Programmation algorithmique de base

Cours d'initiation à
- l'algorithmique de base.
- la technique des fichiers.
- Les algorithmes types de traitements.

Qu'est-ce qu'un algorithme ?

Un algorithme sert à décrire les actions élémentaires nécessaires à la réalisation d'un traitement particulier. De par sa définition un algorithme **doit** être un langage universel. Tout programmeur dans n'importe quel langage doit très facilement pouvoir traduire un algorithme dans un langage de programmation quelconque. Malheureusement il est très rare de tomber sur deux algorithmes ayant la même structure (norme d'écriture). C'est pourquoi à travers cet ouvrage nous allons vous présenter l'une des normes les plus courantes mais aussi vous avertir des différentes façons d'écrire chaque structure.

TABLE DES MATIERES :

①Les différents éléments d'un algorithme :

➠ Format général d'un algorithme :
Cette structure est très variable mais dans l'ensemble on doit retrouver au minimum ceci :

Nom de l'algorithme général.
Fichiers.
Nous expliquerons cette rubrique plus tard. Cette rubrique est facultative.
Déclaration des constantes.
On appelle constante toute donnée à caractère non modifiable. Une constante peut être utilisée dans un algorithme mais ni un algorithme ni un programme ne peut en changer sa valeur. Une constante est définie une fois pour toutes. Une constante se définit de la façon suivante :
Nom_de_la_constante : Valeur.
Une fois définie, dès que le nom_de_la_constante sera rencontré dans l'algorithme elle sera automatiquement remplacée par sa valeur.
Variables.
Une variable algorithmique n'est rien de plus qu'une variable mathématique. Cette variable à un nom que l'on appelle *identificateur*. Ce nom fait référence à son contenu. Son contenu que l'on appelle *type* informe l'utilisateur et surtout la machine de la nature de la variable. En effet la machine doit savoir interpréter ce qui se trouve dans l'identificateur : Est-ce un nombre ? Est-ce un nombre réel 2,51, est-ce un nombre entier ? C'est dans cette rubrique que seront explicitées ces informations de la façon suivante :
type : *liste des identificateurs séparés par une virgule.*
Exemple :
entier : a,b,c ⇦ Déclare dans tout l'algorithme, 3 variables a b et c pouvant chacune recevoir un entier. Le contenu d'une variable peut être modifié par l'algorithme.
Remarque : Une fois déclarée une variable contient n'importe quoi, c'est à vous d'y placer la bonne valeur. On dit qu'il faut initialiser la variable (mettre une valeur dedans). **Déclarer une variable, c'est lui donner un nom et un type. Initialiser une variable, c'est attribuer une valeur à la variable déclarée plus haut.**
Actions.
C'est dans cette rubrique que l'on décrit les actions élémentaires relatives à un traitement.

➟ Les différents types de données :

① Le type entier : Un entier est une valeur numérique sur laquelle la machine peut effectuer des calculs. Un entier ne contient pas de virgule (ils sont de la famille des entiers naturels). Si on place la valeur 1,5 dans une variable entière celle-ci ne recevra que la valeur 1 (tout ce qui est après la virgule est sans intérêt).

② Le type flottant ou réel : Un réel est une valeur numérique sur laquelle la machine peut effectuer des calculs. Un réel comme son nom l'indique peut recevoir toute valeur appartenant à l'ensemble des réels. Le seul problème qui en découle est un défaut d'arrondi. En effet le résultat d'une opération telle que la division, le cosinus, etc... entraîne un résultat approché. Il faut savoir que la machine ne travaille **jamais** en valeur exacte mais en valeur approchée. Les problèmes de précisions sont en général à la charge du programmeur sauf s'ils font l'objet d'un algorithme particulier.

③ Le type caractère : Il définit comme son nom l'indique un caractère c'est à dire une lettre, un chiffre ou un caractère spécial. Les caractères sont toujours mentionnés entre guillemet simple ‘’. Exemple ‘a’ est un caractère ‘9’ est un caractère, ‘&’ est aussi un caractère. La machine **ne peut faire aucun calcul sur un caractère**, elle peut seulement faire des comparaisons ou des affectations (nous verrons ce que signifie ces deux termes dans le prochain chapitre).

④ Le type logique : Il définit une variable de type booléenne (relative à l'algèbre de Boole) qui ne peut contenir que le résultat d'une condition c'est à dire VRAI ou FAUX (0 ou 1).

On dit que ces 4 premiers types sont des types simples ou encore que se sont des scalaires (ils n'ont aucune dimension).

⑤ Le type tableau (ou table) : Un tableau est une suite d'éléments simples ou composés rangés les uns à la suite des autres. Chaque élément est distingué par son rang (c'est sa place dans le tableau). A un rang précis on trouve le contenu d'une variable de type défini.
On appelle origine l'indice du premier élément dans le tableau et on définit un tableau comme suit :
Tableau nom_du_tableau[origine...fin] : Tableau de n type(s) = {listes des éléments respectifs contenu dans le tableau de origine à fin}.

Exemple :
Tableau T[1...10] : Tableau de 10 entiers = {144;225;31;2;7;12;24;7;5;0}
Dans cet exemple T est un tableau qui contient 10 entiers. L'origine du tableau est 1 (c'est à dire que le premier entier est en position 1). Le rang ou la position dans un tableau est toujours un nombre entier.
Le tableau T que nous avons déclaré ci-dessus à la forme suivante :

Indices	1	2	3	4	5	6	7	8	9	10
T[indices]	144	225	31	2	7	12	24	7	5	0

On accède au contenu d'une case du tableau grâce à son indice. Et on note ce contenu nom_du_tableau[indice]. Dans l'exemple ci-dessus T[1] contient un entier dont la valeur est 144, T[3] contient la valeur 31. T[0] n'existe pas et T[11] non plus.
L'initialisation du tableau est facultative (On peut déclarer un tableau sans y mettre de valeur, cela signifie que ce dernier contient forcément n'importe quoi, par conséquent il doit être renseigné plus tard dans l'algorithme).
Renseigner un tableau c'est remplir un tableau avec des valeurs de son type. Selon le type des éléments d'un tableau la machine peut effectuer ou non des calculs sur ses éléments. Enfin un tableau peut avoir plusieurs dimensions. On peut imaginer un tableau à 2 dimensions (lignes + colonnes), des lors c'est grâce à un N° de ligne et un N° de colonne que l'on accède à un élément de type du tableau. On les déclare comme suit :
Tableau nom_du_tableau[origine_ligne...fin_ligne][origine_colonne...fin_colonne] : tableau de n type(s) = { {contenue de la 1ere ligne},{contenue de la 2ème ligne},...}.
Exemple :
Tableau Z[1...5][1...3] : tableau de 15 entiers = { {3;4;6},{155;12;2},{9;11;7}, {11;9;5},{7;11;8} }
Le tableau Z que nous avons déclaré ci-dessus est un tableau à 2 dimensions dont la forme est la suivante :

indices	**1**	**2**	**3**
1	3	4	6
2	155	12	2
3	9	11	7
4	11	9	5
5	7	11	8

Légende : Les nombres soulignés en gras sont les indices respectifs des lignes et des colonnes.

De même que dans l'exemple précédent on accède à un élément du tableau par son N° de ligne et son N° de colonne. Ainsi Z[3][2] contient 11, Z[5][3] contient 8. Il existe aussi une autre notation permettant d'accéder à un élément d'un tableau c'est la notation suivante : T(indice) ou Z(ligne ; colonne). Par exemple Z(3;2) contient 12, Z(5;3) contient 8. Dans l'exemple vu plus haut avec T : T(4) contient 2.
Il est rare mais tout à fait possible de définir des tableaux à 3, 4, 5 voir 6 dimensions et plus (au delà de 3 dimensions c'est extrêmement rare).

⑥ Les chaînes de caractères : Une chaîne de caractères est un tableau à une dimension (c'est à dire avec un indice) dont le type est caractère. Une chaîne de caractères est déclarée comme un tableau mais peut être traitée de deux façons différentes. On l'utilise soit pour ranger des caractères indépendants que l'on traite ultérieurement un par un, grâce à leurs indices, soit comme un ''tout logique'' c'est à dire comme un mot ou une phrase.

Exemple de chaîne de caractères :
Chaîne de caractères P[] = ''Bonjour ''
Remarquez que l'origine et la fin de la chaîne ne sont pas mentionnées (cela signifie que dans le traitement réalisé par l'algorithme on ne s'intéresse pas à chacune des lettres du mot mais au mot entier). Si l'on a besoin d'accéder à un caractère spécifique dans la chaîne on doit spécifier l'origine et la fin de la chaîne et signaler le nombre de caractère(s) qu'il y a dans la chaîne.
Exemple :
Chaîne de caractères P[1...7] : Chaîne de 7 caractères = ''Bonjour''
Non seulement grâce à cette définition on peut s'intéresser a chacun des caractères de la chaîne (c'est à dire P[1] qui contient 'B', P[2] qui contient 'o', P[3] qui contient 'n', etc...) mais en plus on peut traiter cette chaîne comme un ''tout logique'' c'est à dire comme le mot ''Bonjour'' que l'on peut comparer à un autre mot.

⑦ Les structures : Une structure est un mélange de type. Elles servent à décrire des objets de la vie courante. Lorsqu'un objet à plusieurs caractéristiques de natures différentes on le décrira à travers une structure. Par exemple un C.D. audio a un titre, il est réalisé par un compositeur, il contient un certain nombre de chansons et enfin il a une durée totale fixe en minute. Si on veut représenter un C.D. audio ayant les caractéristiques précédemment décrites dans une machine on utilisera une structure.
Comment déclarer une structure :

```
Structure nom_de_l'_objet_concerné
{
  type_1 : liste des variables,
  type_2 : liste des variables,...
  ...
  type_n : liste des variables.
} Fin de structure.
```

Exemple : Essayons de mettre en oeuvre notre problème de C.D. audio maintenant...
Structure CD
{
  Chaîne de caractères : titre[1...30] chaîne de 30 caractères,
  Chaîne de caractères : compositeur[1...60] chaîne de 60 caractères,
  Entier : nbr_chanson,
  Entier : durée.
} Fin de structure.

