Ko), ***MultiLine*** est alors à *True*. Dans ce dernier cas, on peut rajouter des barres de défilement (propriété ***ScrollBars***). La donnée saisie dans une zone de texte est de type *String* et est contenue dans la propriété ***Text***. Cependant on peut la récupérer et la convertir si besoin est.

Exemple :

```
ZT1.Text = ''1234''
Var1%= Val(ZT1.Text)
```

La fonction **Val** convertit un texte en nombre. Ici la variable entière *Var1* contiendra le nombre entier 1234.
Si on veut afficher le nombre 4567 dans une zone de texte ZT2, on écrira :

Exemple :

```
ZT2.Text = Str(4567)
```

La fonction **Str** convertit le nombre 4567 en la chaîne de caractère ''4567''.
Pour effacer le contenu d'une zone de texte, il suffit d'affecter le caractère '''' à la propriété ***Text***.

La propriété ***MaxLength*** permet de limiter le nombre de caractères pouvant être saisi. Alors que ***PasswordChar*** masque les caractères saisis à l'aide d'un caractère affecté à cette propriété.
La sélection est la partie choisie du texte affiché dans une zone de texte. On peut indiquer le début de la sélection dans la chaîne par ***SelStart*** , savoir sa longueur par ***SelLength*** et son contenu par ***SelText***.
Dès qu'on clique dans une zone de texte, le focus (ce qui indique qu'un contrôle est actif) est situé sur celle-ci. Un événement ***GotFocus*** est alors généré. Lorsqu'on quitte une zone de texte, l'événement ***LostFocus*** est alors produit. La méthode ***SetFocus*** permet de maintenir le focus sur un contrôle.

Exemple :

```
ZT1.SetFocus
```

Le focus est positionné dans la zone de texte *ZT1*.
Dès l'appui sur une touche l'événement ***KeyPress*** est généré suivi par ***Change*** lorsque ce premier caractère est traité.
A ces événements peuvent être associés des actions que l'on programme dans les procédures-événement correspondantes.

5.2.2. L'étiquette

C'est un contrôle qui permet d'afficher un texte. Il accompagne généralement la zone de texte pour préciser son utilisation. La propriété ***Caption*** contiendra le texte à afficher.

- Fig.10 : Zone de texte associée à une étiquette..

La méthode ***Print*** permet également d'afficher un texte sur une feuille. Il faut préciser son emplacement, définir son format etc.

Exemple

Form1.Print ''Texte du message à afficher : ''

5.2.3. Le bouton de commande

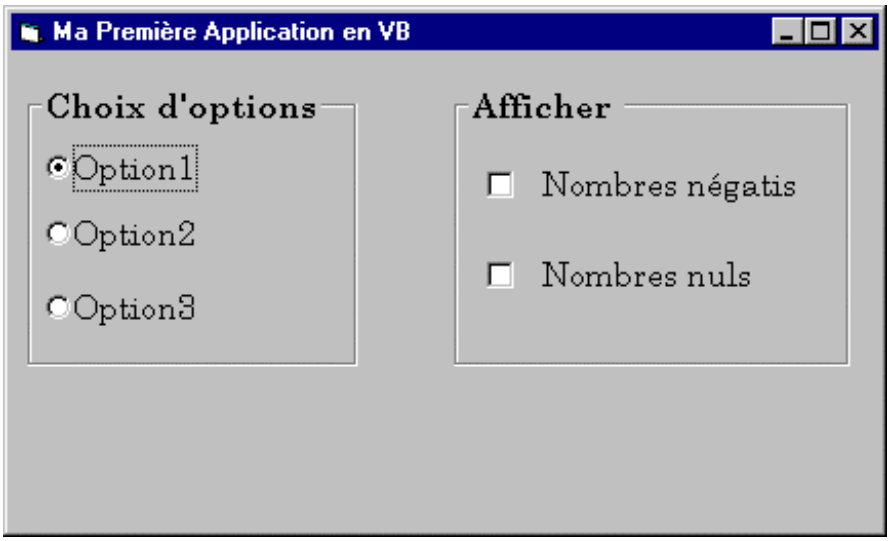
Le bouton de commande est utilisé pour signifier une commande d'utilisateur pour déclencher une action à exécuter. Lorsqu'on clique sur un bouton de commande un événement ***Click*** est généré. C'est dans la procédure-événement relative à celui-ci qu'est écrit le programme correspondant à l'action à exécuter.
Dans la propriété ***Caption,*** on définit le texte à afficher sur le bouton. Il est possible de modifier la police et la taille des caractères, mais pas la couleur ni celle du fond, qui sont fixées par *Windows* (voir le fichier *Win.ini*).

