



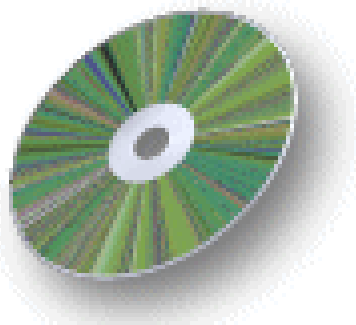
MBODJ.SYSTEM®

Adama MBODJI

COURS ET EXERCICES CORRIGES

Les secrets de la programmation

Tableaux & vecteurs
Algorithmes de tris
Chaînes de caractères
Listes linéaires
Piles & Files
Arbres
Fichiers



CADEAU SUR LE CD-ROM

MSAlgoPascal : Traducteur automatique d'algorithme en langage Pascal

Dédicace

Je dédie cet ouvrage à mon père

Souleymane MBODJI

Sommaire

Dédicace	2
Sommaire	3
Avant-propos	5
Préface	6
1. Introduction générale	7
1.1 Notions d'algorithme et de programme	7
1.2 Structure générale d'un algorithme	8
1.3 Les outils	9
2. Présentation de MSAalgoPascal	19
2.1 Configuration requise	19
2.2 Installation de MSAalgoPascal	19
2.3 Pas à pas sur MSAalgoPascal	21
3. Tableaux et vecteurs	31
3.1 Tableaux simples (une dimension)	31
3.2 Tableaux à plusieurs dimensions	32
3.3 Opérations classiques sur les tableaux	32
Exercices	38
4. Les algorithmes de tri	40
4.1 Définition	40
4.2 Tri par sélection	40
4.3 Tri par insertion simple	42
4.4 Tri à bulles	42
5. Les chaînes de caractères	44
5.1 Définition	44
5.2 Accès à un caractère	44
5.3 Concaténation de chaînes	44
5.4 Fonctions utiles pour le traitement des chaînes	45
Exercices	46
6. Les enregistrements	47

6.1 Déclaration d'une structure	47
6.2 Affectation de valeurs	48
Projets	50
 7. L'allocation dynamique : listes, piles, files	 51
7.1 Définition	51
7.2 Déclaration d'une variable de type pointeur	51
7.3 Allocation et Désallocation	52
7.4 Les listes	53
Projets	67
 8. Les arbres	 72
8.1 Définition de quelques concepts	73
8.2 Déclaration d'un arbre binaire	74
8.3 Opérations classiques sur les arbres binaires	74
Exercices	85
 9. Les fichiers	 86
9.1 Le concept de fichier	86
9.2 L'accès aux fichiers	86
9.3 La création d'un fichier typé	87
9.4 La création d'un fichier texte	88
9.5 La lecture d'un fichier typé	88
9.6 La lecture d'un fichier texte	88
9.7 Les types d'organisations	89
9.8 Les opérations classiques dans un fichier	90
9.9 La mise à jour d'un fichier	92
Exercices	95
 Projet final	 96
 Bibliographie	 99

Avant-propos

Ce guide de l'étudiant est un ouvrage destiné aux étudiants du premier cycle d'informatique, aux débutants et à tous ceux qui veulent connaître les notions de bases de la programmation.

Il aborde brièvement les thèmes les plus classiques et les plus utilisés en informatique : les tableaux, les algorithmes de tri, les chaînes de caractères, les enregistrements, les listes linéaires, les piles, les files, les arbres et les fichiers. De nombreux exercices et projets y sont disponibles et les corrigés sont au niveau du CD-Rom.

Nous avons remarqué que les compétences des débutants se limitent à l'analyse des problèmes et à l'écriture des algorithmes ; dès lors, il leur semble fastidieux de codifier, de corriger les syntaxes et de déboguer leurs programmes. La difficulté majeure qu'ils rencontrent le plus souvent est d'effectuer correctement le passage d'un algorithme au code correspondant dans un langage de programmation donné, en particulier le Pascal.

C'est dans cet optique que nous avons inclus dans ce guide un **compilateur** du nom de **MSAlgoPascal[®]** qui a la possibilité de :

- vérifier les expressions parenthésées.
- vérifier les clauses arborescentes.
- vérifier la déclaration des variables, des types, des procédures et des fonctions.
- envoyer des messages d'erreur à l'utilisateur si nécessaire.
- effectuer la traduction automatique de l'algorithme vers le langage Pascal.

Les étudiants auront la possibilité d'exécuter tous les algorithmes étudiés en classe en passant par le générateur de codes.

MSAlgoPascal[®] a été largement expérimenté en 2004 au niveau des étudiants de première année de l'Institut de Formation Professionnelle (I.F.P) et les résultats obtenus sont dignes d'intérêt.

Je remercie vivement Gervais MENDY, Ousmane DIENE, Emmanuel KABOU, Samuel OUYA, M. ALAOUI pour la relecture et la correction de ce guide.

Que M. Moussa BA professeur du langage Pascal à l'I.F.P, qui après avoir mis en pratique le logiciel MSAlgoPascal[®] avec ses étudiants de première année et qui a largement participé à son débogage, trouve ici l'expression de ma profonde gratitude.

Mille mercis à ma sœur Sourouro Belly MBODJ et à mon oncle Abou NDIANOR pour le soutien et les critiques qu'ils ont apporté.

Pour finir, je ne citerai pas de noms, au risque d'en omettre certains, pour exprimer mes sincères remerciements à tous ceux qui, de près ou de loin, ont contribué à la réalisation de MSAlgoPascal[®].

Adama MBODJI, Janvier 2005

Préface

Chapitre I Introduction Générale

1.1 Notion d'algorithme et de programme

Aujourd'hui avec le développement des sciences et des technologies, nous sommes parvenus au monde de la programmation à la 4^{ème} génération. Nous sommes passés du binaire à l'assembleur puis des langages procéduraux aux langages événementiels et objets.

Derrière toutes ces innovations, aussi complexes qu'elles soient, nous répétons toujours le même processus pour résoudre un problème en informatique. Cette résolution peut être schématisée ainsi qu'il suit :

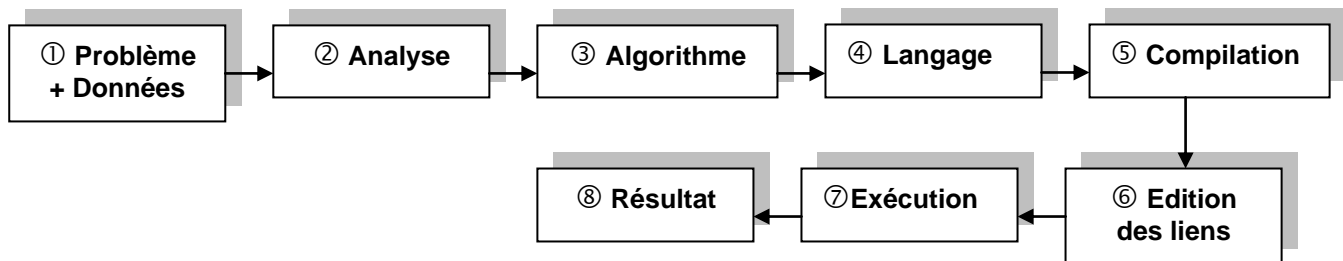


Figure 1. Cycle de vie de la résolution d'un problème en informatique.

Nous pouvons retenir que la résolution d'un problème en informatique passe par la production d'un texte appelé algorithme. Ce dernier décrit l'ensemble des opérations élémentaires qui seront exécutées par un ordinateur *via* un langage de programmation pour obtenir la solution informatique.

« L'algorithme est la description d'un ensemble fini d'actions et d'informations élémentaires destinées à être exécutées par un ordinateur pour résoudre un problème.

L'algorithme est une suite d'actions ordonnées sur un ensemble fini d'objets. Il doit être défini sans ambiguïté, et son exécution doit s'arrêter après un nombre fini d'opérations élémentaires ». DIOP, **Kéba.- Algorithmique et Structures de données Tome 1.- Presse de l'université de l'UNIS page 8.**

Supposons que l'on veuille automatiser la résolution d'une équation du second degré. Pour ce faire, il faut la « mettre » dans un modèle général qui est le suivant : $AX^2 + BX + C = 0$ (avec $A \neq 0$). Résoudre ce problème *via* l'ordinateur revient à lui faire « comprendre » les différentes étapes qu'il doit suivre pour aboutir à un résultat. Ce phénomène logique est appelé **Algorithme**. Ce dernier une fois traduit dans un langage de programmation demeure toujours « incompréhensible » par l'ordinateur. Il faut cependant passer à une autre traduction qui convertit le code écrit en langage machine : c'est la **compilation**.

Ce présent manuel (guide de l'étudiant) présente les algorithmes classiques les plus utilisés en informatique. Il nous permet d'écrire correctement des algorithmes et de les traduire en langage Pascal. Le passage de l'algorithme vers le langage de programmation sera automatique grâce au générateur de code **MSAlgoPascal**.

1.2 Structure générale d'un algorithme

```
Algo ou Algorithme ou Programme Nom_Algo
Constante ou Const nom_constante = Valeur
Type
    % définition de type de données %
Variable ou Var
    % liste des variables %

Procédure Nom_Procédure1 (donnée Nom_Var1 : Type1 ; Résultat Nom_Var2 : Type2)
Début
    % Instruction %
FinProcédure

Fonction Nom_Fonction (donnée Nom_var i : Type i) : Type de résultat
Début
    % Instruction %
    Nom_Fonction = Valeur retour
FinFonction

% PROGRAMME PRINCIPAL %
Début
    % Instruction %
FinAlgo ou FinAlgorithme ou FinProgramme
```

Donc la structure générale d'un algorithme se présente ainsi :

- en-tête de l'algorithme.
- corps encore appelé **bloc**.

Un algorithme commence toujours par les mots réservés suivants : **Algo**, **Algorithme** ou **Programme** suivi de son Nom. Son nom est un identificateur ; il permet de nommer l'algorithme.

Un bloc est toujours constitué d'une partie déclarative et d'une partie instruction. Il est subdivisé en quatre (04) sous parties :

- déclaration de constantes.
- déclaration de types.
- déclaration de variables.
- déclaration de procédures et de fonctions.

Tous les algorithmes doivent se terminer par :

- **FinAlgo** s'ils débutent par Algo.
- **FinProgramme** s'ils débutent par Programme.
- **FinAlgorithme** s'ils débutent par Algorithme.

1.2.1 Les identificateurs

Les identificateurs sont des mots qui servent à désigner, à nommer et à identifier les entités, les objets, les actions, les procédures et les fonctions manipulés dans un programme.

Les identificateurs ne se créent pas n'importe comment car ils doivent respecter l'ordre des **diagrammes syntaxiques** encore appelés **diagrammes de CONWAY** : un identificateur débute

toujours par une lettre ou un caractère de soulignement qui peut être suivi de lettres, de chiffres ou de caractères de soulignement de façon optionnelle.

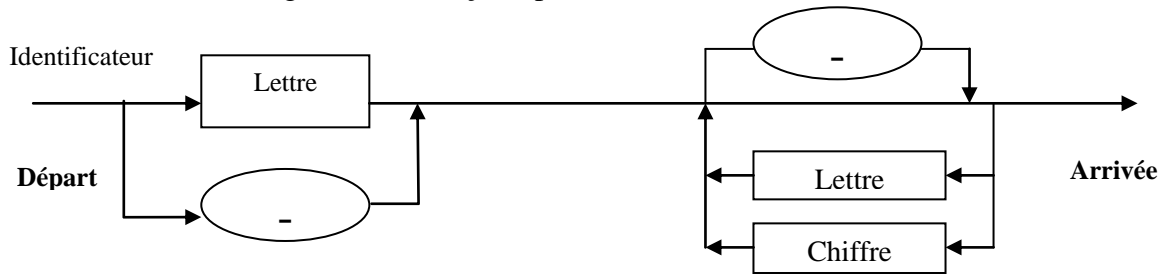


Figure 2. *Diagramme syntaxique d'un identificateur* NOUATIN, Théophile.- *Les bases de la PROGRAMMATION avec C++*.- ITSS SOFT page 34.

1.2.2 Les expressions

Elles sont formées par une combinaison d'opérateurs et d'opérandes. Les opérandes peuvent être des constantes, variables, chaînes de caractères. Les opérateurs sont arithmétiques, logiques et booléens, de chaînes de caractères, d'ensemblistes, etc.

1.2.3 Les constantes

Ce sont des identificateurs qui gardent toujours la valeur qu'on leur a attribué durant tout l'algorithme. Leur valeur ne peut être modifiée. Une constante se déclare de la manière suivante :

Const ou **Constante** **Identificateur** = **Valeur** ou **expression**

1.2.4 Les variables

Une variable est une zone mémoire où l'on peut stocker des informations identifiées par l'identificateur. Elle peut être modifiée dans le corps de l'algorithme. Une variable est déclarée *via* l'instruction Var ou Variable suivi de l'identificateur et du type de la variable.

1.2.5 Les commentaires

Dans l'algorithme, pour éclaircir certains passages, l'utilisateur (le programmeur) peut y « glisser » des commentaires. Les commentaires débutent et se terminent par le symbole %.

Remarque : Pour afficher le symbole % au niveau de l'algorithme en tant que caractère, il faut nécessairement le faire précéder de l'esperluette &. Exemple : Afficher "10 &%"

1.3 Les outils

1.3.1 Les opérateurs arithmétiques

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
Div	Division entière
Mod ou Modulo	Modulo (Reste de la division entière)

1.3.2 Les opérateurs logiques et binaires

Opérateur	Signification
Pas	Négation
Et	Et logique
Ou	Ou logique
OuExclusif	Ou Exclusif logique

Les opérandes associés à ces opérateurs peuvent être des entiers (on parle alors d'opérateurs binaires) ou des booléens (on parle d'opérateurs logiques).

1.3.3 Les opérateurs de chaînes de caractères

Opérateur	Signification
+	Concaténation de deux chaînes de caractères.

Pour concaténer deux chaînes de caractères, nous utilisons l'opérateur « + ». Le résultat est du type chaîne.

1.3.4 Les opérateurs relationnels

Opérateur	Signification
=	Egal
<>	Différent
<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à

Ces opérateurs fournissent toujours un résultat booléen (Vrai ou Faux).

1.3.5 Les opérateurs sur les pointeurs

Opérateur	Signification
^ ou ↑	Crée un pointeur sur une variable

« La notion fondamentale de pointeur est de nommer une entité (une variable) dont le contenu n'est pas une donnée de l'application, mais une adresse mémoire. A cette adresse est stockée la donnée. C'est donc un accès direct. ». BINZINGER, Thomas.- Borland DELPHI 6.- CampusPress page 258.

Les pointeurs permettent de créer des structures dynamiques comme les listes chaînées, les arbres, etc. Le mot réservé NIL (Not In List) est utilisé pour marquer la fin de la liste.

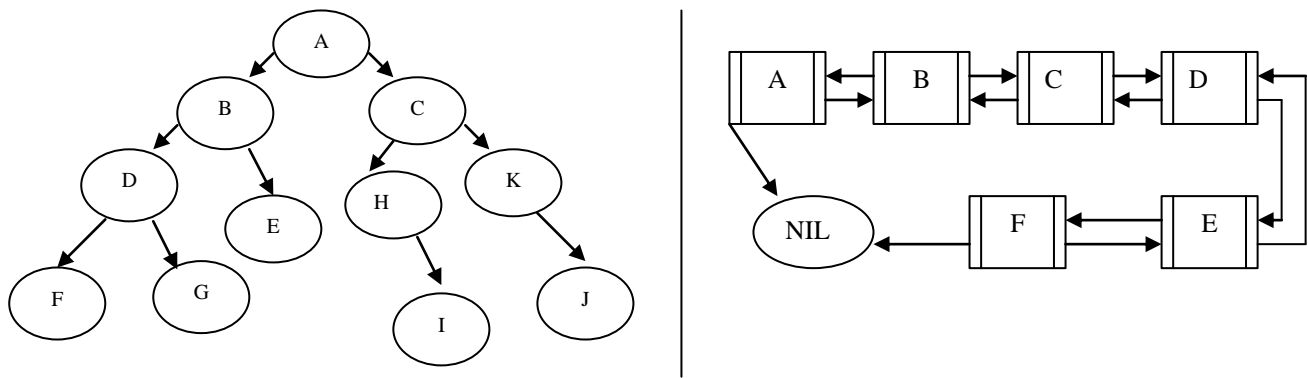


Figure 3. Arbre binaire et structure chaînée (bidirectionnelle) formées grâce aux pointeurs

Les éléments d'un arbre se nomment **nœuds** et ceux d'une liste **cellules**.

Pour l'arbre, le nœud « A » représente la tête, les nœuds « B », « C », « D », « H » et « K » sont intermédiaires et les nœuds « E », « G », « I », « J » et « F » se nomment feuilles.

Pour la liste chaînée, les cellules « A » et « F » représentent à la fois la tête et la queue. Les autres cellules sont appelées cellules intermédiaires.

1.3.6 Les types de données

1.3.6.1 Les entiers

En algorithmique, nous utiliserons ces cinq types de données.

Type	Domaine de définition	Taille mémoire
EntierCourt	[-128, 127]	1 octet
Entier	[-32768, 32767]	2 octets
EntierLong	[-2147483648, 2147483647]	8 octets
Mot	[0, 65535]	2 octets
Octet	[0, 255]	1 octet

1.3.6.2 Les caractères

Le type caractère correspond à l'ensemble des caractères de la table ASCII.

Variable

Touche : Caractère

La variable Touche peut recevoir n'importe quel caractère appartenant à la table ASCII.

Nous pouvons par exemple écrire : Touche = "A" ; Touche = ASCII (65)

1.3.6.3 Les booléens

Les variables du type booléen ne peuvent recevoir que deux valeurs (Vrai ou Faux).

NB : Généralement, Faux peut être codifié par 0 et Vrai par 1.

1.3.6.4 Les énumérés

Un type énuméré permet de définir une donnée correspondant à un ensemble ordonné d'éléments ou de valeurs. Chaque élément est défini au moyen d'un identificateur.

Exemple : **Type** Sexe = (Masculin, Féminin)

1.3.6.5 Les intervalles

Un type intervalle permet de définir des données en fournissant simplement la borne inférieure et la borne supérieure d'un ensemble d'entiers, de chaînes de caractères ou d'éléments définis de type énuméré.

Exemple : Type Chiffre = 0. . 9

1.3.6.6 Les réels

Le langage Pascal met à notre disposition cinq types de réels différents. En algorithmique, nous n'utiliserons que le type réel standard.

Type	Domaine de définition	Taille mémoire
Réel	$[10^{-38}, 10^{38}]$	6 octets

1.3.6.7 Les chaînes de caractères

Un type chaîne de caractères permet de définir des données correspondant à une suite de caractères (appelée chaîne de caractères) dont la longueur peut être spécifiée dans la définition. Si elle n'est pas spécifiée, le système lui attribue une longueur égale à 255 caractères.

Exemple : Variable Expression = Chaîne [15]

Dans cet exemple, la variable **Expression** a été spécifiée comme étant un type chaîne pouvant contenir au maximum 15 caractères.

1.3.6.8 Les tableaux

Un type tableau permet de définir des données composées d'un nombre fixe d'éléments ayant tous le même type.

Syntaxe Générale : NomDuTableau : Tableau [Min .. Max] de Type avec (min > 0)

Définition d'un tableau

```
Tab1 : Tableau [1...10] de Entier
Tab2 : Tableau [1...3,1...8] de Réel
Tab3 : Tableau [1...5,1...5] de Caractère
```

Dans cet exemple, Tab1 est un vecteur de 1 à 10 éléments de type entier. Tab2 est une matrice contenant des réels et Tab3 contient des caractères. Pour se positionner à une cellule du tableau, il suffit d'indiquer le numéro de cellule. Ce numéro commence par l'indice 1.

Exemple : Tab1 [1] = 3
 Tab2 [1,1] = 3.1415
 Tab3 [5,1] = "A"

Représentation physique d'un tableau de caractères

Indice sur le tableau	1	2	3	4	5	6	7	8	9	10	11
	M	B	O	D	J	S	Y	S	T	E	M

Tab [1] = "M"
 Tab [11] = "M"
 Tab [3] = "O"
 Tab [25] = Erreur ! Dépassement de capacité

NB : Si l'indice d'un tableau dépasse la borne maximale de sa déclaration, un message d'erreur est envoyé à l'utilisateur par le compilateur pour lui signifier que cette zone n'appartient pas au tableau. Ce phénomène peut provoquer un blocage de l'ordinateur.

1.3.6.9 Les enregistrements

Le type enregistrement permet de définir des données composées d'un nombre fixe ou variable d'éléments pouvant être de types différents. Chaque élément d'un enregistrement est appelé un CHAMP.

Exemple : **Type** SClient = **Structure**
 Nom : Chaîne [15]
 Prénom : Chaîne [20]
 FinStructure

A partir de ce type, nous pouvons définir une variable enregistrement ainsi :

Variable Client : SClient

Pour accéder à une valeur d'un champ, il suffit de faire précéder le nom de ce champ par l'identificateur du type enregistrement suivi d'un point.

Exemple :

Client.Nom ← "MBODJI"
 Client.Prénom ← "Oumar Soulé"

1.3.6.10 Les ensembles

Le type ensemble permet de définir des données représentant une collection d'objets de même type (entier, booléen, caractère, énuméré ou intervalle). Nous pouvons spécifier un ensemble vide en utilisant [].

Exemple de déclaration : Variable
 Chiffre : Ensemble De 0..9
 Lettre : Ensemble De "A" .. "Z"

L'ensemble Lettre peut recevoir les valeurs suivantes : Lettre = ["I", "U", "O", "A", "E", "Y"]

1.3.6.11 Les fichiers

Les fichiers représentent une collection de données de même type. Celles-ci peuvent être par exemple des enregistrements (structures), des entiers, des réels, etc. Les informations sont généralement stockées sur disque.

On distingue 3 types de fichiers :

1. Les fichiers typés qui sont généralement des fichiers d'enregistrement. Pour déclarer un fichier de ce type, il suffit de spécifier les mots réservés **FICHIER DE** suivis du type d'enregistrement.

Exemple : Type SClient = **Structure**
 Nom : Chaîne [15]
 Prénom : Chaîne [20]
 FinStructure
 FichClient = **Fichier de** SClient

Le fichier est une collection d'enregistrements de type SClient.

2. Les fichiers non typés, déclarés simplement en utilisant le mot réservé **FICHIER**, se distinguent par un accès direct aux informations stockées sur le disque.

Exemple : Type Fich = **FICHIER**

3. Les fichiers Texte permettent de stocker des informations de taille variable. Elles sont séparées les unes des autres par des identificateurs de fin de ligne (caractère retour chariot). Pour déclarer un fichier texte, il suffit de spécifier le mot réservé **TEXTE**.

Exemple : FichText : Fichier de **TEXTE**

Le fichier FichText est un fichier de type Texte.

1.3.6.12 Les pointeurs

Un type pointeur permet de définir une variable dont la valeur est l'adresse en mémoire d'une autre variable. Pour déclarer un type pointeur, il suffit d'utiliser le symbole **^** ou **↑** suivi du type de la variable pointée. Dans l'exemple suivant, le type pointeur permet de créer des variables dont le contenu correspond à une adresse pointant sur un enregistrement **ARTICLE**.

Exemple : **Type** Pointeur = **↑** Article
 Article = **Structure**
 Information : Entier
 Precedent : Pointeur
 Suivant : Pointeur
 FinStructure
 Variable
 Point : Pointeur

La variable **Point** contiendra une adresse en mémoire pointant sur un enregistrement de type Article. Dans cet enregistrement, les champs *Precedent* et *Suivant* sont également du type Pointeur. Leurs contenus respectifs seront également une adresse d'un enregistrement de type **Article**.