Remarque : En théorie la déclaration d'une structure n'est pas la déclaration d'une variable de la forme de cette structure (cela veut dire que le mot CD n'est pas une variable mais un type de donnée nouveau). Dans certains algorithmes, la déclaration d'une structure correspond en même temps à la déclaration d'une variable de même type que cette structure et de même nom. En bref si on ne vous impose aucune restriction vous pouvez déclarer une structure de nom N (ici CD) qui correspond en même temps à une et une seule nouvelle variable de nom N. Dans ce cas vous n'avez plus le droit de déclarer une autre variable ayant les caractéristiques de la structure CD. Si vous êtes rigoureux vous distinguerez la **définition** (et non la déclaration) de la structure d'une part (ici CD) et la **déclaration** (et non la définition) d'une variable pouvant supporter ce nouveau type de données (par exemple : CD a,b,c ⇦ Pour déclarer trois variables ayant les caractéristiques d'un CD mais appelées a,b et c).
DONC : Si je n'ai besoin que d'un seul CD je me sers du nom de la structure comme nom de la variable associé à cette structure. Si j'ai besoin de plus d'une structure CD, je dit que CD est un nouveau type de données et après avoir défini la structure, je déclarerais les variables pouvant supporter ce type de données.

Pour accéder à l'un des éléments de la structure on opère comme suit :
Nom_de_la_structure est le nom propre de la structure ou le nom d'une variable correspondant à la structure indiquée.
Nom_de_la_structure.nom_d'_une_variable_dans_la_structure.
ou
Nom_de_la_structure→nom_d'_une_variable_dans_la_structure.

Exemple d'utilisation : (Dans ces exemples CD est structure et variable à la fois)
◆ CD.nbr_chanson ou CD→nbr_chanson ⇦ Contient un entier correspondant au nombre de chansons contenu dans un C.D.
◆ CD.durée ou CD→durée ⇦ Contient un entier correspondant à la durée totale d'un C.D.
◆ CD.titre ou CD→titre ⇦ Contient une chaîne de caractères contenant le titre d'un C.D.
◆ CD.titre[1] ou CD→titre[1] ⇦ Contient le 1er caractère issu de la chaîne de caractères CD.titre ou CD→titre.

Définition et accès à un tableau de structure :
Tableau : mes_cd[1...10] : tableau de 10 CD.
Dans un tableau de structure on considère la structure comme un nouveau type.
On accède à l'un de ces éléments comme suit :
tableau[indice].variable_dans_la_structure
ou
tableau[indice]→variable_dans_la_structure.
Exemple :
◆ mes_cd[3]→titre ou mes_cd[3].titre ⇦ Contient la chaîne de caractères contenant le titre du C.D. N° 3 dans le tableau de C.D.
◆ mes_cd[4]→titre[2] ou mes_cd[4].titre[2] ⇦ Contient le 2ème caractère issu de la chaîne de caractères CD.titre du C.D. N° 4 dans le tableau de C.D.

On dit que ces trois derniers types (5, 6 et 7) sont des types composés.
Comme un tableau est constitué d'éléments simples ou composés on peut imaginer des tableaux de tableaux, des tableaux de structures,...
Comme une structure est composée de type simple ou composé on peut aussi imaginer de belles réalisations.

Conseils : Donnez des noms à vos variables qui signifient quelque chose (faites en sorte que le nom de vos variables soient ''parlant''). Par exemple n'appelez pas une variable zz2xy si ce nom n'a aucune signification relative à l'exercice. Appelez vos variables plutôt compteur, nom_client, date_j (pour date du jour), nqt (pour quantité), etc... Ne déclarez **jamais** deux variables avec un même nom. Gardez sur une feuille à part la signification exacte du nom de vos variables dans un algorithme (cela s'appelle un dictionnaire des variables). Ne commencez jamais un algorithme par la partie déclarative (sauf si vous êtes un expert) mais penchez vous d'abord sur la rubrique ''actions''. Vous déclarerez vos variables au fur et à mesure que vous en aurez besoin dans la partie ''actions''. Enfin si des éléments donnés dans l'exercice sont susceptibles de ne jamais changer (exemple : taux de TVA) inscrivez les dans la rubrique ''constantes'' ou paramètre.

Conclusion :

La déclaration d'une constante se fait toujours avec sa valeur et ne change jamais. La déclaration d'une variable quelque soit son type nécessite d'être initialisé (de recevoir une 1ère valeur) sinon cette variable contient n'importe quoi. Les caractères se notent entre guillemet simple (exemple 'a') et les chaînes de caractères entre guillemet double (exemple ''Bonjour''). Un tableau contient des éléments d'un et un seul type, on accède à l'un de ses élément grâce à son rang (ou indice). Un rang est toujours une valeur numérique de type entier. Un tableau peut avoir plus d'une dimension et dans ce cas nécessite plusieurs indices (exemple : ligne + colonne) pour atteindre l'un de ses éléments. On se sert d'une structure pour représenter un objet quelconque ayant des caractéristiques de nature différente. Les structures sont très utilisées pour récupérer des informations dans un fichier. Il est important de dissocier le nom de la structure qui fait référence à sa définition (ou description) et le nom des variables pouvant recevoir des données de la forme de cette structure. C'est pourquoi pour définir un tableau de structure *a* on considère la structure *a* comme un nouveau type de données.
La machine ne peut effectuer de calculs que sur des entiers ou des flottants, elle peut toujours comparer des entiers entre eux, des flottants entre eux, des caractères entre eux ou des chaînes de caractères entre elles. Un tableau peut être renseigné (initialisé) tout de suite après sa définition **pas une variable de type simple, une structure ou un tableau de structure**. Renseigner ou initialiser un tableau consiste à mettre dans chacune de ses cases (dans chacun de ses éléments) une valeur appropriée (de type requis). Les types qui n'ont pas de dimension sont des types simples (encore appelés scalaires), les autres sont appelés des types composés.
La déclaration des variables peut s'écrire d'une autre façon :
liste des variables : type.

➠ Qu'est-ce qu'une **CONDITION** :
Une condition est un élément de réponse de type VRAI / FAUX. On dit aussi que le résultat d'une condition est un booléen (en rapport avec l'algèbre de Boole) et qu'il peut être stocké dans une variable de type logique. En machine le VRAI est caractérisé par une valeur numérique entière égale à 1, le FAUX est caractérisé par une valeur numérique entière égale à 0. Certains langages font une distinction fondamentale entre les valeurs entière 0 et 1 et les valeurs logiques VRAI et FAUX (comme par exemple le PASCAL qui dispose d'un type de donnée booléen); d'autres ne font aucune distinction entre le type entier et le type logique (comme par exemple le C). Dans un algorithme nous considérerons qu'un entier n'est pas un logique (nous parlons de type bien sur) et qu'un logique n'est pas un entier. Donc une variable de type logique ne pourra prendre que deux états (deux valeurs) : VRAI ou FAUX **mais pas 0 ni 1**.

❶ Les opérateurs de comparaison : Ce sont les opérateurs qui permettent de comparer des éléments entre eux. Un opérateur de comparaison ne peut s'appliquer qu'entre deux éléments de types similaires ou compatibles. Cela signifie au minimum que a et b sont de types identiques.

a = b ⇦ Opérateur d'égalité. Il renvoie VRAI, si et seulement si, a est égal à b sinon il renvoie FAUX. **Il ne faut surtout pas le confondre avec l' opérateur d'affectation que nous verrons un peu plus loin !**

a > b ⇦ Opérateur de supériorité. Il renvoie VRAI, si et seulement si, a est supérieur à b, sinon il renvoie FAUX. Avec le type caractère il faut se reporter à la table de codification des caractères dans la machine (par exemple la table des caractères selon le code ASCII ou EBCDIC ou autre). Ainsi le caractère 'A' qui est le 65ème caractère de la table ASCII étendue de l'I.B.M. P.C. et compatible est supérieur au caractère '1' qui vaut 49 exprimé en décimal dans cette même table. Pour les chaînes de caractères il en est de même selon la table des caractères utilisée. Par exemple en prenant la table ASCII précédemment citée la chaîne de caractères ''ZZZ'' est supérieure à la chaîne de caractères ''AAA''.

a < b ⇦ Opérateur d'infériorité. Il renvoie VRAI, si et seulement si, a est inférieur à b, sinon il renvoie FAUX. Pour les caractères et les chaînes de caractères se reporter à la remarque relative aux tables de codification des caractères dans la machine ci-dessus.

a <> b⇦ Opérateur d'inégalité. Il renvoie VRAI, si et seulement si, a est différent de b, sinon il renvoie FAUX. Pour les caractères et les chaînes de caractères se reporter à la remarque relative aux tables de codification des caractères dans la machine ci-dessus.

a <= b⇦ Cet opérateur n'existe pas dans tous les langages de programmation. Il renvoie VRAI, si et seulement si, a est inférieur à b **OU** a est égal à b, sinon il renvoie FAUX. Pour les types caractères et chaînes de caractères se reporter aux remarques faites précédemment sur la table de codification des caractères dans la machine.

a >= b⇦ Cet opérateur n'existe pas dans tous les langages de programmation. Il renvoie VRAI, si et seulement si, a est supérieur à b **OU** a est égal à b, sinon il renvoie FAUX. Pour les types caractères et chaînes de caractères se reporter aux remarques faites précédemment sur la table de codification des caractères dans la machine.