- Fig.11 : Zone de texte associée à une étiquette et bouton de commande.

On peut également appliquer la méthode ***SetFocus*** pour sélectionner le bouton. La propriété ***Default***, si elle est à *True* , indique lors de l'exécution que le bouton a le focus. L'appui sur la touche *Entrée* génère l'événement ***Click***. De même la propriété ***Cancel***, si elle est à *True*, l'appui sur la touche *Echap* génère l'événement ***Click.***

5.2.4. Le cadre

Le cadre est utilisé pour regrouper d'autres contrôles : des cases à options (boutons radio) ou à cocher.

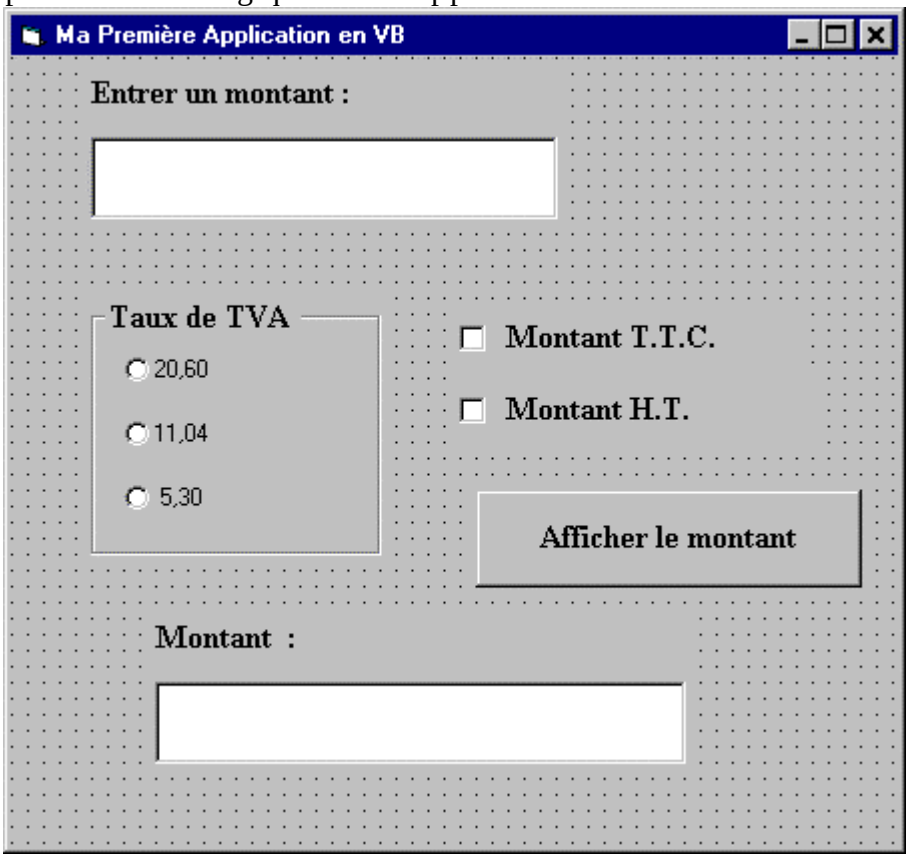


- Fig.12 : Exemple de cadre regroupant des boutons d'option et des cases à cocher.

5.2.5. Le bouton d'option

Ce contrôle permet de choisir une option parmi plusieurs. Plusieurs boutons doivent être définis pour matérialiser le choix. Ils doivent être placés dans un contenant qui peut être soit une feuille, soit une image soit alors un cadre. Deux propriétés sont essentiels : ***Caption*** pour libeller l'option et ***Value*** qui prend la valeur *False* ou la valeur *True.*

Dans une série de boutons d'option, seul un bouton peut être sélectionné (***Value*** à *True*), dès lors les autres ont leur propriété ***Value*** mise à *False* par VB. A chaque changement de ***Value***, un événement ***Click*** est produit et pourra être exploité immédiatement ; ou alors l'état de chaque bouton sera examiné dans une autre procédure pour traiter la logique d'une application.



- Fig.13 : Exemple de boutons d'option et des case à cocher.

Différents boutons d'option peuvent être définis dans un groupe (*Copier* et *Coller* du premier bouton d'option dans le même contenant) et porter ainsi le même nom. C'est par la propriété ***Index*** qu'on les distinguera. Cette méthode permet de tester l'état de chaque bouton dans une instruction répétitive.

