# [FICHES DE REVISION]

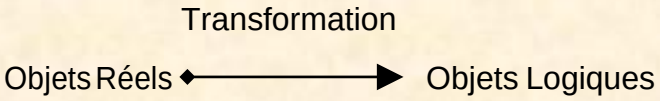
**Ghanmi Alaa eddine**

# Table des matières

# Démarche de résolution

## INTRODUCTION :

La tâche la plus importante dans l'informatisation d'une application ou d'un problème réel donné est la phase de transformation des objets réels en objets logiques, qui peuvent être utilisées et exploités de façon optimale par l'ordinateur.




Cette transformation nécessite trois étapes :

→ La définition exacte du problème.

→ La formalisation du problème.

→ La programmation.

## LES ETAPES DE RESOLUTION D'UN PROBLEME :

### a) La définition du problème (pré analyse) :

La tâche se présente en effet le plus souvent de manière floue. Il est donc nécessaire dans un premier temps d'affiner sa description. Il est indispensable de définir précisément les données qu'on dispose, les objets qu'on souhaite atteindre et éventuellement les limitations de ces objectifs.

D'une façon générale, la définition d'un problème consiste à prévoir des réponses à tous les cas envisageables.

### b) L 'analyse du problème :

Il existe plusieurs approches pour analyser un problème comme l'analyse **descendante** et **ascendante**.

### c) Elaboration d'un algorithme :

Cette phase appelée phase algorithmique, assure la réécriture dans l'ordre d'une suite finie d'actions. Un **algorithme** est une suite structurée et finie d'actions ou d'instructions permettant de résoudre un problème donné.

### d) La programmation :

C'est le passage du modèle logique qui est l'algorithme au modèle directement exploitable par l'ordinateur. Pour cela, il faut traduire l'algorithme en un programme écrit dans un langage de programmation choisi par l'utilisateur. Le langage de programmation dépendra de la nature du problème ou de l'application à automatiser.

## ANALYSE MODULAIRE :

**a) Définition :**

Cette approche d'analyse consiste à diviser un problème en sous problème de moins de difficultés qui peuvent être aussi décomposé, jusqu'on arrive à un niveau de difficulté moindre.

**b) Intérêt :**

→ Il est plus efficace de séparer les difficultés et les tâches

→ Simplifier la correction du programme

→ L'évolution du programme

## LES ETAPES DE PASSAGE DE L'ALGORITHME AU PROGRAMME :

**a) Ecriture avec un éditeur :**

Cette étape consiste à traduire l'algorithme dans un langage de programmation à l'aide de son éditeur de texte.

**b)** Après avoir écrit notre programme, on fait recoure à l'une des deux étapes suivantes :

- **Interprétation :**

Consiste à faire la traduction au langage machine instruction par instruction jusqu'à la fin du programme (interprète et exécute). S'il y a une erreur syntaxique ou sémantique l'interprétation s'arrête. Il n'y a pas production de programme objet (exécutable).

- **Compilation :**

Consiste dans la production de programme exécutable objet ; à fin de passer à l'étape d'exécution il faut que la syntaxe du programme soit correcte. Le programme d'origine s'appelle **programme source** et le programme compilé est dit **exécutable**.

**Remarque :** Tout langage est muni d'un traducteur en langage machine (interpréteur ou compilateur).

**c) Exécution et tests :**

Une fois le programme est écrit, on passe à l'exécution. Il est toujours utile de le tester avec un jeu d'essai.

## EXEMPLES DE LANGAGES DE PROGRAMMATION :

Il y'a plusieurs langage de programmation exemple : PASCAL, JAVA, APL, FORTRAN, LISP, ALGOL, COBOL, BASIC, PROLOG, PERL, LUA, C, C++, PHP…

# Structures de données

Les **identificateurs** servent à désigner les différentes entités manipulées par le programme telles les constantes, variables, fonctions…, ils sont formés d'une suite de caractères choisis parmi les lettres ou les chiffres, le premier d'entre eux étant nécessairement une lettre.

## LES CONSTANTES :

### a) Définition :

On appelle constante un objet ayant une valeur inchangée tout le long d'un algorithme. Elle est caractérisée par :

→ Son nom (un identificateur unique).
→ Sa valeur.

**Exemple:** NomConstante = 1.17

Le compilateur attribue automatiquement à la constante le type de base le plus proche.

### b) Déclaration :

| ALGORITHME | PASCAL |
|---|---|
| **Tableau de déclaration des objets**<br>**Objet** \| **Type/Nature**<br>NomConstante \| Constante = 1.17 | **CONST** NomConstante **=** 1.17 **;**<br>**TYPE** …… **;**<br>**VAR** …… **;** |

Au niveau du programme **Pascal**, de préférable sa déclaration précède la déclaration des variables.

## LES VARIABLES :

### a) Définition :

On appelle variable tout objet pouvant prendre différents valeurs tout le long d'un algorithme. Elle est caractérisée par :

→ Son nom (un identificateur unique).
→ Son type.
→ Sa valeur.

### b) Déclaration :

| ALGORITHME | PASCAL |
|---|---|
| **Tableau de déclaration des objets**<br>**Objet** \| **Type/Nature**<br>NomVariable \| …… | **CONST** …… **;**<br>**TYPE** …… **;**<br>**VAR** NomVariable **:** Type_variable **;** |

Au niveau du programme **Pascal**, de préférable sa déclaration succède la déclaration des constantes.

## LES OPERATEURS :

**a) Opérateurs relationnels** : < , > , ≤ , ≥ , ≠ , = , DANS. (≠, DANS en Turbo Pascal<>, IN)

**b) Opérateurs arithmétiques** : + , * , - , / , DIV , MOD. (DIV /MOD : quotient/reste)

**c) Opérateurs logiques** :

| ALGORITHME | PASCAL | Signification |
|---|---|---|
| NON | NOT | Négation |
| OU | OR | Disjonction |
| ET | AND | Conjonction |
| OUex | XOR | Ou exclusif |

**Exemples :**

**NOT** False = True

False **OR** False = False

True **AND** True = True

True **XOR** False = True

**d) Priorité des opérateurs :** NON * / MOD DIV ET + - OU OUex = < > ≤ ≥ ≠ DANS

1 2 3 4

**En cas de priorités identiques (1 ou 2 ou 3 ou 4), les calculs s'effectuent de gauche à droite. Les parenthèses permettent d'outrepasser ces règles de priorité, en forçant le calcul préalable de l'expression qu'elles contiennent.**
**Les opérateurs arithmétiques unaires + et – (opposé) sont de priorités 1.**

## LES TYPES DE DONNEES :

Le type permet de définir la nature des données (valeurs) qui peuvent être affectées à la variable en question, ainsi que l'ensemble des opérateurs valides sur cette variable. Les plus connus sont:

**a) Le type BOOLEEN :** Ce type contient les deux valeurs logique **VRAI** (TRUE) et **FAUX** (FALSE). **(VRAI > FAUX)**

**Type Scalaire**

| ALGORITHME | PASCAL |
|---|---|
| **Tableau de déclaration des objets**<br>Objet : V<br>Type/Nature : Booléen | **VAR** V **: BOOLEAN ;** |

On obtient un résultat de type booléen quand on est amené à comparer des expressions entre elles, au moyen des **opérateurs relationnels** et des **opérateurs logiques**.

**b) Le type ENTIER :** Désigne les valeurs des nombres entiers relatifs.

**Type Scalaire**

| ALGORITHME | PASCAL |
|---|---|
| **Tableau de déclaration des objets**<br>Objet : V<br>Type/Nature : Entier | **VAR** V **: INTEGER ;** |

Turbo pascal fournit cinq types entiers prédéfinis. Chacun d'eux concerne un sous ensemble particulier des nombres entiers : Integer, LongInt et ShortInt (entier, entier long et entier court) : Entier **signé**.
Word et Byte (mot et octet) : Entier **non signé**.

**c) Le type REEL (FLOTTANT) :** Désigne les valeurs des nombres réels.

| ALGORITHME | PASCAL |
|---|---|
| **Tableau de déclaration des objets**<br><br>**Objet** \| **Type/Nature**<br>V \| Réel | **VAR** V **: REAL ;** |

Turbo pascal fournit cinq types réels prédéfinis. Chacun d'eux a un domaine de valeurs et une précision spécifiques : Real, single, double, extended et comp. (réel, réel simple, réel double, réel étendu et réel compatible)

**d) Le type CARACTERE :**

Désigne tous les caractères alphanumériques imprimables de l'alphabet latin ainsi que les caractères spéciaux non imprimables ayant des significations particulières tel que : le retour chariot, l'Echappe (Escape), le Bip sonore, etc. Tous ces caractères sont ordonnés selon leur code ASCII.

Type Scalaire

| ALGORITHME | PASCAL |
|---|---|
| **Tableau de déclaration des objets**<br><br>**Objet** \| **Type/Nature**<br>V \| Caractère | **VAR** V **: CHAR ;** |

**e) Le type CHAINE :**

Une chaîne de caractère est une succession de n caractères avec n compris entre 0 et 255, c'une suite d'octets correspondant à chacun de ses caractères.
Les variables chaînes de caractères sont définies par une déclaration pouvant indiquer le nombre maximal de ces caractères.

Le type chaîne est en fait un **tableau de caractères** (à une dimension) de taille la longueur de cette chaîne (maximum 255), le premier caractère de la chaîne a pour indice 1.

| ALGORITHME | PASCAL |
|---|---|
| **Tableau de déclaration des objets**<br><br>**Objet** \| **Type/Nature**<br>AB \| CHAINE [15]<br>XY \| CHAINE | **VAR** AB **: STRING [** 15 **] ;**<br><br>XY **: STRING ;** |

**AB** : Désigne une chaîne de caractère contenant jusqu'à 15 caractères.
**XY** : Désigne un texte.

**f) Le type TABLEAU :**

- **Tableau à une dimension :**

Un tableau de dimension **1** appelé aussi vecteur, est déclaré comme un **type particulier** de données. Un tableau permet de ranger un nombre fini d'éléments de même type dans une structure fixe. L'accès à chaque élément sera par le biais d'un indice. Un tableau se compose d'au moins de deux éléments de même type. Un tableau possède un **nom** (identificateur) et un **type** fixe, mais il est associé à un ensemble de valeurs. Chaque valeur est repérée par le **nom** du tableau suivi d'un **indice** (dont la valeur appartient à l'intervalle des indices) qui peut être de type scalaire.

