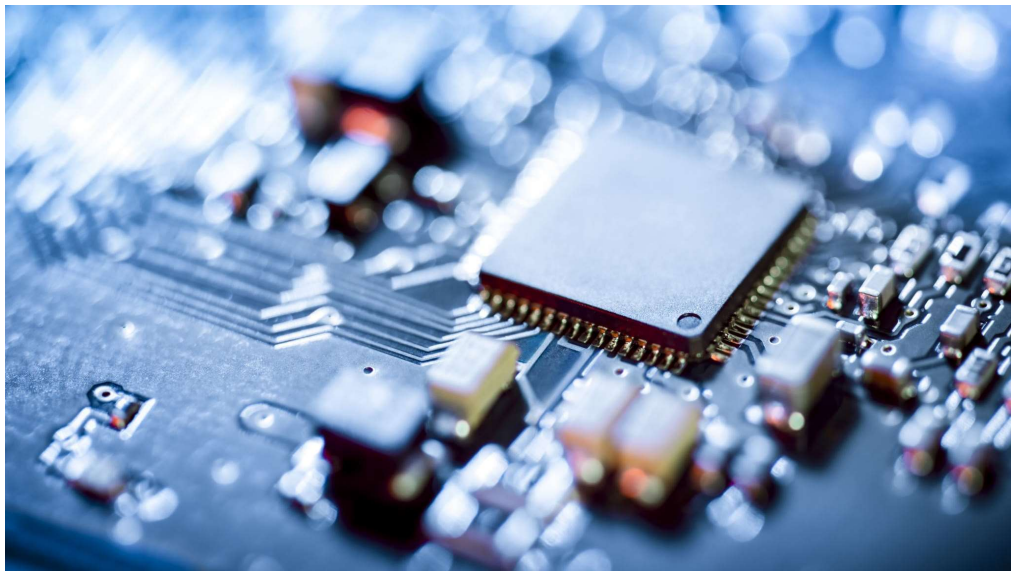
ALGORITHMIQUE



Intervenant

Habib NDIAYE

Plan

- Partie I : Algorithme
  - Chapitre 1 : Introduction
  - Chapitre 2 : Notions de base
    - Variables
    - Types
    - Opérateurs
    - Expressions
  - Chapitre 3 : Instructions
    - Affectation
    - Lecture/Ecriture
    - Structures de contrôle

Plan

- Chapitre 4 : Structures de données
  - Tableaux
  - Enregistrements
- Chapitre 5 : Sous-programmes
  - Procédures
  - Fonctions

Introduction

Quelques définitions.

- Algorithmique : science qui étudie les algorithmes et leurs propriétés.
- Algorithme :

  – une suite d'instructions qui, exécutées les unes à la suite des autres, permet de résoudre un problème donné;

  – enchaînement des actions nécessaires à la résolution d'un problème;
- Langage de programmation : formalisme basé sur une langue et qui définit une manière de donner des ordres à l'ordinateur;
- Programme :

  – algorithme écrit dans un langage de programmation;

  – ensemble d'instructions exécutables par un ordinateur;
- Programmeur : celui qui donne à des ordres à l'ordinateur en écrivant des programmes.

Introduction

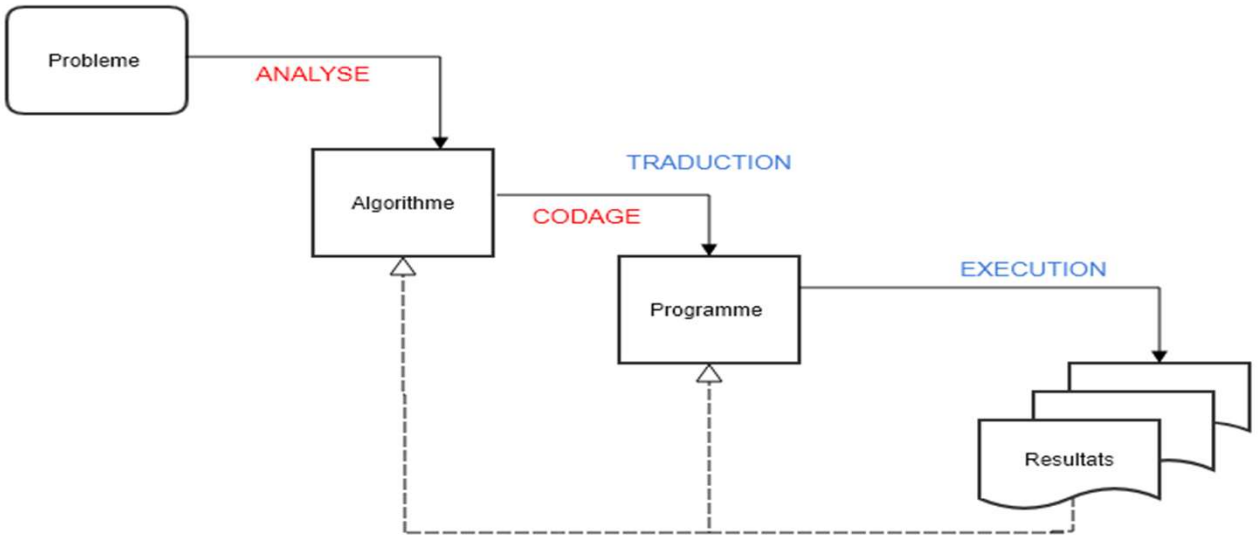
En quoi a-t-on besoin d'un langage spécial, distinct des langages de programmation compréhensibles par les ordinateurs ?