5.2.6. Les cases à cocher

Contrairement au contrôle précédent, on peut effectuer plus d'un choix à l'aide des cases à cocher. Celles-ci peuvent être dans un état *coché* ou *non coché* et parfois même dans un troisième cas (*coché grisé*) pour indiquer qu'on ne peut pas déterminer l'état de la case. Un simple click avec la souris permet de passer d'un état à un autre. Le changement d'état d'une case génère un événement click pouvant être exploité immédiatement ou traité dans une autre procédure conformément à la logique d'une application. Les propriétés ***Value***, ***Caption***, ***Index*** et ***Name*** sont les plus utilisés.

5.2.7. Les listes

Une liste permet de présenter plusieurs éléments et le choix d'un d'entre eux. Il existe plusieurs types de listes :

- les listes simples
- les listes combinées
- les listes de fichiers
- les listes de répertoires
- les listes d'unités de lecteurs

5.2.7.1. La liste simple

La liste simple contient une suite d'items (éléments de la listes : chaînes de caractères). La liste peut être initialisée en remplissant la propriété ***List*** par une suite de chaînes de caractères qui seront les items initiaux de la liste. Ces derniers peuvent être triés en mettant ***Sorted*** à *True*. Le premier item de la liste a le rang (ou indice) 0 . Le nombre d'items d'une liste est déterminé par la propriété ***ListCount***. La propriété ***ListIndex*** contient l'indice de l'item sélectionné. La méthode ***AddItem*** rajoute un item à la liste.

Exemple

L1.AddItem ''élém 8''

L'item *élém 8* est rajouté à la liste *L1*.

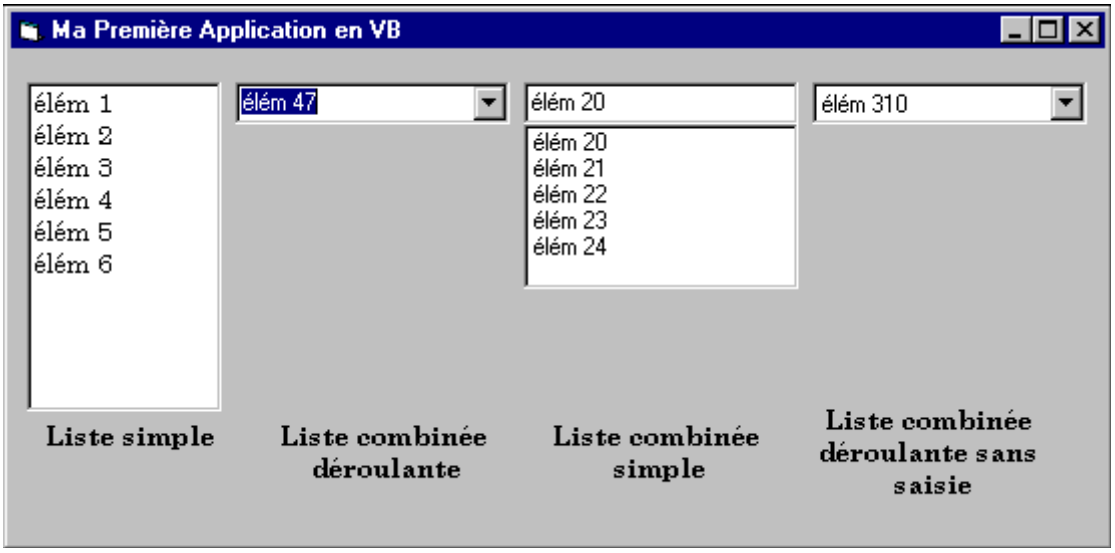
Pour enlever un item d'une liste on indique son indice à la méthode ***RemoveItem***.

Exemple

L3.RemoveItem 3

L'item *élém 23* est supprimé de la liste *L3*.

La méthode ***Clear*** permet d'effacer tous les éléments de la liste.
Exemples de listes simple et combinées :



- Fig.14 : Les différents types de liste.

5.2.7.2. La liste combinée

Une liste modifiable (ou combinée) est une combinaison de liste et d'une zone de texte. Il existe 3 sortes de listes combinées qu'on définit à l'aide de la propriété ***Style*** :

- Liste combinée déroulante (***Style = 0***) : Comprend une liste déroulante et une zone de texte. L'utilisateur peut sélectionner un item dans la liste ou taper ce qui convient dans la zone de texte.
- Liste combinée simple ***(Style = 1)*** : Comprend une zone de texte et une liste non déroulante. L'utilisateur peut sélectionner un item de la liste ou taper ce qui convient dans la zone de texte. Augmentez la valeur de la propriété ***Height*** pour afficher une plus grande partie de la liste.
- Liste combinée déroulante sans saisie (***Style = 2***) : Ce style permet seulement à l'utilisateur de sélectionner une option dans la liste déroulante, sans pouvoir saisir de nouveaux items dans la liste.