Remarque : Vous ne pouvez comparer à l'aide de ces opérateurs que deux éléments de type identiques. La comparaison de caractères ou de chaînes de caractères nécessite la connaissance d'une table de codification des caractères dans une machine. En général ces tables répondent à un critère humain qui correspond à l'ordre d'apparition des caractères dans l'alphabet (par exemple : 'A' est avant 'B' donc 'A' est plus petit que 'B'). Quant à la position relative des chiffres par rapport aux lettres, l'ordre d'apparition des lettres minuscules par rapport aux lettres majuscules, sont fonctions de cette table. Dans les cas les plus fréquents le caractère espace (' ') est le premier caractère de l'alphabet avant le 'A' et le 'a'. La comparaison s'effectue avec le contenu d'une variable et non avec son nom.

❷ Les opérateurs logiques : Ce sont des opérateurs qui permettent de combiner des comparaisons. Ils s'intercalent entre les comparaisons pour formuler de nouveaux résultats (on souhaite par exemple des résultats dépendants de plusieurs variables). Soit R1 une comparaison quelconque (son résultat est donc VRAI ou FAUX), R2 une autre comparaison (son résultat est donc VRAI ou FAUX), op un opérateur logique. Le résultat de R1 op R2 (par exemple A=B **OU** C<D) répondra aux critères suivants selon op. Les opérateurs (op) sont les suivants :

◆ ET : Renvoie VRAI si et seulement si R1 et R2 sont VRAIS ensemble sinon il renvoie FAUX. Exemple 4<5 ET 6=6 se lit : 4<5 ⇨ VRAI, 6=6 ⇨ VRAI, VRAI ET VRAI ⇨ VRAI.

R1	R2	R1 **ET** R2
FAUX	FAUX	FAUX
FAUX	VRAI	FAUX
VRAI	FAUX	FAUX
VRAI	VRAI	VRAI

◆ OU : Renvoie VRAI si au moins une des comparaisons donne VRAI sinon il renvoie FAUX. Exemple : 4<5 OU 6=4 se lit : 4<5 ⇨ VRAI, 6=4 ⇨ FAUX, VRAI OU FAUX ⇨ VRAI.

R1	R2	R1 **OU** R2
FAUX	FAUX	FAUX
FAUX	VRAI	VRAI
VRAI	FAUX	VRAI
VRAI	VRAI	VRAI

◆ NON (ou PAS) : Renvoie le résultat contraire d'une comparaison. Donc PAS R1 = VRAI si R1 est FAUX et PAS R1 = FAUX si R1 est VRAI.

R1	**PAS** R1
FAUX	VRAI
VRAI	FAUX

Ces opérateurs peuvent être combinés à condition que pour le ET et le OU il y ait un résultat à droite et à gauche de l'opérateur du type VRAI / FAUX.
Le NON (ou le PAS) n'admet qu'un argument à sa droite.
Par exemple on peut écrire (a < b et c = d) ou e = f. Dans ce cas on effectue d'abord a < b ⇨R1, puis c = d ⇨R2, puis R1 et R2 ⇨R3, ensuite on évalue e = f ⇨ R4 et pour finir R3 ou R4 ⇨R5 résultat final.

Note : il existe un autre opérateur (le OUX) qui ne sera pas abordé dans ce chapitre.
Les tableaux ci-dessus s'appellent des tables de vérité.

Conseils : Faites attention à ce que vous écrivez, ne comparez pas des pommes de terre avec du fromage ! Rappelez-vous que vous ne pouvez comparer que des variables de types identiques (il est donc impossible de comparer un entier à un caractère **même si un langage le permet,** ou encore de comparer un tableau de flottants à un flottant). Si vous combinez plusieurs opérateurs logiques (ET, OU, NON), n'hésitez pas à utiliser des parenthèses pour pouvoir déterminer qui est à droite et qui est à gauche de l'opérateur concerné.

➟ Qu'est-ce qu'un **TRAITEMENT** :
Un traitement est une suite d'actions élémentaires écrites dans le but de réaliser une fonctionnalité précise référençable par un nom simple.
Exemple : On vous donne un prix hors taxe et un taux (comme 18.6) et vous devez rendre un prix toutes taxes comprises. L'un des traitements demandés consistera à calculer le montant de la TVA. Ce premier exercice peut se décomposer en 4 traitements qui sont les suivants :
◆ Traiter la saisie du prix hors taxe et du taux.
◆ Calculer le montant de la TVA.
◆ Ajouter le montant de la TVA au prix hors taxe.
◆ Donner le résultat du calcul précédent.
On dit qu'il faut repousser la difficulté (c'est ce que nous avons fait, nous venons d'éclater **un problème** compliqué en **plusieurs petits problèmes** simples).
Il existe de nombreux algorithmes types de traitement relatif aux fichiers. Ils seront explicités plus loin.

Comment faire un algorithme type ?
Pour réaliser un algorithme type convenable il faut :
⇨Comprendre l'énoncé du problème (si nécessaire faire un ou deux exemples très rapides pour comprendre ce qui se passe dans l'exercice). **Il est inutile de se lancer dans un algorithme si vous n'avez pas compris ce que vous devez faire. SI TEL EST LE CAS REPORTEZ VOUS ALORS AU CHAPITRE DES ALGORITHMES TYPES (car vous n'avez plus d'autre solution).**
⇨Repoussez le problème de la difficulté en imaginant ce que vous avez besoin de faire (vous pouvez avoir besoin par exemple de saisir une date, de contrôler une date, d'afficher un tableau, de traiter un client, d'analyser une ligne d'un tableau, etc...). Sachez que même si vous ne savez pas comment traiter un client il est **très important** de savoir qu'il y a des clients à traiter ! Voyez un peu comment sont faits les algorithmes types : On vous indique des traitements abstraits sans vous dire ce qu'il y a dedans (C'est vague ''traiter un client''). Même si cela n'est pas la solution complète d'un problème sachez que c'est mieux que rien.
⇨Recherchez maintenant tous les traitements qui se répètent, essayez de déterminer s'il est possible de savoir combien de fois ils se répètent.

Vous pouvez après avoir répondu à ces 3 questions vous lancer maintenant dans la partie ACTIONS d'un algorithme.

Conseils pratiques : A partir de maintenant pour écrire des algorithmes utilisez toujours un crayon à papier. C'est tellement pratique de pouvoir effacer plusieurs fois !

➟ La rubrique **ACTIONS** de l'algorithme :
Je crois que maintenant vous avez un ordre d'idée de ce que vous allez trouver dans cette rubrique. A savoir essentiellement des actions élémentaires. Commençons par apprendre quelques définitions puis voyons ensuite les structures algorithmiques de base...

Définitions :
➔ Affectation : C'est donner une valeur à une variable. On dit que l'on affecte une valeur à une variable. On dit aussi que l'on initialise une variable (initialiser c'est plutôt lui donner une première valeur). Le symbole de l'affectation est ←.
Ce qui se trouve à droite du signe ← est évalué puis transféré dans ce qui se trouve à gauche de ce même signe. Exemple :
a est de type entier, *b* est de type réel, *c* est de type logique, *T* et *Z* sont les tableaux pris pour exemple dans le chapitre précédent.
a← 3
b← 4.56
c← VRAI
On dit que *c* reçoit la valeur VRAI, *a* reçoit la valeur 3...
T[2] ← 5
Z[2][1] ← 44
Si *w* est une structure de type CD alors je peux écrire :
w→nbr_chanson← 21 ou w.nbr_chanson← 21
NOTE IMPORTANTE : la déclaration d'une structure est une chose, la création d'une variable (ici *w*) du type de cette structure rend cette structure exploitable à travers cette variable *w*. La déclaration d'une structure n'est pas la déclaration d'une variable. Si m est de type logique je peux écrire :
m← a < 5
Dans ce cas *m* reçoit VRAI.
m← a = 5
Dans ce cas *m* reçoit FAUX.
Si vous souhaitez remplir un tableau d'entiers ou de flottants (quelque soit ses dimensions), vous pouvez écrire :
RAZ T
On dit que l'on effectue une **R**emise **A Z**éro du tableau. Si le tableau n'est ni de type entier, ni de type flottant on dit qu'il faut le **M**ettre **A B**lanc. Par exemple si on reprend le tableau de CD : mes_cd déclaré plus haut on peut écrire :
MAB mes_cd
Et surtout pas RAZ mes_cd. L'opération de Mise A Blanc remplit le tableau avec des caractères espaces ' '.

A RETENIR : Il ne faut surtout pas confondre l'opérateur d'affectation ← avec l'opérateur d'égalité =.

➔ Incrémentation : C'est augmenter le contenu d'une variable numérique de type entière ou flottante de une unité entière. Autrement dit c'est ajouter 1 au contenu d'une variable.

Exemple :
Soit les variables *a* de type entier contenant 3 et *b* de type flottant contenant 4.56, j'incrémente *a* puis *b* :

a← a + 1
b← b + 1

Après exécution de ces deux lignes la variable *a* qui contenait 3 avant contient désormais 4 et la variable *b* qui contenait 4.56 contient désormais 5.56.

➔ Décrémentation : C'est diminuer le contenu d'une variable numérique de type entier ou flottant de une unité entière. Autrement dit c'est soustraire 1 au contenu d'une variable.

Exemple :
Soit les variables *a* de type entier contenant 3 et *b* de type flottant contenant 4.56, je décrémente *a* puis *b* :

a← a - 1
b← b - 1

Après exécution de ces deux lignes la variable *a* qui contenait 3 avant contient désormais 2 et la variable *b* qui contenait 4.56 contient désormais 3.56.

Conclusion : Affecter une valeur à une variable c'est remplacer son contenu actuel par une nouvelle valeur. Incrémenter c'est ajouter un au contenu d'une variable et décrémenter c'est soustraire un au contenu d'une variable. Ces deux dernières opérations affectent généralement le comportement des boucles (des traitements qui se répètent en machine).