Une telle définition de structure permet de créer des listes d'éléments comme ci-dessous :

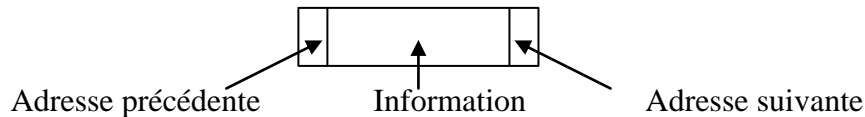


Figure 4. Définition d'une structure chaînée

1.3.7 Les instructions

1.3.7.1 Introduction

Un algorithme se compose d'un certain nombre d'instructions. Ces dernières sont classées par catégories. Nous étudierons tout d'abord les instructions standard traitant l'information : l'assignation, la lecture et l'écriture. Ensuite, celles définissant l'ordre d'exécution d'un programme : la séquence, le choix et la boucle. Nous terminerons par les sous programmes (fonctions et procédures).

1.3.7.2 L'assignation

L'assignation est une instruction qui permet d'attribuer une valeur à une variable ou à un identificateur de fonction afin d'en renvoyer le résultat. Cette valeur retournée, doit être du même type que la variable. La syntaxe est la suivante :

```
<Variable> / <identificateur de fonction> = (Expression)
ou <variable> / <identificateur de fonction> := (Expression)
```

Exemple : `A := A+1 ;` ou `A = A+1`
`Delta = B*B - 4 *A*C`

1.3.7.3 Les instructions d'Entrée/Sortie

- Les instructions d'entrée

L'instruction d'entrée **Saisir** : elle offre à l'utilisateur la possibilité d'entrer plusieurs valeurs (données) dans des variables.

La syntaxe générale est la suivante : `SAISIR (<Nom_Var1>, <Nom_Var2>, ... <Nom_VarN>)`

- Les instructions de sortie

Nous avons deux types d'instructions de sortie : **Afficher** et **AfficherLigne**.

L'instruction **Afficher** affiche les données et reste sur la même ligne ; tandis que **AfficherLigne** place le curseur après l'affichage à la ligne suivante (retour chariot).

Syntaxe générale :

```
Afficher ou AfficherLigne (<Nom_Var1>, . . . , <Nom_Var2>)
```

1.3.7.4 Les instructions conditionnelles

Les instructions conditionnelles permettent de modifier l'ordre de la séquence d'un algorithme. Dans une séquence, les instructions sont exécutées les unes à la suite des autres sans interruption. Nous disposons de deux instructions conditionnelles : SI ... ALORS et SELON ... DE

- **L'instruction SI ... Alors (structure alternative)**

La structure alternative correspond à un choix entre deux possibilités. Suivant la valeur issue de la condition spécifiée dans l'instruction, l'ordinateur exécute une suite d'instructions A ou une suite d'instructions B. En aucun cas, l'ordinateur n'exécute à la fois les instructions A et B.

Syntaxe générale :

SI <Condition> ALORS Instruction FINSI	SI Moyenne < 10 ALORS Afficher « Redouble » FINSI
Ou	Ou
SI <Condition> ALORS Instruction A SINON Instruction B FINSI	SI Moyenne < 10 ALORS Afficher « Redouble » SINON Afficher « Passe » FINSI

Exemple : <Condition> est une expression logique.

- **Le choix multiple**

L'instruction se présente comme une alternative ; en effet, elle n'offre que deux choix possibles dépendant de la valeur d'une condition (valeur vraie ou fausse). En algorithme, nous disposons aussi d'une instruction permettant d'effectuer un choix entre plusieurs décisions.

L'instruction SELON ... DE

Syntaxe Générale

```
SELON <Expression> DE  
    Cas <Constante 1>, <Constante 2>, <Constante 3> : Instruction A  
    Cas <Constante 4>  
    Début  
        Instruction 1  
        ...  
        Instruction N  
    FIN  
CASSINON  
    Instruction B  
FINSELON
```

<Instruction> peut être : soit une instruction, soit un bloc d'instructions. Les comparaisons sont effectuées au fur et à mesure en commençant par la première ; dès qu'une comparaison s'avère vraie, les instructions correspondantes sont exécutées et le programme se branche à l'instruction qui suit FINSELON (les autres comparaisons ne sont donc pas effectuées) ; si aucune comparaison ne s'avère vraie, les instructions correspondant à CASSINON sont effectuées (si la clause CASSINON existe).

1.3.7.5 Les instructions répétitives

Dans un programme plusieurs instructions peuvent se répéter ; il est alors plus intéressant d'écrire les instructions une seule fois et de les exécuter plusieurs fois. Cette action est ce que l'on appelle BOUCLE.

En algorithmique, nous possédons trois Boucles.

- **La boucle REPETER ... JUSQUA**

Cette boucle permet d'exécuter les instructions comprises entre REPETER et JUSQUA jusqu'à ce que la condition du JUSQUA soit vérifiée.

Syntaxe générale :

```
REPETER
    Instruction 1
    Instruction 2
    Instruction N
JUSQUA <Condition>
```

Les instructions allant de 1 à N vont s'exécuter jusqu'à ce que <Condition> soit vérifiée. Mais il faut noter que cette boucle exécute au moins une fois ces instructions avant de tester la condition.

- **La boucle TANTQUE**

C'est la boucle la plus utilisée en informatique. Son fonctionnement est simple. Elle exécute toutes les instructions comprises entre les mots réservés TANTQUE et FINTANTQUE tant que la condition de départ reste vérifiée.

La syntaxe générale est la suivante :

```
TANTQUE <Condition> FAIRE
    Action(s)
FINTANTQUE
```

L'ordinateur commence par vérifier si la condition est vraie. Si c'est le cas, il exécute les instructions de la boucle. Si ce n'est pas le cas les instructions suivant le FINTANTQUE sont exécutées.

NB : Pour éviter une boucle infinie, il faut modifier la variable du test à l'intérieur de la boucle.

- **La boucle POUR**

Cette boucle permet d'exécuter un certain nombre de fois une suite d'instructions.

Syntaxe générale :

```
POUR <Nom_Var> = <Borne minimale> à <Borne maximale> FAIRE
    <Action>
FINPOUR
```

L'ordinateur exécute l'instruction <Action> Borne maximale – Borne minimale +1 fois.

1.3.8 Les sous programmes

1.3.8.1 Définition

Un sous-programme est rédigé de façon telle que son exécution puisse être commandée par un programme. Celui-ci est appelé programme appelant. Il fournit des données au sous-programme et récupère les résultats de ce dernier.

On distingue deux types de sous programmes : les procédures et les fonctions.

La différence est qu'une fonction renvoie une valeur alors qu'une procédure ne renvoie pas de valeur.

1.3.8.2 Les procédures

Les procédures sont composées d'un en-tête de procédure et d'un bloc d'instructions : c'est la partie déclarative et le corps de la procédure. La syntaxe générale est la suivante :

```
Procédure <identificateur> (<liste de paramètres>)  
Début  
    Instruction(s)  
FinProcédure
```

L'appel d'une procédure se fait au sein du programme principal avec une instruction composée de l'identificateur de la procédure suivi des paramètres effectifs. Les paramètres permettent à un programme appelant de transmettre à un sous-programme des données lors de l'appel de ce dernier.

Un sous-programme est rédigé de façon à pouvoir recevoir des données du programme appelant ; cela est possible grâce aux paramètres. Ils sont appelés formels lors de la définition et effectifs lors de l'appel. Il existe deux types de paramètres : les paramètres transmis par valeur et les paramètres transmis par adresse.

1.3.8.3 Les fonctions

Les fonctions sont constituées d'un en-tête de fonction (partie déclaration) et d'un bloc d'instructions (corps de la fonction). Les fonctions effectuent certaines opérations avant de renvoyer un résultat ; elles sont donc appelées dans une expression.

La syntaxe générale d'une fonction est la suivante :

```
Fonction <identificateur> (<liste de paramètres>) : <Type  
résultat>  
Début  
    Instruction(s)  
    <Identificateur> = Valeur retour  
FinFonction
```

Chapitre II Présentation de MSAlgoPascal

Les différents algorithmes que nous verrons par la suite pourront directement être mis en application au niveau du logiciel MSAlgoPascal®. MSAlgoPascal® est un générateur de codes. Il traduit automatiquement un algorithme vers un langage évolué : le **Pascal**.

Suivez pas à pas les notes qui sont signalées pour en tirer profit.

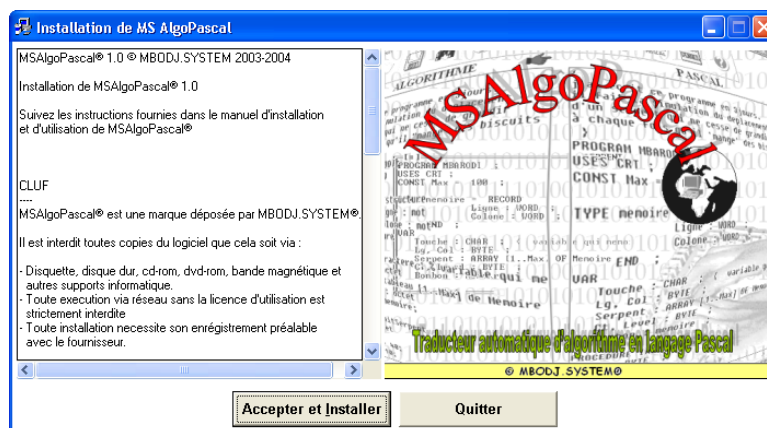
2.1 Configuration requise

Pour une meilleure utilisation du logiciel MSAlgoPascal®, vous devez disposer de :

- Processeur : PC Pentium 100 MHz au minimum
- Système d'exploitation : Parfaitement compatible sur toutes les versions de Windows® sous 32 Bits (*Windows 95, Windows 98, Windows NT, Windows Millenium, Windows 2000 et Windows XP...*).
- Ram (mémoire vive ou centrale) : 32 Mo.
- Définition de l'écran : minimum 256 couleurs.
- Espace disque : 3.5 Mo espace disque.
- Périphérique : Carte son et un lecteur de cd-rom.

2.2 Installation de MSAlgoPascal

Le logiciel MSAlgoPascal® est livré sur cd-rom. Insérez le cd-rom d'installation dans votre lecteur, puis patientez un moment. Cette fenêtre s'affiche en premier lieu.



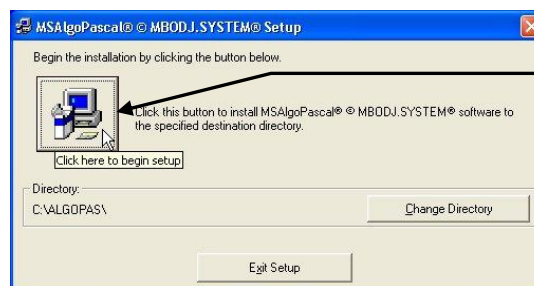
Cliquez sur le bouton **Installer**, pour démarrer l'installation du logiciel MSAlgoPascal®.

Au clic, voici la fenêtre qui s'affiche.



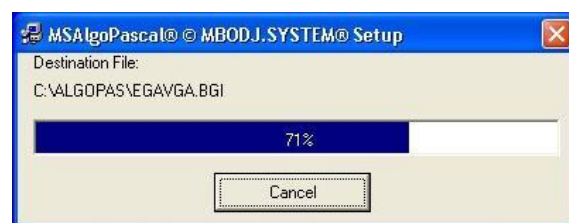
Cliquer sur le bouton **OK**

Une nouvelle fenêtre se présente et vous demande de choisir votre répertoire d'installation. Pour son bon fonctionnement, MSAIgoPascal® doit être installé dans le répertoire **C : \ALGOPAS**.



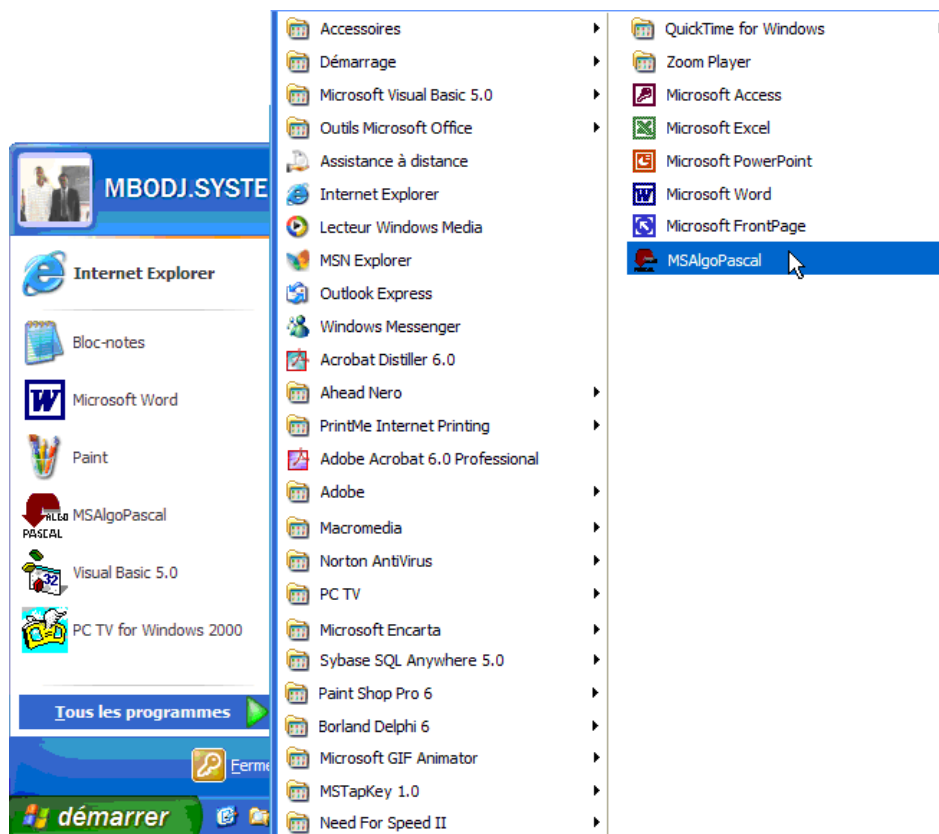
Validez toutes ces opérations en cliquant sur le bouton suivant :

Vous êtes à présent au bout ; la copie des fichiers se lance automatiquement. Patientez jusqu'à ce que la barre des pourcentages soit à 100 %.



Tous les fichiers sont désormais sur votre disque dur, vous pouvez maintenant retirer le cd-rom d'installation de MSAIgoPascal® de votre lecteur.

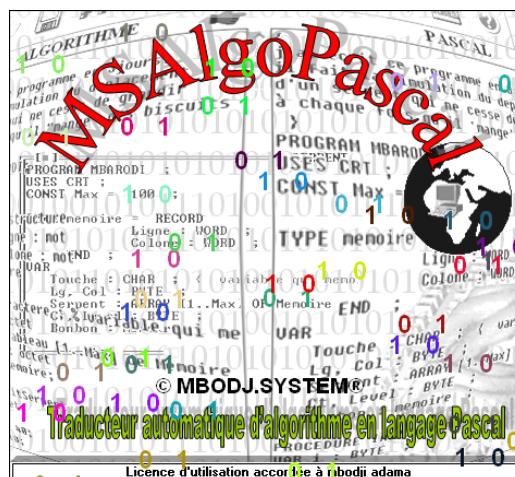
Cliquez sur le bouton Démarrer de Windows, ensuite sur *Tous les programmes (pour les autres versions de Windows (Win9x, WinNT, 2000) vous avez « programmes » au lieu de « Tous les programmes »)* puis sur le groupe de Travail MSAIgoPascal et enfin MSAIgoPascal.



Lancement de MSAlgoPascal® depuis le menu démarrer

2.3 Pas à pas sur MSAlgoPascal®

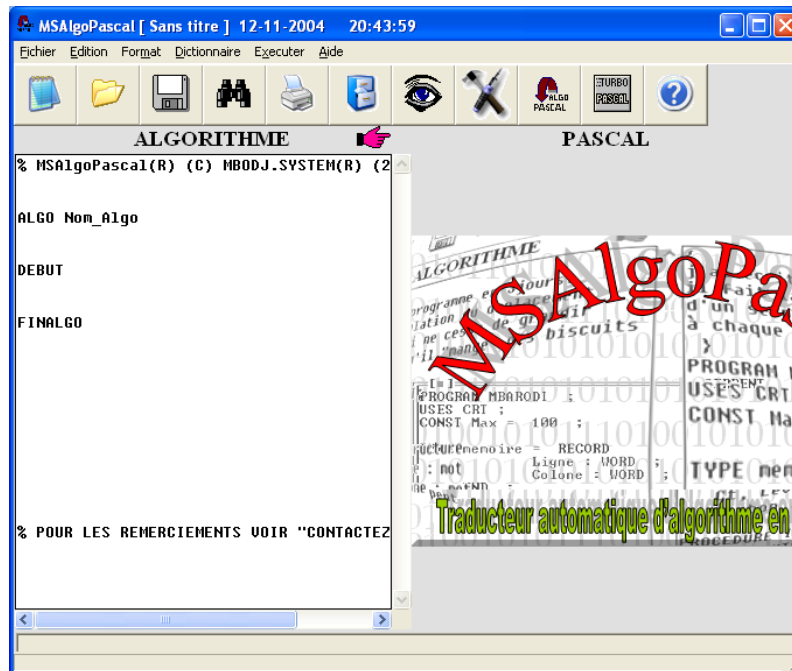
Vous avez enfin installé MSAlgoPascal®. Vous pouvez lancer MSAlgoPascal® à tout moment ; depuis le menu démarrer de Windows. Cette fenêtre s'affiche en premier lieu :



Chargement de la base de données en mémoire centrale

Cette fenêtre d'animation s'affiche pour une petite durée, elle charge la base de données (dictionnaire) en mémoire.

A la fin du traitement, la fenêtre principale s'affichera à l'écran.



Interface de MSAIgoPascal®

Cette fenêtre présente l'environnement général. Elle contient une barre d'outils avec des boutons sur la partie supérieure de la fenêtre et une zone de texte sur la partie gauche. C'est à partir de cette zone de texte (*éditeur de texte*) que l'utilisateur saisit son algorithme.

2.3.1 Comment écrire un algorithme ?

L'écriture d'un algorithme est très simple. MSAIgoPascal® dispose d'un éditeur de texte standard que l'utilisateur utilise pour saisir ses algorithmes. Cette zone de texte est similaire à Bloc-notes de Windows.

Par défaut, un bout de code est affiché pour préparer l'algorithme à venir.

```
ALGO Nom_Algo

DÉBUT

FINALGO
```

Pour saisir un algorithme, l'utilisateur peut changer le nom *Nom_Algo* par le nouveau nom qu'il introduit, ensuite déclarer les *variables*, *types*, *constantes* si nécessaire. Puis entre *Début* et *FinAlgo*, il introduit les instructions de l'algorithme principal.

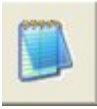
2.3.2 Présentation des objets de l'environnement

2.3.2.1 La barre d'outils



Le logiciel MSAIgoPascal® dispose d'une barre d'outils complète qui permet à l'utilisateur de dialoguer avec le système.

- **Le bouton Nouveau**



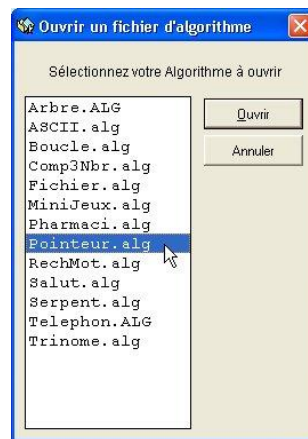
Il sert à créer de nouveaux algorithmes. L'ordinateur affiche une page vierge et invite l'utilisateur à saisir son nouvel algorithme.

- **Le bouton Ouvrir**



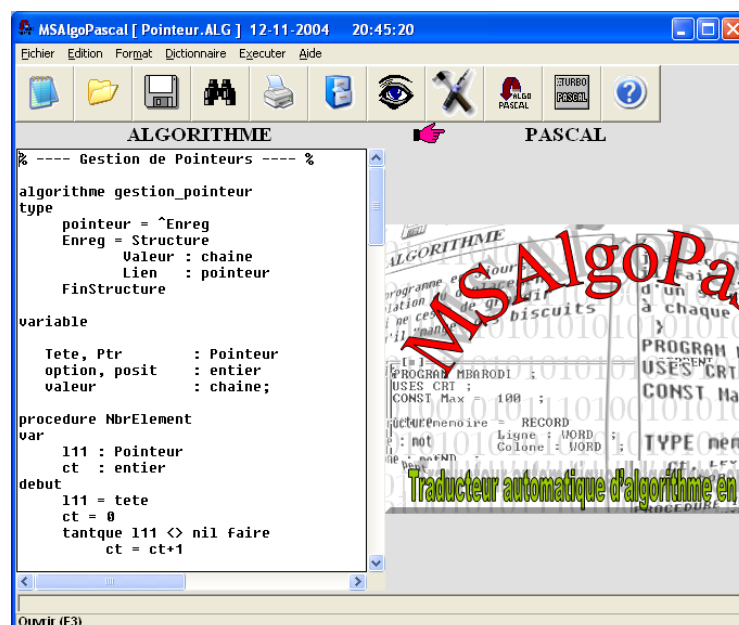
Ce bouton nous permet d'accéder au répertoire des fichiers d'algorithme sauvegardés sur le disque dur et de les ouvrir.

A son clic, voici la fenêtre qui s'affiche.



L'utilisateur peut sélectionner l'un des algorithmes précédemment sauvegardés pour l'éditer à nouveau.

Après sélection, voici l'apparence de la fenêtre principale.



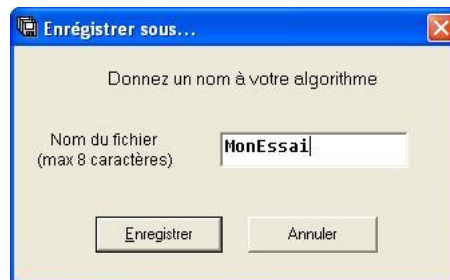
- **Le bouton Enregistrer**



Le bouton Enregistrer sert à sauvegarder l'algorithme courant sur le disque dur.

Si l'algorithme porte déjà un nom, le fichier source est automatiquement mis à jour selon les modifications effectuées. Mais s'il s'agit d'un nouvel algorithme, une fenêtre d'enregistrement s'affiche et invite à nouveau l'utilisateur à donner un nom à l'algorithme.

Cette fenêtre d'enregistrement se présente ainsi :

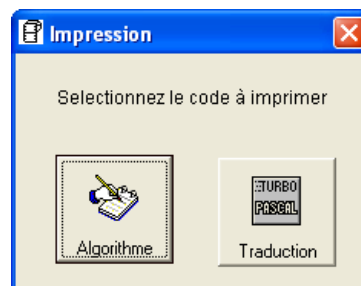


L'utilisateur est appelé à donner un nom de fichier n'ayant pas plus de 8 caractères (à cause du système de gestion de fichier sur MS-DOS). Ce fichier ne doit comporter ni de caractères spéciaux ni d'espaces. Une fois saisi, on clique sur le bouton Enregistrer pour valider cette opération.