Les mêmes propriétés et méthodes permettent, en général, de manipuler les listes simples et combinées. Les événements ***Change, Click, DblClick***, etc, peuvent être utilisés pour appliquer des actions aux listes.

5.2.8. La grille

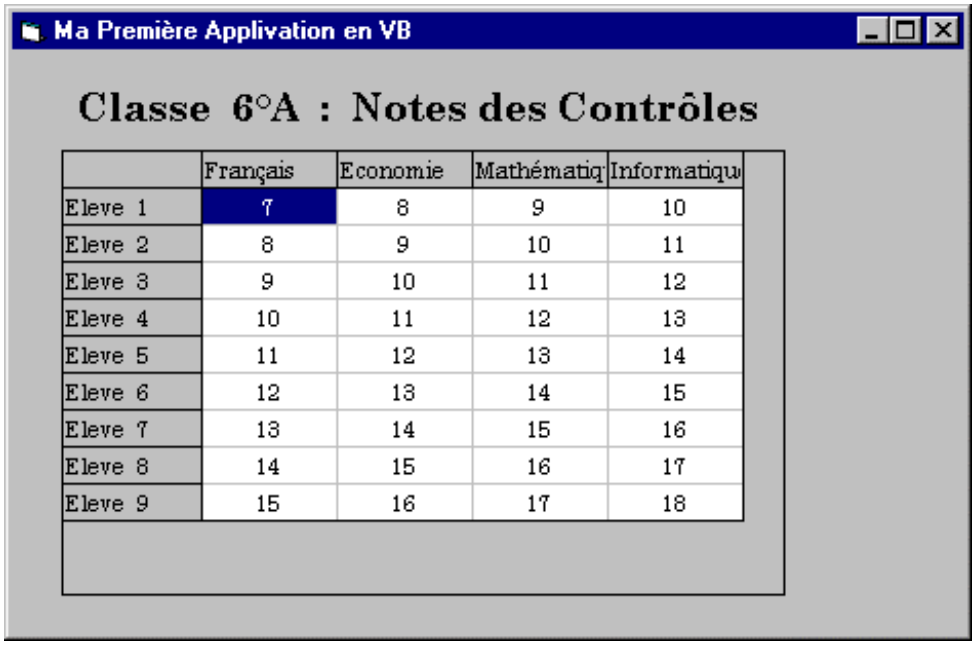
C'est le contrôle qui permet d'afficher un tableau à 2 dimensions. Les propriétés ***Cols*** et ***Rows*** définissent le nombre de colonnes et de lignes. Il est possible de fixer des colonnes et des lignes est précisant la quantité dans les propriétés ***FixedCols*** et ***FixedRows***. Les largeurs de colonnes et les hauteurs de lignes sont stockées dans des

tableaux représentés respectivement par les propriétés ***ColWidth*** et ***RowHeight*** . les coordonnées de la cellule active sont à définir dans les propriétés ***Col*** et ***Row***.

La procédure suivante remplit le tableau de la figure 14.

```
Private Sub Form_Load( )
    Dim i
    Dim j

    G.Row = 0
    G.Col = 1
    For i = 1 To G.Cols - 1
      G.ColAlignment(i) = 2
      G.ColWidth(i) = 980
    Next
    G.Text = "Français"
    G.Col = 2
    G.Text = "Economie"
    G.Col = 3
    G.Text =
    "Mathématiques"
    G.Col = 4
    G.Text = "Informatique"
    G.Col = 0
    G.ColWidth(0) = 1000

    For i = 1 To G.Rows - 1
        G.Row = i
        G.Text = "Eleve " + Str(i)
    Next i

    For i = 1 To G.Cols - 1
        G.Col = i
    For j = 1 To G.Rows - 1
        G.Row = j
        G.Text = i + 5 + j
    Next j
    Next i
    G.Col = 2
    G.Row = 2

    End Sub
```

Les propriétés ***Text*** et ***Picture*** permettent de lire ou modifier le contenu de chaque cellule du tableau. ***Clip*** permet de lire ou modifier le contenu des cellules sélectionnées. Les propriétés ***SalStartCol***, ***SelStartRow, SelEndCol, SelEndRow*** et ***SelChange*** traitent la sélection de cellules.
L'alignement des données dans une colonne se fait à'aide de la propriété ***ColAlignment.***



- Fig.15 : Tableau représenté par une grille.