Les structures de contrôle de base d'un algorithme :

❶ La structure alternative :

Si vous souhaitez effectuer un traitement si une condition est remplie (une condition est remplie si son résultat est VRAI), vous utiliserez la structure suivante :

```
Si (condition)
| alors traitement
Fin si
```

La condition est évaluée puis, si son résultat est VRAI, le traitement situé après le *alors* est exécuté une seule fois. Dans le cas où la condition serait FAUSSE le traitement situé après le *alors* n'est pas exécuté et le programme se poursuit après le *Fin si*.

Si vous souhaitez exécuter un traitement si une condition est remplie ou un autre traitement si cette condition n'est pas remplie (une condition n'est pas remplie si son résultat est FAUX), vous utiliserez la structure suivante :

```
Si (condition)
| alors  traitement_1
| sinon traitement_2
Fin si
```

La condition est évaluée puis, si son résultat est VRAI, *traitement_1* est exécuté une seule fois. Une fois *traitement*_1 terminé le programme se poursuit après le *Fin si*. Si la condition est FAUSSE, *traitement_1* n'est pas exécuté, *traitement_2* lui est exécuté puis une fois ce dernier terminé, le programme se poursuit après le *Fin si*.

Quand utiliser le SI alors Fin si ? Quand vous devez exécuter un traitement dépendant d'une condition particulière.

Quand utiliser le Si alors sinon Fin si ? Quand deux traitements sont dépendants d'une même condition.

Les erreurs à ne pas commettre : Si vous trouvez une ou plusieurs actions élémentaires identiques dans le traitement du *alors* **et** du *sinon*. Vous devez dans ce cas les éliminer des deux traitements et les placer avant ou après le *Si alors sinon Fin si*. Exemple :

```
Si (m>0)
| alors  x ← x+1
|        m← g+4
| sinon m← 31
|        x ← x+1
Fin si
```

Notez la redondance du *x ← x+1* dans le *alors* et dans le *sinon*. Vu que l'emplacement de l'affectation de x dans le *alors* et dans le *sinon* n'importe pas sur la prise de sa nouvelle valeur (qui est une incrémentation), Je peux écrire *x ← x+1* soit avant ou après le *Si alors sinon Fin si* et l'enlever du *alors* et du *sinon*.

Ce qui donnerait :

```
x← x+1
Si (m>0)
| alors  m← g+4
| sinon m← 31
Fin si
```

Conseils pratiques : Si vous décrivez un traitement complètement ou partiellement dans un *Si alors Fin si* ou dans un *Si alors sinon Fin si* pensez à aligner vos actions pour des raisons évidentes de lisibilité. Par exemple n'écrivez jamais comme ceci :

```
Si (m>0) alors  m← g+4
| Z←m
| sinon p← 31
| Z← p - m
Fin si
```

Mais plutôt comme cela :

```
Si (m>0)
| alors  m← g+4
|         Z← m
| sinon p← 31
|         Z←p - m
Fin si
```

Enfin si vous voyez des actions redondantes dans un *alors* et dans un *sinon* interrogez-vous pour savoir si vous pouvez oui ou non extraire ces actions du *Si alors sinon Fin si* et si oui, demandez vous si vous allez écrire ces actions avant le *Si alors sinon Fin si* ou après.

❷ Les structures itératives :

Ces structures permettent au programmeur de décrire des traitements répétitifs. Un traitement qui se répète s'appelle aussi une itération. Pour écrire l'une de ces structures il faut déterminer deux choses vitales. La première : Peut-on déterminer avec exactitude le nombre de fois que l'on va répéter un même traitement (OUI ou NON) ? La seconde : Comment progressent les répétitions de ce traitement (de n en n, de 1 en 1, variable selon le traitement) ?

Si on répond OUI a la première et qu'à la seconde on répond de n en n ou de 1 en 1 on choisira **POUR**. Sinon on choisira **TANT QUE**.

TRES IMPORTANT : Si la progression de la répétition est variable (si la répétition ne dépend pas d'un nombre de fois à faire un traitement), ou si la progression change à sa guise le nombre de fois à faire ce traitement alors on choisira un **TANT QUE**. Enfin si vous hésitez à choisir parmi les deux structures qui vous sont proposées alors choisissez toujours le **TANT QUE** car cette structure fonctionnera tout le temps. Après avoir écrit votre itération autour d'un **TANT QUE** vous pourrez peut-être la transformer en **POUR**.

◆ La structure TANT QUE :

```
Tant Que (condition) répéter
|  traitement
Fin Tant Que
```

Dès l'exécution de la ligne *Tant Que (condition) répéter*, la condition est évaluée. Si cette condition est remplie alors la partie *traitement* est exécutée complètement (Sauf cas particulier que nous étudierons à la fin). Le traitement terminé, la phrase *Fin Tant Que* est exécutée à son tour rendant le contrôle **IMMEDIATEMENT** à la phrase *Tant que (condition) répéter*. La condition est de nouveau évaluée et si cette dernière est toujours VRAI alors la partie *traitement* est de nouveau exécutée. Et ainsi de suite jusqu'à ce que la *condition* de la phrase *Tant Que* soit FAUSSE. Dans ce dernier cas la partie *traitement* n'est pas exécutée, le programme poursuit alors sa course immédiatement après la phrase *Fin Tant Que*.
Remarque très importante : Il va de soi que dans ce type d'itération le *traitement* doit modifier le résultat de la condition de telle sorte que la suite du programme puisse s'exécuter sinon vous ne sortirez jamais de ce type de boucle.
Cette itération nécessite impérativement d'être initialisée avant son exécution (assurez-vous qu'au moment où le programme va exécuter votre *Tant Que* la condition qui en dépend soit correctement initialisée) sinon vous risquez de ne jamais exécuter votre *traitement*. L'altération des variables constituant la condition du *Tant Que* doit apparaître dans la partie *traitement* proprement dite et **SURTOUT PAS** dans un sous traitement de ce dernier (C'est à dire que vous ne devez pas dans votre traitement modifier les variables de la condition dans un traitement appelé par le *traitement* du *Tant Que*).

◆ La structure POUR:
La structure *Pour* n'est rien de plus qu'une structure *Tant Que* ayant une caractéristique particulière. Cette structure est très utilisée pour traiter des tableaux car son but est de générer une série de nombre successif (par exemple : 1, 2, 3, 4,... ou 1.5, 1.6, 1.7, 1.8, 1.9, 2.0... ou encore 6, 4, 2, 0...). Cette structure travaille à l'aide d'une variable de type numérique entier ou flottant (C'est cette variable qu'elle va faire progresser comme indiqué ci-dessus). C'est la structure qui va se charger pour nous de faire progresser le contenu de cette variable et par conséquent **NOUS N'AVONS PAS LE DROIT D'EN CHANGER LE CONTENU**. Nous avons simplement le droit d'en apprécier le contenu. Nous verrons plus loin comment passer d'une structure *Pour* à une structure *Tant Que* et vice versa.

Présentation générale :

```
Pour variable de origine à fin au pas de n répéter
| traitement
Fin pour
```

Description :
variable est une variable numérique de type entier ou flottant.
origine est une valeur numérique entière ou flottante quelconque (dans les limites de la machine). C'est la première valeur que prendra la variable ci-dessus.
fin est une valeur numérique entière ou flottante quelconque (dans les limites de la machine). C'est la dernière valeur que prendra la variable ci-dessus.
au pas de n Ce morceau de la phrase est facultatif si l'incrément (la façon dont progresse la variable ci-dessus) est 1. C'est à dire que la variable prend toutes les valeurs de *origine* à *fin* en progressant de 1 en 1. Si vous souhaitez partir de 5 jusqu'à 1 en allant de -1 en -1 vous devez stipuler *au pas de -1* (on dit que l'incrément est de -1). Vous pouvez ainsi faire varier votre variable de *n* en *n*.
Si on ne stipule aucun pas (un pas est un incrément) alors la variable progressera de 1 en 1. Le pas est obligatoirement une valeur ou une variable de type entier ou flottant (on peut progresser de .67 en .67 si ça vous chante !).

Comment ça marche ?
Lorsque le programme exécute la phrase *pour z de 1 à 5 répéter* pour la première fois il initialise *z* à 1 (c'est à dire qu'il fait un : $z \leftarrow 1$). Puis aussitôt après il évalue *z <= fin*. Si cette condition est vraie alors il exécute le *traitement* une fois sinon le programme se poursuit à l'instruction après le *Fin Pour*. Le traitement terminé (dans le cas où *z <= fin* est vrai) la phrase *Fin pour* est exécutée provoquant automatiquement l'incrémentation de la variable $z \leftarrow z+pas$ (ici pas de pas stipulé donc $z \leftarrow z+1$). Ceci fait la condition *z <= fin* est de nouveau évaluée et si cette dernière est toujours VRAIE alors le traitement est exécuté de nouveau avec une nouvelle valeur dans *z*. Si la condition devient FAUSSE alors le *Fin Pour* n'exécute pas une nouvelle fois le traitement et laisse le programme se poursuivre après cette dernière phrase. Notez que dans cette structure **il n'est pas permis** d'affecter ou de modifier le contenu de la *variable* pendant son *traitement*. Si vous deviez réaliser une telle action utilisez alors la structure *Tant Que Fin Tant Que*.

Remarque : Il existe une relation évidente entre la structure *Pour Fin Pour* et *Tant Que Fin Tant Que*. Cette relation permet de passer d'une structure à l'autre chaque fois que c'est possible. En fait un *Pour* est un *Tant Que*, sa seule particularité réside dans le problème d'incrément déjà abordé plus haut. Pour comprendre pourquoi ces deux structures existent au lieu d'une seule reportez-vous à un cours de technologie relatif au langage d'assemblage.