La taille (dimension) du tableau doit être une constante ou une expression constante.

| ALGORITHME | PASCAL |
|---|---|
| **Tableau de déclaration des nouveaux types**<br>**Type**<br>Vecteur **= Tableau** de **taille** N<br>de **type** Type_éléments | **TYPE** Vecteur **= ARRAY [** INP **..** IND **] OF** Type_éléments **;**<br>**VAR** V **:** Vecteur ; |

**[ INP .. IND ]** : Désigne l'intervalle des indices.

**IND – INP + 1** : Désigne la **Taille** du tableau (**N**).

**Type_éléments** : Désigne le type des éléments du tableau.

***Tableau de déclaration des objets***

| Objet | Type/Nature |
|---|---|
| V | Vecteur |

- **Tableau à deux dimensions :**

Lorsqu'un traitement utilise plusieurs tableaux à une dimension, ayant le même nombre d'éléments et subissant le même traitement, on utilise un seul tableau à **deux dimensions (un tableau de tableaux)**. Cette nouvelle forme de tableau possède un **identifiant** unique. Chaque élément est identifié par **deux indices**: l'un indiquant la **ligne** et l'autre la **colonne.**

| ALGORITHME | PASCAL |
|---|---|
| **Tableau de déclaration des nouveaux types**<br>**Type**<br>Vecteur **= Tableau** de NL **lignes**<br>de NC **colonnes**<br>de **Type** Type_éléments | **TYPE** Vecteur **= ARRAY [** INPL **..** INDL **,** INPC **..** INDC **]**<br>**OF** Type_éléments **;**<br>**VAR** V **:** Vecteur ; |

**INDL – INPL + 1** : Désigne le **nombre de lignes** du tableau (**NL**).

**INDC – INPC + 1** : Désigne le **nombre de colonnes** du tableau (**NC**).

**Type_éléments** : Désigne le type des éléments du tableau.

**g) Les types définis par l'utilisateur :**

Les types simples comme entier, réel, caractère, booléen, chaîne de caractères et tableau ; sont des types prédéfinis d'un langage de programmation.

Tous les langages de programmation offrent à l'utilisateur la possibilité de définir des nouveaux types de données plus sophistiqués.

**Exemple :**

- **Le type Enuméré :** liste de valeur donnée par l'utilisateur, exemple :

**Type Scalaire**

**ALGORITHME**

**Tableau de déclaration des nouveaux types**

| Type |
|---|
| Element = ( e1 , e2 , e3 ) |

**Tableau de déclaration des objets**

| Objet | Type/Nature |
|---|---|
| V | Element |

**PASCAL**

```
TYPE  Element = (e1,e2,e3);
VAR   V : Element ;
```

Une variable énumérée ne peut être lue ou écrite par les instructions READ ou WRITE.

- **Le type Intervalle :** valeurs scalaires comprises entre deux bornes, exemple :

**Type Scalaire**

**ALGORITHME**

**Tableau de déclaration des nouveaux types**

| Type |
|---|
| Element = "A" .. "Z" |

**Tableau de déclaration des objets**

| Objet | Type/Nature |
|---|---|
| V | Element |

**PASCAL**

```
TYPE  Element = 'A' .. 'Z' ;
VAR   V : Element ;
```

Chaque variable définie correspond bien à un sous-type d'un type scalaire (classique ou énuméré défini), mais ne peut en aucun cas contenir une valeur dépassant les bornes de l'intervalle.

# Structures simples

Les échanges d'informations entre l'utilisateur et la machine sont appelées opérations d'entrée-sortie.

- **LA LECTURE DE DONNEES :** Entrée d'information (une lecture) en règle générale, au clavier, afin de l'affecter à une variable.

| ALGORITHME | PASCAL |
|---|---|
| **Lire (** V **)** | **Read (** V **) ;**<br>ou<br>**Readln (** V **) ;** |

Avec **V** un identificateur de variable.

- **L'ECRITURE DE DONNEES :** Opération de sortie ; on parle d'écriture, même si le résultat apparait sur l'écran.

| ALGORITHME | PASCAL |
|---|---|
| **Ecrire (** V **)** | **Write (** V **) ;**<br>ou<br>**Writeln (** V **) ;** |
| **Ecrire ("** message **")** | **Write ('** message **') ;** |
| **Ecrire ("** message **",** V**)** | **Write ('** message **',** V**) ;** |

Avec **V** une expression ou un identificateur de variable.

**Remarque :**

En Pascal pour E entier, R réel et C chaine (ou caractère) :

- **Write (E : n) / Write (C : n)** ⇨ affiche la valeur entière/chaine sur n positions (largeur minimale=n caractères) avec insertion d'espacement à gauche pour atteindre cette largeur. S'il faut plus de n caractères pour afficher cette valeur, alors la largeur utilisée sera augmentée en conséquence.
- **Write (R : n)** ⇨ affiche le nombre en notation scientifique sur n positions précédé d'un espacement.
- **Write (R : n : d)** ⇨ affiche le nombre sans espace avant sur n positions avec d décimales.

- **L'AFFECTATION :** Il s'agit d'une affectation du résultat d'une expression à une variable. La partie droite d'une expression d'affectation est toujours calculée avant que l'affectation n'ait lieu. Il faut vérifier que les opérandes sont du même type que ce que vous souhaitez obtenir.

| ALGORITHME | PASCAL |
|---|---|
| V1 ← V2 | V1 **:=** V2 **;** |

Avec **V1** un identificateur de variable.
Et **V2** un identificateur de variable ou expression de même type ou compatible à la variable **V1**.

# Structures conditionnelles

**« Condition » :** expression booléenne quelconque c'est-à-dire condition quelconque ou variable de type booléen.
**« Instructions_0_1_2…N »** désignes toujours n'importe quelles instructions (structures) simples, structurées (choix, boucles) ou un bloc.

- **STRUCTURE CONDITIONNELLE SIMPLE (forme réduite) :** Exécuter une instruction ou un bloc d'instructions si une condition est vérifiée.

| ALGORITHME | PASCAL |
|---|---|
| **SI** Condition<br>**ALORS**<br>Instructions<br>**FINSI** | **IF** Condition<br>**THEN**<br>**BEGIN**<br>Instructions **;**<br>**END ;** |

- **STRUCTURE CONDITIONNELLE SIMPLE (forme alternative) :** Réaliser un choix parmi deux possibilités.

| ALGORITHME | PASCAL |
|---|---|
| **SI** Condition<br>**ALORS**<br>Instructions1<br>**SINON**<br>Instructions2<br>**FINSI** | **IF** Condition<br>**THEN**<br>**BEGIN**<br>Instructions1 **;**<br>**END**<br>**ELSE**<br>**BEGIN**<br>Instructions2 **;**<br>**END ;** |

Un **ELSE** se rapporte toujours au dernier IF rencontré auquel un ELSE n'a pas encore été attribué.
Le **end** terminant le then n'est pas suivi d'un point virgule.

- **STRUCTURE CONDITIONNELLE À CHOIX MULTIPLES :** Comparer un objet à toute une série de valeurs.

| ALGORITHME | PASCAL |
|---|---|
| **SELON** V **FAIRE**<br>Valeur 1 **:** Instructions1<br>Valeur 2 **:** Instructions2<br>…<br>Valeur N **:** InstructionsN<br>**SINON** Instructions0<br>**FINSELON** | **CASE** V **OF**<br>Valeur 1 **:** Instructions1 **;**<br>Valeur 2 **:** Instructions2 **;**<br>…<br>Valeur N **:** InstructionsN **;**<br>**ELSE** Instructions0 **;**<br>**END ;** |

**V** est une variable ou expression de type scalaire.
**Valeur_1_2…N** peuvent se présenter sous forme de : une Valeur, Liste de valeurs ou Intervalle.

# Structures itératives

Une itération consiste à exécuter un bloc d'instructions un certain nombre de fois.

- **LA BOUCLE REPETER :** On répète l'exécution du bloc d'instructions tant que Condition = **FAUX**.

| ALGORITHME | PASCAL |
|---|---|
| **REPETER**<br>\| Instructions<br>**JUSQU'A** Condition | **REPEAT**<br>Instructions ;<br>**UNTIL** Condition ; |

- **LA BOUCLE TANT QUE :** On répète l'exécution du bloc d'instructions tant que Condition = **VRAI**.

| ALGORITHME | PASCAL |
|---|---|
| **TANT QUE** Condition **FAIRE**<br>\| Instructions<br>**FINTANTOUE** | **WHILE** Condition **DO**<br>BEGIN<br>Instructions ;<br>END; |

- **LA BOUCLE POUR :**

**1ère forme :** Le nombre de passages dans la boucle est connu au préalable (= **F-D+1**).
D, F peuvent être des constantes ou des expressions scalaires.

| ALGORITHME | PASCAL |
|---|---|
| **POUR** *C* **DE** *D* **A** *F* **FAIRE**<br>\| Instructions<br>\|<br>**FINPOUR** | **FOR** *C* **: =** *D* **TO** *F* **DO**<br>BEGIN<br>Instructions;<br>END; |

**C** est une variable de type scalaire qui sera **incrémenté** après chaque parcours.

**2ème forme :** La valeur du pas peut être négative, dans ce cas la valeur initiale (F) sera supérieure à la valeur finale (D).

| ALGORITHME | PASCAL |
|---|---|
| **POUR** *C* **DE** *F* **A** *D* **PAS = -1 FAIRE**<br>\| Instructions<br>**FINPOUR** | **FOR** *C* **: =** *F* **DOWNTO** *D* **DO**<br>BEGIN<br>Instructions;<br>END; |

**C** est une variable de type scalaire qui sera **décrémenté** après chaque parcours.

# Les sous programmes

## 1. DEFINITIONS :

- **Procédure / Fonction** : un module de programme auquel on peut faire référence par son nom.
- **Procédure** : suite de commandes ou d'instructions décrivant une action simple ou composée a laquelle on donne un **nom** et qui permet de **retourner plusieurs résultats**.
- **Fonction** : procédure particulière qui ne transmet qu'un **seul résultat**, une **seule valeur**, au programme appelant.
- **Variables globales** : variables déclarées dans le programme principal, utilisable dans les instructions du programme principal ainsi que dans les procédures et les fonctions.
- **Variables locales** : variables déclarées dans la procédure ou la fonction, utilisable uniquement à l'intérieur de la procédure ou la fonction.
- **Paramètres formels** : figurent dans l'en-tête de la déclaration de la procédure ou la fonction. Ces paramètres sont utilisés dans les instructions de la procédure ou la fonction et là seulement. ils correspondent a des variables locales.
- **Paramètres effectifs (réels)** : figurent dans l'appel de la procédure ou la fonction. Ces paramétres sont substitués aux paramètres formels au moment de l'appel de la procédure ou la fonction, il sajit de variable, constante ou d'une expression.
- On peut créer une procédure ou une fonction sans paramètres formels.
- **Passage de paramètres** : La substitution des paramètres effectifs aux paramètres formels s'appelle passage de paramètres elle correspond a un transfert de données entre le programme principal et La procédure.
- **Deux modes de passage de paramètres :**

  **Passage par valeur :** Le transfert d'information est effectué dans un seul sens, du programme principal vers la procédure.
  Permet au programme appelant de transmettre une valeur à la procédure appelé. Toute modification du paramêtre formel est **sans consequence** sur le paramètre effectif.

  **Passage par variable (référence) :** le paramétre formel est précédé de **VAR** .
  Le transfert d'information est éffectué dans les deux sens, du programme principal vers la procédure et inversement.
  Permet au programme de transmettre une valeur à la procédure et inversement. Toute modification du paramètre formel entraine automatiquement la **modification** de la valeur du paramètre effectif.

- Les paramètres effectifs transmis par variable ne peuvent pas être des constantes ou des expressions. Il ne peut s'agir que des variables.
- On interdit l'emploi de paramètres variables dans les fonctions ainsi que la modification des valeurs des variables globales.