- **Le bouton Imprimer**



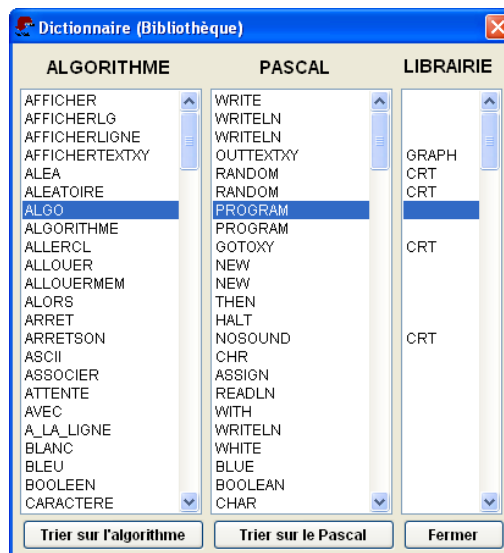
Ce bouton permet à l'utilisateur d'imprimer ses codes sources (aussi bien l'algorithme que sa traduction en langage Pascal). A son clic, une fenêtre s'affiche et l'invite à choisir le code à imprimer. La fenêtre d'invite se présente ainsi :



- **Le bouton Dictionnaire**



Il est très important et permet d'accéder à la base de données du système. A son clic, la fenêtre de relation s'affiche. Les données sont triées sur l'algorithme par défaut.

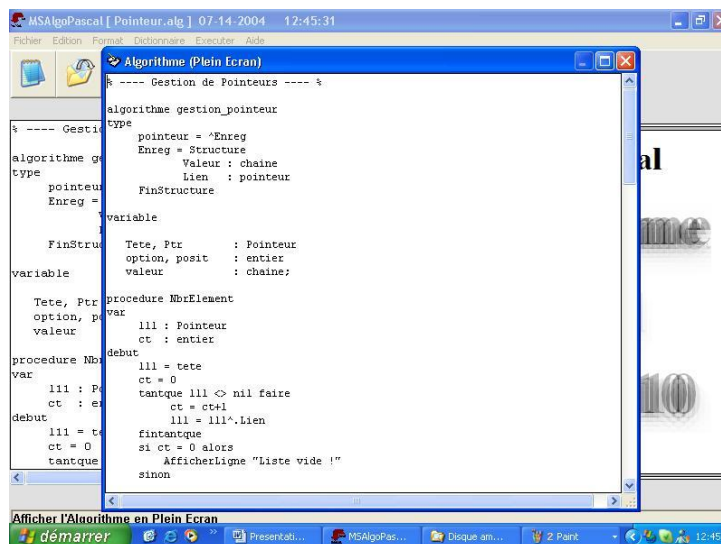


A chaque clic sur un enregistrement (occurrence), le système pointe automatiquement sur l'enregistrement auquel il correspond. Par exemple, en cliquant sur "**ALGO**", l'ordinateur sélectionne parmi les mots listés en Pascal "**PROGRAM**" et un vide sur la liste des Librairies. Cela veut simplement dire que "**ALGO**" en algorithme signifie "**PROGRAM**" en Pascal et son utilisation ne nécessite pas de "**librairie**".

- **Le bouton Zoom**



Il permet d'agrandir l'algorithme ou la traduction en plein écran et facilite l'écriture de l'algorithme.

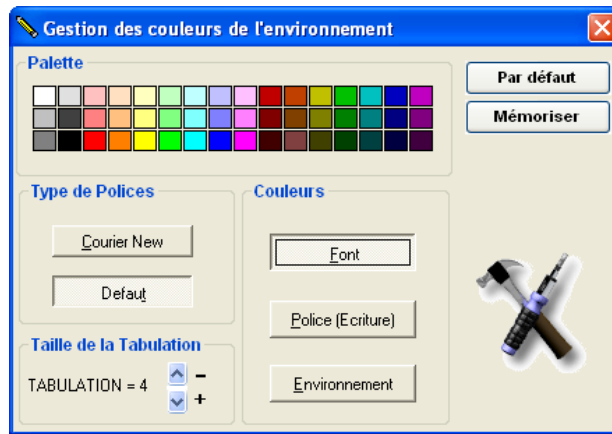


Zoom sur l'algorithme

- **Le bouton Gestion de l'interface de l'environnement**



La gestion de l'interface de l'environnement permet à l'utilisateur de personnaliser l'interface générale. Cette fenêtre s'affiche par la suite :



Gestionnaire des couleurs de l'environnement

Grâce à cette fenêtre l'utilisateur peut changer la couleur de la police, la couleur de fond et de l'environnement. Cette fenêtre permet aussi de changer le type de police et la taille de tabulation. Pour conserver ces modifications, il suffit de cliquer sur le bouton Mémoriser.

A chaque redémarrage du logiciel, MSAlgoPascal® conserve les dernières modifications de la gestion de l'interface de l'environnement.

- **Le bouton Traduire en langage Pascal**



Pour traduire notre algorithme en langage Pascal, il suffit de cliquer sur ce bouton. C'est ce dernier qui déclenche l'ensemble des procédures et des fonctions vues antérieurement pour avoir une traduction correcte. Plusieurs messages d'erreur peuvent être envoyés à l'utilisateur pour qu'il les corrige. En voici quelques exemples :

Exemple 1

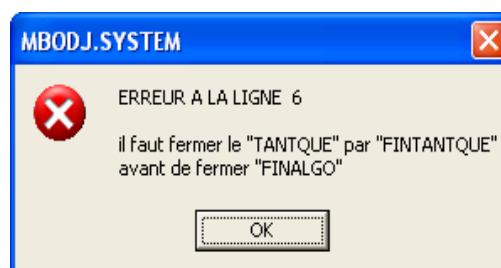
```
Algo exemple1
Var
    A, b : entier
Début
    Saisir (a1)
    Saisir (b)
FinAlgo
```



La variable a1 n'a pas été définie.

Exemple 2

```
Algo exemple2
Var
    A : entier
Début
    A = 20
    TantQue A > 0 Faire
        Afficher (« A= », A)
        A = A - 1
FinAlgo
```



La clause ouvrante TantQue n'a pas été fermée.

Exemple 3

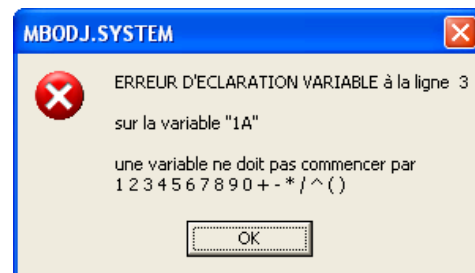
```
Algo exemple3
Var
    K : entier
Début
    A1 = 0
    TantQue A <= 20 Faire
        Afficher (K) ;
        K = K +1
    FinTantque
FinAlgo
```



A1 est non identifié.

Exemple 4

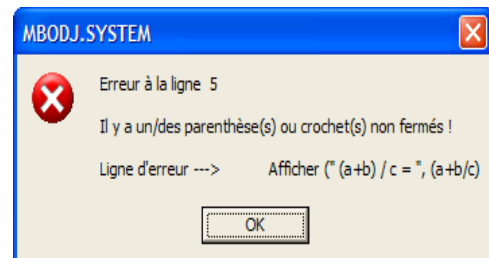
```
Algo Exemple4
Var
    1A : entier
Début
    Saisir (1A)
    Si 1A = 0 alors
        Afficher « un nombre nul »
    FinSi
FinAlgo
```



Une variable ne doit pas débiter par 1 2 3 4 5 6 7 8 9 0 + - * / ^ ()

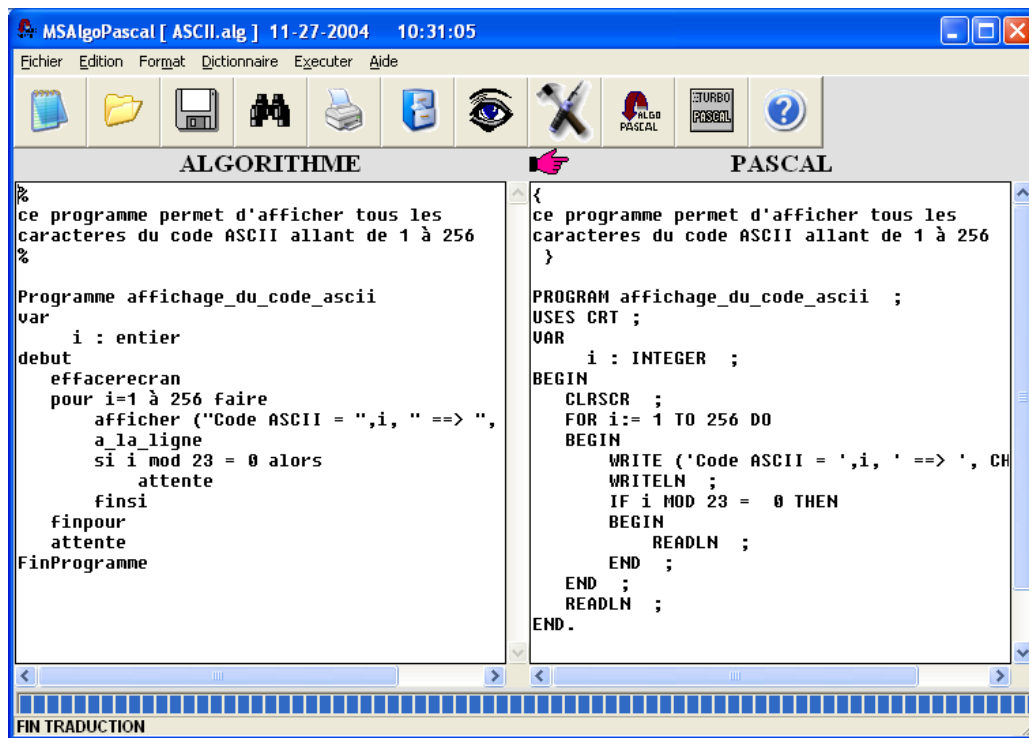
Exemple 5

```
Algo Exemple5
Var a, b, c : entier
Début
    Saisir (a, b, c)
    Afficher (« (a+b) / c = », (a+b/c)
FinAlgo
```



Expression mal parenthésée

Une fois que l'algorithme est correct, le système affiche sa traduction en langage Pascal dans la zone de texte située à droite de votre algorithme de sorte que vous puissiez mieux voir la traduction.

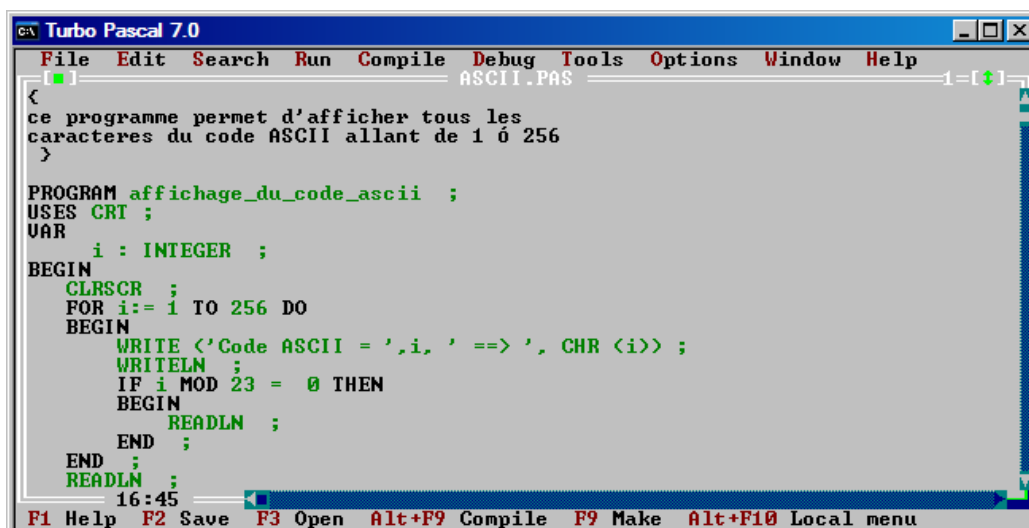


Traduction d'un exemple d'algorithme en langage Pascal

- **Le bouton Exécuter en langage Pascal**

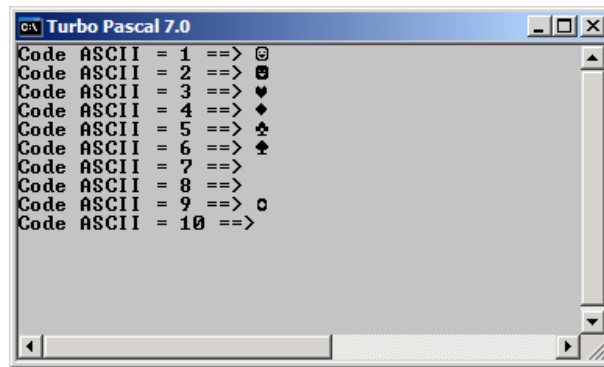


MSAIgoPascal® est muni du compilateur de Borland Turbo Pascal 7.0. Celui-ci nous permet d'éditer et d'exécuter nos algorithmes traduits en langage Pascal. Au clic de ce bouton, la fenêtre du compilateur Turbo Pascal 7.0 s'affiche.



Traduction de l'algorithme éditée à partir de l'éditeur de Borland Turbo Pascal 7.0

A partir de Turbo Pascal 7.0, l'utilisateur peut exécuter ses programmes en appuyant simultanément sur les touches **CTRL + F9**. Voici ce que nous avons lors de l'exécution d'un algorithme par Turbo Pascal.

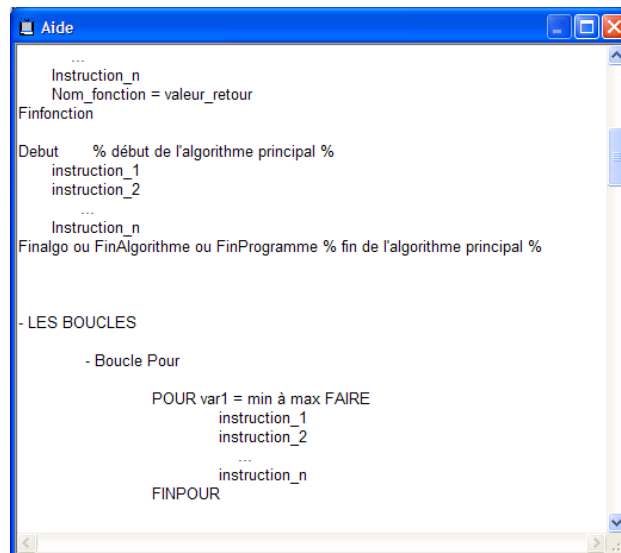


Exécution du programme

- **Le bouton Aide**



Pour savoir comment écrire un algorithme correct, il suffit de cliquer sur ce bouton. Une aide s'affiche et explique comment faire.

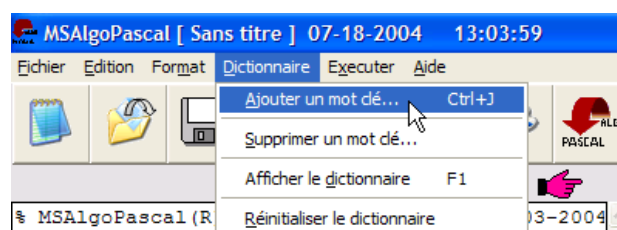


Fenêtre d'aide

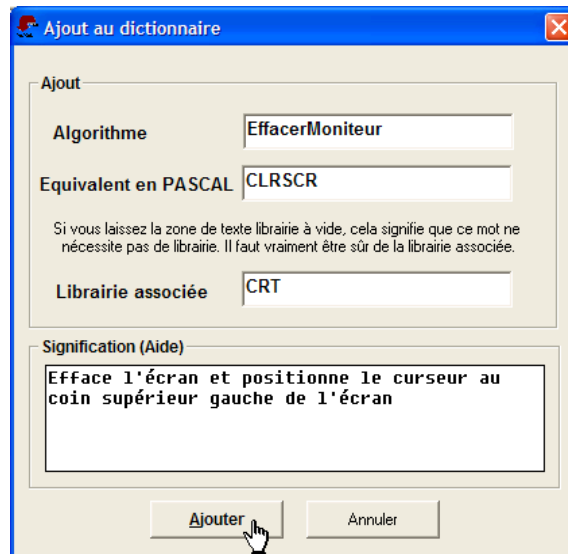
2.3.2.2 Autres options du logiciel MSAIgoPascal®

- **Opération d'ajout**

Il peut arriver qu'un mot ne soit pas traduit par MSAIgoPascal®, il faut donc ajouter ce dernier dans le dictionnaire et donner son équivalence en Pascal ; ainsi que la librairie associée. Pour ce faire, cliquer sur le menu "Dictionnaire" puis sur "ajouter un mot clé"



Ainsi la fenêtre d'ajout s'affiche.

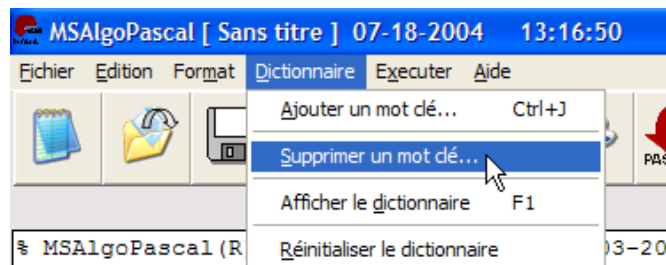


Fenêtre d'ajout d'un mot dans le dictionnaire

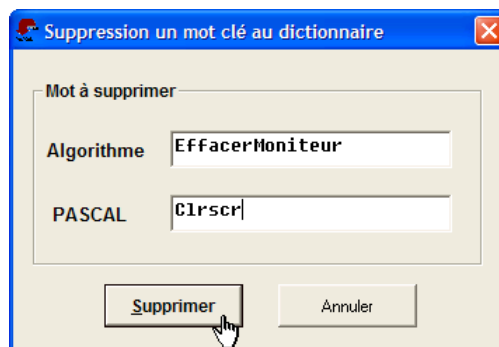
Dans cet exemple, le mot "EffacerMoniteur" a été ajouté et son équivalence en Pascal n'est rien d'autre que "CLRSCR". Or la procédure "CLRSCR" se trouve dans la librairie standard "CRT".

- **Opération de suppression**

Pour supprimer un mot au sein de la base de données, il suffit d'aller au menu « Dictionnaire », puis sur « Supprimer un mot clé ».



A son clic, voici la fenêtre qui s'affiche.



Le système nous invite à saisir le mot à supprimer en algorithme et sa signification en langage Pascal.

Chapitre III Tableaux et vecteurs

Dans un tableau, toutes les données élémentaires qui le constituent doivent être du même type. De plus, l'accès à un élément du tableau ne se fait que par son approche numérique, en indiquant entre crochets le rang de l'élément à lire ou à écrire ; c'est-à-dire son indice.

3.1 Tableaux simples (une dimension)

Un tableau permet de regrouper une quantité importante d'éléments (valeurs) de même type. Voici comment déclarer un tableau appelé TAB en algorithme et en langage Pascal.

Déclaration en algorithme

```
Variable  
TAB : Tableau [1..10] de entier
```

Déclaration en langage Pascal

```
Var  
TAB : Array [1.. 10] of integer ;
```

De ce fait, pour déclarer un tableau, il faut donner ses bornes (minimale et maximale).

```
TAB : TABLEAU [min .. max]  
                ↑       ↓  
            Indice minimal indice maximal
```

Dans l'exemple, nous avons déclaré d'un seul coup 10 variables de type entier. Pour accéder à la 6^{ème} variable, il suffit d'écrire

```
TAB [6]
```

Cet indice appartient naturellement à l'intervalle des bornes (min et max).

Faciliter la programmation

Imaginez que l'on veuille attribuer 15 valeurs à 15 personnes. Pour ce faire, nous déclarons 15 variables de type entier et nous passons à la saisie des valeurs.

```
Variable  
    val1, val2, val3, val4, val5,  
    val6, val7, val8, val9, val10,  
    val11, val12, val13, val14, val15 : Entier  
Début  
    Saisir (val1)  
        (...)  
    Saisir (val15)  
FinAlgo
```

Pour la méthode tableau, il suffit d'écrire :

```
Variable TAB : Tableau [1..15] de Entier  
           i : Entier
```

Début

```
    Pour i=1 à 15 faire  
        Saisir (TAB [i]) ;
```

```
    FinPour
```

```
% simple non ! %
```

Pour s'en convaincre, essayons pour 1000 valeurs.

```
TAB : TABLEAU [1..1000] de entier % c'est tout %
```

3.2 Tableau à plusieurs dimensions

Déclaration en algorithme

```
Variable  
TAB : TABLEAU [1..5, 1..5] de entier
```

Déclaration en Pascal

```
Var  
TAB : Array [1..5, 1..5] of integer;
```

Nous avons déclaré un tableau de 5 x 5 éléments.

Représentation graphique

Plusieurs opérations et structures de données peuvent être réalisées grâce aux tableaux. Ces opérations peuvent être entre autres :

- les tris, les recherches, les compressions de données
- les piles, les files, les arbres, etc.

3.3 Opérations classiques sur les tableaux

3.3.1 Affichage des données d'un tableau

Il suffit de parcourir toutes les cellules du tableau et d'afficher les données qui s'y trouvent.

Exemple 1 : soit un tableau $\times 1$ de taille N, le sous-programme suivant nous affiche alors tous les éléments de ce dernier.


```

Procédure AfficheTab (donnée TabR : TAB ; N : entier)
Variable
    i : entier
Début
    Pour i = 1 à N Faire
        AfficherLigne (TabR [i])
    FinPour
FinProcédure

```

NB : s'il s'agit d'un tableau dont les éléments sont des enregistrements, l'instruction d'affichage devient alors

```

AfficherLigne (TabR[i].NomChamp)

```

Exemple : Soit la structure suivante

```

Type Etudiant = Structure
    Matricule : Chaîne [3]
    Nom       : Chaîne [10]
    Prenom    : Chaîne [15]
FinStructure

```

Pour afficher le matricule, le nom et le prénom de l'étudiant d'indice 5, il suffit d'écrire

1^{er} cas

```

    AfficherLigne (TabR [5].Matricule)
    AfficherLigne (TabR [5].Nom)
    AfficherLigne (TabR [5].Prenom)

```

2^{ème} cas

```

    Avec TabR [5] Faire
        AfficherLigne (Matricule)
        AfficherLigne (Nom)
        AfficherLigne (Prenom)
    FinAvec

```

3.3.2 Opération d'ajout

Cette opération ne peut se faire qu'au cas où il y a de l'espace libre dans le vecteur. Il faut au moins une cellule non utilisée pour la réaliser.

Cas 1

Soit un tableau x 1 (1 dimension) de taille n. Ecrire l'algorithme qui ajoute n valeurs VAL au tableau.

```

Algorithme AjoutValeur
Constante n = 10
TYPE TAB = Tableau [1..10] de Entier
Variable
    Nbr, ValeurLue : Entier
    TabR : TAB

```

```

Procédure Ajout (Donnée Val, Nbr : Entier ; Résultat TabR : TAB)
Début
    Si (Nbr = n) alors
        AfficherLigne "Ajout impossible"
    Sinon
        Nbr = Nbr + 1
        TabR [Nbr] = Val
    FinSi
FinProcédure
DÉBUT
    Nbr ← 0
    Repeter
        Saisir (ValeurLue)
        Ajout (ValeurLue, Nbr, TabR)
    Jusqua (Nbr = Max)
FinAlgorithme

```

NB : nous remarquons que le cas ci-dessus ne s'intéresse pas à l'ordre ; notre souci était simplement d'ajouter un élément donné. Or, dans la pratique, pour la plupart du temps, l'ordre est très important.