◆ La relation entre TANT QUE et POUR :
Cette structure équivaut à la structure *Pour variable de origine à fin répéter Fin Pour*. Il faut modifier le contenu de la variable *pas* si vous ajoutez la rubrique *au pas de n*.

```
pas←1
variable←origine
Tant Que (variable <= fin) répéter
| Traitement
| variable←variable+pas
Fin Tant Que
```

Grâce à cette ‘’passerelle’’ vous pourrez jongler avec *Pour* et *Tant Que* sans difficulté.

◆ La structure REPETER JUSQU’A:
Cette structure est la plus ancienne de toute pourtant elle n’est pas complètement obsolète puisqu’on la retrouve dans de nombreux langages de programmation évolué. L’utilisation de cette structure nécessite une grande expérience dans l’art de l’algorithmique car son emploi est très hasardeux.

DEBUTANT ! EVITEZ CETTE STRUCTURE CHAQUE FOIS QUE VOUS LE POURREZ !

Son inconvénient majeur est d’exécuter un traitement dépendant d’une condition au moins une fois. Cela signifie qu’à chaque fois que vous faites appel à cette structure c’est que le traitement à exécuter **DOIT** être exécuté une première fois avant de se demander si on l’exécute une fois suivante. DONC la condition doit être issue du traitement et non de l’extérieur (imaginez donc ce qui se passerait avec ce type de structure si vous ne devez pas exécuter votre traitement du tout à cause d’une condition produite par un traitement plus haut !). A chaque fois que vous utiliserez cette structure vous devrez être sûr à 100 % que le traitement à exécuter dans cette boucle **DOIT ETRE EXECUTE UNE FOIS**.

NOTE TRES IMPORTANTE : Cette structure est source de nombreuses erreurs algorithmiques ! Elle n’est pas à bannir mais à employer avec une extrême précaution !

Présentation générale :

```
Répéter
 | Traitement
Jusqu'à (condition)
```

Lorsque la phrase *répéter* est exécutée le traitement est exécuté une première fois. Le traitement terminé la phrase *Jusqu'à* est exécutée à son tour provoquant l'évaluation de la *condition*. Si la *condition* n'est pas remplie (est FAUSSE) alors l'exécution du programme reprend à la phrase *Répéter* relançant le *traitement* une nouvelle fois sinon l'exécution du programme se poursuit après la phrase *jusqu'à*.

◆ La structure SELON :
Cette structure est extrêmement pratique pour sélectionner un (ou plusieurs) traitement parmi un ensemble de valeurs pris par une variable. Imaginez que, vous deviez exécuter un traitement en fonction d'une touche entrée au clavier par l'utilisateur, que cette touche soit stockée dans une variable de type caractère appelé c. Vous souhaitez savoir si la personne qui a saisi une touche au clavier a pressé la touche [A], [B], [?] ou une autre que faire ? Première solution : Ecrire une séquence de *Si alors sinon Fin Si* en cascade. C'est beau mais ça fait mal à la tête ! Deuxième solution : utiliser une structure sélective *Selon Fin Selon*.

Présentation générale :

```
Selon (variable)
| cas VAL_1   : traitement_1  Fin cas
| cas VAL_2   : traitement_2  Fin cas
| cas VAL_3   : traitement_3  Fin cas
| cas échéant : traitement_E  Fin cas
Fin Selon
```

variable : Est une variable de type quelconque.
VAL_n (*VAL_1*, *VAL_2*,...) : Sont des valeurs possibles de cette variable (donc de même type !).

Comment ça marche ?
C'est très simple ! Lorsque la phrase *Selon (variable)* est exécutée le programme scrute les différents *cas* possibles. Si l'un des *cas* correspond au contenu de la *variable* alors le traitement qui lui est associé est exécuté jusqu'à la rencontre de l'instruction *Fin cas*. Si aucun cas ne correspond alors le programme exécutera le traitement du *cas échéant* si il à été stipulé (le *cas échéant* est facultatif) sinon le programme poursuivra après l'instruction *Fin Selon*. Dans tous les cas une fois l'instruction *Fin cas* rencontrée ou si aucun *cas* ne correspond, le programme se poursuit après l'instruction *Fin Selon*. Il existe de très nombreuses astuces à mettre en oeuvre avec cette structure comme par exemple la gestion de traitement commun.

Mettons en oeuvre notre problème de caractère saisi au clavier présenté ci-dessus :

```
Selon (c)
| cas ‘A’         : traiter caractère A;                                        Fin cas
| cas ‘B’         : traiter caractère B;                                        Fin cas
| cas ‘?’         :  Afficher l’aide;                                           Fin cas
| cas échéant : Afficher ‘’Vous n’avez pas appuyé sur la bonne touche !’’;  Fin cas
Fin Selon
```

Imaginons que le traitement des touches A, B et ? soit commun :

```
Selon (c)
| cas ‘A’         : traitement_1
| cas ‘B’         : traitement_2
| cas ‘?’         : traiter les caractères;                                     Fin cas
| cas échéant : Afficher ‘’Vous n’avez pas appuyé sur la bonne touche !’’;  Fin cas
Fin Selon
```

Dans ce cas si la variable *c* contient ‘A’ alors *traitement_1* est exécuté puis *traitement_2* est exécuté puis *traiter les caractères* est exécuté à son tour.
Si la variable *c* contient ‘B’ alors *traitement_2* est exécuté puis *traiter les caractères* est exécuté. Si la variable c contient ‘?’ alors *traiter les caractères* est exécuté. Dans tous les autres cas c’est le *cas échéant* qui est exécuté.
En général au moment ou un cas est exécuté c’est l’instruction *Fin Cas* qui termine l’exécution du Selon. Il est donc possible de traverser plusieurs *cas* avec une seule valeur de la variable (il suffit de choisir un ordre d’apparition). Il faut savoir que la recherche des cas s’effectues une seule fois pour choisir l’exécution du traitement à réaliser. C’est le *Fin Cas* qui met un terme à la poursuite des traitements dans un *Selon*. Tant que cette instruction n’a pas été lue le programme peut traverser tous les *cas*. Si au dernier *cas* exécuté *Fin cas* n’a pas été trouvé (c’est une erreur algorithmique), l’exécution se poursuit après *Fin Selon*. Enfin si vous n’écrivez pas de *cas échéant* et que, la variable ne contient aucune des valeurs spécifiées dans chaque *cas* alors aucun traitement entre les phrases S*elon* et *Fin Selon* ne sera exécuté, le programme se poursuivra après l’instruction *Fin Selon*.

Un jeu d'instructions théoriquement parfait :

Il existe très peu d'instructions en algorithmique. On considère ces instructions comme parfaites, c'est à dire qu'elles remplissent tous les critères de qualité exigés (sauf algorithme spécifique pour refaire l'une de ces fonctions). On compte 3 instructions primordiales qui sont les suivantes :

Saisir(variable)
Afficher(quelque chose)
Faire un_traitement

L'instruction *Saisir* par exemple peut recevoir n'importe quel type de variable, on considérera que la variable a reçu une valeur tolérable SAUF si on vous demande explicitement de contrôler son contenu. On peut saisir directement des tableaux et même des structures comme par exemple Saisir(mes_CD[1]). L'instruction *Afficher* peut afficher n'importe quoi séparé par des virgules. Comme par exemple : Afficher(''La valeur de i est '',I,'' et de Z est '', Z). On peut afficher un tableau ou une structure sans aucune difficulté comme par exemple Afficher(mes_CD[2]).
L'instruction *Faire* permet d'exécuter un traitement. Plutôt que de marquer un nom de traitement tout seul on écrit **Faire traitement** et plus loin après la fin du programme principal on écrit un paragraphe décrivant ce traitement. Exemple :
...
Faire calcul de la TVA.
Fin programme.

Calcul de la TVA**.** ⇦ Ceci s'appelle un paragraphe.
Somme← somme * 18.6 / 100

Si on écrit plusieurs instructions sur une même ligne on sépare ces instructions par un point virgule.

Procédure et Fonction :

En algorithmique on ne les considère pas comme telle. On parle plutôt de paragraphe (ou de sous programme). Leur écriture emploie des syntaxes spécifiques que nous ne considérerons pas comme standard à l'algorithmique. Nous allons toutefois voire ensemble l'une des formes possibles d'écriture de fonctions et de procédures. Pour cela, il suffit d'avoir conscience des différents problèmes nécessaires au codage de ces éléments dans tous les langages de programmations évolués (quand c'est possible !).

Une notion de base : Local/Global :

On dit qu'un composant d'un algorithme (tel qu'une variable, un sous programme, etc...) est global si ce composant est accessible à tous les autres composants de l'algorithme. On dit qu'un composant d'un algorithme est local si ce composant n'est accessible que par un certain nombre d'autres composants de l'algorithme. Par exemple : Dans tout ce que nous avons vu jusqu'à présent les variables que vous déclarez dans la partie ''Variables :'' sont des variables globales, donc accessibles à tous.

Procédures :

Une procédure est un type de traitement localisé. On peut la considérer comme un sous programme. Ses caractéristiques sont les suivantes :

- ◆ Elle peut recevoir des paramètres de celui qui l'appelle.
- ◆ Elle peut localiser ou partager ses variables.
- ◆ **Une procédure ne retourne aucune valeur à celui qui l'a appelée**.

Déclaration d'une procédure :
Etant donné qu'une procédure est en quelque sorte un petit morceau d'algorithme, une procédure se déclare comme un algorithme. Notez l'absence de la partie ''Fichiers''.

Procédure : nom de la procédure.
Déclaration des constantes.
Constantes locales (définies uniquement dans cette procédure et nulle part ailleurs).
Variables locales.
Variables locales (définies uniquement dans cette procédure et nulle part ailleurs).
Variables reçues.
Liste des variables reçues dans l'ordre de transmission. Ces variables sont locales et doivent avoir un nom différent de toutes celles déjà définies. Les types doivent impérativement correspondre entre l'appelant et la procédure appelée.
Variables globales utilisées ou modifiées par la procédure.
Liste des variables globales de l'algorithme utilisées par la procédure. Ici les types et les noms de variables doivent correspondre avec ceux de l'algorithme général.
Actions.
Partie traitement.