## 2. LES PROCEDURES :

- **Déclaration d'une procédure** : (PROCEDURE ou DEFPROC)

| ALGORITHME | PASCAL |
|---|---|
| (0) **PROCEDURE** NomP (Paramètres_formels)<br>(1) instruction<br>.<br>.<br>.<br>(n) **FIN NomP** | **PROCEDURE** NomP (Paramètres_formels) ;<br>**VAR**<br>Déclaration_des_variables_locales ;<br>**BEGIN**<br>Instructions ;<br>**END;** |

- **Appel d'une procédure** :

| ALGORITHME | PASCAL |
|---|---|
| **PROC** NomP (Paramètres_effectifs) | NomP (Paramètres_effectifs) ; |

## 3. LES FONCTIONS :

- **Déclaration d'une fonction :** (FONCTION ou DEFFN)

| ALGORITHME | PASCAL |
|---|---|
| (0) **FONCTION** NomF (Paramètres_formels) : TypeF<br>(1) instruction<br>.<br>.<br>.<br>(n-1) **NomF** ← Valeur<br>(n) **FIN NomF** | **FUNCTION** NomF (Paramètres_formels) : TypeF ;<br>**VAR**<br>Déclaration_des_variables_locales ;<br>**BEGIN**<br>Instructions ;<br>**. . .**<br>**NomF : =** Valeur ;<br>**END;** |

**TypeF** est le type du résultat retourné (doit être différent d'un tableau ou d'un enregistrement).

**Valeur** est un identificateur de variable ou expression de même type ou compatible à TypeF.

Le résultat de la fonction est la **denière valeur** qui a été affectée à son identificateur (**NomF**).

- **Appel d'une fonction** : une fonction s'utilise comme on utiliserait une variable du type de la valeur retournée par la fonction, on peut donc l'utiliser dans une expression ; exemples :

| ALGORITHME | PASCAL |
|---|---|
| V ← **FN** NomF (Paramètres_effectifs ) | V : = NomF (Paramètres_effectifs) ; |
| Ecrire ( **FN** NomF (Paramètres_effectifs) ) | Write ( NomF (Paramètres_effectifs ) ) ; |

**V** est une variable de même type que le résultat retourné par la fonction.

# Fonctions et procédures prédéfinies

## LES FONCTIONS MATHEMATIQUES :

| ALGORITHME | PASCAL | RÔLE | TYPE OPERANDES | TYPE DU RESULTAT |
|---|---|---|---|---|
| ABS (x) | **ABS (x)** | Valeur absolue de x. | ENTIER / REEL | ENTIER / REEL |
| CARRE (x) | **SQR (x)** | Carré de x. | ENTIER / REEL | ENTIER / REEL |
| RACINECARRE (x) | **SQRT (x)** | Racine carré de x. | ENTIER / REEL | REEL |
| SIN (x) | **SIN (x)** | Sinus de x (en radians). | ENTIER / REEL | REEL |
| COS (x) | **COS (x)** | Cosinus de x (en radians). | ENTIER / REEL | REEL |
| ARCTAN (x) | **ARCTAN (x)** | Arctangente de x (en radians). | ENTIER / REEL | REEL |
| EXP (x) | **EXP (x)** | Exponentielle de x. | ENTIER / REEL | REEL |
| LN (x) | **LN (x)** | Logarithme népérien de x. | ENTIER / REEL | REEL |
| Π | **Pi** | Valeur de Pi = 3, 14. | | REEL |
| ENT (x) | **INT (x)** | Partie entière. | REEL | REEL |
| FRAC (x) | **FRAC (x)** | Partie décimale. | REEL | REEL |
| ARRONDI (x) | **ROUND (x)** | Arrondie de x a la plus proche valeur. | REEL | ENTIER |
| TRONC (x) | **TRUNC (x)** | Supprime la partie décimale. | REEL | ENTIER |
| ALEA | **Randomize; RANDOM** | Valeur aléatoire $\in 0..1$ | | REEL |
| ALEA(x) | **Randomize; RANDOM (X)** | Valeur aléatoire $\in 0..x-1$ | ENTIER | ENTIER |

## LES FONCTIONS PREDEFINIES SUR LES CARACTERES :

| ALGORITHME | PASCAL | RÔLE | TYPE DU RESULTAT |
|---|---|---|---|
| ORD (A) | **ORD (A)** | Renvoie le **code ASCII** du caractère A. | ENTIER |
| CHR (A) | **CHR (A)** | Renvoie le **caractère** dont le code ASCII est A. | CARACTERE |
| SUCC (A) | **SUCC (A)** | Renvoie le caractère **successeur** du caractère A. | CARACTERE |
| PRED (A) | **PRED (A)** | Renvoie le caractère **prédécesseur** du caractère A | CARACTERE |
| MAJUS (A) | **UPCASE (A)** | Convertit le caractère A en **majuscule**. | CARACTERE |

**Remarque :** ORD, SUCC et PRED sont aussi utilisables sur les variables d'un type énuméré ou booléen (sont respectivement **numéro d'ordre (le 1$^{er}$ = 0)**, **successeur** et **prédécesseur** d'une valeur donnée).

**Exemples** : ORD(False) = 0, ORD(True) = 1, SUCC (False) = True, PRED (True) = False

## LES FONCTIONS PREDEFINIES SUR LES CHAINES :

| ALGORITHME | PASCAL | RÔLE | TYPE DU RESULTAT |
|---|---|---|---|
| CONCAT (mot1, mot2,..., motn) | **CONCAT (mot1, mot2, ..., motn)** | Concaténation d'un groupe de chaîne. | CHAINE |
| SOUS_CHAINE (mot, pos, nb) | **COPY (mot, pos, nb)** | Extrait de **mot , nb** caractères a partir de la position **pos**. | CHAINE |
| LONG (mot) | **LENGTH (mot)** | Renvoie la **longueur** de La variable **mot**. | ENTIER |
| POSITION (mot1, mot2) | **POS (mot1, mot2)** | Retourne un entier contenant la position de la première occurrence de **mot1** dans **mot2**, ou **0** si elle n'existe pas. | ENTIER |

**Remarque :** En Turbo PASCAL, l'opérateur « **+** » permet la concaténation de deux chaines.
La comparaison entre deux chaines se fait en utilisant les « opérateurs relationnels ».

## LES PROCEDURES PREDEFINIES SUR LES CHAINES :

| ALGORITHME | PASCAL | RÔLE |
|---|---|---|
| EFFACE (mot, pos, nb) | **DELETE (mot, pos , nb) ;** | Enlève **nb** caractères de la variable chaîne **mot** à partir de la position **pos**. |
| INSERE (mot1, mot2, pos) | **INSERT (mot1, mot2, pos) ;** | Insérer **mot1** dans **mot2** a la position **pos**. |
| CONVCH (valeur, mot) | **STR (valeur, mot) ;** | Convertit une valeur **numérique** en une **chaîne**. |
| VALEUR (mot, valeur, ERREUR) | **VAL (mot, valeur, ERREUR) ;** | Convertit une **chaîne** de caractère en une valeur **numérique**. (**SI** conversion = VRAI **Alors** ERREUR = 0 **Sinon** ERREUR = la position de la faute) |

# Enregistrements et fichiers

## LES ENREGISTREMENTS :

### a) Définition :

Un enregistrement (ou structure) permet de désigner sous un seul nom un ensemble de champ pouvant être de types différents.

### b) Déclaration :

**ALGORITHME**

**Tableau de déclaration des nouveaux types**

| **Type** |
|---|
| Element **= ENREGISTREMENT**<br>Champ1 : type1<br>Champ2 : type2<br>...<br>ChampN : typeN<br>**FIN** Element |

**Tableau de déclaration des objets**

| **Objet** | **Type/Nature** |
|---|---|
| V | Element |

**PASCAL**

```
TYPE  Element =  RECORD
                    Champ1 : type1 ;
                    Champ2 : type2 ;
                     ...
                    ChampN : typeN ;
      END ;

VAR   V : Element ;
```

**Element** est un type (modèle) d'enregistrement (structure), qui précise le nom et le type de chacun Des **champs** constituant cette structure (**Champ_1_2…N**).

Avec **type_1_2…N** : sont des types simples (prédéfinis du langage de programmation), définis par l'utilisateur ou enregistrement.

**V** un identificateur de variable correspondant à cette structure.

**c) Accès à un enregistrement :**

| | ALGORITHME | PASCAL |
|---|---|---|
| Accès **global** | V1 ← V2 | V1 **: =** V2 ; |
| Accès a un **champ** | V1**.**Champ ← Valeur | V1**.**Champ **: =** Valeur ; |
| L'instruction **AVEC** | **AVEC** V1 **FAIRE**<br>Champ1 ← Valeur1 Champ2 ← Valeur2<br>...<br>ChampN ← ValeurN<br>**FINAVEC** | **WITH** V1 **DO**<br>BEGIN<br>Champ1 **: =** Valeur1 ;<br>Champ2 **: =** Valeur2 ;<br>...<br>ChampN **: =** ValeurN ;<br>END **;** |

Avec **V1** un identificateur de variable et **V2** une variable ou expression de même type que la variable **V1**.
**Valeur_1_2…N** sont des identificateurs de variable ou expressions de même type ou compatible avec **Champ_1_2…N**.

## LES FICHIERS :

**a) Définition :** Un fichier est un ensemble d'informations regroupées sous forme d'enregistrements de même nature stockées sur un support de mémoire de masse.

**b) Eléments attachés à un fichier :**

- **Nom interne** (logique) d'un fichier : Nom sous lequel est identifié dans un programme.
- **Nom externe** (physique) d'un fichier : Nom sous lequel est identifié en mémoire secondaire.
- **Tampon ou Buffer** d'un fichier : zone de la mémoire principale pouvant contenir un enregistrement du fichier.

**c) Caractéristiques d'un fichier :**

- **Types de fichiers** : Fichiers Texte, Typés (à enregistrements), graphiques et exécutables.
- **Mode d'organisation** : il existe plusieurs modes dont l'organisation séquentielle et directe (relative).
- **Mode d'accès** :

  **Accès séquentiel :** pour atteindre l'élément de rang **n**, il est nécessaire de parcourir les (**n-1**) éléments précédents.

  **Accès direct :** retrouver directement l'élément recherché, à condition que l'on connaisse son numéro d'ordre (c'est-à-dire sa position dans le fichier).