Cas 2

Ecrire le sous-programme qui ajoute la valeur VAL si possible à la position POS du tableau tout en respectant l'ordre de tri.

```

Procédure Ajout (Donnée Val, Nbr, Pos : Entier ; Résultat TabR :
TAB)
Variable
    i : Entier
Début
    Si (Nbr = max) Alors
        Afficherligne "Ajout impossible"
    Sinon
        Si (Pos > Max) Alors
            Afficherligne "Position incorrecte"
        Sinon
            Pour i=Nbr à Pos Faire
                TabR [i+1] = TAB [i]
            FinPour

            TabR [Pos] = Val
            Nbr = Nbr +1
        FinSi
    FinSi
FinProcédure

```

Note : Nbr est le nombre d'éléments présents dans le vecteur (tableau). Le principe consiste à déplacer tous les éléments en partant de la position Nbr à Pos vers Nbr+1 à Pos+1.

Autrement dit, il faut faire l'application $F(x) \rightarrow F(x+1)$ tel que x appartient à $[Nbr-1, Pos]$; puis attribuer $F(Pos)$ la valeur à ajouter : $F(Pos) \leftarrow Val$.

Nous verrons par la suite que Pos est déterminé par un algorithme de recherche.

3.3.3 Opération de suppression

Ayant compris l'opération d'ajout, celle de suppression devient facile à réaliser. Cette dernière est son inverse.

Ecrire la Procédure SupprimeVal (donnée TabR : TAB, Pos, Nbr : entier ; Résultat TabR : TAB) qui supprime l'élément d'indice Pos.

```
Procédure SupprimeVal (donnée TabR : TAB, Pos, Nbr : entier ;  
Résultat TabR : TAB)  
  
Var i : entier  
Début  
    Si (Pos > Max) Alors  
        AfficherLigne "Position incorrecte"  
    Sinon  
        Si (Nbr = 0) Alors  
            AfficherLigne "Tableau vide"  
        Sinon  
            Pour i=Pos à nbr-1 faire  
                TabR [i] ← TabR [i+1]  
            FinPour  
            Nbr ← (Nbr - 1)  
        FinSi  
    FinSi  
FinProcédure
```

NB : Dans la pratique, si nous avons affaire à des tables de taille importante, l'opération de suppression est souvent optimisée car la réorganisation de la table nécessite un temps énorme. Il est préférable de récupérer les enregistrements (*cellules*) supprimés lors de la création de nouveaux enregistrements. Cette opération se fait à l'aide d'un *marqueur de présence*.

3.3.4 Opérations de recherche

Elles font partie des opérations les plus utilisées en informatique. Les plus fréquentes sont : la recherche séquentielle et la recherche dichotomique.

3.3.4.1 La recherche séquentielle

La recherche consiste à partir du début d'un tableau, à comparer l'élément rencontré à celui que l'on cherche. Si ce dernier est différent alors on passe à l'indice suivant ; sinon on arrête l'opération car l'information recherchée est trouvée.

Ecrire la **Fonction RechercheValeur** (donnée Val, Nbr : entier, TabR : TAB) : Booléen qui cherche la valeur Val dans le tableau TabR.

```

Fonction RechercheValeur (donnée Val, Nbr : entier ; TabR : TAB) :
Booléen
Variable
    Trouver    : Booléen
    i          : entier
Début
    Trouver ← faux
    i ← 1
    TantQue (i <= Nbr) ET (Trouver = FAUX) FAIRE
        Si TabR [i] = Val Alors
            Trouver ← Vrai
        Sinon
            i ← (i + 1)
        FinSi
    FinTantQue
    RechercheValeur ← Trouver
FinFonction

```

Remarque : Cette recherche est relativement lente. Imaginons que l'élément que nous cherchons se trouve à la position 150000. Nous sommes obligés de parcourir 150000 éléments pour le trouver. Si l'information recherchée ne se trouve pas au niveau du vecteur, l'algorithme le parcourt du début à la fin pour être sûr que l'information ne s'y trouve pas.

Ce problème nous amène à optimiser l'algorithme tout en utilisant un autre beaucoup plus rapide.

3.3.4.2 La recherche dichotomique

Le principe de la recherche dichotomique réside dans la division successive de l'espace de recherche en deux dans un vecteur **ordonné sur le critère de recherche**.

Ecrire la **Fonction RechercheValeur (donnée Val, Nbr : entier, TabR : TAB) : Booléen** qui cherche dichotomiquement la valeur Val dans le tableau TabR.

```

Fonction RechercheValeur (donnée Val, Nbr : entier, TabR : TAB) : Booléen
Variable
    Trouver : Booléen
    BorneInferieur, BorneSuperieur, Milieu : entier
Début
    BorneInferieur ← 1 ; BorneSuperieur ← Nbr ; Trouver ← Faux
    TantQue (Trouver = Faux) ET (BorneInferieur <= BorneSuperieur) Faire
        Milieu ← (BorneInferieur + BorneSuperieur) / 2
        Si TabR [Milieu] < Val Alors
            BorneInferieur ← (Milieu + 1)
        Sinon
            Si TabR [Milieu] > Val Alors
                BorneSuperieur ← (Milieu - 1)
            Sinon
                Trouver ← Vrai
            FinSi
        FinSi
    FinTantQue
FinFonction

```

3.3.4.3 La recherche de la position d'une information

La recherche séquentielle et la recherche dichotomique peuvent nous permettre de déterminer la position où il faut insérer l'information.

Ainsi pour la recherche séquentielle nous aurons :

```
Procédure RecherchePosition (donnée Val, Nbr : entier ; TabR :  
TAB ; Résultat Position : Entier)  
Variable  
    Trouver    : Booléen  
    i          : entier  
Début  
    Trouver ← faux  
    i ← 1 ; Position ← 1  
    TantQue (i <= Nbr) ET (Trouver = FAUX) FAIRE  
        Si TabR [i] <= Val Alors  
            i ← (i + 1)  
        Sinon  
            Trouver ← Vrai  
            Position ← i  
        FinSi  
    FinTantQue  
FinFonction
```

Et pour la recherche dichotomique, l'algorithme devient :

```
Procédure RecherchePosition (donnée Val, Nbr : entier, TabR : TAB ; Résultat  
Position : entier)  
Variable  
    Arret : Booléen  
    BorneInferieur, BorneSuperieur, Milieu : entier  
Début  
    BorneInferieur ← 1  
    BorneSuperieur ← Nbr  
    Arret ← Faux  
    Position ← 1  
    TantQue (Arret = Faux) ET (BorneInferieur <= BorneSuperieur) Faire  
        Milieu ← (BorneInferieur + BorneSuperieur) / 2  
        Si TabR [Milieu] < Val Alors  
            BorneInferieur ← (Milieu + 1)  
        Sinon  
            Si TabR [Milieu] > Val Alors  
                BorneSuperieur ← (Milieu - 1)  
            Sinon  
                Arret ← Vrai  
            FinSi  
        FinSi  
    FinTantQue  
    Position ← Milieu  
FinFonction
```

Exercices

Exercice 1 : Ecrire la **fonction Palindrome** (donnée **Mot : Chaîne**) : **Booléen** qui vérifie si un mot est palindrome ou pas. Nous rappelons qu'un palindrome est une suite de caractères qui se lit indifféremment de gauche à droite ou de droite à gauche en donnant le même texte ou un texte différent. Exemple : "radar", "matam", "sexes", "Laval"

Exercice 2 : Ecrire la **Procédure TransposeMatrice** (donnée **Résultat T1**) qui crée la transposée de la matrice carrée **T1**.

Exercice 3 : Ecrire la **Procédure Symétrie** (donnée **T1, Résultat T2**) qui crée la symétrie de la matrice carrée **T1**.

Exercice 4 : Ecrire le sous-programme qui transforme un tableau à deux dimensions en un vecteur (tableau à une dimension).

Exemple : l'exemple ci-dessous transforme le tableau TAB en un vecteur appelé VECT.

TAB

0	10	1	5	3
4	7	8	0	0
0	4	2	8	3
4	9	9	0	0
1	0	0	1	0

VECT

0	10	1	5	3	4	7	8	0	0	0	4	2	8	3	4	9	9	0	0	1	0	0	1	0
---	----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Exercice 5 : Faites la réciproque de l'exercice précédant (quitter VECT vers TAB).

Exercice 6 : Ecrire un programme en Pascal qui vérifie si un tableau x 2 est symétrique ou antisymétrique. Nous rappelons que ce tableau doit être carré.

Exemple de matrice carrée symétrique :

2	4	0	9
4	4	1	0
0	1	-5	8
9	0	8	7

Remarque : Quelque soit a_{ij} avec $i \neq j$ on a $a_{ij} = a_{ji}$

Exemple de matrice carrée antisymétrique :

0	1	0	9
-1	0	-7	-4
0	7	0	8
-9	4	-8	0

Remarque : Quelque soit a_{ij} avec $i \neq j$ on a $a_{ij} = -a_{ji}$. Et nous remarquons que les valeurs situées sur la diagonale principale sont nulles.

Exercice 7 : Ecrire un programme en Pascal qui mémorise l'adresse des valeurs non nulles d'une matrice par les couples (i, j, a_{ij}) dans un vecteur VECT ou i représente la ligne, j la colonne et a_{ij} l'information.

Exemple

TAB

0	0	0	9
0	8	0	-4
0	0	0	0
0	4	0	0

VECT devient alors

1	4	9	2	2	8	2	4	-4	4	2	4
---	---	---	---	---	---	---	---	----	---	---	---

Exercice 8 : Ecrire la réciproque de l'exercice précédent. Nous quitterons le vecteur VECT pour construire le tableau TAB.

Exercice 9 : Ecrire un programme qui fait la somme et le produit des nombres d'un tableau à deux dimensions.

Exercice 10 : Ecrire un programme qui calcule les totaux par ligne et par colonne d'un tableau à deux dimensions.

Exemple

Total par ligne

0	0	0	9	9
0	8	0	-4	4
0	0	0	0	0
0	4	0	0	4

Total par colonne

0	12	0	5
---	----	---	---

Exercice 11 : Soit un vecteur de taille N (avec N un multiple de 3) formé par les chiffres suivants : $\{1, 2, 3\}$ dans le désordre. Ecrire la procédure qui réorganise ce vecteur dans l'ordre de $\{1\ 2\ 3\ 1\ 2\ 3\ 1\ 2\ 3\ \dots\ 1\ 2\ 3\}$.

NB : il y a autant de « 1 » de « 2 » que de « 3 ».

Exemple :

Vecteur initial

1	3	1	2	3	2	3	1	3	2	2	1
---	---	---	---	---	---	---	---	---	---	---	---

Vecteur final

1	2	3	1	2	3	1	2	3	1	2	3
---	---	---	---	---	---	---	---	---	---	---	---

Chapitre IV Les algorithmes de tri

4.1 Définition

Le tri est une opération qui consiste à classer des informations, des données selon un ou plusieurs critères ou selon un certain ordre. L'opération de tri est très importante et très utilisée dans les programmes informatiques utilisant les algorithmes de recherche. Imaginez que l'on dispose d'un dictionnaire non trié, comme il serait fastidieux de trouver un mot. Combien de temps nous faudra-t-il pour retrouver le mot que l'on cherche ?

Il existe plusieurs algorithmes de tri. Les plus courants sont :

- tri par sélection
- tri par insertion
- tri à bulles

Au niveau des algorithmes qui vont suivre, nous supposons que nous disposons d'un vecteur de Nbr éléments (de type entier).

4.2 Le tri par sélection

Le principe des méthodes de tri par sélection est de rechercher le minimum dans le vecteur donné, de le placer en tête et de recommencer sur le reste de la liste. A la $i^{\text{ème}}$ étape, le sous-vecteur $[1..i-1]$ est trié. L'élément de clé minimale, trouvée entre le rang i et le rang n , est placé au rang i , et le tri se poursuit à l'étape $i+1$.

Exemple : Soit le vecteur `VECT [1..5]`.

Première itération

11	26	45	3	68
i			j	

$i = 1$

$j = i + 1$

Le plus petit élément du sous-vecteur (sous-matrice) $[2,5]$ est 3. Alors nous le permutons avec l'élément d'indice

$i = 1$.

Le vecteur devient alors

3	26	45	68	11
---	----	----	----	----

Deuxième itération

3	26	45	11	68
	i		j	

$i = 2$

$j = i + 1$

Le plus petit élément du sous-vecteur [3,5] est 11. Ce dernier sera permuté avec l'élément d'indice $i=2$.

Le vecteur devient alors

3	11	45	68	26
---	----	----	----	----

Troisième itération

3	11	45	26	68
---	----	----	----	----

i j

$i = 3$

$j = i + 1 \text{ \% } j = 4 \text{ \% }$

Le plus petit élément du sous-vecteur [4,5] est 26. Ce dernier sera permuté avec l'élément d'indice $i= 3$.

Le vecteur devient alors

3	11	45	26	68
---	----	----	----	----

Quatrième et dernière itération

$i = 4$

$j = i + 1$

Aucune permutation ne sera effectuée car $26 < 68$.

3	11	45	26	68
---	----	----	----	----

Nous utiliserons le sous-programme qui réalise la permutation de deux entités de même type.

```

Procédure Permute (donnée Résultat Valeur1, Valeur2 : entier)
Variable Tampon : entier
Début
    Tampon  $\leftarrow$  Valeur1
    Valeur1  $\leftarrow$  Valeur2
    Valeur2  $\leftarrow$  Tampon
FinProcédure
  
```

L'algorithme de tri par sélection est le suivant :

```

Procédure TriSelection (Donnée Nbr : entier; Résultat TabR : TAB)
Variable
    i, j : entier
Début
    Pour i=1 à Nbr-1 Faire
        Pour j = i+1 à Nbr Faire
            Si TabR [i] > TabR [j] Alors
                Permute (TabR [i], TabR [j])
            FinSi
        FinPour
    FinPour
FinProcédure
  
```

4.3 Le tri par insertion simple

On choisit un élément du vecteur, on trie les autres et on insère l'élément initialement choisi à la bonne place en parcourant le tableau. Chaque élément sera inséré à sa place.

```
Procédure TriInsertion (Donnée Nbr : entier, TabR : TAB ; Résultat
TabR : TAB)
Variable
    i, j, Tampon : entier
Début
    i ← 2
    TantQue (i ≤ Nbr) Faire
        Tampon ← TabR [i]
        j ← i - 1
        TantQue (j > 0) ET (Tampon < TabR [j]) Faire
            TabR [j+1] ← TabR [j]
            j ← j - 1
        FinTantQue
        TabR [j + 1] ← Tampon
        i ← i + 1
    FinTantQue
FinProcédure
```

Exercice : Utilisez le vecteur suivant :

45	33	5	26	7
----	----	---	----	---

et à l'aide de votre crayon déroulez l'algorithme. (ici Nbr = 5)

4.4 Le tri à bulles

On parcourt le tableau en comparant deux éléments consécutifs ; s'ils sont mal placés, on les permute. Cela revient à faire remonter le plus grand élément à chaque parcours. Comme une bulle d'air qui remonte à la surface de l'eau, d'où le nom de tri à bulles. (Il est aussi appelé tri par permutation ou tri par échanges).

4.4.1 La première méthode

```
Procédure TriInsertion (Donnée Nbr : entier ; Résultat TabR : TAB)
Var
    i, Dernier : entier
Début
    Dernier ← Nbr - 1
    Repeter
        Pour i=1 à dernier Faire
            Si TabR [i] > TabR [i+1] Alors
                Permute (TabR [i], TabR [i+1])
            FinSi
        FinPour
        Dernier ← dernier - 1
    Jusqua (dernier = 1)
FinProcédure
```

Parfois les dernières itérations sont inutiles, elles ne décèlent aucune inversion et n'effectuent donc aucune permutation. Ce phénomène peut être évité en arrêtant l'algorithme dès qu'il n'est plus possible d'effectuer une inversion.

4.4.2 La deuxième méthode

```
Procédure TriInsertion (Donnée Nbr : entier ; Résultat TabR : TAB)
Var
    i, Dernier : entier
    Echange      : booléen
Début
    Dernier ← Nbr - 1
    Repeter
        Echange ← faux
        Pour i=1 à dernier Faire
            Si TabR [i] > TabR [i+1] Alors
                Permute (TabR [i], TabR [i+1])
                Echange ← vrai
            FinSi
        FinPour
        Dernier ← dernier - 1
    Jusqua (Echange = faux)
FinProcédure
```

Chapitre V Les chaînes de caractères

5.1 Définition

Une chaîne de caractères n'est rien d'autre qu'un vecteur de caractères. Auparavant elle était déclarée :

NomChaine : Tableau [1..N] de caractere

Son utilisation accrue a permis au concepteur de compilateurs tel que le Pascal de lui donner un type à part : **String**. Pour déclarer une chaîne de 50 enregistrements en Pascal, il suffit de dire **NomChaine : String [Taille]** où *Taille* est un entier positif inférieur ou égal à 255.

En algorithmique, la déclaration est la suivante : **NomChaine : Chaîne [Taille]**.

Si le programmeur ne précise pas lors de la déclaration la taille de la chaîne, le compilateur lui réserve directement la taille maximale (255). Il faut noter qu'en cours d'exécution la longueur d'une chaîne peut varier.

Nous verrons au chapitre consacré à l'allocation dynamique comment faire pour disposer des chaînes supérieures à 255 caractères.

5.2 Accès à un caractère

Tous les traitements effectués sur les vecteurs sont possibles sur les chaînes de caractères. Ainsi pour accéder au $i^{\text{ème}}$ caractère supposé existant, il suffit de dire **NomChaine [i]**.

Exemple

```
Variable Nom : Chaîne [10]
Début
    Nom ← "NDIANOR"
    AfficherLigne "Le premier caractère de votre nom est", Nom [1]
```

Exercice :

Ecrire la fonction qui compte le nombre de syllabes contenues dans une chaîne donnée.

NB : La fonction **Longueur (NomChaine)** renvoie la longueur d'une chaîne donnée. Cette fonction est **Length** en Pascal.

```
Fonction NombreSyllabe (Donnée Ligne : Chaîne) : entier
Variable
    i, Nbr : entier ; % i est l'indice de parcours %
Début
    Nbr ← 0
    Pour i=1 à Longueur (Ligne) Faire
        Selon Ligne [i] Faire
```

```

        Cas "i", "u", "o", "a", "e", "y" :
            Nbr ← Nbr + 1
    FinSelon
FinPour
NombreSyllabe ← Nbr
FinFonction

```

5.3 Concaténation de chaînes

La concaténation ou l'addition de deux ou plusieurs chaînes donne comme résultat la juxtaposition de ces chaînes. L'opérateur de concaténation est le « + ».

Exemple : Soit A, B, C et R quatre chaînes de caractères, on a :

```

A ← "Bonjour"
B ← "mes"
C ← "amis"
R ← A + " " + B + " " + C

```

A la fin de cette opération, R donne la valeur suivante : "Bonjour mes amis"

NB : une chaîne peut être vide. ($R \leftarrow ""$). Ceci est très utilisé en informatique surtout lors de l'initialisation des variables de type chaîne. La longueur d'une chaîne vide vaut zéro.

5.4 Les fonctions utiles pour le traitement des chaînes

- **Longueur (NomChaine : Chaîne) : entier** renvoie la longueur d'une chaîne.
- **CVNombre (NomChaine) : entier** convertit une chaîne en valeur numérique.
- **CVChaine (ValeurNumerique) : chaîne** convertit une valeur numérique en chaîne de caractères.
- **ToucheAppuie** lit une touche saisie au clavier.
- **LireTouche** renvoie la touche saisie au clavier.
- **Pos (Chaine1, Chaine2 : chaîne) : entier** retourne la position de la première occurrence d'une sous-chaîne de caractères dans une chaîne de caractères.
- **InsertChaine (chainel1, chaine2 : chaîne, Posit : entier)** insère la sous-chaîne (chainel1) dans une chaîne de caractères (chaine2) à partir de la position Posit.

Exercices

Exercice 1 : Ecrire la fonction **Pos (chaine1, chaine2 : chaîne) : entier** qui retourne la position de la première occurrence d'une sous-chaîne de caractères dans une chaîne de caractères.

Exercice 2 : Ecrire la Procédure **InsertChaine (Donnée chaine1 : chaîne ; Posit : entier ; résultat chaine2 : chaîne)** qui insère la sous-chaîne (chaine1) dans la chaîne de caractères (chaine2) à partir de la position Posit.

Exercice 3 : Ecrire la Procédure **OccurrenceAlphabet (donnée Ligne : chaîne, Résultat Alphabet : Tableau [1..26] de Entier)** qui compte le nombre d'occurrences du caractère Alphabet [i] de la ligne Ligne.

NB : Alphabet [1] = "A", Alphabet [2] = "B", ..., Alphabet [26] = "Z"

Exemple : Ligne = "je suis un informaticien"

Représentation du vecteur Alphabet après le déroulement de l'algorithme.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
2	0	1	0	2	1	0	0	4	1	0	0	0	2	1	0	0	1	2	1	2	0	0	0	0	0

Exercice 4 : Ecrire la Fonction **RechercheMot (donnée Ligne : chaîne, Posit : entier) : Chaîne** qui affiche le premier mot rencontré de la ligne Ligne en partant de la position Posit.

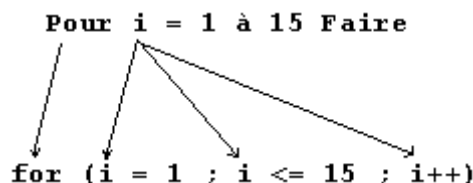
Exemple :

Ligne = "je suis un débutant en informatique"

Posit = 12

RechercheMot (ligne, Posit) = "débutant"

Exercice 5 : Lors de la conception du générateur de codes MSAIgoC[®], il arrive de transformer des instructions de l'algorithme en langage C. Par exemple : l'instruction (boucle Pour) suivante :
Pour i = 1 à 15 Faire donne en C **for (i = 1 ; i <= 15 ; i++)**.



Analyser attentivement le schéma ci-dessus puis écrire la Procédure (donnée BouclePourAlgo : chaîne, Résultat BouclePourC : chaîne) qui transforme une boucle **Pour** en boucle **for**.

Chapitre VI Les enregistrements

Il est souvent intéressant de pouvoir manipuler dans un tableau plusieurs types différents. Par exemple pour gérer les 50 étudiants d'une école comprenant les informations suivantes :

```
(Matricule : entier ; Nom : Chaîne [10] ; Prénom : Chaîne [15] ;  
Adresse : Chaîne [35] ; etc.)
```

Etant donné les types formés par ces informations sont différents, la première idée qui nous vient à l'esprit est d'utiliser quatre (4) tableaux :

```
Constante Max = 50  
Type TabMatri      = Tableau [1..Max] de entier  
      TabNom        = Tableau [1..Max] de Chaîne [10]  
      TabPrenom     = Tableau [1..Max] de Chaîne [15]  
      TabAdress     = Tableau [1..Max] de Chaîne [35]  
Variable % Puis déclarer les variables tableaux...%  
      TMatricule    : TabMatri  
      TNom          : TabNom  
      TPrenom       : TabPrenom  
      TAdresse      : TabAdress  
      % ouf ! Difficile à écrire %
```

Quels sont les inconvénients d'une telle définition ?

- La déclaration est fastidieuse. Imaginez pour des informations qui nécessitent 15 tableaux combien de lignes nous faudrait-il ?
- Au niveau des algorithmes de recherche, de suppression, d'ajout, de modification et de tri, tous les tableaux doivent avoir le même indice. Sinon il y aurait des informations incohérentes.
- Toute modification apportée sur l'un de ces tableaux doit impérativement être effectuée sur les autres.

6.1 Déclaration d'une structure

Pour réaliser ces opérations à l'aide d'un tableau, nous utiliserons un nouveau type de données appelé **enregistrement** ou **structure**.