Attention : Si votre procédure partage des variables avec l'algorithme général vous devez le stipuler impérativement.

Fonctions :

Une fonction est un type de traitement localisé. On peut la considérer comme un sous programme à la différence près qu'elle renvoie une valeur à celui qui l'appelle. Ses caractéristiques sont les suivantes :

- ◆ Elle peut recevoir des paramètres de celui qui l'appelle.
- ◆ Elle peut localiser ou partager ses variables.
- ◆ **Une fonction retourne une valeur à celui qui l'a appelée.**

Déclaration d'une fonction :
Etant donnée qu'une fonction est en quelque sorte un petit morceau d'algorithme, une fonction se déclare comme un algorithme. Notez l'absence de la partie ''Fichiers''.

Fonction : nom de la fonction.
Déclaration des constantes.
Constantes locales (définies uniquement dans cette fonction et nulle part ailleurs).
Variables locales.
Variables locales (définies uniquement dans cette fonction et nulle part ailleurs).
Variables reçues.
Liste des variables reçues dans l'ordre de transmission. Ces variables sont locales et doivent avoir un nom différent de toutes celles déjà définies. Les types doivent impérativement correspondre entre l'appelant et la fonction appelée.
Variables globales utilisées ou modifiées par la fonction.
Liste des variables globales de l'algorithme utilisées par la fonction. Ici les types et les noms de variables doivent correspondre avec ceux de l'algorithme général.
Variable retournée :
C'est le nom et le type d'une variable locale à cette fonction qui est retournée à l'appelant. Cette variable peut être une variable partagée déclarée dans la rubrique '' Variables globales utilisées ou modifiées par la fonction''.
Actions.
Partie traitement avec au moins une instruction Renvoyer(une_variable).

Attention : Si votre fonction partage des variables avec l'algorithme général vous devez le stipuler impérativement. L'instruction de renvoie d'une variable met fin à la fonction, son rôle est de transmettre **IMMEDIATEMENT** à celui qui a appelé la fonction le contenu de la variable *une_variable*. Cette variable doit être de même type et de même nom que celle décrite dans la rubrique ''Variable retournée''.

Mécanisme de transmission des paramètres :

Lorsqu'une procédure ou une fonction est appelée toutes les variables transmises à cette dernière sont recopiées dans les nouvelles variables locales de celle-ci. Seules les variables figurant dans la rubrique ‘’Variables globales utilisées ou modifiées par la (fonction/procédure)‘’ restent modifiables de part et d'autre. Si votre fonction ou procédure modifie une variable figurant dans cette rubrique alors cette variable est modifiée pour tout le reste de l'algorithme. Lorsque vous retournez une valeur par une fonction le contenu de la variable retournée est recopié dans la variable réceptrice de l'appelant.

Exemple d'appel de fonction et de procédure :

Les fonctions et procédures ne peuvent être exécutées qu'à partir d'une rubrique ‘’Actions’’. Ces dernières peuvent s'appeler mutuellement. Toutes les procédures et toutes les fonctions seront déclarées pour vous comme globales (c'est à dire qu'il n'y aura pas de fonction ou de procédure locale ou imbriquée).

Pour appeler une fonction ADD(entier a, entier b) qui reçoit deux entiers et renvoie la somme des deux, on procède comme suit dans la partie ‘’Actions’’ :
On considère que *m* est de type entier.
m← ADD(3;4)
Après l'exécution de cette ligne *m* contiendra 7.
Si on considère que *h* et *i* sont deux variables de type entier on peut écrire ceci :
h ← 3
i ← 4
m← ADD(h;i)

Si nous avions une fonction ret_touche() qui ne reçoit aucun paramètre et qui renvoie un caractère correspondant à une touche lu sur le clavier nous pourrions écrire ceci :
On considère que *c* est de type caractère.
c← ret_touche()

Pour appeler une procédure CURSEUR(entier x, entier y) qui reçoit deux entiers x et y et qui positionne le curseur graphique en coordonnées (x,y) sur l'écran alors nous pourrions écrire ceci :
CURSEUR(5;7)

Remarque : Si dans vos algorithmes vous avez des conflits entre les différents noms de vos variables vous devez prendre garde au recouvrement possible de celles-ci. Pour cela sachez qu'il est possible de déclarer plusieurs fois une variable x ou y dans des fonctions ou procédures **différentes** pourvu que ces variables soient **locales** (c'est à dire qu'elles figurent dans la rubrique ''Variables locales''). Il faut juste savoir qu'à l'appel d'une fonction ou d'une procédure toutes les variables ne figurant pas dans la rubrique ''Variables globales utilisées ou modifiées par la fonction/procédure'' sont sauvegardées pendant l'exécution de la fonction ou procédure puis, sont restituées à la fin de celle-ci (Le renvoi d'une valeur par une fonction se fait en dernier). La rubrique où vous devrez faire le plus attention est donc la rubrique qui rend une variable commune à un ou plusieurs traitements : ''Variables globales utilisées ou modifiées par la fonction/procédure''. Prenez garde au degré de localisation de vos données si l'une de vos fonctions ou procédures déclare une variable locale puis la partage à différents niveaux entre d'autres fonctions ou procédures que celle-ci appelle.
BREF quand vous utilisez des fonctions ou des procédures soyez très vigilants à ce que vous écrivez ! Restez simple ce sera beaucoup mieux pour tout le monde !

②La gestion des fichiers :

➠ Points à prendre en considération :

Un fichier est une sorte de tableau enregistré sur un support de données. Ce support admet certaines caractéristiques techniques à prendre en compte (est-ce un lecteur de bande, un lecteur de disquette ou un disque dur ?). De plus le système sur lequel vous allez mettre en oeuvre vos fichiers comportera ce que l'on appelle un SGF (Système de Gestion de Fichiers), c'est lui qui définira ce que vous pourrez faire avec vos fichiers selon le support employé.

❶ Définitions à connaître :

- ◆ L'organisation d'un fichier : C'est le mode d'implémentation du fichier sur son support. C'est à dire la façon dont sont rangés les enregistrements sur un support.

- ◆ L'accès à un fichier : C'est le type de procédure de recherche des enregistrements.

- ◆ Nature d'un fichier : On classe les fichiers par catégorie selon le type d'information qu'ils contiennent.
  - ➔ Permanent : Ce sont des fichiers de base, ils contiennent des informations stables et sont valables plusieurs mois ou plusieurs années.
  - ➔ Mouvement : Ils ne sont valables que pour une période déterminée et doivent être reconstruits une fois la période terminée.
  - ➔ Travail : Ils sont créés par la machine pour stocker des informations provisoires, ils sont détruits à la fin de l'exécution du programme. On les appelle aussi fichiers temporaires.
  - ➔ Table : Ce sont des petits fichiers, ils contiennent quelques enregistrements, ils sont chargés en mémoire centrale dans un tableau au début de l'exécution d'un programme.
  - ➔ Liaison : Ils contiennent des informations qu'un programme communique à un autre programme (par exemple : édition différée).

- ◆ Un support adressable : C'est un support qui permet d'atteindre une information grâce à son adresse sur le support.

- ◆ Un support réutilisable : C'est un support que l'on peut utiliser plusieurs fois.

- ◆ Un support utilisable en lecture/écriture : C'est un support qui autorise la lecture et l'écriture.

❷ Composition et définition d'un fichier :

➟ Composition d'un fichier :

Un fichier est composé d'enregistrements. Chaque enregistrement est caractérisé par une structure. Imaginons un fichier élèves. Chaque fiche (ou enregistrement) de ce fichier contient : un numéro d'élève, un nom, un prénom, une date de naissance, la classe fréquentée cette année par l'élève et une langue vivante. Tous les enregistrements de ce fichier suivent la même structure (cela signifie que l'on peut charger chaque élève dans une seule structure).
Soit la structure élève suivante :

Structure élève
{
  Chaîne de caractères : numéro[1...5] chaîne de 5 caractères,
  Chaîne de caractères : nom[1...20] chaîne de 20 caractères,
  Chaîne de caractères : prénom[1...20] chaîne de 20 caractères,
  Chaîne de caractères : date_naiss[1...6] chaîne de 6 caractères,
  Chaîne de caractères : classe[1...5] chaîne de 5 caractères,
  Entier : code_LV.
} Fin de structure.

◆ Un article : C'est un enregistrement (ou une fiche) concernant un élève.

◆ Une rubrique : C'est un élément (ou morceau) d'articles. Par exemple *élève.prenom* est une rubrique de la structure élève.

◆ Un indicatif : C'est ce qui permet d'identifier un article précis. Dans ce cas l'indicatif est forcément le numéro d'élève : *élève.numéro*.

◆ Un enregistrement logique : C'est la représentation d'un article sur son support.

➟ Caractérisation d'un fichier :

On définit parfaitement un fichier en énonçant les cinq propriétés suivantes :

◆ Son nom,
◆ Sa nature,
◆ Son Organisation,
◆ Son ou Ses articles avec leur critère de tris éventuels,
◆ Sa ou ses clefs si il y en a.

❸ Les différentes opérations possibles sur les fichiers :

On distingue deux sortes d'opérations liées aux fichiers. Celles qui ont trait au fichier de façon générale (c'est à dire qui peuvent porter sur tout ses enregistrements à la fois). Et celles qui ont trait aux enregistrements du fichier de façon individuelle.

➟ Opérations générales sur les fichiers :

◆ La création : Elle consiste comme son nom l'indique à créer un fichier. Pour cela le SGF crée les informations du label du fichier sur un support. Cette opération peut être effectuée par :

- ➔ Un éditeur de fichier.
- ➔ Une base de données (Gestionnaire de fichiers).
- ➔ Un programme.
- ➔ Une saisie (sur un encodeur).