**d) Opérations sur les fichiers à accès séquentiel et direct (Fichiers Typés):**

- **Déclaration :**

**ALGORITHME**

**Tableau de déclaration des nouveaux types**

**Type**

Element **= ENREGISTREMENT**

Champ1 : type1
Champ2 : type2
...
ChampN : typeN

**FIN** Element

TypFile = **FICHIER DE** Element

**Tableau de déclaration des objets**

| Objet | Type/Nature |
|---|---|
| **Nom_interne** | TypFile |
| V | Element |

**PASCAL**

```
TYPE
        Element  =  RECORD
                        Champ1 : type1 ;
                        Champ2 : type2 ;
                         ...
                        ChampN : typeN ;
                    END ;

        TypFile = FILE OF Element ;

VAR     Nom_interne : TypFile ;
        V : Element ;
```

**Nom_interne** est le nom interne (logique) du fichier.

**V** est une variable tampon.

- **Commandes sur les fichiers à accès séquentiel et direct (Fichiers Typés):**

| | ALGORITHME | PASCAL |
|---|---|---|
| 1 | **ASSOCIER** ( Nom_interne , Nom_externe ) | **ASSIGN** ( Nom_interne , Nom_externe ) ; |
| | **P** - Associer le Nom interne (logique) d'un fichier à son Nom externe (physique). | |
| 2 | **RECREER** ( Nom_interne ) | **REWRITE** ( Nom_interne ) ; |
| | **P** - Créer un fichier s'il n'existe pas sinon écrasement du contenu. | |
| 3 | **OUVRIR** ( Nom_interne ) | **RESET** ( Nom_interne ) ; |
| | **P** - Ouvrir le fichier s'il existe; le pointeur se positionne au début du fichier. | |
| 4 | **LIRE** ( Nom_interne , V ) | **READ** ( Nom_interne , V ) ; |
| | **P** - Lecture d'un enregistrement du fichier sur lequel le pointeur est positionné (place le pointeur de sur l'élément suivant). | |
| 5 | **ECRIRE** ( Nom_interne , V ) | **WRITE** ( Nom_interne , V ) ; |
| | **P** - Ecriture ou modification d'un enregistrement sur le fichier (place le pointeur de sur l'élément suivant) | |
| 6 | **FERMER** ( Nom_interne ) | **CLOSE** ( Nom_interne ) ; |
| | **P** - Fermeture d'un fichier ouvert. | |
| 7 | **EFFACER** ( Nom_interne ) | **ERASE** ( Nom_interne ) ; |
| | **P** - Effacer (supprimer du disque) un fichier. | |
| 8 | **RENOMMER** (Nom_interne , new_Nom_externe) | **RENAME** (Nom_interne , new_Nom_externe) ; |
| | **P** - Renommer le nom externe d'un fichier fermé. | |
| 9 | **FIN_FICHIER** ( Nom_interne ) | **EOF** ( Nom_interne) |
| | **Fn** - Détecte la fin du fichier, et retourne la valeur Vrai ( True ) sinon Faux ( False ). | |
| 10 | **TAILLE_FICHIER** ( Nom_interne ) | **FILESIZE** ( Nom_interne ) |
| | **Fn** - Renvoie la taille (Nombre d'enregistrements) du fichier. | |
| 11 | **POINTER** ( Nom_interne , numero_ordre ) | **SEEK** ( Nom_interne , numero_ordre ) ; |
| | **P** - Accéder directement à un enregistrement par son numéro d'ordre. (**0** pour le 1er enregistrement). | |
| 12 | **TRONQUER** ( Nom_interne ) | **TRUNCATE** ( Nom_interne ) ; |
| | **P** - Tronque un fichier à la position courante du pointeur, c a d tout ce qui se trouve après la position courante du pointeur est supprimé. | |
| 13 | **POSITION_FICHIER** ( Nom_interne ) | **FILEPOS** ( Nom_interne ) |
| | **Fn** - Renvoie la position courante (numéro d'ordre) du pointeur du fichier. | |

### e) Opérations sur les fichiers texte :

- **Définition :**

Les fichiers texte sont des fichiers **séquentiels** qui contiennent des caractères organisés en lignes terminées par un caractère Retour Chariot (noté **CR** dont le code ASCII est 13) qui marque la fin de chaque ligne.
Une marque de fin de fichier (Ctrl-Z) termine la séquence de lignes.
Les commandes associer, recréer, écrire, lire, fin_fichier, renommer, effacer et fermer sont identiques à celle appliquées sur les fichiers à enregistrements (la commande **reset** ouvre un fichier texte uniquement en lecture).

**Remarque** : Dans un fichier texte chaque octet représente un caractère.

- **Déclaration :**

<table>
<tr><th>ALGORITHME</th><th>PASCAL</th></tr>
<tr><td>Tableau de déclaration des objets<br><br><table><tr><th>Objet</th><th>Type/Nature</th></tr><tr><td>Nom_interne</td><td>Texte</td></tr><tr><td>V</td><td>Chaine</td></tr></table></td><td>VAR Nom_interne:TEXT;<br><br>V : STRING;</td></tr>
</table>

**Nom_interne** est le nom interne (logique) du fichier.

**V** est une variable tampon.

- **Commandes sur les fichiers texte :**

| | ALGORITHME | PASCAL |
|---|---|---|
| 1 | **FIN_LIGNE** ( Nom_interne ) | **EOLN** ( Nom_interne ) |
| | **Fn** - Renvoie le statut de fin de ligne, retourne la valeur Vrai ( True ) si fin de ligne atteint sinon Faux (False). | |
| 2 | **CHERCHER_FIN_LIGNE** (Nom_interne) | **SEEKEOLN** ( Nom_interne ) |
| | **Fn** - Renvoie le statut de fin de ligne, retourne la valeur Vrai ( True ) si fin de ligne atteint sinon Faux (False). Avant ce test, la fonction supprime tout espace et caractère de tabulation. | |
| 3 | **CHERCHER_FIN_FICHIER** ( Nom_interne ) | **SEEKEOF** ( Nom_interne ) |
| | **Fn** - Détecte la fin du fichier, et retourne la valeur Vrai ( True ) sinon Faux ( False ). Avant ce test, la fonction supprime tout espace et caractère de tabulation. | |
| 4 | **AJOUTER** ( Nom_interne ) | **APPEND** ( Nom_interne ) ; |
| | **P** - Ouvre un fichier pour ajout (le pointeur se positionne à la fin du fichier). On ne peut pas **écrire** dans un fichier Texte ouvert avec **Ouvrir/Reset.** | |
| 5 | **LIRE_NL** ( Nom_interne ) | **READLN** ( Nom_interne ) ; |
| | **P** - Amène à la ligne suivante. | |
| 6 | **LIRE_NL** ( Nom_interne , V ) | **READLN** ( Nom_interne , V ) ; |
| | **P** - Lecture de la ligne courante (et amène à la suivante). | |
| 7 | **ECRIRE_NL** ( Nom_interne ) | **WRITELN** ( Nom_interne ) ; |
| | **P** - Ajoute un marqueur de fin de ligne. | |
| 8 | **ECRIRE_NL** ( Nom_interne , V ) | **WRITELN** ( Nom_interne , V ) ; |
| | **P** - Ecriture dans la ligne courante (et ajoute un marqueur de fin de ligne). | |

# La Récursivité

## NOTION DE RECURSIVITE :

Une procédure ou fonction est dite récursive si son corps contient un ou plusieurs appels à elle-même : (Récursivité directe)

0) Procédure **NomP** (paramètres)
1) …
   Proc **NomP** (valeurs)
   ...
n) Fin NomP

Dans l'écriture de tout module récursif, il sera nécessaire de mettre en évidence d'une part la forme de l'appel récursif et d'autre part la condition d'arrêt.

## MECANISME DE FONCTIONNEMENT DE LA RECURSIVITE :

Soit la fonction suivante :

**0) Fonction Factorielle (n : entier) : entier**
**1) Si** n = 0 **Alors** Factorielle ← 1
   **Sinon** Factorielle ← n * **Fn** Factorielle (n – 1)
   **FinSi**
**2) Fin Factorielle**

**Exemple** : calcul de 4! (n = 4) par la fonction récursive Factorielle :

Factorielle(4) renvoie 4 * Factorielle(3)

Factorielle(3) renvoie 3 * Factorielle(2)

Factorielle(2) renvoie 2 * Factorielle(1)

Factorielle(1) renvoie 1 * Factorielle(0)

Factorielle(0) renvoie 1 **(arrêt de la récursivité)**

Factorielle(1) renvoie 1 * 1 = 1

Factorielle(2) renvoie 2 * 1 = 2

Factorielle(3) renvoie 3 * 2 = 6

Factorielle(4) renvoie 4 * 6 = 24

## RECURSIVITE CROISEE OU INDIRECTE :

**Récursivité croisée** un sous programme A, appelle un sous programme B qui appelle A.

**Récursivité indirecte** un sous programme A, appelle un sous programme B qui appelle un Sous programme C… qui appelle A.

# Les algorithmes de Tri

**Soit à trier un tableau V de n entiers (vecteur) selon un ordre croissant.**

## TRI PAR SELECTION :

1. On se pointe à la 1ère case du tableau et on parcourt la totalité de V afin de sélectionner l'indice du plus petit.
2. Echanger le plus petit élément trouvé avec le premier élément.
3. Refaire les étapes 1 et 2 et chercher le plus petit élément de la liste sauf le premier puis l'échanger avec le second.

**Algorithme** **0) Procédure Tri_Selection (Var V : vecteur ; n : entier)**

```
1) Pour i De 1 A n – 1 Faire
   |        Indice ← i
   |        Pour k De i + 1 A n Faire
   |        |        Si V [ Indice ] > V [ k ] Alors
   |        |        |        Indice ← k
   |        |        FinSi
   |        FinPour
   |
   |        Si Indice ≠ i Alors
   |        |        aux ← V[ i ]
   |        |        V [ i ] ← V [ Indice ]
   |        |        V [ Indice ] ← aux
   |        FinSi
   FinPour

2) Fin Tri_Selection
```

**Indice, i, k, aux** : *entier*.

## TRI PAR SELECTION PROCEDE RECURSIF :

**Algorithme** **0) Procédure Tri_Selection (Var V : vecteur ; i, n : entier)**

```
1) Indice ← i
   Pour k De i + 1 A n Faire
   |        Si V [ Indice ] > V [ k ] Alors
   |        |        Indice ← k
   |        FinSi
   FinPour

2) Si Indice ≠ i Alors
   |        aux ← V[ i ]
   |        V [ i ] ← V [ Indice ]
   |        V [ Indice ] ← aux
   FinSi

3) Si i+1 ≠ n Alors Proc Tri_Selection (V , i+1 , n)
   FinSi

4) Fin Tri_Selection
```

**Indice, k, aux** : *entier*.

- **TRI A BULLES :**

1. Comparer le premier pair d'éléments.
2. Si V [ 1 ] > V [ 2 ] alors permuter V [ 1 ] et V [ 2 ] et tenir compte de cette action.
3. Aller au pair suivant et répéter les étapes 1 et 2 jusqu'à comparer le dernier pair.
4. Si une permutation a été réalisé (ou plusieurs) alors répéter ce qu'on vient de faire, sinon la liste est triée.

**Algorithme** **0) Procédure Tri_a_bulles (Var V : vecteur ; n : entier)**

```
1) Répéter
|       Ok ← Vrai
|       Pour i De 1 A n – 1 Faire
|       |       Si V [ i ] > V [ i + 1 ] Alors
|       |       |       aux ← V[ i ]
|       |       |       V [ i ] ← V [ i + 1 ]
|       |       |       V [ i + 1 ] ← aux
|       |       |       Ok ← Faux
|       |       FinSi
|       FinPour
|
Jusqu'a (Ok = Vrai)

2) Fin Tri_a_bulles
```

**i, aux** : *entier*.
**Ok** : *booléen*.

- **TRI A BULLES PROCEDE RECURSIF :**

**Algorithme** **0) Procédure Tri_a_bulles (Var V : vecteur ; n : entier)**

```
1) Ok ← Vrai

   Pour i De 1 A n – 1 Faire
   |       Si V [ i ] > V [ i + 1 ] Alors
   |       |       aux ← V[ i ]
   |       |       V [ i ] ← V [ i + 1 ]
   |       |       V [ i + 1 ] ← aux
   |       |       Ok ← Faux
   |       FinSi
   FinPour

2) Si Ok = Faux Alors
   |       Proc Tri_a_bulles (V, n)
   FinSi

3) Fin Tri_a_bulles
```

**i, aux** : *entier*.
**Ok** : *booléen*.

- **TRI PAR INSERTION :**

1. On commence par le deuxième élément.
2. Comparer l'élément choisi avec tous ses précédents dans la liste et l'insérer dans sa bonne position.
3. Répéter l'étape 2 pour l'élément suivant jusqu'à arriver au dernier.