La structure générale de la déclaration d'un enregistrement est la suivante :

```
Type NomEnregistrement = Structure  
      Champ1      : Type1  
      Champ2      : Type2  
      (...)       :  
      Champn     : Typen  
FinStructure  
Variable NomTableau : Tableau [1..Limite] de NomEnregistrement.
```

Ainsi la déclaration ci-dessus devient alors

```
Constante Max = 50
Type Etudiant = Structure
    Matricule : entier
    Nom       : Chaîne [10]
    Prenom    : Chaîne [15]
    Adresse   : Chaîne [35]
FinStructure
Variable
    TabEtudiant : Tableau [1..Max] de Etudiant

% simple non par rapport au précédant ! %
```

Désormais, tous les champs sont « logés » au niveau d'un seul tableau. Pour accéder à un champ voir le *chapitre 1.3.6.9*.

Représentation graphique

Valeur des indices	1				2			
Nom champs	Matricule	Nom	Prénom	Adresse	Matricule	Nom	Prénom	Adresse
Exemple	A001	AW	Mamadou	Hamo3	A002	BA	Aïda	Sicap

6.2 Affection de valeurs

Si deux enregistrements ont la même structure (Enreg1 et Enreg2) pour effectuer l'affection de Enreg2 à Enreg1, nous pouvons utiliser l'instruction suivante :

Enreg1 ← Enreg2

Si ces derniers sont différents alors l'affection se fera champs par champs.

Exemple :

```
Type Enreg1 = Structure
    Nom       : Chaîne [10]
    Prenom    : Chaîne [15]
    Sexe      : Caractere
FinStructure

Type Enreg2 = Structure
    Nom       : Chaîne [10]
    Prenom    : Chaîne [15]
FinStructure
```

L'affectation précédente se fera de la manière suivante :

```
Enreg1.Nom ← Enreg2.Nom
Enreg1.Prenom ← Enreg2.Prenom
```


Application

Voici à présent une opération de tri par sélection sur le tableau précédemment défini. Les enregistrements sont triés sur le matricule de l'étudiant.

```
Procédure Permute (donnée Résultat Valeur1, Valeur2 : Etudiant)
Variable Tampon : Etudiant
Début
    Tampon ← Valeur1
    Valeur1 ← Valeur2
    Valeur2 ← Tampon
FinProcédure
```

L'algorithme de tri par sélection est le suivant :

```
Procédure TriSelection (Donnée Nbr : entier; Résultat TabR : TAB)
Variable
    i, j : entier
Début
    Pour i=1 à Nbr-1 Faire
        Pour j = i+1 à Nbr Faire
            Si TabR [i].Matricule > TabR [j].Matricule Alors
                Permute (TabR [i], TabR [j])
            FinSi
        FinPour
    FinPour
FinProcédure
```

Projets

Projet n°1

Nous décidons de réaliser un agenda numérique portant les informations suivantes :

(N° ordre, Nom, Prénom, Adresse, Téléphone Domicile, Téléphone Cellulaire et Téléphone Bureau)

Les opérations à faire sont les suivantes :

- affichage de toutes les personnes.
- affichage des personnes commençant par les 3 premiers caractères que nous saisissons.
- recherche d'une personne.
- ajout d'une personne.
- suppression d'une personne.

Correction : Pour la correction, voir au niveau de MSAlgoPascal le projet du nom de **Agenda.ALG**

Projet n°2

Il nous est soumis de gérer une salle de classe de 30 étudiants. Chaque étudiant dispose des informations suivantes : nom (15 caractères), prénom (20 car.), et les matières suivantes : Physique, Math et Informatique variant entre 0 et 20.

1. Donnez la structure de la table étudiant.
2. Réaliser les procédures suivantes :
 - a. saisir les noms, prénoms des étudiants.
 - b. saisir les notes des étudiants pour chaque matière.
 - c. afficher les noms et prénoms des étudiants.
 - d. rechercher et afficher les notes d'un étudiant.
 - e. calculer la moyenne de tous les étudiants en les affichant.
 - f. trier la table en ordre croissant selon la moyenne obtenue.
 - g. supprimer au niveau de la table tous les étudiants n'ayant pas la moyenne supérieure ou égale à 10.

Correction : Pour la correction, voir au niveau de MSAlgoPascal le projet du nom de **Classe.ALG**

Chapitre VII L'allocation dynamique : listes, piles, files

Au chapitre 3, nous avons découvert les structures de type tableau. Ces structures souffrent d'un gros handicap (inconvenient) : **ils sont statiques**. Leurs tailles ne peuvent être modifiées. Le programmeur doit connaître dès l'écriture du programme la taille exacte du tableau (le nombre de cellules que doit contenir le tableau). Or dans certains cas cela est quasi impossible.

Les opérations d'ajout sur une table ordonnée et de suppression nécessitent la juxtaposition de n cellules pour en libérer ou en écraser une. Par conséquent ces opérations sont très coûteuses en temps machines.

Pour contourner ces problèmes, on utilise le concept de pointeur et d'allocation dynamique.

7.1 Définition

Un pointeur est une variable dont le contenu est une **adresse mémoire d'une autre variable**.

7.2 Déclaration d'une variable de type pointeur

```
Variable VarPointeur : ↑TypePointé
```

```
Exemple : Variable PEntier : ↑Entier
```

PEntier ne peut contenir que des adresses mémoires des variables de type entier.

La fonction Adresse (Var) renvoie l'adresse mémoire de la variable Var.

Pour accéder au contenu de la variable pointeur, il suffit d'écrire VarPointeur↑ (variable pointeur suivi de l'accent circonflexe).

Application 1

```
1.  Algo App1
2.  Variable
3.      PEntier : ↑Entier
4.      A      : Entier
5.  Début
6.      A ← 10
7.      PEntier ← Adresse (A)
8.      AfficherLigne (A)
9.      AfficherLigne (PEntier↑)
10. FinAlgo
```

Explication de l'application 1

PEntier est une variable pointeur sur le type entier. La variable A est du type entier.

A la ligne 6, nous affectons la valeur 10 à la variable A.

A la ligne 7, nous affectons la variable pointeur PEntier à l'adresse en mémoire de la variable A contenant la valeur 10.

Aux lignes 8 et 9, nous affichons la valeur de la variable A.

Application 2

```
1.  Algo App2
2.  Variable
3.      PEntier    : ↑Entier
4.      A          : Entier
5.  Début
6.      A ← 10
7.      AfficherLigne (A)
8.      PEntier ← Adresse (A)
9.      PEntier↑ ← A + PEntier↑
10.     AfficherLigne (A)
11. FinAlgo
```

Exercice : Analyser ce que produit l'application 2

7.3 Allocation et Désallocation

Le système d'exploitation met à la disposition du programmeur une zone mémoire suffisante appelée TAS ou HEAP (en anglais). Cette zone n'est accessible qu'avec les variables de type pointeur. La variable pointée est une variable dynamique car elle sera créée pendant l'exécution du programme.

7.3.1 Allocation

Le programmeur peut faire une demande de réservation d'un espace mémoire au niveau du HEAP grâce à la procédure CREER (VariablePointeur). Si la demande est satisfaite, une zone mémoire est réservée pour la variable pointeur sinon la variable pointeur aura comme valeur NIL (Not In List). NIL signifie que le pointeur ne pointe sur aucune variable.

7.3.2 Désallocation

L'opération de désallocation est aussi appelée libération. Pour libérer (supprimer) une zone mémoire occupée par une variable pointeur, nous utilisons la procédure LIBERER (variable pointeur).

Grâce à ces opérations plusieurs structures de données prennent naissance :

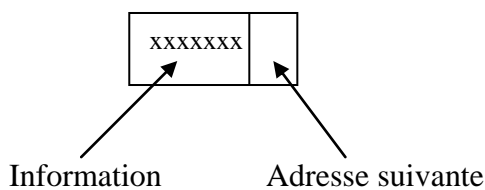
- Liste linéaire (monodirectionnelle, bidirectionnelle)
- Pile dynamique
- File dynamique
- Arbre dynamique

7.4 Les listes

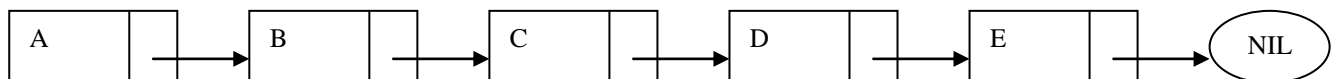
Les structures linéaires sont un des modèles de données les plus élémentaires utilisés dans les programmes informatiques. Elles organisent les données sous forme de séquence non ordonnée d'éléments accessibles de façon séquentielle. Tout élément d'une séquence, sauf le dernier, possède un successeur.

Les opérations d'ajout et de suppression sont des opérations de bases sur les listes. Selon la méthode utilisée, on distingue plusieurs sortes de listes : piles, files, etc.

7.4.1 Représentation graphique



7.4.2 Les listes linéaires



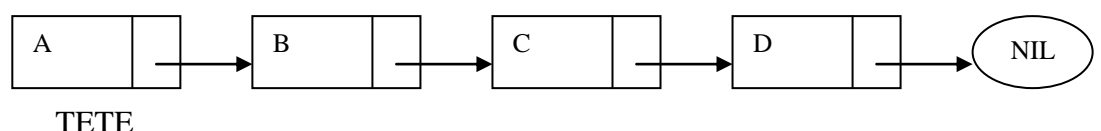
Une liste linéaire n'est rien d'autre qu'une succession de cellules reliées entre elles par des pointeurs.

7.4.3 Déclaration d'une structure linéaire

```
Type Pointeur = ↑NomEnregistrement
    NomEnregistrement = Structure
        Champ      : Type
        Suivant    : Pointeur
    FinStructure
```

7.4.4 Opérations sur les listes linéaires monodirectionnelles

Soit la liste ci-dessous.



TETE est l'adresse de la première cellule de la liste. Elle ne doit jamais être déplacée en cours d'application.

7.4.4.1 Affichage d'informations

Pour afficher la valeur contenue d'une cellule, il suffit d'écrire

```
AfficherLigne (VariablePointeur↑.Champ)
```

7.4.4.2 Passage d'une adresse à la suivante

Le passage d'une cellule à une autre se fait de la manière suivante :

```
(VariablePointeur) ← VariablePointeur↑.Suivant
```

7.4.4.3 Opération de parcours

Pour parcourir la liste, il suffit d'initialiser la variable P à partir de la tête de liste (**TETE**) et de passer au suivant jusqu'à **NIL**.

Ecrire la fonction LongListe (donnée Liste : Pointeur) : entier qui calcule le nombre d'éléments d'une liste linéaire.

```
Fonction LongListe (donnée TETE : Pointeur) : entier
Variable
    P          : Pointeur
    NbrElement : Entier
Début
    P ← TETE
    NbrElement ← 0
    TantQue (P <> Nil) Faire
        NbrElement ← NbrElement + 1
        P ← P↑.Suivant
    FinTantQue
    LongListe ← NbrElement
FinFonction
```

Ecrire la fonction qui donne le nombre d'occurrence de la valeur Val de la liste TETE.

```
Fonction NombreOcc (donnée TETE : Pointeur; Val : entier) : Entier
Variable
    P : Pointeur
    Nbr : Entier
Début
    Nbr ← 0
    P ← TETE
    TantQue (P <> Nil) Faire
        Si (P↑.valeur = val) Alors
            Nbr ← Nbr + 1
        Finsi
        P ← P↑.Suivant
    FinTantQue
    NombreOcc ← Nbr
FinFonction
```

Ecrire la Fonction RapportHommeFemme (Donnée TETE : Pointeur) : Réel qui calcule le rapport des hommes par rapport aux femmes de la liste TETE.

Voici la structure de la liste

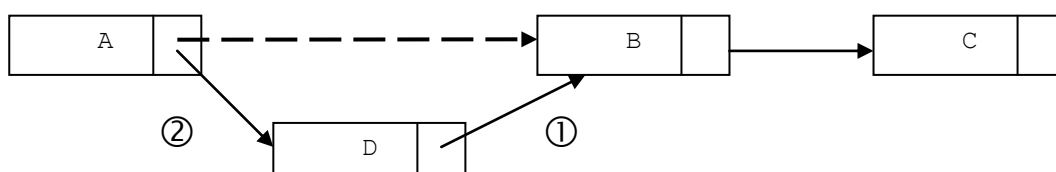
```
Type Pointeur = ↑NomEnregistrement
  NomEnregistrement = Structure
    Sexe      : Caractere % "F" = Féminin, "M" = Masculin %
    Suivant   : Pointeur
  FinStructure
```

```
Fonction RapportHommeFemme (Donnée TETE : Pointeur) : Réel
Variable
  P          : Pointeur
  NbrH, NbrF : Entier
Début
  NbrH ← 0 ; NbrF ← 0
  P ← TETE
  TantQue (P <> Nil) Faire
    Si P↑.Sexe = "H" Alors
      NbrH ← NbrH + 1
    Sinon
      NbrF ← NbrF + 1
    FinSi
  FinTantQue
  Si NbrF = 0 Alors
    AfficherLigne "la liste n'est constituée que d'hommes"
    RapportHommeFemme ← (-1)
  Sinon
    RapportHommeFemme ← NbrH / NbrF
  FinSi
FinFonction
```

7.4.4.4 Opération d'ajout

Supposons que l'on ait une liste formée en ordre par les cellules nommées : {A, B et C}
Il nous est demandé d'ajouter à l'intersection de A et B la cellule de valeur D.

Le schéma suivant montre comment ajouter cette cellule dans une liste.



Processus : il faut d'abord créer la cellule D. Se positionner sur la cellule A.

- ① : créer le lien entre D et B.
- ② : modifier le lien entre A et B par le nouveau lien A, D

Ecrire la Procédure Chaînage (Donnée A, B, D : Pointeur) qui dessine le schéma ci-dessus.

```
Procédure chaînage (Donnée Résultat A, B, D : Pointeur)
Début
    D↑.Suivant ← B
    A↑.Suivant ← D
FinProcédure
```

Ecrire la Procédure AjoutCellule (Donnée TETE, Pnt : Pointeur; Posit : entier; Résultat TETE : Pointeur) qui ajoute le pointeur Pnt à la position Posit sur la liste.

Plusieurs cas peuvent être possible. Nous utiliserons la fonction **LongListe** précédemment vue.

- **Cas 1** : Posit = 1 ; alors Pnt devient la tête de liste.
- **Cas 2** : 1 < Posit < **LongListe (TETE)** ; alors nous appliquons le schéma précédent.
- **Cas 3** : Posit > **LongListe (TETE)** ; opération impossible

```
Procédure AjoutCellule (Donnée TETE, Pnt : Pointeur, Posit, N :
entier ; Résultat TETE : Pointeur)
Variable
    P, Precedent : Pointeur
    Nbr          : entier % le nombre de cellules parcourus %
Début
    Si Posit = 1 Alors % ajout sur la tête de liste %
        Pnt↑.Suivant ← TETE
        TETE ← Pnt
    Sinon
        Si Posit > LongListe (TETE) Alors
            AfficherLigne "Opération impossible"
        Sinon
            P ← TETE
            Nbr ← 1
            TantQue (Nbr < Posit) Faire
                Nbr ← Nbr + 1
                Precedent ← P
                P ← P↑.Suivant
            FinTantQue
            Chaînage (Precedent, P, Pnt)
        FinSi
    FinSi
FinProcédure
```

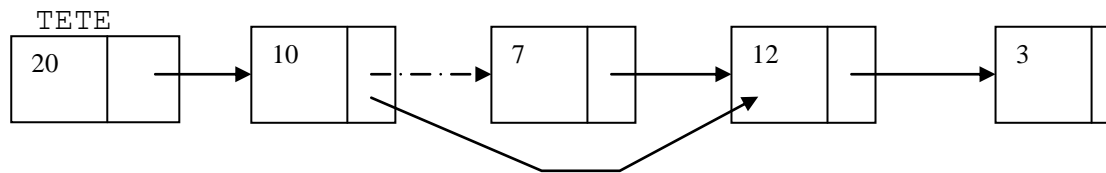
7.4.4.5 Opération de suppression

Le schéma suivant illustre comment supprimer un élément d'une liste. La suppression peut se faire de deux manières :

- **la suppression physique** : la cellule à supprimer est physiquement supprimée dans le support.

- **la suppression logique** : la cellule à supprimer est ignorée. Mais elle existe au niveau du HEAP.

Le schéma suivant montre comment procéder pour la suppression logique. Supprimons la cellule portant la valeur 7.



La suppression logique est une opération très facile, il suffit de faire « *un bond* » d'une cellule à une autre.

Ecrire la Procédure SupprimeChainage (Données Résultat A, B : Pointeur) qui supprime logiquement la cellule B.

Processus :

- ① : Il faut d'abord se positionner sur la cellule B tout en mémorisant la cellule A.
- ② : Créer le lien entre A et le suivant de B.

Procédure SupprimeChainage (Donnée Résultat A, B : Pointeur)
Début
 A↑.Suivant ← B↑.Suivant
FinProcédure

Ecrire la Procédure SupprimeCellule (données TETE : Pointeur ; Val : entier; Résultat TETE : Pointeur) qui supprime la première occurrence de la valeur VAL de la liste TETE.

Deux cas peuvent être possibles :

Cas 1. Val = TETE↑.Valeur ; Alors TETE devient son suivant : (TETE ← TETE↑.Suivant)

Cas 2. Val entre TETE et la fin de la liste, ainsi il faut alors chercher cette valeur puis utiliser la procédure SupprimeChaine.

Procédure SupprimeCellule (Donnée TETE : Pointeur ; Val : entier ;
Résultat TETE : Pointeur)
Variable
 P, Precedent : Pointeur
 Trouver : Booléen; % Permet d'arrêter le parcours si l'élément est trouvé %
Début
 Si TETE↑.Valeur = Val **Alors**
 TETE ← TETE↑.Suivant
 Sinon
 Trouver ← Faux
 P ← TETE

```

    TantQue (Trouver = Faux) ET (P <> Nil) Faire
        Precedent ← P
        P ← P↑.Suivant
        Si P <> Nil Alors
            Si P↑.Valeur = Val Alors
                SupprimeChainage (Precedent, P)
                Trouver ← Vrai
            FinSi
        FinSi
    FinTantQue
FinSi
FinProcédure

```

7.4.4.6 La création d'une liste

Nous utiliserons la Procédure CREER (VariablePointeur) pour créer dynamiquement une cellule au niveau du HEAP.

Ecrire la Procédure CreerListe (TETE : Pointeur ; N : entier) qui crée une liste de N cellules.

```

Procédure CreerListe (Donnée TETE : Pointeur; N : entier; Résultat
TETE : Pointeur)
Variable
    P, Precedent : Pointeur
    i : Entier
Début
    Pour i=1 à N Faire
        Si i=1 Alors
            CREER (TETE)
            Si TETE <> Nil Alors
                Saisir (TETE↑.Valeur)
                TETE↑.Suivant ← NIL
                Precedent ← TETE
            Sinon
                AfficherLigne "Mémoire insuffisante"
            FinSi
        Sinon
            CREER (P)
            Si P <> NIL Alors
                Saisir (P↑.Valeur)
                Chaînage (Precedent, NIL, P)
                Precedent ← P
            Sinon
                AfficherLigne "Mémoire insuffisante"
            FinSi
        FinSi
    FinPour
FinProcédure

```

7.4.4.7 La suppression physique

Nous utiliserons la Procédure LIBERER (VariablePointeur) pour supprimer, au niveau du TAS, une variable pointeur.

Ecrire la Procédure SupprimOccurrence (Donnée TETE : Pointeur ; VAL : Entier ; Résultat TETE : Pointeur) qui supprime physiquement toutes les occurrences de valeur VAL au niveau de la liste TETE.

```
Procédure SupprimOccurrence (Donnée TETE : Pointeur ; VAL :  
Entier ; Résultat TETE : Pointeur)  
Variable  
    P, Precedant, Pnt : Pointeur  
Début  
    % suppression de toutes les valeurs Val de tête. %  
    TantQue (TETE↑.Valeur = VAL) ET (TETE <> NIL) Faire  
        Precedant ← TETE  
        TETE ← TETE↑.Suivant  
        Liberer (Precedant)  
    FinTantQue  
    P ← TETE ; Precedant ← P  
    TantQue (P <> NIL) Faire  
        Si P↑.Valeur = Val Alors  
            TantQue (P↑.Valeur = Val) ET (P <> NIL) Faire  
                Pnt ← P  
                P ← P↑.Suivant  
                Liberer (Pnt)  
            FinTantQue  
            Precedant↑.Suivant ← P  
            Precedant ← P  
        Sinon  
            P ← P↑.Suivant  
        FinSi  
    FinTantQue  
FinProcédure
```

7.4.4.8 Chaîne de caractères dynamiques

Ecrire la procédure qui crée une chaîne de caractères qui pourrait dépasser la taille 255.

```
Algo ChaineDynamique  
Type PCaractere = ↑Caract  
    Caract = Structure  
        C          : Caractère  
        Suivant    : PCaractere  
    FinStructure  
Variable  
    TETE, P          : PCaractere  
    N                : Entier %Taille de la chaîne%
```

Début

```

Afficher "Donnez la longueur de votre chaîne :"
Saisir (N)
TETE ← NIL
CreerListe (TETE, N)
% affichage de la chaîne de caractères %
P ← TETE
TantQue P <> NIL Faire
    AfficherLigne (P↑.Valeur)
    P ← P↑.Suivant
FinTantQue
FinAlgorithme

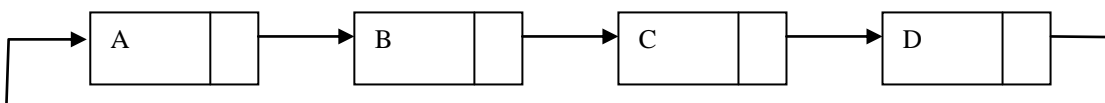
```

Note : Ce type de chaîne est très utilisé en informatique à cause de la limitation du type String. Les langages C et C++ ne disposent pas de type String ; ils utilisent les chaînes dynamiques.

7.4.5 Liste circulaire

On appelle liste circulaire une liste dont le dernier élément pointe sur le premier. Grâce à une telle définition, il n'est plus utile de connaître l'adresse de la tête de liste pour accéder aux autres éléments.

7.4.5.1 Représentation physique



Ecrire la fonction NBRElement qui énumère le nombre de cellules contenues dans la liste circulaire CListe.

Mise en garde : Attention à la boucle infinie.

```

Fonction NBRElement (Donnée CListe : Pointeur) : entier
Variable
    P      : Pointeur
    Nbr    : Entier
Début
    Si CListe = Nil Alors
        Nbr ← 0
    Sinon
        P ← CListe
        Nbr ← 1
        P ← P↑.Suivant
        TantQue P <> CListe Faire
            Nbr ← Nbr + 1
            P ← P↑.Suivant
        FinTantQue
        NbrElement ← Nbr
    FinSi
FinFonction

```

7.4.6 Les Piles

Une pile est une liste linéaire particulière dont l'ajout se fait toujours à partir du dernier élément. La lecture se fait à partir du dernier élément de la liste (celle-ci est appelée SOMMET). Les piles sont très utilisées dans la conception des compilateurs, des évaluations des expressions, dans les logiciels de traitements de texte, etc.