◆ L'ouverture : Elle consiste à vérifier les informations du label et à établir un lien logique entre un programme par exemple et le SGF.

◆ La fermeture : Elle consiste à libérer un lien logique établi sur un fichier entre un programme par exemple et le SGF.

◆ La destruction : Elle consiste à éliminer la présence d'un fichier sur un support. Elle peut s'effectuer par une commande ou par un utilitaire. Elle peut être logique ou physique.

◆ La fusion : Elle consiste à regrouper deux fichiers en un seul.

◆ L'éclatement : Il consiste à scinder un fichier en deux fichiers.

◆ Le tri : Il consiste à mettre les enregistrements dans un ordre précis selon un ou plusieurs **critères de tris**. Cette opération peut être effectuée par un programme ou par un utilitaire.

➠ Opérations générales sur les enregistrements :

◆ La création : Elle consiste à insérer un nouvel article dans un fichier. Plus précisément il s'agit d'allouer (ou de réserver) la place nécessaire sur le support pour un article supplémentaire.

◆ La consultation : Elle consiste à rechercher un article dans un fichier et si sa présence est confirmée d'extraire son contenu vers une zone mémoire appropriée.

◆ L'écriture : Elle consiste à introduire les informations concernant un nouvel article dans un fichier. Cette opération est généralement liée à la création.

◆ La réécriture : Elle consiste à revaloriser les informations d'un article concernant un fichier.

◆ L'ajout d'un enregistrement : Elle se conçoit sous deux formes distinctes qui sont :
- ➔ L'adjonction: On insère un article à la fin du fichier.
- ➔ L'insertion : On insère un article entre plusieurs articles déjà existants.

◆ La suppression : Elle se conçoit sous deux formes distinctes qui sont :
- ➔ La suppression logique : L'article n'est pas supprimé du support, un drapeau spécial indique simplement au SGF que l'enregistrement n'existe plus.
- ➔ La suppression physique : L'article est supprimé physiquement du support.

◆ La mise à jour : C'est une réécriture qui porte sur un ou plusieurs articles dans un fichier.

❹ Les différents types d'organisations et d'accès :

L'organisation définit les différents types d'accès possibles sur un fichier.

➠ Les organisations :

◆ L'organisation Séquentielle (SQ) :
Les enregistrements sont rangés les uns derrière les autres. Cette organisation est possible sur tous les supports de données.

◆ L'organisation Séquentielle Indexée (SI) :
Elle a les mêmes caractéristiques que l'organisation Séquentielle. De plus chaque enregistrement est identifié par une clef unique. Cela signifie que les enregistrements sont rangés dans l'ordre des clefs. Cette organisation n'est possible que sur des supports de données adressables. On peut y définir des clefs secondaires. **La ou les clefs font parties de l'enregistrement**.

◆ L'organisation Directe (D) :
Chaque enregistrement est identifié par une clef unique. Cette organisation n'est possible que sur des supports de données adressables. Cette organisation est préférable à l'organisation (SI) si vous avez peu d'enregistrement à traiter et ce toujours de façon directe (c'est à dire toujours en accès DIRECT).

◆ L'organisation Relative (R):
Chaque enregistrement est identifié par son rang comme dans un tableau à une dimension. Deux enregistrements ne peuvent occuper la même place. Cette organisation n'est possible que sur des supports de données adressables. Cette organisation n'existe pas partout. **La clef ne fait pas partie de l'enregistrement**.

➠ Les modes d'accès :

◆ L'accès Séquentiel (SQ) :
La lecture des enregistrements se fait séquentiellement. Si on souhaite lire le 15ème enregistrement d'un fichier il faut d'abord lire les 14 premiers. Cet accès est possible avec les organisations Séquentielles, Séquentielles indexées et Relatives.

◆ L'accès Direct (D) :
Pour lire un enregistrement on renseigne une clef puis on lit directement l'enregistrement. Cet accès est possible avec les organisations Séquentielles Indexées, Directes et Relatives.

◆ L'accès Dynamique (DY) :
Il permet le positionnement d'un pointeur de fichier sur un enregistrement particulier afin de lire séquentiellement une suite d'enregistrements. Cet accès est possible seulement avec l'organisation Séquentielle Indexée et Relative. De plus il n'est pas permis par tous les SGF.

❺ Les différents types de supports :

① Les bandes : Les lecteurs de bandes permettent de lire des bandes magnétiques. Surtout utilisés pour **archiver** des données, ce sont des supports réutilisables (c'est à dire qu'il est possible de lire et d'écrire plusieurs fois sur une bande). Les bandes sont des supports **non adressables**.

② Les disquettes magnétiques : De plus faibles capacités que les bandes, ces disquettes sont utilisables en lecture et en écriture. Ce sont des supports **adressables** et réutilisables.

③ Les CD-ROM : De très grosses capacités, un CD n'est inscriptible qu'une seule fois à partir d'un graveur de CD-ROM. Les CD-ROM sont des supports **adressables**.

④ Les disques magnéto optiques : Plus rares que les lecteurs de CD-ROM, les lecteurs magnéto optiques permettent de lire des disquettes lasers (un petit CD miniature est enfermé dans le boîtier d'une disquette 3 pouces ½). De grande capacité ces supports sont **adressables**, parfois réutilisables. Ils sont généralement inscriptibles une seule fois, puis accessibles en lecture seul (comme les WORM par exemple), mais peuvent aussi être accessibles en lecture/écriture (ces derniers étant un peu plus coûteux).

⑤ Les disques durs : Ce sont des supports de grandes capacités accessibles en lecture et en écriture. Ils sont **adressables** et réutilisables. Très répandus, ils permettent la plupart des traitements relatifs aux fichiers.

❻ Listes des instructions liées aux fichiers :

Pour accéder à un fichier il existe un certain nombre d'instructions qu'il est souhaitable de connaître. Tout d'abord souvenez vous de l'entête d'un algorithme...

Nom de l'algorithme général.
Fichiers.
Nous expliquerons cette rubrique plus tard. Cette rubrique est facultative.
Déclaration des constantes.
On appelle constante...

◆ Qu'est-ce que la rubrique *Fichiers* ?
Cette rubrique sert à déclarer toutes les informations nécessaires à l'exploitation de fichiers dans un algorithme. On y trouve pour chaque fichier les informations suivantes :

Fichier : Nom_du_fichier,
C'est le nom logique ou plus rarement physique de votre fichier.
Organisation : (SQ)/(SI)/(D)/(R),
C'est l'organisation de votre fichier.
Accès : (SQ)/(D)/(DY),
C'est le type d'accès que vous effectuerez dans votre algorithme. Plusieurs accès peuvent être stipulés.
Nature : Sa_Nature (cette information est facultative),
C'est la nature de votre fichier (par exemple PERMANENT).
Enregistrement : Nom_de_la_structure_d_un_enregistrement,
C'est le nom d'une ou des structure(s) reflétant la structure des enregistrements de votre fichier.
Clef : Nom_des_variables_clefs (la présence de cette information dépend de l'organisation du fichier).
C'est le nom des variables clefs de votre fichier s'il y en a. Vous devez stipuler la clef primaire et les clefs secondaires explicitement.

◆ Exemple :

Fichier : ELEVE
Organisation : Séquentielle Indexée
Accès : Séquentiel
Enregistrement : élève
Clef : élève.numéro.

Fichier : CAMIONS
Organisation : Séquentielle Indexée
Accès : Direct
Enregistrement : véhicule
Clef : Primaire : véhicule.NOcamion / Secondaire : véhicule.NOimmatriculation.

➠ Le jeu d'instructions :

◆ Instructions d'ouverture d'un fichier :

OUVRIR nom_Fichier EN mode_d_ouverture.
Cette opération est préalable à tout traitement avec un fichier.
Les modes d'ouverture sont les suivants :

Lecture	:	Vous ne pourrez dans ce cas que lire des enregistrements.
Ecriture	:	Le contenu du fichier ouvert dans ce mode est écrasé (vidé). Seules les opérations d'écriture seront possibles.
Lecture/Ecriture	:	Ce mode permet d'ouvrir un fichier en lecture sans en altérer son contenu à la source mais aussi de s'octroyer les droits en écriture. Dans ce mode les fonctions de lecture et d'écriture sont permises ensembles.
Adjonction	:	Ce mode permet d'ouvrir un fichier en vue d'y ajouter des enregistrements en partant de la fin. Ce mode n'est possible que sur des fichiers organisés en Séquentiels. Tous les langages de programmation ne permettent pas ce mode.

Exemple d'ouverture d'un fichier :
OUVRIR ELEVE EN Lecture.

◆ Instructions de lecture d'un enregistrement dans un fichier :

LIRE nom_du_fichier SUR CLEF nom_de_clef DANS nom_d_enregistrement.
Cette opération permet de lire directement un enregistrement du fichier qui porte le nom *nom_du_fichier* et de placer son contenu dans une zone de la mémoire centrale appelé *nom_d_enregistrement*. La clause *SUR CLEF nom_de_clef* est facultative. Elle doit apparaître explicitement uniquement si il peut y avoir une ambiguïté sur le nom de la clef.

LIRE nom_du_fichier DANS nom_d_enregistrement (Si **FF** alors Action Fsi).
Cette opération permet de lire séquentiellement un enregistrement du fichier qui porte le nom *nom_du_fichier* et de placer son contenu dans une zone de la mémoire centrale appelé *nom_d_enregistrement*.
La clause (*Si **FF** alors Action Fsi*) permet d'exécuter une action lorsque la fin du fichier est atteinte.