**Algorithme** **0) Procédure Tri_Insertion (Var V : vecteur ; n : entier)**

**1) Pour** i **De** 2 **A** n **Faire**

aux ← V[ i ]

P ← i – 1

*aux est la valeur à insérer dans l'endroit approprié du Tableau. On décale toutes les valeurs du Tableau < aux à droite pour vider*

**Tantque** ( P > 0 ) **Et** ( V [ P ] > aux ) **Faire**

V [ P + 1 ] ← V [ P ]

P ← P – 1

**FinTantque**

V [ P + 1 ] ← aux

*Finalement la valeur **aux** est insérée à son emplacement adéquat.*

**FinPour**

**2) Fin Tri_Insertion**

**i, P, aux** : *entier*

- **TRI PAR INSERTION PROCEDE RECURSIF :**

**Algorithme** **0) Procédure Tri_Insertion (Var V : vecteur ; n : entier)**

**1) Si** n > 1 **Alors**

**Proc** Tri_Insertion ( V, n – 1 )

**Si** V[n] < V[n–1] **Alors**

aux ← V [ n ]

i ← n

**Répéter**

V [ i ] ← V [ i – 1 ]

i ← i – 1

**Jusqu'a** ( i = 1 ) **Ou** ( aux > V [ i – 1 ] )

V [ i ] ← aux

**FinSi**

**FinSi**

**2) Fin Tri_Insertion**

**Aux, i** : *entier*

- **TRI SHELL :** Le tri Shell trie chaque liste d'éléments séparés de « **pas** » **positions** chacun avec le **tri par insertion**. L'algorithme effectue plusieurs fois cette opération **en diminuant « pas » jusqu'à** « **pas=1** » ce qui équivaut à trier tous les éléments ensemble. (**pas** = espacements entre les éléments, Pas ou Gap)

**Algorithme 0) Procédure Tri_Shell (Var V : vecteur ; n : entier)**

**1)** pas ← 0

**Tantque** ( pas < n ) **Faire**

    pas ← 3 * pas + 1

**FinTantque**

*Recherche du Pas optimal qui est le résultat de la suite récurrente : $U_n = 3*U_{n-1} + 1$ Tel que $U_n < n$ Nombre d'éléments du tableau.*

**2)** **Tantque** ( pas ≠ 0 ) **Faire**

    pas ← pas div 3 *On affine peu à peu le Pas, Pas = 1 → Tri par Insertion ordinaire.*

    **Pour** i **De** pas+1 **A** n **Faire**

        aux ← V [ i ] *Valeur à insérer.*

        P ← i

        **Tantque** (P > pas) **Et** (V[P – pas] > aux) **Faire**

            V[P] ← V[P – pas]

            P ← P – pas

        **FinTantque** *Recherche de la position d'insertion.*

        V[P] ← aux *Insertion de la valeur à son emplacement.*

    **FinPour**

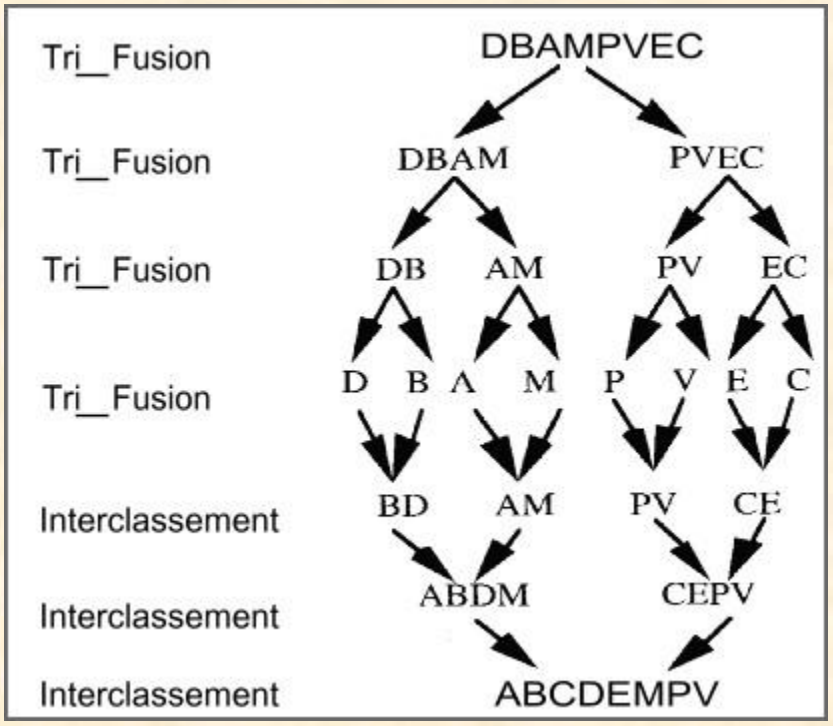
**FinTantque**

**3)** **Fin Tri_Shell**

**pas, aux, i, P** : *entier.*

- **TRI PAR FUSION :** A chaque étape on trie la première moitié du vecteur et la seconde moitié, puis on interclasse les deux sous-vecteur triés.

**Exemple :** Soit à trier le vecteur V de taille 8 de type caractère (D, B, A, M, P, V, E, C)




**Algorithme** **0) Procédure Tri_Fusion (Var V : vecteur ; inf, sup: entier);**

**1)** **Si** sup > inf **Alors**

milieu ← (inf + sup) div 2

**Proc** Tri_Fusion (V, inf, milieu)

**Proc** Tri_Fusion (V, milieu+1, sup)

**Proc** Interclassement (V, inf, milieu, V, milieu+1, sup, V )

**FinSi**

**2)** **Fin Tri_Fusion**

**milieu** : *entier*.

**Interclassement** : *procédure*.

**0) Procédure Interclassement (V1 : vecteur; i, n1 : entier; V2 : vecteur; j, n2 : entier; Var V3 : vecteur)**

**1)** k ← i

**Tantque** (i≤n1) **Et** (j≤n2) **Faire**

**Si** V1[i] ≤ V2[j] **Alors**

V3[k] ←V1[i]

i ← i + 1

**Sinon**

V3[k] ← V2[j]

j ← j + 1

**FinSi**

k ← k +1

**FinTantque**

**2) Tantque** i ≤ n1 **Faire**

V3[k] ← V1[i]

i ← 1+ i

k ← k + 1

**FinTantque**

**3) Tantque** j ≤ n2 **Faire**

V3[k] ← V2[j]

j ← j + 1

k ← k + 1

**FinTantque**

**4) Fin Interclassement**

**K** : *entier*.

- **TRI PAR COMPTAGE :** L'algorithme de tri consiste à construire un vecteur intermédiaire **ind** dont chaque élément **ind[i]** indique la place de l'élément **V[i]** dans le vecteur trié correspondant.

**Algorithme** **0) Procédure Tri_Comptage (V : vecteur; Var Vtrié : vecteur ; n : entier)**

**1) Pour** i **De** 1 **A** n **Faire**
    Ind [i] ← 1
**FinPour**

**2) Pour** i **De** 1 **A** n-1 **Faire**
    **Pour** k **De** i +1 **A** n **Faire**
        **Si** V[ k ]<V[ i ] **Alors** ind [ i ] ← ind [ i ] + 1
        **Sinon** ind [ k ] ← ind [ k ] + 1
        **FinSi**
    **FinPour**
**FinPour**

**3) Pour** i **De** 1 **A** n **Faire**
    Vtrié [ ind[ i ] ] ← V[ i ]
**FinPour**

**4) Fin Tri_Comptage**

**Vtrié** : *tableau contenant tous les éléments de* **V** *triés en ordre croissant.*

**ind :** *vecteur.*
**i, k** : *entier.*

# Méthodes de recherche

**Soit un tableau V de n entiers (vecteur) et X l'élément à rechercher.**

- **RECHERCHE SEQUENTIELLE :** Comparer **X** aux différents éléments du tableau jusqu'à trouver **X** ou atteindre la fin du tableau.

**Algorithme** **0) Fonction Recherche (V : vecteur; n, X : entier) : booléen**

**1)** i ← 1

**Tantque** (i<n) **et** (V[i] ≠ X) **Faire**

i ← i + 1

**FinTantque**

**2)** Recherche ← V[i] = X

**3) Fin Recherche**

**i** : *entier.*

- **RECHERCHE SEQUENTIELLE PROCEDE RECURSIF :**

**Algorithme** **0) Fonction Recherche (V : vecteur; n, X : entier) : booléen**

**1) Si** n = 0 **alors** Recherche ← Faux

**Sinon** **Si** V[n] = X **Alors** Recherche ← Vrai

**Sinon** Recherche ← **Fn** Recherche (V, n-1, X)

**FinSi**

**FinSi**

**2) Fin Recherche**

- **RECHERCHE SEQUENTIELLE (VECTEUR TRIE – ORDRE CROISSANT) :**

**Algorithme** **0) Fonction Recherche (V : vecteur; n, X : entier) : booléen**

**1)** i ← 1

**Tantque** (i<n) **et** (V[i] < X) **Faire**

i ← i + 1

**FinTantque**

**2)** Recherche ← V[i] = X

**3) Fin Recherche**

**i** : *entier.*

## ▪ RECHERCHE DICHOTOMIQUE (VECTEUR TRIE – ORDRE CROISSANT) :

Diviser l'intervalle de recherche par 2 à chaque itération. Pour cela, on procède de la façon suivante:
Soient **inf** et **sup** les extrémités gauche et droite de l'intervalle dans lequel on cherche la valeur **X**, on calcule **m**, l'indice de l'élément médian : **m = (inf + sup) div 2.**
Il y a 3 cas possibles:

- **X = V [m]** : l'élément de valeur X est trouvé, la recherche est terminée.
- **X < V [m]** : l'élément X, s'il existe, se trouve dans l'intervalle **[inf .. m - 1].**
- **X > V [m]** : l'élément X, s'il existe, se trouve dans l'intervalle **[m + 1 .. sup].**

La recherche dichotomique consiste à itérer ce processus jusqu'à ce que l'on trouve **X** ou que l'intervalle de recherche soit vide.