On distingue deux représentations de piles :

- Représentation statique
- Représentation chaînée (dynamique)



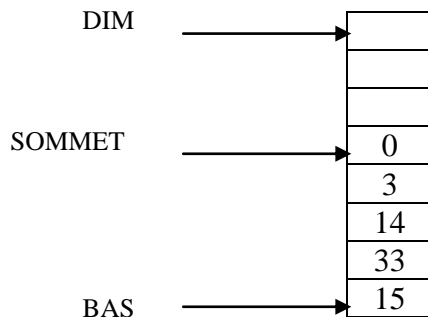
Pile de livres

7.4.6.1 Représentation d'une pile statique

```
Constante TailleMax = Valeur
Type Pile = Tableau [1..TailleMax] de TypeDonnée
Variable
    P      : Pile
```

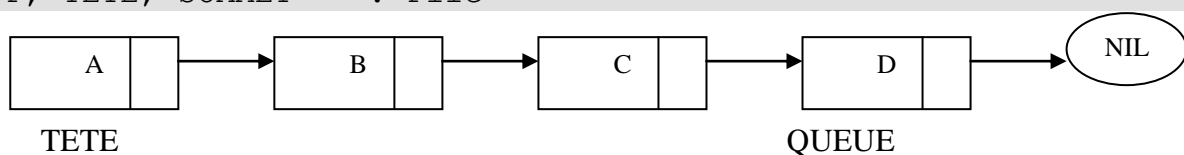
Exemple

```
Constante DIM = 30
Type Pile = Tableau [1..DIM] de Entier
Variable
    P          : Pile
    QUEUE, SOMMET : Entier
```



7.4.6.2 Représentation d'une pile chaînée (dynamique)

```
Type Pointeur = ↑Element
Element = Structure
    Valeur      : Entier
    Suivant     : Pointeur
FinStructure
Variable
    P, TETE, SOMMET : Pile
```



7.4.6.3 Opération sur les piles

- **Opération d'ajout** : Cette opération est aussi appelée opération d'empilement.

Ecrire la **Procédure Empiler** (Donnée P : Pile ; Val, SOMMET : Entier ; Résultat P) qui ajoute la valeur VAL sur une pile.

Cas d'une pile statique

```
Procédure Empiler (Données P : Pile; Val, SOMMET : Entier;
Résultat P)
Début
    Si SOMMET = SOMMET Alors
        AfficherLigne "Pile Pleine"
    Sinon
        SOMMET ← SOMMET + 1
        P [SOMMET] ← Val
    FinSi
FinProcédure
```

Cas d'une pile chaînée

```
Procédure Empiler (Données TETE : Pile; Val : Entier; Résultat
TETE)
Variable Pnt : Pile
Début
    CREER (Pnt)
    Si Pnt <> NIL Alors
        Pnt↑.Valeur ← Val
        Pnt↑.Suivant ← NIL
        Si TETE = NIL Alors
            Tete ← Pnt
            SOMMET ← Pnt
        Sinon
            SOMMET↑.Suivant ← Pnt
            SOMMET ← Pnt
        FinSi
    FinSi
FinProcédure
```

- **Lecture au sommet**

Dans une pile, nous ne pouvons lire que l'élément au sommet. C'est comme sur la pile de livres, seul le livre au sommet peut être retiré.

Ecrire la **Procédure ValSommet** (P, SOMMET : Pile ; Résultat Valeur) qui donne la valeur du sommet de la pile.

Cas d'une pile statique

```
Fonction ValeurSommet (donnée P, SOMMET : Pile ; Résultat Valeur)
Début
    Si Sommet = 0 Alors
        AfficherLigne "Pile vide"
    Sinon
        Valeur ← P [SOMMET]
    FinSi
FinProcédure
```

Cas d'une pile chaînée

```
Fonction ValeurSommet (donnée TETE, SOMMET : Pile ; Résultat
Valeur)
Début
    Si TETE = NIL Alors
        AfficherLigne "Pile vide"
    Sinon
        Valeur ← SOMMET↑.Valeur
    FinSi
FinProcédure
```

- Opération de lecture

Cette opération est aussi appelée dépilage. Elle donne la valeur du sommet qui devient par la suite son précédent.

Ecrire la Procédure Depiler réalise cette opération.

Cas d'une pile statique

```
Fonction Depiler (donnée P, SOMMET : Pile ; Résultat Valeur)
Début
    Si Sommet = 0 Alors
        AfficherLigne "Pile vide"
    Sinon
        ValeurSommet [P, SOMMET, Valeur]
        SOMME ← SOMMET -1
    FinSi
FinProcédure
```

Cas d'une pile chaînée

```
Procédure Depiler (donnée TETE, SOMMET : Pile ; Résultat Valeur)
Variable
    P, AvantDernier : Pile
Début
    P ← TETE
    AvantDernier ← NIL
```

```

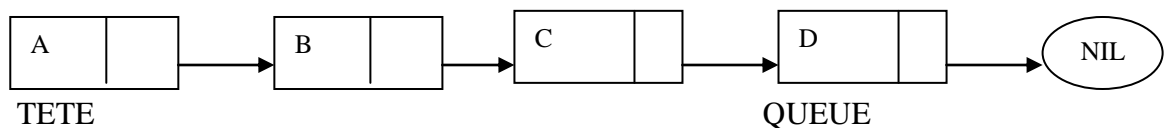
TantQue (P↑.Suivant <>NIL) Faire
    AvantDernier ← P
    P ← P↑.Suivant
FinTantQue
ValeurSommet (TETE, SOMMET, Valeur)
Si SOMMET = TETE Alors
    TETE ← AvantDernier
Sinon
    SOMMET ← AvantDernier
FinSi
FinProcédure

```

7.4.7 Les Files

Une file est une liste linéaire particulière dont l'ajout se fait à partir de la QUEUE (dernière cellule d'une liste) et la suppression à partir de la TETE.

7.4.7.1 Représentation chaînée



7.4.7.2 Déclaration d'une file chaînée

```

Type Pile = ↑Cellule
Cellule = Structure
    Valeur : Type
    Suivant : Pile
FinStructure
Variable
    TETE, QUEUE : Pile

```

7.4.7.3 Représentation statique

TETE					QUEUE					DIM				
15	8	3	7	17	3									

7.4.6.4 Déclaration d'une file statique

```

Const Dim = 15
Type File = Tableau [1..Dim]
Variable
    TETE, QUEUE : Entier

```


7.4.7.5 Opérations classiques sur les files

CAS D'UNE FILE DYNAMIQUE

Opération d'ajout

Ecrire la Procédure qui ajoute une valeur Val dans la file FListe.

```
Procédure Ajout (donnée File ; Val : Type-valeur ; Résultat File)
Variable P : Pointeur
Début
    Créer (P)
    Si P = NIL Alors
        AfficherLigne "pas d'espace"
    Sinon
        P↑.Valeur ← Val
        P↑.Suivant ← NIL
        Si TETE = NIL Alors
            QUEUE ← P
            TETE ← P
        Sinon
            QUEUE↑.Suivant ← P
            QUEUE ← P
        FinSi
    FinSi
FinProcédure
```

Opération de suppression

Ecrire la procédure qui supprime une valeur dans une file.

```
Procédure Supprime (Donnée File ; Résultat Val)
Variable
    Precedant : Pointeur
Début
    Si TETE = NIL Alors
        AfficherLigne "impossible de supprimer"
    Sinon
        Val ← TETE↑.Valeur
        Precedant ← TETE
        TETE ← TETE↑.Suivant
        Libérer (Precedant)
        Si TETE = NIL Alors
            QUEUE ← NIL
        FinSi
    FinSi
FinProcédure
```

CAS D'UNE FILE STATIQUE

Opération d'ajout

```
Procédure Ajout (Donnée File ; Val : Type-valeur ; Résultat File)
Début
    Si Queue = DIM Alors
        QUEUE ← 1
    Sinon
        QUEUE ← QUEUE + 1
    FinSi
    Si QUEUE = TETE Alors
        AfficherLigne "File Pleine"
    Sinon
        File [QUEUE] ← Val
        Si TETE = 0 Alors
            TETE ← 1
        FinSi
    FinSi
FinProcédure
```

Opération de suppression

Ecrire le sous-programme qui supprime une valeur dans une file circulaire.

```
Procédure Supprime (Donnée File : File; Résultat Val : Type-
Valeur)
Début
    Si TETE = 0 Alors
        Afficher "Erreur suppression"
    Sinon
        Val ← File [TETE]
        Si TETE = QUEUE Alors
            TETE ← 0
            QUEUE ← 0
        Sinon
            Si TETE = DIM Alors
                TETE ← 1
            Sinon
                TETE ← TETE + 1
            FinSi
        FinSi
    FinSi
FinProcédure
```

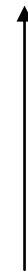
Projets

APPLICATIONS SUR LES PILES

Application 1 : Réalisation d'un programme qui convertit un nombre décimal en binaire.

Exemple : 128_{10} donne 10000000_2 . Comment le réaliser ?

Opérations	Quotient	Reste
128 : 2 =	64	0
64 : 2 =	32	0
32 : 2 =	16	0
16 : 2 =	8	0
8 : 2 =	4	0
4 : 2 =	2	0
2 : 2 =	1	0
1 : 2 =	0	1



Sens de lecture

Il faut procéder à une division successive du quotient par 2 jusqu'à trouver un quotient nul tout en empilant les restes dans une pile. L'opération de dépile de la pile donne l'équivalence du nombre en binaire.

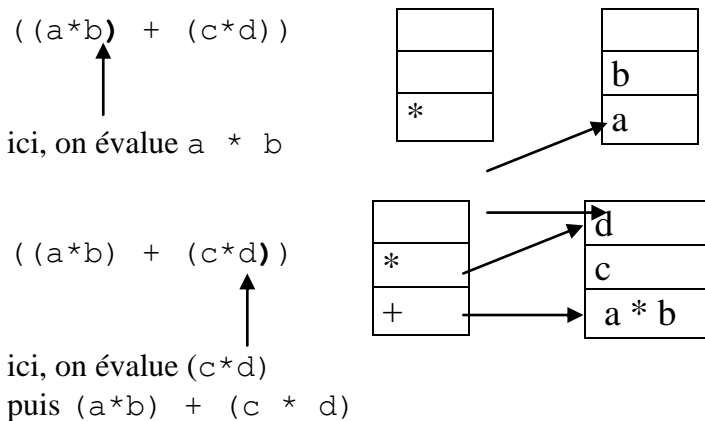
```
Procédure DecimalBinaire (Donnée Nombre : Entier)
Variable
    P                : Pile
    Reste            : Entier
Début
    SOMMET  $\leftarrow$  0 % initialise le sommet de la pile %
    Repeter
        Empile (P, Nombre Div 2)
        Nombre  $\leftarrow$  Nombre Div 2
    Jusqua (Nombre = 0)
    TantQue SOMMET > 0 Faire
        Depile (Pile, Reste)
        Afficher (Reste)
    FinTantQue
FinProcédure
```

Application 2 : Réalisation d'une calculette

Evaluation d'une expression complètement parenthésée

L'analyse d'une expression complètement parenthésée se fait de gauche à droite. Chaque opérande rencontré est mis dans une pile dite « pile des opérandes », chaque opérateur étant stocké dans une « pile des opérateurs ».

Chaque fois que l'on rencontre une parenthèse fermante, on évalue le résultat de l'opération entre les deux opérandes du haut de la pile avec l'opérateur du haut de la pile, puis le résultat est rangé (empilé) à la place des deux précédents opérandes.



Ecrire la **Fonction EvalParenthese** (Donnée **ExPar** : chaîne) : Réel qui évalue l'expression complètement parenthésée ExPar.

Algorithme :

Toutes les variables ainsi que les opérateurs sont empilés au niveau de la pile opérateur. Les opérandes sont empilés au niveau de la pile opérande.

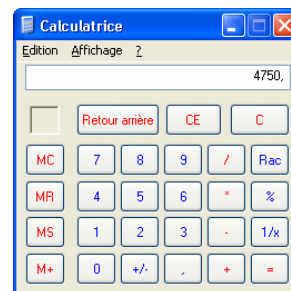
L'opération de calcul se déclenche à la rencontre d'une parenthèse fermante et le résultat est empilé à nouveau au niveau de la pile opérande.

Correction : Pour la correction, voir au niveau de MSAlgoPascal le projet du nom de **EvalExp.ALG**

Application 3 : Passage d'une expression complètement parenthésée à une Notation Polonaise Postfixée (n.p.p)

Exemple :

$(a + b)$ donne $a b +$
 (\sqrt{a}) donne $a \sqrt{}$
 $((a + b) / c)$ donne $a b + c /$



Certaines calculettes utilisent des processeurs à n.p.p. Ils utilisent une pile pour analyser les instructions.

Processus de traduction : Pour faire le passage d'une expression complètement parenthésée à une Notation Polonaise Postfixée, nous utilisons une pile de caractères.

La chaîne à analyser sera lue de gauche à droite. L'algorithme de passage est le suivant :

- les parenthèses ouvrantes sont ignorées.
- les opérandes rencontrés sont immédiatement écrits vers la chaîne résultat.
- les opérateurs sont empilés.
- la rencontre d'une parenthèse fermante provoque l'écriture sur la chaîne résultat l'élément dépilé au niveau de la pile

Exemple : Soit à traduire l'expression $((A + (B * C)) * (\sqrt{D + E}))$

Les étapes de traduction sont les suivantes :

Chaîne d'entrée	PILE	Chaîne de sortie
((A + (B * C)) * (√ (D + E)))	vide	vide
A + (B * C)) * (√ (D + E)))	vide	vide
+ (B * C)) * (√ (D + E)))	vide	A
(B * C)) * (√ (D + E)))	+	A
* C)) * (√ (D + E)))	+	AB
C)) * (√ (D + E)))	++	AB
) * (√ (D + E)))	++	ABC
) * (√ (D + E)))	+	ABC*
* (√ (D + E)))	vide	ABC*+
(√ (D + E)))	*	ABC*+
(D + E)))	*√	ABC*+
+ E)))	*√	ABC*+D
E)))	*√+	ABC*+D
)))	*√+	ABC*+DE
))	*√	ABC*+DE+
)	*	ABC*+DE+√
vide	vide	ABC*+DE+√X

Exercice : Ecrire un sous-programme qui effectue la traduction d'une expression parenthésée à une notation polonaise postfixée.

Correction : Pour la correction, voir au niveau de MSAIgoPascal le projet du nom de **ParPost.ALG**

Application 4 : Evaluation d'une Notation Polonaise Postfixée (n.p.p)

L'analyse d'une chaîne de symboles en n.p.p se fait en suivant les règles que voici :

- Au début la pile est initialisée à vide.
- Le premier symbole de la chaîne est toujours un opérande. Il est empilé.
- Ensuite, nous analysons symbole par symbole, les éléments de la chaîne.
- Si le symbole rencontré est un opérateur dyadique ou binaire, il faut qu'il y ait au moins deux éléments dans la pile. Nous dépilons ces deux éléments et leur appliquer la fonction représentée par l'opérateur dyadique et empiler le résultat.
- Pour un opérateur n-adique (avec $n > 0$), on dépile n élément(s) de la pile, on applique la fonction à ces n éléments et on empile le résultat.
- A la fin du processus, il doit rester un seul élément dans la pile. C'est le résultat de l'expression.

Exercice : Ecrire un sous-programme qui évalue une notation polonaise postfixée.

Correction : Pour la correction, voir au niveau de MSAIgoPascal le projet **EvalPost.ALG**

APPLICATION SUR LES FILES

Application : Réalisation du célèbre jeu vidéo **SNAKE** (serpent) de Nokia.





Introduction : Au niveau des téléphones portables Nokia, il existe un jeu intitulé "**SNAKE**" (*serpent*) qui se déplace au niveau de l'écran tout en essayant de manger des biscuits pour grandir.

Le principe est très simple :

- Les touches directionnelles de votre clavier font changer le sens (direction) de déplacement du serpent.
- Le jeu s'arrête si la tête du serpent heurte une partie de son corps ou les bords de l'écran.

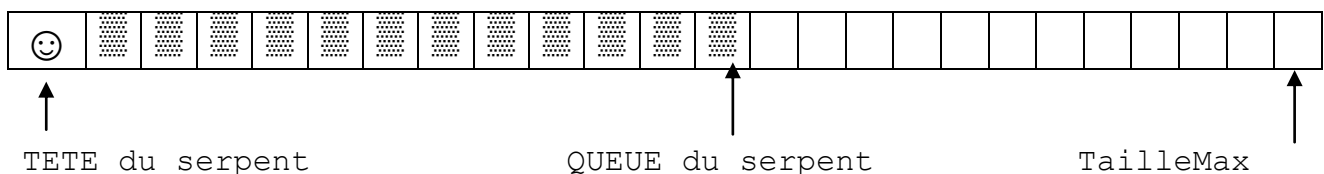
NB : On ne peut arrêter le serpent ni le faire reculer.

Le serpent n'est rien d'autre qu'une file particulière dont chaque cellule est formée par les couples de coordonnées (x, y) où x représente l'abscisse et y l'ordonnée. Pour stimuler le déplacement avec les touches directionnelles de votre clavier on a :

Boutons du clavier	Evénements
	$y \leftarrow y - 1$
	$x \leftarrow x + 1$
	$y \leftarrow y + 1$
	$x \leftarrow x - 1$

Voici la structure du serpent :

```
Constante TailleMax = 50
Type Cellule = Structure
    x      : Entier
    y      : Entier
FinStructure
Variable Serpent : Tableau [1..TailleMax] de Cellule
    Biscuit : Cellule
```

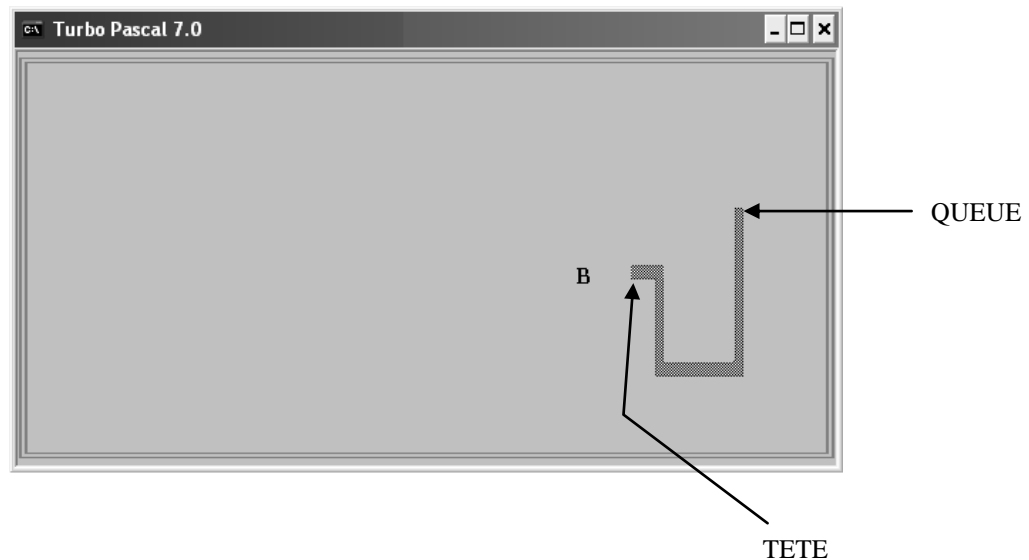


Tous les déplacements sont effectués à partir de la tête du serpent (Tête de file). A chaque modification des coordonnées de la tête, nous mémorisons ces coordonnées antérieures qui deviennent les nouvelles coordonnées de la deuxième cellule. Cette dernière aussi mémorise ces coordonnées antérieures qu'elle passe à la troisième cellule. Et le processus continue jusqu'à la

queue. Nous affichons une case vide au niveau des coordonnées antérieures de la queue pour créer l'illusion du déplacement du serpent.

Au début nous disposons de N cellules ($N < \text{TailleMax}$). A chaque fois que l'on mange un biscuit (coordonnée de la tête = coordonnée du biscuit), nous déclenchons l'algorithme d'ajout dans une file et N devient N+1 (le serpent grandi d'une cellule).

A chaque déplacement, nous vérifions si `serpent[i].x` et `serpent[i].y` (avec $1 < i \leq N$) ne coïncide pas avec la tête c'est-à-dire `serpent[1].x` et `serpent[1].y` mais aussi nous garderons l'œil sur le bord de l'écran.



Simulation du jeu SNAKE

Exercice : Ecrire le programme qui réalise cette simulation.

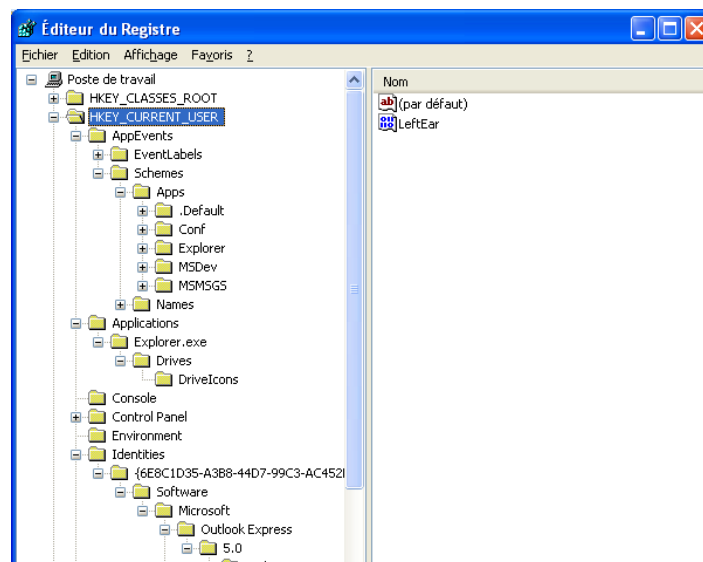
Correction : Pour la correction, voir au niveau de MSAalgoPascal le projet du nom de **Serpent.ALG**

Chapitre VIII Les arbres

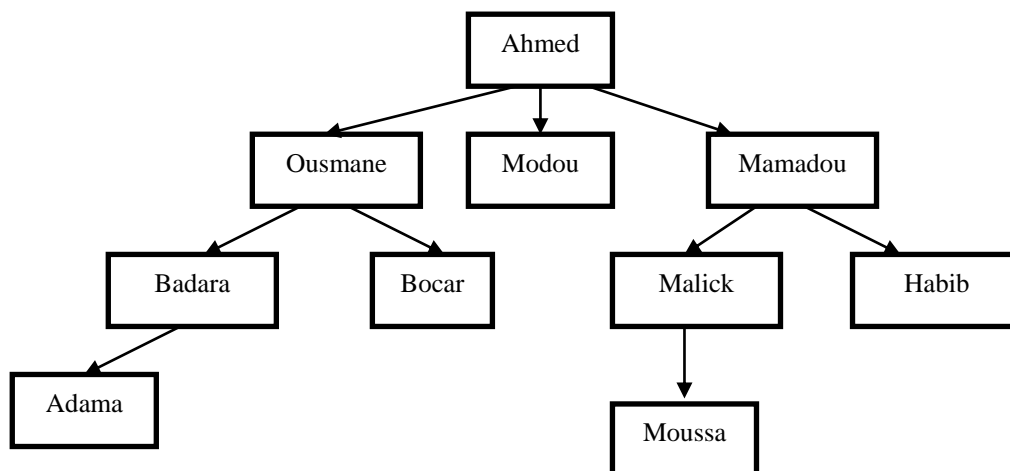
On appelle arbre une collection d'informations formant une structure hiérarchique. Les informations arborescentes sont très utilisées dans la vie courante. Nous pouvons citer :

L'armée : Dans le commissariat de l'armée de terre, on distingue les commissaires généraux de division et de brigade, les commissaires-colonels, les lieutenants-colonels, les commandants, les capitaines, les lieutenants et les sous-lieutenants ; dans la marine, les commissaires généraux de première classe, les commissaires généraux de deuxième classe, les commissaires en chef de première classe, les commissaires en chef de deuxième classe, les commissaires principaux et les commissaires de première classe.