LIRE nom_fi DANS n_e ENREGISTREMENT SUIVANT (Si **FF** alors Action Fsi).
Cette opération est utilisée dans le cas d'un fichier ouvert en accès dynamique. Un enregistrement du fichier *nom_fi* est lu séquentiellement puis son contenu est placé dans une zone de la mémoire centrale appelé *n_e*.
La clause (*Si **FF** alors Action Fsi*) permet d'exécuter une action lorsque la fin du fichier est atteinte.

NOTE : Lorsqu'un fichier est lu, écrit, effacé ou réécrit en accès direct il est **IMPERATIF** d'avoir valorisé sa clef avant d'y accéder. On ne peut accéder à un et un seul enregistrement si on ne sait pas de quel enregistrement on souhaite parler précisément. La clause (*Si **FF** alors Action Fsi*) peut être omise lors d'une lecture séquentielle à condition de contrôler la fin du fichier avec une fonction spéciale qui renvoie une valeur logique VRAI si le dernier enregistrement à été lu. Cette fonction est : Logique FF(nom_du_fichier). On peut ainsi écrire ceci :

```
Ouvrir F1 en Lecture
Lire F1
Tant Que ( Pas FF(F1) ) répéter
| Traitement
| Lire F1
Fin Tant Que
Fermer F1
```

Exemple de lecture d'un enregistrement dans un fichier :
LIRE ELEVE DANS élève (Si **FF** alors EOF ← 1 Fsi).

◆ Instructions d'écriture d'un enregistrement dans un fichier :

ECRIRE nom_d_enregistrement DANS nom_du_fichier SUR CLEF nom_clef.
Cette opération permet d'écrire directement un enregistrement contenu dans une zone de la mémoire centrale appelé *nom_d_enregistrement* dans le fichier qui porte le nom *nom_du_fichier*. La clause *SUR CLEF nom_clef* est facultative. Elle doit apparaître explicitement uniquement si il peut y avoir une ambiguïté sur le nom de la clef.

ECRIRE nom_d_enregistrement DANS nom_du_fichier.
Cette opération permet d'écrire séquentiellement un enregistrement contenu dans une zone de la mémoire centrale appelé *nom_d_enregistrement* dans le fichier qui porte le nom *nom_du_fichier*. Cela signifie que les enregistrements seront stockés les uns derrière les autres.

NOTE : Lorsque le fichier est ouvert en mode Adjonction l'enregistrement écrit vient se placer après le dernier enregistrement du fichier.

Exemple d'écriture d'un enregistrement dans un fichier :
Valoriser la structure élève avec les données concernants un élève
élève.numéro←''12345''
ECRIRE élève DANS ELEVE SUR CLEF élève.numéro.

◆ Instructions de réécriture d'un enregistrement dans un fichier :

REECRIRE nom_d_enregistrement DANS nom_du_fichier SUR CLEF nom_clef.
Cette opération permet de remplacer directement un enregistrement situé dans un fichier qui porte le nom *nom_du_fichier* par un enregistrement figurant dans une zone de la mémoire centrale appelée *nom_d_enregistrement*. La clause *SUR CLEF nom_clef* est facultative. Elle doit apparaître explicitement uniquement si il peut y avoir une ambiguïté sur le nom de la clef.

REECRIRE nom_d_enregistrement DANS nom_du_fichier.
Cette opération permet de remplacer séquentiellement un enregistrement situé dans un fichier qui porte le nom *nom_du_fichier* par un enregistrement figurant dans une zone de la mémoire centrale appelée *nom_d_enregistrement*. Cette opération porte sur le dernier enregistrement lu.

NOTE : La réécriture d'un enregistrement ne peut s'effectuer que sur un enregistrement déjà existant.

Exemple de réécriture d'un enregistrement dans un fichier :
Revaloriser élève
élève.numéro ←''12345''
REECRIRE élève DANS ELEVE SUR CLEF élève.numéro.

◆ Instruction d'effacement d'un enregistrement :

EFFACER nom_du_fichier SUR CLEF nom_de_clef.
Cette opération permet d'effacer directement un enregistrement dans le fichier qui porte le nom *nom_du_fichier*. La clause *SUR CLEF nom_clef* est facultative. Elle doit apparaître explicitement uniquement si il peut y avoir une ambiguïté sur le nom de la clef.

EFFACER nom_du_fichier.
Cette opération permet d'effacer séquentiellement un enregistrement dans le fichier qui porte le nom *nom_du_fichier*. Cette opération porte sur le dernier enregistrement lu.

NOTE : L'effacement d'un enregistrement ne peut s'effectuer que sur un enregistrement déjà existant.

Exemple d'effacement d'un enregistrement dans un fichier :
élève.numéro ←''12345''
EFFACER ELEVE SUR CLEF élève.numéro.

◆ Instruction de positionnement :

POSITIONNER nom_du_fichier SUR CLEF [opl] nom_de_clef.
Cette opération n'est possible qu'en accès dynamique. Son but est de positionner le pointeur de fichier *nom_du_fichier* sur un enregistrement répondant à une condition *[opl]* de la clef primaire ou secondaire *nom_de_clef*.

Exemple :
élève.numéro ← ''00010''
Positionner ELEVE SUR CLEF >= élève.numéro.

Après ce positionnement les lectures séquentielles de ELEVE rapporteront en mémoire centrale tous les élèves dont le numéro est supérieur ou égal à ''00010'' dans la table des caractères interne de la machine.

◆ La clause « clef invalide » en accès direct ou dynamique :

Il est toujours possible d'ajouter la close suivante :
(Si **CLEF INVALIDE** alors Action Fsi)
dans n'importe laquelle des phrases nécessitant une clef pour accéder à un enregistrement. Ainsi dans le cas d'une lecture, d'un effacement ou d'une réécriture une action peut être exécutée si l'enregistrement requis est inexistant. Dans le cas d'un positionnement cette clause est exécutée si le positionnement à échoué.

Exemple :
ERR ← 0 ; élève.numéro ← ''00010''
LIRE ELEVE DANS élève SUR CLEF élève.numéro (Si **CLEF INVALIDE** alors ERR ← 1 Fsi).

◆ Instruction de fermeture d'un fichier :

FERMER nom_du_fichier.
Cette opération clôture un fichier en indiquant au SGF que le fichier qui porte le nom *nom_du_fichier* ne sera plus utilisé.

NOTE : On peut stipuler plusieurs fichiers au lieu d'un seul dans une même phrase de fermeture. Un fichier fermé ne peut plus faire l'objet d'aucune opération sauf si il est ouvert de nouveau. Si vous avez besoin d'accéder à un fichier dans deux modes différents (Exemple : Ouverture en lecture puis Ouverture en Adjonction), vous devez fermer le fichier après l'avoir utilisé dans un mode puis l'ouvrir de nouveau dans un autre mode.

Exemple de fermeture d'un fichier :
FERMER ELEVE.

❼ Formation des clefs & récapitulatif :

Un fichier organisé en Séquentiel Indexé comporte obligatoirement une clef dite **primaire** et éventuellement une ou plusieurs clefs dites **secondaires**. Les valeurs contenues dans une clef primaire sont uniques car elles permettent d'identifier un et un seul enregistrement. **Une clef primaire ne contient pas de doublon (ou homonyme)**. Une clef secondaire peut contenir des doublons. Toutes clefs (primaires ou secondaires) peuvent être décomposées en une ou plusieurs parties de votre choix. Vous vous devez alors de respecter la taille globale de la clef. Un positionnement permet donc d'extraire des enregistrements d'un fichier selon un morceau de clef. Un accès Direct est possible à partir d'une valeur d'une clef secondaire à condition que cette dernière n'admette pas de doublon.

Un fichier organisé en Relatif ne dispose que d'une clef primaire (donc sans doublon) correspondant au rang d'un enregistrement dans le fichier. Cette clef ne fait pas partie de l'enregistrement du fichier. Elle ne peut pas être décomposée.

Un fichier traité en séquentiel doit **toujours** faire apparaître une marque de fin soit à l'aide d'une fonction spéciale (*Logique FF(nom_de_fichier)*), soit à l'aide de la close (*Si* ***FF*** *alors Action Fsi*).

③Algorithmes types de traitements :

Voici une liste d'algorithmes types conçus pour réaliser un certain nombre de tâches :

◆ Edition d'un document à partir de la totalité d'un fichier :

```
Ouvrir Fichiers
Initialiser les variables
EOF ← 0
Faire Entête
Lire Fichier-Directeur (Si FF alors EOF ← 1 Fsi)
Tant Que (EOF=0) répéter
|  Faire Traitement d'un enregistrement
|  Lire Fichier-Directeur (Si FF alors EOF ← 1 Fsi)
Fin Tant Que
Faire Calculs Globaux
Faire Pied de page
Fermer Fichiers
```

◆ Edition d'un document à partir de plusieurs enregistrements d'un fichier :

```
Ouvrir Fichiers
Initialiser les variables
EOF ← 0
Lire Fichier-Directeur (Si FF alors EOF ← 1 Fsi)
Tant Que (EOF=0) répéter
|  Faire Entête
|  Initialisation d'une condition
|  Tant Que (EOF=0 et condition vraie) répéter
|  |  Faire traitement d'un enregistrement
|  |  Lire Fichier-Directeur (Si FF alors EOF ← 1 Fsi)
|  Fin Tant Que
|  Faire Pied de page
Fin Tant Que
Faire Calculs Globaux
Fermer Fichiers
```

◆ Edition d'un document pour chaque enregistrement d'un fichier :

```
Ouvrir Fichiers
Initialiser les variables
EOF ← 0
Lire Fichier-Directeur (Si FF alors EOF ← 1 Fsi)
Tant Que (EOF=0) répéter
|  Faire Entête
|  Faire Corps
|  Faire Pied de page
|  Lire Fichier-Directeur (Si FF alors EOF ← 1 Fsi)
Fin Tant Que
Fermer Fichiers
```