**Algorithme** **0) Fonction Recherche (V : vecteur; n, X : entier) : booléen**

**1)** trouve ← Faux

**Si** (X≥V*1+) **et** (X≤V*n+) **Alors**

inf ← 1

sup ← n

**Tantque** ( trouve = Faux) **et** ( inf ≤ sup ) **Faire**

m ← (inf + sup) div 2

**Si** X = V [ m ] **Alors** trouve ← Vrai

**Sinon**

**Si** X < V [ m] **Alors** sup ← m - 1

**Sinon** inf ← m +1

**FinSi**

**FinSi**

**FinTantque**

**FinSi**

**2)** Recherche ← trouve

**3) Fin Recherche**

**inf, sup, m** : *entier*.
**trouve** : *booléen*.

- **RECHERCHE DICHOTOMIQUE (VECTEUR TRIE – ORDRE CROISSANT) :**

**Algorithme** **0) Fonction Recherche (V : vecteur; n, X : entier) : booléen**

**1)** **Si** X > V[n] **Alors** Recherche ← Faux

**Sinon**

inf ← 1

sup ← n

**Tantque** inf < sup **Faire**

m ← (inf + sup) div 2

**Si** X ≤ V [ m] **Alors** sup ← m

**Sinon** inf ← m + 1

**FinSi**

**FinTantque**

Recherche ← X = V [ sup ]

**FinSi**

**2)** **Fin Recherche**

**inf, sup, m** : *entier*

- **RECHERCHE DICHOTOMIQUE PROCEDE RECURSIF (VECTEUR TRIE – ORDRE CROISSANT) :**

**Algorithme** **0) Fonction Recherche (V : vecteur; inf, sup, X : entier) : booléen**

**1)** **Si** inf > sup **Alors** Recherche ← Faux

**Sinon**

m ← (inf + sup) div 2

**Si** X = V [ m ] **Alors** Recherche ← Vrai

**Sinon**

**Si** X < V [ m] **Alors** sup ← m - 1

**Sinon** inf ← m +1

**FinSi**

Recherche ← **Fn** Recherche (V, inf, sup, X)

**FinSi**

**FinSi**

**2)** **Fin Recherche**

**m** :*entier*.

# Algorithmes de mise à jour

**Soit un tableau V de n entiers (vecteur) et X l'élément à insérer/supprimer.**

- **INSERTION (VECTEUR TRIE) :**

**Algorithme** **0) Procédure Insertion (Var V : vecteur ; Var n : entier ; X : entier)**

```
1) Si n=0 Alors V[1] ← X
   Sinon
        i ← n
        Tantque (i≠0) et (V[i]>X) Faire
        |                          i ← i-1
        FinTantque

        p ← i+1

        Pour i De n A p Pas = -1 Faire
        |                          V[i+1] ← V[i]
        FinPour

        V[p] ← X
   FinSi
2) n ← n+1
3) Fin Insertion
```

- **SUPPRESSION (VECTEUR) :**

**recherche_position** : Fonction permettant de rechercher la position (indice) d'une occurrence de la valeur **X** à supprimer.

**Algorithme** **0) Fonction Recherche_position (V : vecteur ; n, X : entier) : entier**

```
1) Tantque ( n ≠ 0 ) et ( V*n+ ≠ X ) Faire
   |                          n ← n-1
   FinTantque
2) Recherche_position ← n
3) Fin Recherche_position
```

**i, p** : *entier*.

**Algorithme** **0) Procédure Suppression (Var V : vecteur ; Var n : entier ; X : entier)**

```
1) p ← Fn Recherche_position (V, n, X)
2) Si p > 0 Alors
        Pour i De p+1 A n Faire
        |                  V[i-1]← V[i]
        FinPour

        n← n-1
   FinSi
3) Fin Suppression
```

**Recherche_position** : *fonction*.
**i, p** : *entier*.

## ▪ INSERTION (FICHIER TRIE) :

**Soit un Fichier d'entiers (nomi), un tableau T de n entiers (vecteur) et X l'élément à insérer/supprimer.**

**Algorithme** **0) Procédure Insertion (Var nomi : TypFile ; X : entier)**

**1)** i ← 0
verif ← Faux
**Tantque** Non fin_fichier(nomi) **Faire**
i ← i +1
lire (nomi, F)
**Si** (F ≤ X) **OU** (verif = Vrai) **Alors** T[i] ← F
**Sinon**
T[i] ← X
i ← i+1
T[i] ← F
verif ← Vrai
**FinSi**
**FinTantque**

**2)** **Si** verif = Faux **Alors**
pointer(nomi, taille_fichier(nomi) )
écrire(nomi, X)
**FinSi**

**3)** **Si** verif = Vrai **Alors**
recréer(nomi)
**Pour** k **De** 1 **A** i **Faire**
écrire (nomi, T[k])
**FinPour**
**FinSi**

**4)** fermer(nomi)

**5)** **Fin Insertion**

**, k, F** : *entier*.
**T** : *vecteur*.
**verif :** *booléen*.

- **SUPPRESSION (FICHIER) M1 :**

**Algorithme**

```
0) Procédure Suppression (Var nomi : typfile ; X : entier)
1) ouvrir(nomi)
2) n ← 0
   Tantque Non fin_fichier(nomi) Faire
   |                                  lire(nomi, F)
   |                                  Si F≠X Alors
   |                                  |        n ← n+1
   |                                  |        T[n] ← F
   |                                  FinSi
   FinTantque

3) Si n≠taille_fichier(nomi) alors    recréer(nomi)
   |                                  Pour i De 1 A n Faire
   |                                  |        écrire(nomi, T[i])
   |                                  FinPour
   FinSi
4) fermer(nomi)
5) Fin Suppression
```

**i, n, F** : *entier*.
**T** : *vecteur*.

- **SUPPRESSION (FICHIER) M2 :**

**Algorithme**

```
0) Procédure Suppression (Var nomi : typfile ; X : entier)
1) ouvrir(nomi)
2) recréer(nomi1)

3) Tantque Non fin_fichier(nomi) Faire
   |                                  lire(nomi, F)
   |                                  Si F≠X Alors
   |                                  |        écrire(nomi1, F)
   |                                  FinSi
   FinTantque

4) Si taille_fichier(nomi1) ≠ taille_fichier(nomi) alors
   |                                  recréer(nomi)
   |                                  ouvrir(nomi1)
   |                                  Tantque Non fin_fichier(nomi1) Faire
   |                                  |        lire(nomi1, F)
   |                                  |        écrire(nomi, F)
   |                                  FinTantque
   FinSi
5) fermer(nomi1)
6) fermer(nomi)

7) Fin Suppression
```

**F** : *entier*.
**nomi1** : *typfile*.

- **SUPPRESSION (FICHIER A ACCES DIRECT) M3:**

**Algorithme** **0) Procédure Suppression (Var nomi : typfile ; X : entier)**

**1)** ouvrir(nomi)

**2)** n ← taille_fichier(nomi) - 1

**Pour** i **De** 0 **A** n **Faire**

pointer(nomi, i)

lire(nomi, F)

**Si** (F = X) **Alors**

**Pour** k **De** i+1 **A** n Faire

pointer(nomi, k)

lire(nomi, F)

pointer(nomi, k-1)

écrire(nomi, F)

**FinPour**

tronquer(nomi)

**FinSi**

**FinPour**

**3)** fermer(nomi)

**4) Fin Suppression**

**i, k, n, F** : *entier*.

- **TRI (FICHIER) :**

**proc Tri** : procédure permettant de trier un tableau T de taille n.

**Algorithme** **0) Procédure Tri_Fichier (Var nomi : typfile)**

**1)** ouvrir(nomi)

**2) Pour** i **De** 1 **A** taille_fichier(nomi) **faire** lire(nomi, T[i])

**FinPour**

**3) proc** Tri(T, i)

**4)** ouvrir(nomi)

**5) Pour** i **De** 1 **A** taille_fichier(nomi) **faire** écrire(nomi, T[i])

**FinPour**

**6)** fermer(nomi)

**7) Fin Tri_Fichier**

**i** : *entier*.
**T** : *vecteur*.
**Tri** : *procédure*.

- **TRI (FICHIER A ACCES DIRECT) :**

**Algorithme** **0) Procédure Tri_a_bulles (Var nomi : typfile)**

**1)** ouvrir(nomi)

**2)** n ← taille_fichier(nomi) - 1

**Répéter**

Ok ← Vrai

**Pour** i **De** 0 **A** n – 1 **Faire**

Pointer(nomi, i)

Lire(nomi, X)

Lire(nomi, Y)

**Si** X > Y **Alors**

Pointer(nomi, i)

écrire (nomi, Y)

écrire (nomi, X)

Ok ← Faux

**FinSi**

**FinPour**

**Jusqu'a** (Ok = Vrai)

**3)** fermer(nomi)

**4) Fin Tri_a_bulles**

**i, n, X, Y** : *entier*.
**Ok** : *booléen*.

# Notions complémentaires

- **POINT FIXE :** Un élément X est un point fixe d'une fonction F, si F(X) = X.
- **FACTORIELLE :** Produit de tous les entiers de l'intervalle *1 .. n+. (Notez que 0 ! = 1).

**Algorithme** **0) Fonction Factorielle (n : entier): entier**

**1)** f ← 1

**Pour** i **De** 2 **A** n **Faire**

f ← f * i

**FinPour**

**2)** Factorielle ← f

**3) Fin Factorielle**

**f, i** : *entier.*

- **n! :** Nombre de permutations de n éléments (la factorielle de n).
- **:** Nombre de mots de longueur p (groupement ordonné) parmi n :

$$A_n^p = \frac{n!}{n-p\ !} \quad ; \text{avec } 1 \leq p \leq n.$$

**Exemple** : On choisit 2 livres parmi 3 dénommés A, B, C. Combien de choix sont possibles ?

$A_3^2 = 6$ => AB, BA, AC, CA, BC, CB

- **:** Nombre de mots croissants de longueur p (groupement non ordonné) parmi n:

$$C_n^p = \frac{A_n^p}{p!} = \frac{n!}{p!\ n-p\ !} \quad ; \text{avec } 0 \leq p \leq n.$$

**Exemple :** Dans l'exemple précédent, 3 arrangements représentent une seule et même combinaison (ce sont des arrangements équivalents qui représentent le même ensemble de livres).

$C_3^2 = 3$ => AB, AC, BC.

- **PPCM :** le plus petit commun multiple de 2 entiers A et B. Le plus petit entier multiple à la fois de A et B.

**Algorithme** **0) Fonction PPCM (A, B : entier) : entier**

**1) Si** A > B

**Alors** cm ← A

**Sinon** cm ← B

**Fin Si**

**2) Tantque** (cm mod A ≠ 0) **OU** (cm mod B ≠ 0) **Faire**

cm ← cm + 1

**FinTantque**

**cm** : *entier.*

**3)** PPCM ← cm

**4) Fin PPCM**

- **PGCD :** le plus grand commun diviseur de 2 entiers A et B ; le plus grand entier permettant de diviser A et B.

**Algorithme**

```
0) Fonction PGCD (A, B : entier) : entier
1) Tantque B≠ 0 Faire
   |        R← A mod B
   |        A ← B
   |        B ← R
   FinTantque
2) PGCD ← A
3) Fin PGCD
```

*Méthode de calcul : algorithme d'***Euclide.**

**R** : *entier.*

- **PGCD :** PGCD(A, B) * PPCM(A, B) = A*B

*Méthode de calcul : algorithme de* **la différence.**

**Algorithme**

```
0) Fonction PGCD (A, B : entier) : entier
1) Tantque A≠B Faire
   |        Si A>B Alors A ← A - B
   |             Sinon B ← B - A
   |        FinSi
   FinTantque
2) PGCD ← A
3) Fin PGCD
```

- **NOMBRE PREMIER :** un entier B supérieur à 1 est dit premier s'il n'est divisible que par 1 et par lui-même.