L'informatique avec l'explorateur Windows (exploration de la base de registre)



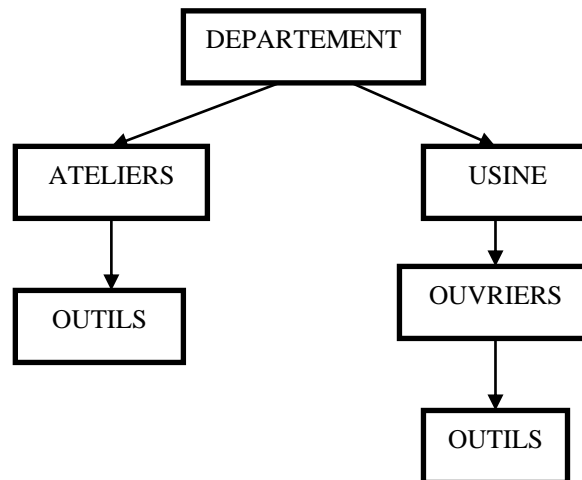
L'arbre généalogique (Exemple : famille FALL)



NB : Nous avons supposé que chaque père de famille a au plus deux enfants. *Cet exemple marche très bien en République Populaire de Chine.*

Les systèmes de gestion de bases de données (SGBD) hiérarchiques

Exemple de base de données d'une entreprise industrielle



8.1 Définition de quelques concepts

Nœud : Un arbre est une collection d'éléments (informations) appelés nœuds.

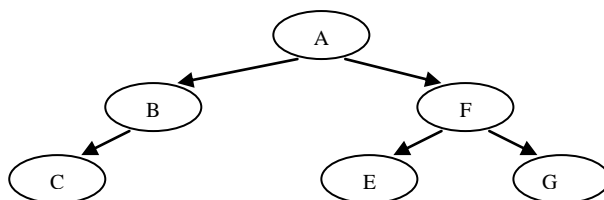
Racine : Un arbre a un nœud particulier appelé racine.

Au niveau de l'armée, la racine représente des **commissaires généraux**.

Pour l'explorateur Windows, la racine représente le lecteur **C:** ou le **poste de travail**

Pour l'arbre généalogique il représente **Ahmed** et sur la base de données de l'entreprise industrielle elle est le **département**.

Soit l'arbre suivant :



Le nœud A désigne la racine.

Pères, fils et feuilles

Les nœuds B et F sont des nœuds intermédiaires.

Les nœuds C, E et G sont appelés feuilles (ils n'ont pas de descendant).

A est le père de B et de F

F est le père de E et de G

C est le fils de B

Degré d'un nœud : Le degré d'un nœud est le nombre de fils associé à ce nœud.

Arbre N-Aire : Un arbre sera dit N-Aire si chaque nœud dispose au plus N fils.

Au niveau des algorithmes qui vont suivre, nous n'utiliserons que les arbres binaires (2-aire).

Propriétés d'un arbre à parcours infixé

Dans un arbre suivant un parcours infixé, on a toujours les propriétés suivantes :

Soient **A** le père des nœuds **B** et **C** (avec B le fils gauche de A et C son fils droit) et **info** un type quelconque (information) porté par ces derniers.

- **$A↑.info \geq B↑.info$**
- **$A↑.info \leq C↑.info$**
- **Tout nœud ajouté devient automatiquement une feuille.**
- **Chaque information est représentée de façon unique au niveau de l'arbre (jamais de redondance).**

8.2 Déclaration d'un arbre binaire

```
Type Pointeur = ↑ Nœud
Nœud          = Structure
    Champ      : Type-Quelconque
    FilsGauche : Pointeur
    FilsDroit  : Pointeur
FinStructure
```

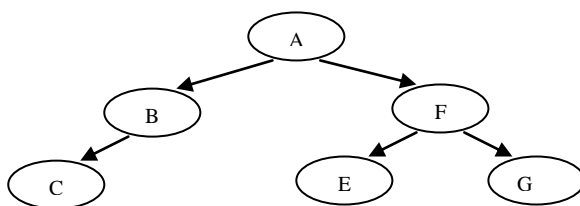
Champ peut être n'importe quelle structure.

8.3 Opérations classiques sur les arbres binaires

8.3.1 Opération de parcours

Nous avons trois types de parcours : préfixé, infixé et postfixé.

Seul le parcours préfixé sera vu dans ce chapitre. *Le lecteur intéressé aux autres parcours pourra se référer aux livres cités à la bibliographie.*



Le parcours préfixé de l'arbre donne : C B A E F G

Pour réaliser ce parcours nous utilisons deux méthodes :

- méthode itérative
- méthode récursive

Méthode itérative

Une pile d'adresses sera utilisée. Voici sa structure générale :

```
Type PileNoeud = ↑ Element
      Element = Structure
                P      : ↑ Noeud %P pointe sur un nœud de l'arbre%
                Suivant : PileNoeud
      FinStructure
```

Algorithme de parcours (Méthode itérative)

Ecrire la procédure `ParcoursInfixe` (Donnée `Racine` : Noeud) qui affiche tous les nœuds de l'arbre.

```
Procédure ParcoursInfixe (Donnée Racine : Pointeur)
Variable
    P      : Nœud
    Pile   : PileNoeud
Début
    Pile ← NIL
    P ← Racine
    TantQue (P <> NIL) ou (Pile <> NIL) Faire
        TantQue (P <> NIL) Faire
            Empiler (P)
            P ← P↑.FilsGauche
        FinTantQue
        Depiler (P)
        AfficherLigne (P↑.valeur)
        P ← P↑.FilsDroit
    FinTantQue
FinProcédure
```

Méthode récursive

Note : il existe des langages de programmation qui n'admettant pas la méthode récursive : c'est le cas du Fortran.

```
Procédure ParcoursInfixe (Donnée Racine : Pointeur)
Début
    Si Racine <> NIL Alors
        ParcoursInfixe (Racine↑.FilsGauche)
        AfficherLigne (Racine↑.Valeur)
        ParcoursInfixe (Racine↑.FilsDroit)
    FinSi
FinProcédure
```

Exercice : Ecrire le sous-programme qui recopie toutes les valeurs d'un arbre dans un vecteur (nous supposons que la taille du vecteur est suffisamment importante pour tenir toutes les informations).

8.3.2 Recherche d'une information dans un arbre

La recherche d'une valeur VAL dans un arbre (à parcours infixé) est très simple.

- Si la valeur cherchée est plus grande que la valeur du nœud alors nous poursuivons la recherche vers la droite.
- Si elle est plus petite que la valeur du nœud alors nous allons vers la gauche.
- Au cas contraire, l'information est trouvée.

```
Fonction Recherche (Donnée RACINE :Nœud;Val :Type-valeur): Booléen
Variable
    P          : Nœud
    Trouver    : Booléen
Début
    Trouver ←Faux
    Si RACINE <> NIL Alors
        P ←RACINE
        TantQue (P <> Nil) ET (Trouver = Faux) Faire
            Si P↑.Valeur > Val Alors
                P ←P↑.FilsGauche
            Sinon
                Si P↑.Valeur < Val Alors
                    P ←P↑.FilsDroit Alors
                Sinon
                    Trouver ←Vrai
                FinSi
            FinSi
        FinTantQue
    Sinon
        AfficherLigne "Arbre vide"
    FinSi
    Recherche ←Trouver
FinProcédure
```

8.3.3 Opération d'ajout

En suivant les propriétés sur les arbres suivant un parcours infixé, nous pouvons aisément ajouter un nœud.

Ecrire le sous-programme qui ajoute la valeur VAL à l'arbre de père RACINE.

Note : il faut d'abord vérifier si l'information à ajouter n'existe pas dans l'arbre.

```
Procédure Ajout (Donnée RACINE : Nœud ; Val : type-valeur ;
résultat RACINE : Nœud)
Variable
    Pere, Nd, P : Nœud
Début
```

```

If Recherche (RACINE, Val, RACINE) = Faux Alors
  CREER (P)
  Si P <> NIL Alors
    P↑.valeur ← Val
    P↑.FilsGauche ← NIL
    P↑.FilsDroit ← NIL
    Si RACINE <> NIL Alors
      Pere ← RACINE
      Nd ← RACINE
      TantQue Nd <> Nil Faire
        Pere ← Nd
        Si Nd↑.Valeur > Val Alors
          Nd ← Nd↑.FilsGauche
        Sinon
          Nd ← Nd↑.FilsDroit
        FinSi
      FinTantQue
      Si Pere↑.Valeur > Val Alors
        Pere↑.FilsGauche ← P
      Sinon
        Pere↑.FilsDroit ← P
      FinSi
    Sinon
      RACINE ← P
    FinSi
  Sinon
    AfficherLigne "Pas d'espace mémoire"
  FinSi
Sinon
  AfficherLigne "Cette valeur existe déjà dans l'arbre"
FinSi
FinProcédure

```

NB : Racine sera égale à P si l'arbre est vide (Racine = NIL) au départ.

8.3.4 Dissolution de deux arbres

La dissolution de deux arbres du même type revient à ajouter tous les nœuds de l'un sur l'autre.

Ecrire le sous-programme DissoudreArbre (Donnée Racine1, Racine2 : Nœud ; Résultat Racine1 : Nœud) qui dissout les arbres de racines Racine1 et Racine2.

Le sous-programme Ajout d'une valeur à un arbre sera utilisé pour la réalisation de l'algorithme.

```

Procédure DissoudreArbre (Donnée Racine1, Racine2 : Nœud ;
Résultat Racine1 : Nœud)
Variable
  P : Nœud
  Pile : PileNoeud
Début

```

```

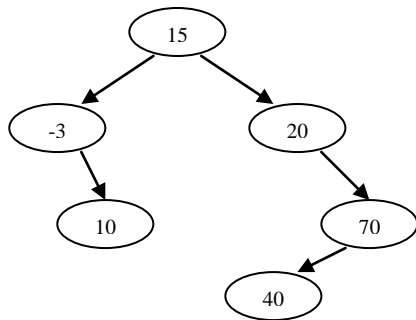
Pile ← NIL
P ← Racine2
TantQue (P <> NIL) ou (Pile <> NIL) Faire
    TantQue (P <> NIL) Faire
        Empiler (P)
        P ← P↑.FilsGauche
    FinTantQue
    Depiler (P)
    Ajout (RACINE1, P↑.Valeur, RACINE1)
    P ← P↑.FilsDroit
FinTantQue
FinProcédure

```

8.3.5 Création d'un arbre

Soit VECT : Tableau [1..N] de Entier un vecteur à valeurs entières de taille N, écrire le sous-programme suivant: Procédure CreerArbre(Donnée VECT ; Résultat Racine)

VECT = {15, -3, 20, 10, 70, 40}



Le sous-programme Ajout d'une valeur à un arbre sera utilisé pour garder l'arbre trié.

```

Procédure CreerArbre (Donnée VECT ; Résultat Racine)
Variable i : Entier
Début
    Pour i=1 à N Faire
        Ajout (Racine, Vect[i], Racine)
    FinPour
FinProcédure

```

8.3.6 Père d'un nœud

Tout nœud à l'exception de la racine a un père dans un arbre. On se propose de trouver le père d'un nœud donné.

Ecrire la fonction qui renvoie le pointeur au père d'un nœud de valeur VAL.

```

Fonction PereNoeud(donnée Racine : Nœud ; VAL :Type-Valeur) : Nœud
Variable
    P, Pere    : Nœud
    Trouver    : Booléen
Début
    Si (Racine = NIL) ou (Racine↑.Valeur = VAL) Alors
        PereNoeud ←NIL
    Sinon
        Trouver ←FAUX
        TantQue (P<>NIL) et (Trouver = Faux) Faire
            P ←Racine
            Si P↑.Valeur > VAL Alors
                P ←P↑.FilsGauche
            Sinon
                Si P↑.Valeur < VAL Alors
                    P ←P↑.FilsDroit
                Sinon
                    Trouver ←Vrai
                FinSi
            FinSi
        FinTantQue
        Si Trouver = Faux Alors
            Afficher "Elément inexistant"
            PereNoeud ←NIL
        Sinon
            PereNoeud ←Pere
        FinSi
    FinSi
FinFonction

```

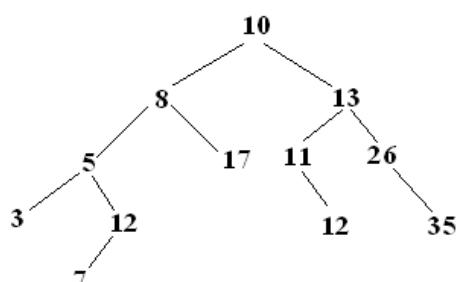
8.3.7 Opération de suppression

La suppression d'un nœud entraîne la déconnexion de l'arbre. Il faut penser alors à le reconnecter.

Cette suppression suppose trois cas :

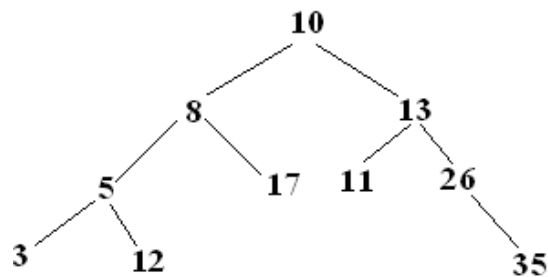
- une feuille
- la racine
- un nœud intermédiaire

Schéma



8.3.7.1 Etude du premier cas : suppression d'une feuille

Soit à supprimer la feuille portant la valeur 7, l'arbre devient ainsi :



Il faut mémoriser le père de la feuille. Ensuite mettre $Pere \uparrow .FilsGauche$ ou $Pere \uparrow .FilsDroit$ à NIL selon la nature du fils. Puis supprimer dynamiquement la feuille.

Soit la Fonction `EstFeuille` (donnée `Racine : Nœud`) : Booléen qui renvoie la valeur VRAI si `Racine` est une feuille, faux dans le cas contraire.

```
Fonction EstFeuille (donnée Racine : Nœud) : Booléen
Début
    Si Racine = NIL Alors
        EstFeuille ← FAUX
    Sinon
        Si (Racine↑.FilsGauche = NIL) et
            (Racine↑.FilsDroit = NIL) Alors
                EstFeuille ← VRAI
            Sinon
                EstFeuille ← FAUX
        FinSi
    FinSi
FinFonction
```

Ecrire la Procédure `SupprimeFeuille` (donnée `P : Nœud ; Val : Type-Valeur ; Résultat Racine : Nœud`) qui supprime la feuille portant la valeur VAL dans l'arbre `Racine`.

```
Procédure SupprimeFeuille (donnée P : Nœud; VAL:Type-Valeur ;
Résultat Racine : Nœud)
Variable
    Pere      : Nœud
Début
    Si P = Racine Alors
        Racine ← NIL
    Sinon
        Si (EstFeuille (P) = Vrai) ET (P↑.Valeur = Val) Alors
            Pere ← PereNoeud (P, Val)
            Si Pere↑.FilsGauche = P Alors
                Pere↑.FilsGauche ← NIL
```



```

        Sinon
            Pere↑.FilsDroit ←NIL
        FinSi
    FinSi
    Libérer (P)
FinSi
FinProcédure

```

8.3.7.2 Etude du deuxième cas : suppression de la racine.

Soit à supprimer la racine d'un arbre, trois cas de figure peuvent se présenter :

Cas 1 : l'arbre a un seul nœud (la racine).

Cas 2 : la racine a un fils (gauche ou droit).

Cas 3 : la racine a deux fils.

Le premier cas ne sera pas traité car si l'arbre est réduit à un seul nœud (Racine), il sera supprimé par le sous-programme précédant SupprimeFeuille car il sera considéré comme étant une feuille.

Pour le second cas, la racine a un fils.

Si la racine a un seul fils, alors ce dernier devient Racine en supprimant son père.

L'instruction de suppression est la suivante selon le cas de figure :

Racine ← Racine↑.FilsGauche ou Racine ← Racine↑.FilsDroit

Pour le dernier cas, la racine a deux fils.

En supprimant la Racine, l'arbre se déconnecte en deux. Nous relierons alors le fils gauche de la racine au dernier descendant gauche du fils droit de la racine. Et la racine devient par la suite son fils droit.

Soit le sous-programme RelierDescendantGauche (Noeud1, Noeud2 : Nœud ; Résultat Noeud2 : Nœud) qui relie le nœud Noeud1 au dernier descendant gauche de Noeud2.

```

Procédure RelierDescendantGauche (donnée Noeud1, Noeud2 : Nœud ;
Résultat Noeud2 : Nœud)
Début
    TantQue Noeud2↑.FilsGauche <> NIL Faire
        Noeud2 ← Noeud2↑.FilsGauche
    FinTantQue
    Noeud2↑.FilsGauche ← Noeud1
FinProcédure

```

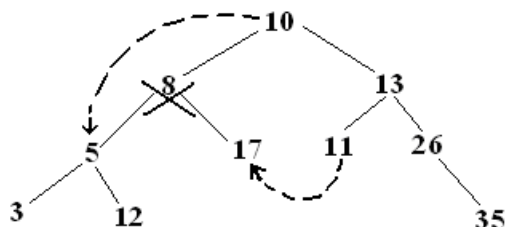
A l'aide de ces trois cas, écrire la procédure SupprimerRacine (donnée Racine : Racine ; VAL Type-Valeur Résultat Racine : Nœud) qui supprime la Racine d'un arbre ayant la valeur VAL.

```

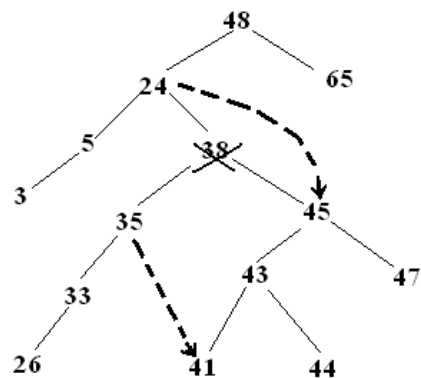
Procédure SupprimerRacine (donnée Racine : Racine ; VAL Type-
Valeur Résultat Racine : Nœud)
Variable
    SvgNoeud : Nœud
Début
    Si Racine↑.Val = VALEUR Alors
        Si EstFeuille (Racine) = Faux Alors
            SvgNoeud ← Racine
            Si Racine↑.FilsGauche = NIL Alors
                Racine ← Racine↑.FilsDroit
            Sinon
                Si Racine↑.FilsDroit = NIL Alors
                    Racine ← Racine↑.FilsGauche
                Sinon
                    RelierDescendantGauche (Racine↑.FilsGauche,
                                            Racine↑.FilsDroit)
                FinSi
            FinSi
            Liberer (SvgNoeud)
        FinSi
    FinSi
FinProcédure

```

8.3.7.3 Etude du troisième cas : Suppression d'un nœud intermédiaire.



Le nœud est le fils gauche de son père



Le nœud est le fils droit de son père

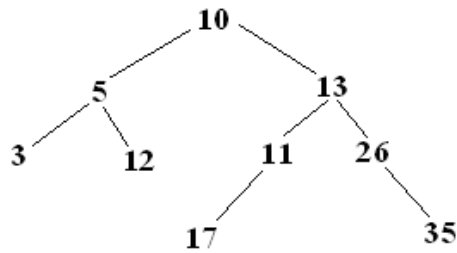
Pour supprimer le nœud intermédiaire de valeur 8, il faut :

- rattacher le fils gauche du nœud à supprimer au fils gauche de son père.
- établir la liaison entre le fils droit du nœud à supprimer au dernier descendant gauche du fils droit du père du nœud à supprimer.

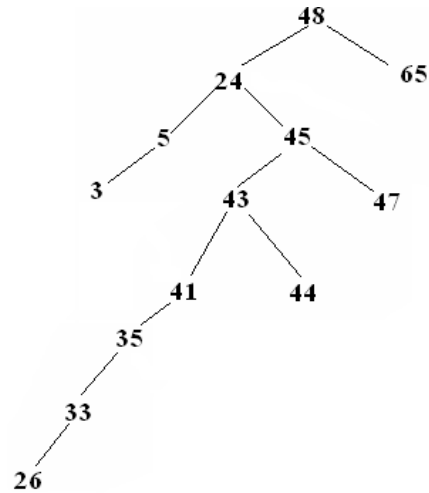
Pour supprimer le nœud intermédiaire de valeur 38, il faut :

- rattacher le fils droit du nœud à supprimer au fils droit de son père.
- établir la liaison entre le fils gauche du nœud au dernier descendant gauche du fils droit du nœud à supprimer.

Les deux schémas deviennent ainsi après la suppression.



Etat de l'arbre après la suppression du nœud de valeur 8



Etat de l'arbre après la suppression du nœud de valeur 38

```

Procédure SupprimeIntermediaire (donnée Racine : Nœud ; Val :
Type-valeur ; Résultat Racine : Nœud)
Variable
    Pere : Nœud
Début
    Si Racine↑.Valeur = VAL Alors
        Pere ← PereNoeud (Racine, Val)
        Si Pere↑.FilsDroit = Racine Alors %s'il s'agit du fils droit %
            Pere↑.FilsDroit ← Racine↑.FilsDroit
            RelierDescendantGauche (Racine↑.FilsGauche,
                                    Racine↑.FilsDroit)
        Sinon % si c'est le cas du fils gauche %
            Pere↑.FilsGauche ← Racine↑.FilsGauche
            RelierDescendantGauche (Racine↑.FilsDroit,
                                    Pere↑.FilsDroit)
        FinSi
        Libérer (Racine)
    FinSi
FinProcédure
  
```

8.3.7.4 Algorithme général pour la suppression d'un nœud

Les algorithmes précédemment vus seront utilisés pour supprimer un nœud de valeur VAL dans un arbre.

Ecrire le sous-programme qui réalise cette opération.

```

Procédure SupprimeNoeud (donnée Racine : Nœud ; Val : Type-
Valeur ; Résultat Racine : Nœud)
Variable
    P : Nœud
    Continuer : Booléen
  
```

```

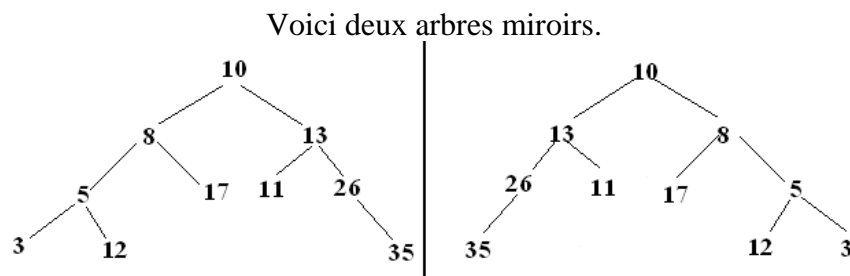
Début
  Si Racine <> NIL Alors
    P ← Racine
    Continuer ← Vrai
    TantQue (Continuer = Vrai) ET (P <> NIL) Alors
      Si P↑.Valeur = VAL Alors
        Continuer ← Faux
        Si EstFeuille (P) = Vrai Alors
          SupprimeFeuille (P, VAL)
        Sinon
          Si P = Racine Alors
            SupprimerRacine (P, VAL)
          Sinon
            SupprimeIntermediaire (P, Val)
          FinSi
        FinSi
      Sinon % nous continuons à chercher le nœud %
        Si P↑.Valeur > VAL Alors
          P ← P↑.FilsGauche
        Sinon
          P ← P↑.FilsDroit % P↑.Valeur < VAL%
        FinSi
      FinSi
    FinTantQue
  Sinon
    AfficherLigne "Arbre vide"
  FinSi
FinProcédure

```

Exercices

Exercice 1 : Soit `Racine` la racine d'un arbre d'entiers donné. Ecrire le sous-programme qui supprime tous les nœuds pairs de l'arbre.

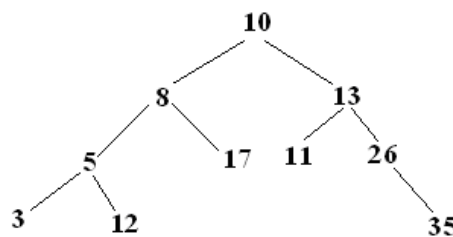
Exercice 2 : Soient deux arbres donnés A et B. Ecrire l'algorithme qui vérifie si l'arbre A est le miroir de l'arbre B.