**Algorithme**

```
0) Fonction Premier (B : entier) : booléen
1) verif ← vrai
   i ← 2
   Tantque (verif = vrai) ET (i dans [2..B div 2]) Faire
   |        verif ← ( B mod i) ≠ 0
   |        i ← i + 1
   FinTantque
2) Premier ← verif
3) Fin Premier
```

**i** : *entier.*
**verif** : *booléen.*

- **FACTEURS PREMIERS :** la décomposition d'un entier B en produit de facteurs premiers consiste à écrire cet entier sous la forme d'un produit de ces diviseurs premiers.

**Algorithme**

```
0) Procédure Fact_premier (B: entier)
1) Tantque (B ≠ 1) Faire
   |        i ← 1
   |        Répéter
   |             i ← i + 1
   |        Jusqu'à ( B mod i = 0 )
   |
   |        écrire ( i , '' * '' )
   |        B ← B div i
   FinTantque
2) écrire(b)
3) Fin Fact_premier
```

**i** : *entier.*

## ▪ CONVERSION DE LA BASE B1 A LA BASE 10 :

**Algorithme**

```
0)  Fonction Convert_b1_10 (ch : chaine ; b1: entier) : entier
1)  D ← 0
    P ← 1
    Pour i De long(ch) A 1 Pas = -1 Faire
        L← ord (ch[i])
        Si L dans[48..57]   Alors D ← D + (L-48) * P
                            Sinon D ← D + (L-55) * P
        FinSi
        P ← P * b1
    FinPour
2)  Convert_b1_10 ← D
3)  Fin Convert_b1_10
```

**L, D, i, P** : *entier*.

## ▪ CONVERSION DE LA BASE 10 A LA BASE B2 :

**Algorithme**

```
0)  Fonction Convert_10_b2 (D, b2 : entier) : chaine
1)  ch ← ""
    Répéter
        r ← D mod b2

        Si r<10      Alors  convch(r,c)
                    Sinon  c ← chr(r+55)
        FinSi

        ch ← c + ch

        D ← D div b2
    Jusqu' à D=0

2)  Convert_10_b2 ← ch
3)  Fin Convert_10_b2
```

**r** : *entier*.
**c, ch** : *chaine*.

**Remarque** : *Le code ASCII de « **0** » = **48**.*
*Le code ASCII de « **A** » = **65**.*

- **CONVERSION DE LA BASE B1 A LA BASE 10 :**

**Algorithme**

**0) Fonction Convert_b1_10 (ch : chaine ; b1: entier) : entier**

**1)** D ← 0
P ← 1
R ← "0123456789ABCDEF"
**Pour** i **De** long(ch) **A** 1 **Pas = -1 Faire**
L ← position (ch[i], R) -1
D ← D + P * L
P ← P * b1
**FinPour**

**2)** Convert_b1_10 ← D

**3) Fin Convert_b1_10**

**L, D, i, P** : *entier*
**R** : *chaine*.

- **CONVERSION DE LA BASE 10 A LA BASE B2 :**

**Algorithme**

**0) Fonction Convert_10_b2 (D, b2 : entier) : chaine**

**1)** R ← "0123456789ABCDEF"
ch ← ""
**Répéter**
ch ← R [ D mod b2 +1 ] + ch
D ← D div b2
**Jusqu'à** D = 0

**R, ch** : *chaine*

**2)** Convert_10_b2 ← ch

**3) Fin Convert_10_b2**

- **REGLES DE DIVISIBILITE :**

Un entier est divisible par :

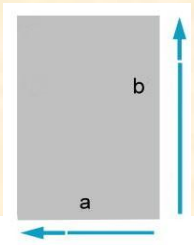
- **2 ( Ξ 5 )**, si son chiffre des unités est divisible **par 2**. (**Ξ par 5** pour 5).
- **3 ( Ξ 9 )**, si la somme des chiffres qui le composent est divisible **par 3**. (**Ξ par 9** pour 9).
- **4 ( Ξ 25 )**, si le nombre composé des deux derniers chiffres est divisible **par 4**. (**Ξ par 25** pour 25).
- **7,** si le nombre **mcdu** est divisible par 7 alors **mcd – 2*u** est divisible par 7, et réciproquement.
- **10**, si son chiffre des unités est égal à **0**.
- **11**, si la différence entre la somme des chiffres de rang pair et celle de rang impair est divisible par **11**.

## ▪ CALCUL D'AIRES $\int_a^b F(x)\,dx$ - METHODES DES RECTANGLES - :

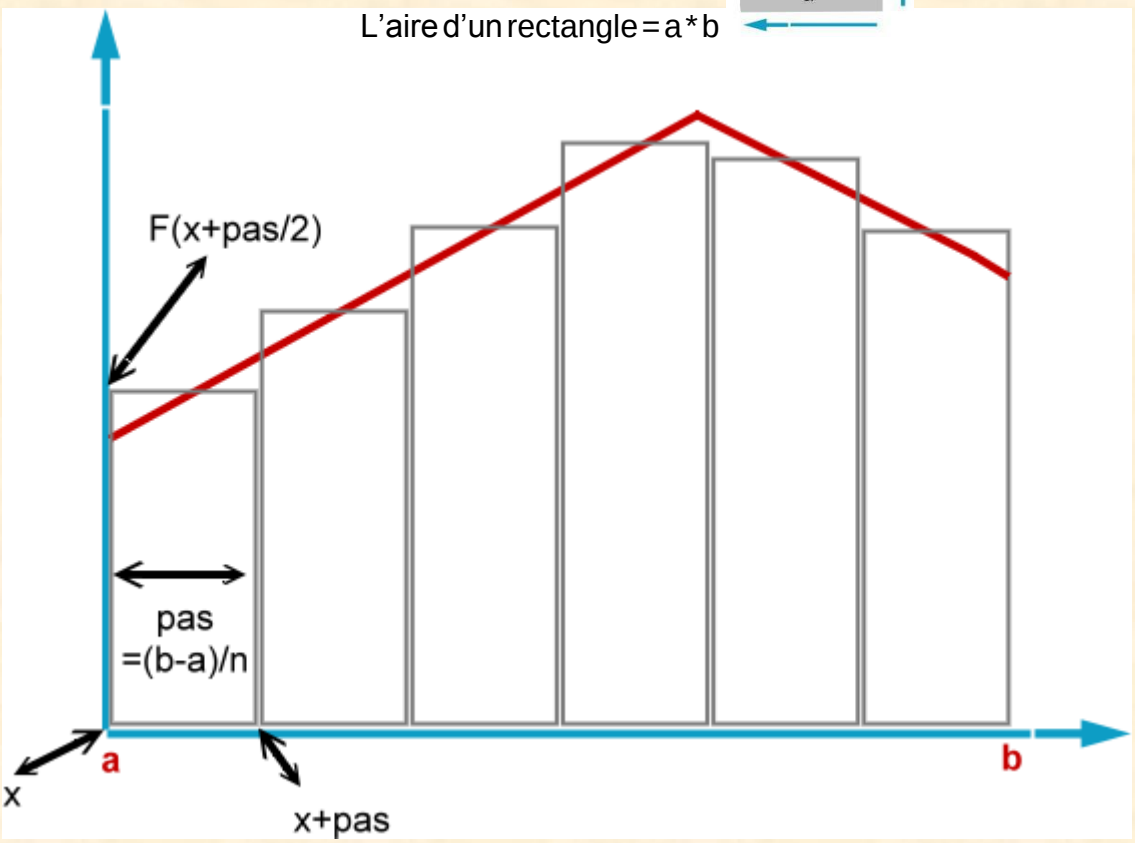
**L'intégrale** d'une fonction réelle positive est la valeur de l'aire du domaine délimité par l'axe des abscisses et la courbe représentative de la fonction.

**Rapprocher la valeur de :** $\int_a^b F(x)\,dx$ Subdiviser l'intervalle **[a,b]** en **n** intervalles avec le **pas** pour chaque intervalle = **b-a/n.**

**Rapprocher** $\int_a^{a+pas} F(x)\,dx \ldots \int_{b-pas}^{b} F(x)\,dx$




L'aire d'un rectangle = a * b




**Algorithme**

**0) Fonction Rectangles (a, b : réel ; n : entier) : réel**

**1)** integrale ← 0

pas ← (b-a) / n

x ← a + pas/2

**Pour** i **De** 1 **A** n **Faire**

integrale ← integrale + **Fn** F(x) * pas

x ← x + pas

**FinPour**

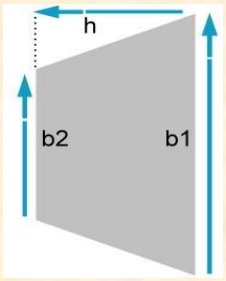
**2)** Rectangles ← integrale

**3) Fin Rectangle**

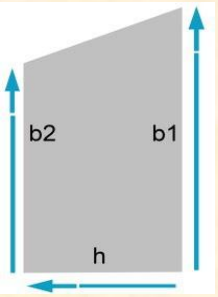
**i** : *entier.*
**pas, integrale, x** : *réel.*
**F** : *fonction qui calcule F(x).*

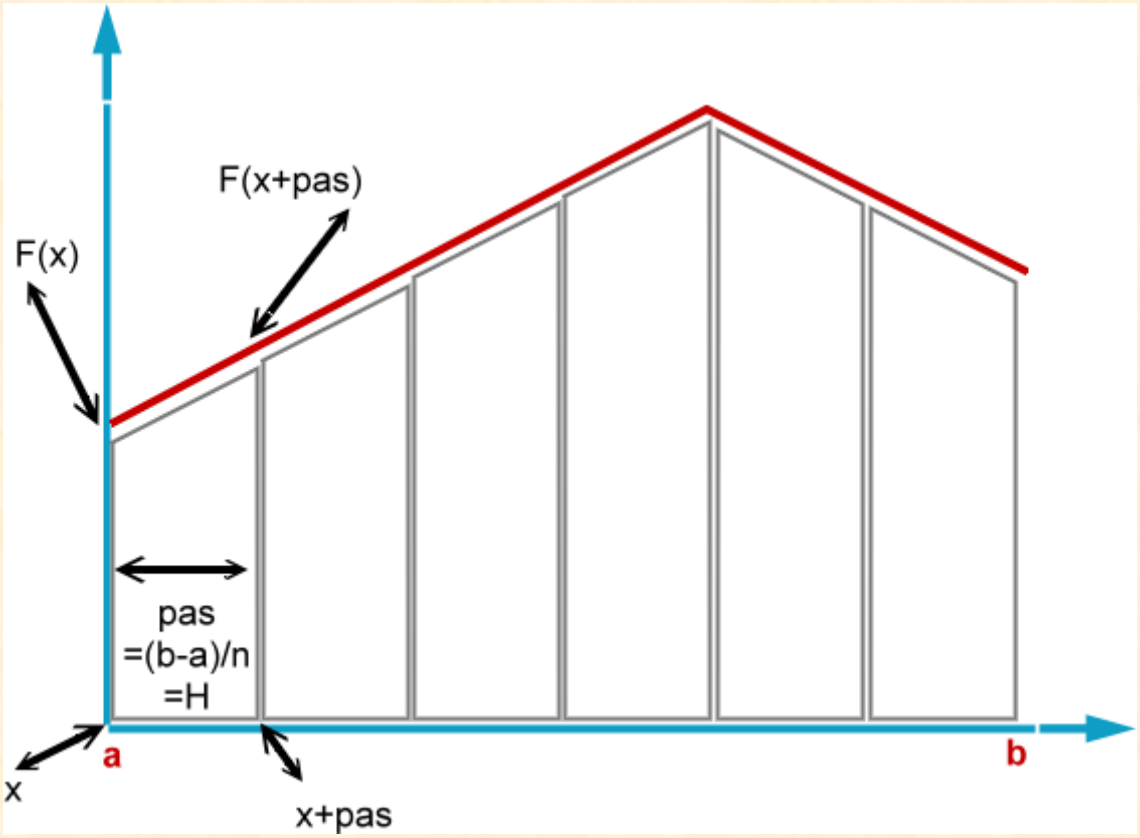
- **CALCUL D'AIR $\int_a^b F(x)\,dx$ - METHODES DES TRAPEZES - :**