Exercice 3 : Soit R un arbre et VAL une valeur supposée existante au niveau de l'arbre. Ecrire le sous-programme qui donne le chemin d'accès allant de la racine au nœud portant la valeur VAL.

Note : le chemin sera porté sur une liste linéaire monodirectionnelle.

Exemple : VAL = 35 ; liste = {10, 13, 26, 35}



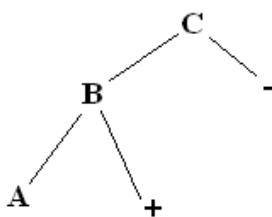
Exercice 4 : Evaluation d'une expression

Dans le chapitre précédant, nous avons vu comment passer d'une expression complètement parenthésée à une notation polonaise postfixée.

Exemple : ((A+B)-C) donne AB+C-

Nous nous proposons de porter la notation polonaise postfixée dans un arbre.

AB+C-



Travail demandé : Ecrire les sous-programmes suivants :

1) Procédure `ArbrePostFixé` (donnée `ExpressionPostFixée` : Chaîne ; Résultat `Racine` : Nœud) qui crée l'arbre de notation polonaise postfixée.

2) Fonction `EvalArbre` (donnée `Racine` : chaîne) : Réel qui évalue l'expression portée par l'arbre.

Chapitre IX Les fichiers

9.1 Le concept de fichier

Un fichier est une séquence de données binaires stockée sous forme d'octets sur un support de stockage permanent tel qu'un disque dur, une disquette ou un autre périphérique du même genre.

Au niveau des systèmes d'exploitation comme MS-DOS, Windows, Unix, Linux, les fichiers sont stockés de façon hiérarchisée. La connaissance du nom ne suffit pas pour retrouver un fichier, il faut aussi son chemin d'accès. Les fichiers sont très différents. Leurs extensions nous permettent de savoir la nature du contenu du fichier. Les fichiers .EXE et .COM sont des exécutables. Les fichiers .TXT, .INI, .BAT contiennent du texte. Les fichiers .BMP, .JPG, .GIF, .TIF, .WMF sont des images, etc.

9.2 Accès aux fichiers

Voici les règles qu'il faut pour exploiter un fichier.

9.2.1 Association

Avant de travailler (lire ou écrire) sur un fichier, il faut d'abord l'assigner par l'instruction : **Associer (VariableFichier, Nomfichier)**.

9.2.2 Ouverture d'un fichier

Avant tout traitement sur un fichier, il faut l'ouvrir suivant un mode donné :

OuvrirLecture (VariableFichier) pour y accéder en lecture,
OuvrirEcriture (VariableFichier) pour l'accès en écriture
OuvrirLectEcrit (VariableFichier) pour l'accès en lecture-écriture

9.2.3 Fermeture d'un fichier

A la fin de tout traitement sur un fichier, il faut le fermer par l'instruction **Fermer (VariableFichier)**.

9.2.4 Ecriture d'un fichier

Pour écrire dans un fichier, il suffit de dire **Ecrire (NomFichier, Information)**
S'il s'agit d'un fichier Texte, il faut utiliser l'instruction : **EcrireLigne (NomFichier, Information)**.
Information est une chaîne de caractères.

9.2.5 Lecture d'un fichier

Pour lire dans un fichier, l'instruction suivante est utilisée :

Lire (VariableFichier, Information).

Pour les fichiers textes nous utilisons l'instruction suivante :

LireLigne (VariableFichier, Information).

NomFichier est l'adresse suivie du nom du fichier associé à la variable fichier lors de l'assignation.

VariableLigne est une chaîne de caractères.

La fonction **FDF (VariableFichier)** renvoie la valeur vraie si nous sommes à la fin d'un fichier.

9.2.6 Déclaration d'une variable fichier

Un fichier de données peut contenir n'importe quel type de données.

VariableFichier : FICHIER DE Type-de-données

Un fichier TEXTE se définit ainsi : **VariableFichier : FICHIER DE TEXTE**

9.3 Création de fichier typé

Soit à créer un fichier pour la conception d'un répertoire téléphonique.

```
Algorithme RepertoireTelephonique
Type Personne = Structure
    Nom      : Chaîne [10]
    Prenom   : Chaîne [15]
    Telephone : Chaîne [12]
FinStructure
Variable
    FichRepertoire : FICHIER DE PERSONNE
    Repertoire     : Personne
    Choix          : Caractere
Début
    Associer (FichRepertoire, "C:\RepTel.dat")
    OuvrirEcriture (FichRepertoire)
    Repeter
        Afficher "Nom = " ;      Saisir (Repertoire.Nom)
        Afficher "Prénom = " ;   Saisir (Repertoire.Prenom)
        Afficher "Téléphone = " ; Saisir (Repertoire.Telephone)
        Ecrire (FichRepertoire, Repertoire)
        Afficher "Voulez-vous continuer ? (O/N) "
        Saisir (Choix)
    Jusqu'à (Choix = "N")
    Fermer (FichRepertoire)
FinAlgorithme
```

9.4 Création de fichier texte

```
Algorithme RepertoireTelephonique
Variable
    FichTexte : FICHIER DE TEXTE
    Ligne      : Chaîne
Début
    Associer (FichTexte, "C:\Texte.txt")
    OuvrirEcriture (FichTexte)
    Repeter
        Saisir (Ligne)
        Ecrire (FichTexte, Ligne)
    Jusqu'à (Ligne = "")
    Fermer (FichTexte)
FinAlgorithme
```

9.5 Lecture d'un fichier typé

Pour lire un fichier typé, il faut obligatoirement connaître et définir sa structure. La procédure lecture nécessite la fonction FDF (VariableFichier) pour savoir si nous avons atteint la fin du fichier.

Soit à lire le fichier du répertoire téléphonique.

```
Algorithme RepertoireTelephonique
Type PERSONNE = Structure
    Nom      : Chaîne [10]
    Prenom   : Chaîne [15]
    Telephone : Chaîne [12]
FinStructure
Variable
    FichRepertoire : FICHIER DE PERSONNE
    Repertoire      : Personne
Début
    Associer (FichRepertoire, "C:\RepTel.dat")
    OuvrirLecture (FichRepertoire)
    TantQue FDF (FichRepertoire) = Faux Faire
        Lire (FichRepertoire, Repertoire)
        Afficher (Repertoire.Nom)
        Afficher (Repertoire.Prenom)
        Afficher (Repertoire.Telephone)
    FinTantQue
    Fermer (FichRepertoire)
FinAlgorithme
```

9.6 Lecture d'un fichier texte

```
Algorithme RepertoireTelephonique
Variable
    FichTexte : FICHIER DE TEXTE
    Ligne      : Chaîne
```



```

Début
  Associer (FichTexte, "C:\Texte.txt")
  OuvrirLecture (FichTexte)
  TantQue FDF (FichTexte) = Faux Faire
    Lire (FichTexte, Ligne)
    AfficherLigne (Ligne)
  FinTantQue
  Fermer (FichTexte)
FinAlgorithme

```

9.7 Les types d'organisations

Il existe plusieurs types d'organisations de fichiers : l'organisation séquentielle indexée, l'organisation relative, l'organisation aléatoire, etc.

Seuls l'organisation séquentielle et l'organisation séquentielle indexée vont nous intéresser dans ce chapitre.

9.7.1 L'organisation séquentielle

9.7.1.1 Principe

Il consiste à ranger les enregistrements logiques les uns à la suite des autres dans l'ordre où ils se présentent.

9.7.1.2 Opération sur les fichiers séquentiels

Ecriture : Les enregistrements logiques sont écrits comme ils arrivent.

Consultation : Les enregistrements logiques sont lus les uns à la suite des autres comme ils ont été écrits. Pour accéder à la n^{ième} enregistrement, il faut nécessairement lire n-1 enregistrements.

Mise à jour : la mise à jour suppose trois cas

- **Ajout** : l'enregistrement logique est inséré entre les deux où il doit se trouver
- **Suppression** : l'enregistrement logique est effacé ou banalisé.
- **Modification** : l'enregistrement est modifié puis remis à sa place.

Ces fonctions de mise à jour dépendent du support.

Par exemple : pour mettre à jour un fichier sur bande magnétique, il est obligatoire de le lire sur une bande et de le recopier sur une autre.

9.7.2 L'organisation séquentielle indexée

9.7.2.1 Principe

C'est un type d'organisation qui permet l'accès séquentiel et l'accès direct à l'enregistrement logique du fichier. Chaque enregistrement logique est repéré par un champ clé dont la transformation en adresse est à la charge du système.

Au moment de la création, le système crée des tables d'index qui permettent d'accéder rapidement aux enregistrements logiques du fichier.

9.7.2.2 Création du fichier

Le fichier est créé en séquentiel dans une zone primaire. Les enregistrements logiques sont écrits par ordre croissant de la clé. Deux enregistrements logiques ne peuvent avoir une même clé.

Au fur et à mesure de sa création, le système crée des tables d'accès.

9.7.2.3 Consultation

L'accès séquentiel : il se fait de la même manière que les fichiers séquentiels.

L'accès direct : Pour accéder à un enregistrement, il faut fournir sa clé.

9.7.2.4 Mise à jour

Les opérations de mise à jour sont très simples.

Pour ajouter un enregistrement, on a la possibilité de nous positionner, grâce à la table d'index, à la fin du fichier puis créer un champ d'information à ce niveau.

Pour la modification, l'enregistrement logique est lu puis modifié en mémoire centrale, ensuite recopié à sa place initiale.

La suppression se fait par un pointeur d'invalidation. L'enregistrement logique est marqué et il n'est plus considéré dans les traitements ultérieurs, mais il occupe toujours sa place au niveau du support.

9.8 Opérations classiques dans un fichier

9.8.1 Opération de recherche dans un fichier séquentiel

Ecrire le sous-programme qui cherche la position de l'enregistrement logique PEnreg dans un fichier séquentiel Fich.

Soit Enreg l'enregistrement logique du fichier Fich.

```
Fonction RecherchePosit (donnée Fich, PEnreg) : Entier
Variable
    Trouver    : Booléen
    Pos        : Entier
Début
    OuvrirLecture (Fich)
    Trouver ← Faux
    Pos ← 0
    TantQue (FDF (Fich) = Faux) ET (Trouver = Faux) Faire
        Lire (Fich, Enreg)
        Pos ← Pos + 1
        Si Enreg = PEnreg Alors
            Trouver ← Vrai
```

```

        FinSi
    FinTantQue
    Fermer (Fich)
    Si Trouver = Faux Alors
        AfficherLigne "Enregistrement introuvable"
    FinSi
    RecherchePosit ← Pos
    Fermer (Fich)
FinFonction

```

9.8.2 Opération d'ajout dans un fichier séquentiel

Nous supposons que nous disposons d'une bande magnétique comme support de travail.

Ecrire le sous-programme qui ajoute tous les enregistrements de valeurs PEnreg du fichier Fich1 dans le fichier Fich2.

Soit Enreg l'enregistrement logique du fichier Fich.

```

Procédure EnregSous (donnée Fich1, Fich2, PEnreg; Résultat
NouveauFich)
Début
    OuvrirLecture (Fich1)
    OuvrirLecture (Fich2)
    OuvrirEcriture (NouveauFich)
    TantQue (Pas (FDF (Fich2)) Faire
        Lire (Fich2, Enreg)
        Ecrire (NouveauFich, Enreg)
    FinTantQue
    TantQue (Pas (FDF (Fich1)) Faire
        Lire (Fich, Enreg)
        Si Enreg = PEnreg Alors
            Ecrire (NouveauFich, PEnreg)
        FinSi
    FinTantQue
    Fermer (NouveauFich)
    Fermer (Fich1)
    Fermer (Fich2)
FinProcédure

```

9.8.3 Opération de suppression dans un fichier séquentiel

Ecrire le sous-programme qui supprime tous les enregistrements PEnreg du fichier Fich.

```

Procédure Supprime (donnée Fich, PEnreg ; Résultat NouveauFich)
Début
    OuvrirEcriture (NouveauFich)
    OuvrirLecture (Fich)
    TantQue FDF (Fich) = Faux Faire

```

```

        Lire (Fich, Enreg)
        Si Enreg <> PEnreg Alors
            Ecrire (NouveauFich, Enreg)
        FinSi
    FinTantQue
    Fermer (NouveauFich)
    Fermer (Fich)
FinProcédure

```

9.8.4 Opération de modification dans un fichier séquentiel

Soit un fichier séquentiel Fich. On se propose de supprimer toutes les occurrences de valeur PEnreg par BEnreg.

Ecrire le sous-programme qui réalise ce travail.

```

Procédure   Modif   (donnée   Fich,   PEnreg,   BEnreg ;   Résultat
NouveauFich)
Début
    OuvrirLecture (Fich)
    OuvrirEcriture (NouveauFich)
    TantQue FDF (Fich) = Faux Faire
        Lire (Fich, Enreg)
        Si Enreg <> PEnreg Alors
            Ecrire (NouveauFich, Enreg)
        Sinon
            Ecrire (NouveauFich, BEnreg)
        FinSi
    FinTantQue
FinProcédure

```

9.9 Mise à jour d'un fichier

La mise à jour d'un fichier consiste à faire les opérations suivantes :

- ajout
- suppression
- modification

Les plus souvent un fichier externe est utilisé pour effectuer plusieurs opérations dans un fichier. Ce fichier se nomme fichier mouvement. Il regroupe toutes les opérations d'ajout, de suppression et de modification du fichier source appelé permanent.

Grâce à des algorithmes classiques sur les fichiers mouvements et le fichier permanent, nous obtenons un nouveau fichier permanent.

L'exemple suivant montre un cas de mise à jour dans un fichier séquentiel.

Soit StatutPersonnel le fichier du personnel d'une entreprise donnée avec Enreg l'enregistrement logique de ce fichier.

```
Type TEnreg = Structure
    Matricule      : Chaîne [3]
    Nom            : Chaîne [10]
    Prénom        : Chaîne [15]
    Adresse        : Chaîne [35]
    SM             : Chaîne [1]
    DateNaissance  : Chaîne [10]
FinStructure
```

Matricule	Nom	Prénom	Adresse	SM	DateNaissance
-----------	-----	--------	---------	----	---------------

SM (Situation matrimoniale) peut prendre les valeurs suivantes :

- M pour les mariés
- C pour les célibataires
- D pour les divorcés

Soit FichMouvement le fichier mouvement et MEnreg son enregistrement ayant la structure suivante :

```
Type TMvt = Structure
    Matricule : Chaîne [3]
    SM        : Chaîne [1]
FinStructure
```

Matricule	SM
-----------	----

Les fichiers sont triés sur le numéro de matricule de l'agent.

Travail demandé : Ecrire l'algorithme qui met à jour la situation matrimoniale des agents de cette entreprise.

NB : Le nouveau fichier aura comme nom NouveauStatutPersonnel et son enregistrement logique NEnreg

```
Procédure Mise_A_Jour (donnée StatutPersonnel, FichMouvement ;
Résultat NouveauStatutPersonnel)
Variable
    FinPersonnel, FinMouvement : Booléen
Début
    OuvrirLecture (StatutPersonnel)
    OuvrirLecture (FichMouvement)
    OuvrirEcriture (NouveauStatutPersonnel)
    FinPersonnel ← Faux
    FinMouvement ← Faux
    Lire (StatutPersonnel, Enreg)
    Si FDF (StatutPersonnel) = Vrai Alors
        FinPersonnel ← Vrai
    FinSi
```

```

Lire (FichMouvement, MEnreg)
Si FDF (FichMouvement)=Vrai Alors
    FinMouvement ←Vrai
FinSi
TantQue (FinPersonnel=Faux) ET (FinMouvement= Faux) Faire
    Si Enreg.Matricule < MEnreg.Matricule Alors
        Ecrire (NouveauStatutPersonnel, Enreg)
        Lire (FichMouvement, MEnreg)
        Si FDF (FichMouvement = Vrai) Alors
            FinMouvement ←Vrai
        FinSi
    FinSi
    Si Enreg.Matricule = MEnreg.Matricule Alors
        NEnreg ←Enreg
        NEnreg.SM = MEnreg.SM
        Ecrire (NouveauStatutPersonnel, Enreg)
        Lire (StatutPersonnel, Enreg)
        Si FDF (StatutPersonnel) = Vrai Alors
            FinPersonnel ←Vrai
        FinSi
        Lire (FichMouvement, MEnreg)
        Si FDF (FichMouvement)=Vrai Alors
            FinMouvement ←Vrai
        FinSi
    FinSi
FinTantQue
TantQue (FDF (StatutPersonnel) = Faux) Faire
    Ecrire (NouveauStatutPersonnel, Enreg)
    Lire (StatutPersonnel, Enreg)
    Si FDF (StatutPersonnel) = Vrai Alors
        FinPersonnel ←Vrai
    FinSi
FinTantQue
Fermer (FichMouvement)
Fermer (NouveauStatutPersonnel)
Fermer (StatutPersonnel)
FinProcédure

```

Principe utilisé :

Le fichier principal ainsi que le fichier mouvement sont tous deux triés sur un même **argument**. Au niveau du précédent algorithme, il s'agit du **matricule** de l'agent.

Deux cas sont observés :

Si **Enreg.Matricule < MEnreg.Matricule** (cela signifie que l'agent dont le matricule est égal à Enreg.Matricule n'a subi aucune modification) alors nous sauvegardons directement l'enregistrement au niveau du nouveau fichier.

Si **Enreg.Matricule = MEnreg.Matricule** (cela signifie que l'agent dont le matricule est égal à Enreg.Matricule a effectué une modification au niveau de son statut) alors les informations portées par le fichier mouvement seront prises en compte.

Exercices

Exercice 1 : Soit **FEtudiant** un fichier d'une école de la place dont chaque enregistrement **Enreg** a la structure suivante :

```
Type Etudiant = Structure
    Numéro      : Chaîne [3]
    Nom           : Chaîne [15]
    Prénom        : Chaîne [20]
    Adresse       : Chaîne [35]
    Sexe          : Chaîne [1]
FinStructure
```

Soient **ArbreG** et **ArbreF** deux arbres binaires dont chaque nœud a la structure suivante :

```
Type Pointeur = ↑Nœud
Nœud = Structure
    Numéro      : Chaîne [3]
    Nom           : Chaîne [15]
    Prénom        : Chaîne [20]
    Adresse       : Chaîne [35]
    Sexe          : Chaîne [1]
    FilsGauche    : Pointeur
    FilsDroit     : Pointeur
FinStructure
```

Travail à faire : Eclater le fichier en deux arbres : **ArbreG** ne contiendra que les garçons et **ArbreF** les filles.

Exercice 2 : Soit **FichTexte** un fichier texte donné.

1) Ecrire l'algorithme qui affiche et mémorise tous les mots de **FichTexte** dans un arbre binaire **ArbreTexte**.

NB : Les espaces ainsi que les signes de ponctuation seront ignorés, les doublons ne seront pas mémorisés.

2) Ecrire l'algorithme qui mémorise tous les mots de **FichTexte** dans une liste linéaire **ListTexte**.

3) Donnez la structure de l'arbre et de la liste linéaire.

Projet final

Réalisation de la traduction de la boucle « Repeter-Jusqua » en « do-while » du langage C.

En algorithmique, la boucle Repeter-Jusqua se fait de la manière suivante :

```
Repeter  
    (Instructions)  
Jusqua (Condition)
```

(Condition) est formée par un ensemble d'opérateurs, d'opérandes et de comparateurs.

Par exemple : **Jusqua** (a + b = c) ; **Jusqua** (i > 100)

Exemple : l'algorithme suivant affiche 100 fois le message « bonjour »

```
Algorithme Bonjour  
Variable  
    i : entier  
Debut  
    i ← 0  
    Repeter  
        AfficherLigne "Bonjour"  
        i ← i+1  
    Jusqua (i=100)  
FinAlgorithme
```

Par ailleurs le compilateur du langage C n'a pas la même structure arborescente de la boucle Repeter-Jusqua de l'algorithme. En C la boucle Repeter-Jusqua se définit de la manière suivante :

```
do                → Repeter  
    (instruction) →    (instruction)  
while (condition) → TantQue (condition)
```

Traduit littéralement, elle devient ainsi :

```
Repeter  
    (instruction)  
Jusqua (condition)
```

L'écriture de l'algorithme précédent devient ainsi en langage C *littéral*.

Attention : on ne traduit pas tout l'algorithme en C mais c'est la boucle qui nous intéresse.


```

Algorithme Bonjour
Variable
    i    : entier
Debut
    i ← 0
    do
        AfficherLigne "bonjour"
        i ← i + 1
    while (i != 100)
FinAlgorithme

```

Remarque : La condition de **Repeter-Jusqua** est l'inverse de la condition de **do-while**. Ce principe permet de dresser le tableau suivant :

Opérateur de l'algorithme	Opérateur du C
= (égal)	!= (différent)
< (inférieur strictement)	>= (supérieur ou égal)
<= (inférieur ou égal)	> (supérieur strictement)
> (supérieur strictement)	<= (inférieur ou égal)
>= (supérieur ou égal)	< (inférieur strictement)
ET	(OU)
OU	&& (ET)
PAS (instruction)	(instruction)
<> (différence)	= (égal)

Travail à faire : Ecrire l'algorithme **Procedure JusquaWhile** (donnee ligneAlgo : chaîne ; resultat ligneC : chaîne) qui réalise cette traduction.

Correction

Analyse :

La procédure **JusquaWhile** reçoit comme paramètre d'entrée une chaîne de caractères contenant la condition de l'instruction **Repeter** et en sortie la condition de **while**.

Exemple :

ENTREE	SORTIE
Pas (a+b)	(a+b)
Pas (a=b)	(a=b)
(a+b > 0)	a + b <= 0
Racine (a) > 8	Racine (a) <= 8
i = 3,1415	i != 3,1415

L'algorithme que nous allons donner ne tient pas en compte de l'opérateur PAS (négation). Nous invitons au lecteur de le réaliser après analyse du premier.

Nous utiliserons la primitive **MotSuivant** (ligne, posit) qui renvoie le premier caractère différent de l'espace rencontré à partir de la position *posit*.

```

Procédure Jusquawhile (donnée ligneAlgo : chaîne ; resultat ligneC : chaîne)
Variable
    I : entier
    Mot : Chaîne
Debut
    I ← 1
    ligneC ← ""
    TantQue (i ≤ Longueur (ligneAlgo)) Faire
        Mot ← RechercheMot (LigneAlgo, i)
        Si (Mot = ">") ET (MotSuivant (ligneAlgo,i) = "=") Alors %cas du >=
            ligneC ← ligneC + "<"
        Sinon
            Si (Mot = "<") ET (MotSuivant (LigneAlgo,i) = "=") Alors
                ligneC ← ligneC + ">"
            Sinon
                Si MotSuivant (ligneAlgo, i) <> "=" Alors
                    ligneC ← ligneC + ">="
                Sinon
                    Si (Mot = "=") Alors
                        ligneC ← ligneC + "!="
                    Sinon
                        ligneC ← ligneC + Mot
                    FinSi
                FinSi
            FinSi
        FinSi
    FinTantQue
FinProcédure

```

Bibliographie

- Kéba DIOP.- *Algorithmique et structures de données*.- Tome 1 et Tome 2, Press de l'université de l'UNIS.
- Jean-Luc STEHLE, Pierre HOCHARD.- *Théorie des langages (Partie II)*.- Ellipses
- Moussa LÔ et Fatou KAMARA.- *Algorithmique & Programmation*.- UGB Saint-Louis
- Microsoft Encarta 2004