L'aire d'un trapèze = (b1+b2)*h/2




ou




**Algorithme**

**0) Fonction Trapèzes (a, b : réel ; n : entier) : réel**

**1)** integrale ← 0

pas ← ( b – a ) / n

x ← a

**Pour** i **De** 1 **A** n **Faire**

integrale ← integrale + ((**Fn** F(x) + **Fn** F(x+pas)) / 2) * pas

x ← x + pas

**FinPour**

**2) Trapèzes** ← integrale

**3) Fin Trapèzes**

**i** : *entier*.
**pas, integrale, x** : *réel*.
**F** : *fonction qui calcule F(x)*.

# Les algorithmes avancés

## ▪ TOURS DE HANOÏ :

déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire » et ceci en un minimum de coups, tout en respectant les règles suivantes :

- on ne peut déplacer plus d'un disque à la fois,
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

**nombre** : nombre de disques utilisés

**but** : emplacement de destination

**dep** : emplacement de départ

**int** : emplacement intermédiaire

**Algorithme**

**0) Procédure Hanoi (nombre, dep, but, int : entier)**

**1) Si** nombre > 0 **Alors Proc** Hanoi (nombre - 1, dep, int, but)

Ecrire (''Bouger un disque de '', dep, '' à '', int)

**Proc** Hanoi (nombre - 1, but, dep, int)

**FinSi**

**2) Fin Hanoi**

$$Nombre\ de\ déplacements = 2^{nombre} - 1$$

## ▪ TRI RAPIDE :

Elle consiste à placer un élément d'un tableau d'éléments à trier (appelé **pivot**) à sa place définitive en permutant tous les éléments de telle sorte que tous ceux qui lui sont inférieurs soient à sa gauche et que tous ceux qui lui sont supérieurs soient à sa droite. Cette opération s'appelle partitionnement. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit triés.

Supposons que nous ayons à trier le vecteur t[Deb. . Fin] avec (Deb < fin), nous allons segmenter ce vecteur en trois sous vecteurs tels que:


tous les éléments se trouvant à gauche de Place (Place ∈ [Deb. .Fin]) soient inférieurs ou égaux à t[place] et que ceux se trouvant à droite de place soient supérieurs à t[place].

On peut écrire: **t [ Deb .. place-1 ] ≤ t [ place ] < t [ place+1 .. Fin ]**

**Algorithme** **0) Procédure Tri_Rapide (deb, fin : entier ; Var t : vecteur)**

```
1) Si (fin > deb) Alors                    { Si fin > deb alors le tableau nécessite d'être trié. }
        pivot ← t [deb ]                   { Choisir le 1er élément du tableau comme pivot. }
        D ← deb + 1
        F ← fin
        Tantque D ≤ F Faire
            Si t [D ] ≤ pivot  Alors D ← D+ 1
                               Sinon
                                   Si pivot ≤ t*F+ Alors  F ← F - 1
                                                  Sinon
                                                   Si D < F Alors
                                                            aux ← t [ D]
                                                            t [ D ] ← t [ F ]
                                                            t [ F ] ← aux
                                                            D ← D + 1
                                                           F ← F - 1
                                                   FinSi
                                   FinSi
            FinSi
        FinTantque
        { Mettre le pivot à la position adéquate c.à.d. à la suite des éléments qui lui sont inférieurs. }
        aux ← t [ F ]
        t [ F ] ← t [ deb ]
        t [ deb ] ← aux
        Proc Tri_Rapide (deb , F – 1 , t)   { Trie le sous tableau à gauche. }
        Proc Tri_Rapide (F + 1, fin , t)    { Trie le sous tableau à droite. }
    FinSi
2) Fin Tri_Rapide
```

{ **D, F, pivot, aux** : *entier* }

## ▪ PROBLEME DU VOYAGEUR DE COMMERCE :

Etant donné un ensemble de villes (des points dans le plan), il s'agit de trouver le plus court chemin fermé passant une fois et une seule par chacun des points.

**Résolution en Turbo Pascal :**

```
PROGRAM Voyageur;
CONST nmax = 20 ;
TYPE  mat  = array [1 .. nmax, 1.. nmax] of real;
      vect1 = array [1 .. nmax] of string;
      vect2 = array [1 .. nmax] of boolean;
VAR   dist          : mat;
      ville         : vect1;
      vv            : vect2;
      vd            : string;
      dp, dmax, dmin : real;
      i, j, n, vc, vi, vs : integer;
BEGIN
```

***dist*** ▶ *Distance entre 2 villes.*
***vv*** ▶ *Ville visitée.*
***vd*** ▶ *Ville de départ.*
***vc*** ▶ *Ville courante.*
***vi*** ▶ *Ville initiale.*
***vs*** ▶ *Ville suivante.*
***dp*** ▶ *Distance totale parcourue.*
***dmax*** ▶ *Distance maximale entre villes.*
***dmin*** ▶ *Distance minimale entre 2 villes.*

```
    writeln('Entrer le nombre de villes à visiter: ');
    readln (n);
```

*Entrer le nombre de villes à visiter (≤ nmax).*

```
    writeln ('Entrer les noms des villes à visiter');
    for i := 1 to n do
                begin
                    readln ( ville[i] );
                    vv[i] := false;
                end;
```

*Entrer les noms des villes et Initialiser le vecteur des villes visitées à Faux.*

```
    writeln ('Entrer le nom de la ville de départ: ');
    readln (vd);
```

*Entrer le nom de la ville de départ.*

```
    dmax:=0;
```

*Initialiser la distance maximale entre les villes à 0.*

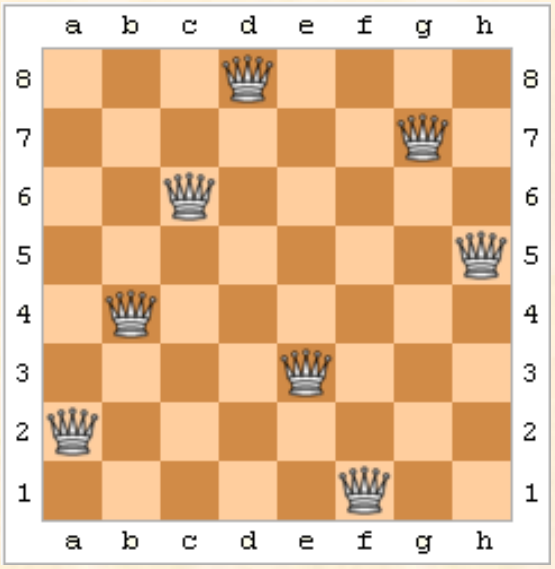
```
  writeln ('Entrer les distances entres les villes');
    for i := 1 to n do
                for j := 1 to i do
                        if i = j then dist[i,j] := 0
                            else
                                begin
                                    write ('Entre ' ,ville[i], ' et ',ville[j],  ': ');
                                    readln ( dist[i,j]);
                                    dist[j,i] := dist[i,j];
                                    if dmax < dist[i,j] then dmax := dist[i,j] ;
                                end;
```

*Entrer les distances entre les villes.*

```
vc := 1;
while ville[vc] <> vd do vc := vc+1;      { Marquer la ville de départ comme ville visitée. }
vv[vc] := true;

vi := vc;

dp := 0 ;        { Initialiser la distance parcourue à zéro. }

writeln('Ville de départ : ',vd);
writeln('L''itinéraire suivi par le voyageur est le suivant : ');
write( vd, '       ');

        { Recherche de la ville suivante "vs" la plus proche et calcul de la distance parcourue. }
 for i := 1 to n-1 do
        begin
            dmin := dmax + 1;
            for j :=1 to n do
                        if (dist[vc,j] < dmin) and (vv[j] = false) then
                                                                    begin
                                                                        dmin := dist[vc,j];
                                                                        vs := j;
                                                                    end;
            dp := dp + dist[vc,vs];
            write    (ville[vs],'    ');
            vv[vs] := true;
             vc := vs;
        end;

dp := dp + dist[vi,vs];
writeln (vd);
write ('La distance totale parcourue est de : ', dp, ' Km.');
END.
```

## PROBLEME DES HUIT DAMES :

Le but du **problème des huit dames**, est de placer huit dames d'un jeu d'échecs sur un échiquier de 8×8 cases sans que les dames ne puissent se menacer mutuellement. Par conséquent, deux dames ne devraient jamais partager la même rangée, colonne, ou diagonale.




### Résolution en Turbo Pascal :

```
PROGRAM Huit_Dames;
CONST Nbr_dames = 8 ;
TYPE  vecteur = array [1 .. Nbr_dames] of integer;
VAR   RESULTAT : vecteur;
      solution, LIGNE : integer;
```

*Dans Le tableau RESULTAT **LIGNE** indique la ligne de l'échiquier, **RESULTAT [LIGNE]** indique la colonne de l'échiquier.*

```
PROCEDURE Affiche (RESULTAT : vecteur; Nbr_dames : integer);
VAR LIGNE, COLONNE : integer;
BEGIN
    for LIGNE := 1 to Nbr_dames do
                begin
                    for COLONNE := 1 to Nbr_dames do
                                        If COLONNE = RESULTAT[LIGNE] then write('1')
                                                                     else write('0');
                    writeln;
                end;
    writeln ;
END;
```

```
PROCEDURE Ajoute_Dame (RESULTAT : vecteur ; dames : integer);
LABEL suivant;
VAR LIGNE, COLONNE : integer;
BEGIN
    if dames = Nbr_dames + 1 then
                                    begin
                                        Affiche(RESULTAT, Nbr_dames);
                                        solution := solution+ 1;
                                    end;

    for LIGNE := 1 to Nbr_dames do
        begin
            for COLONNE := 1 to dames do
                if ( RESULTAT[COLONNE] = LIGNE ) OR
                   ( abs( RESULTAT[COLONNE] - LIGNE ) = abs( COLONNE - dames ))
                        then GOTO suivant ;

            RESULTAT[dames] := LIGNE;
            Ajoute_Dame(RESULTAT, dames+1);

            suivant :
        end;
END ;
```

*Le test **RESULTAT[COLONNE] = LIGNE** permet d'éviter de mettre une dame sur la même ligne ou colonne.*

*Le test **abs( RESULTAT[COLONNE] - LIGNE ) = abs( COLONNE - dames )** permet d'éviter de mettre une dame sur la même diagonale.*

```
BEGIN
    for LIGNE := 1 to Nbr_dames do RESULTAT[LIGNE] := 0 ;
    solution := 0;
    Ajoute_Dame (RESULTAT, 1);
    writeln ('Solution =', solution);
END ;
```