Parce que l'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage.

Apprendre l'algorithmique, c'est apprendre à manier la structure logique d'un programme informatique. Cette dimension est présente quelle que soit le langage de programmation ; mais lorsqu'on programme dans un langage (en C, en Visual Basic, etc.) on doit en plus se colleter les problèmes de syntaxe, ou de types d'instructions, propres à ce langage. Apprendre l'algorithmique de manière séparée, c'est donc sérier les difficultés pour mieux les vaincre.

Introduction

La démarche à suivre :



Introduction : Structure générale d'un algorithme

Un algorithme comprend deux parties : **un entête** et **un corps**.

1- L'entête fournit des informations sur l'algorithme, telles que :

- le nom de l'algorithme;
- les données locales;
- les données prises en entrée et/ou retournées à la sortie.

2- Le corps comprend:

- le mot-clé DEBUT;
- une suite d'instructions indentées;
- et le mot-clé FIN.

Introduction : Propriètes d'un algorithme

- Il ne dépend d'aucun langage de programmation, ni de la machine sur laquelle il sera exécuté;
- Il doit être lisible et sans ambigüités, compréhensible même par un non-informaticien;
- Il doit respecter certaines règles d'écriture;
- Il doit se terminer après un nombre fini d'étapes;
- Il doit être concis : aller à l'essentiel et éviter les étapes inutiles.

Introduction : Règles d'écriture

Pour avoir une portée universelle, un algorithme se doit de respecter quelques règles lors de son écriture :

- avoir une écriture rigoureuse et soignée;
- respecter l'indentation afin de faire ressortir la structure de l'algorithme;
- le nom de l'algorithme doit être parlant, représentatif du rôle de l'algorithme;
- le nom d'un algorithme ne doit contenir ni espaces, ni caractères spéciaux et doit commencer par une lettre. Il ne peut pas commencer par un chiffre.

Eléments de base : Variables

Durant son exécution, un algorithme manipule des données(ou informations). On peut avoir en permanence besoin de stocker provisoirement ces données. Il peut s'agir de données issues du disque dur, fournies par l'utilisateur (frappées au clavier)…

On fait appel alors à la notion de variable pour sauvegarder ces données.

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

Une variable est donc caractérisée par 3 attributs qui sont :

- son identificateur : c'est le nom donné à la variable;
- sa valeur : c'est l'information contenue (stockée) dans la variable;
- son type : c'est la nature de l'information contenue dans la variable.

NB: Une variable ne peut contenir qu'une seule valeur à la fois !

Il existe des variables dont la valeur ne change pas une fois initialisée : ce sont les constantes.

Eléments de base : Types

Selon la nature des informations (du texte, des nombres, des caractères spéciaux etc.) qu'on veut stocker dans les variables, on distingue essentiellement trois types de données:

- le type numérique;
- le type alphanumérique;
- le type booléen.

Eléments de base : Type Numérique

Le type numérique regroupe les entiers d'une part et les réels d'autre part.

- Les entiers : une variable est dite de type entier si elle prend ses valeurs dans l'ensemble $\mathbb{Z}$ des entiers relatifs.

- Les réels : Une variable est dite de type réel si elle prend ses valeurs dans l'ensemble $\mathbb{R}$ des nombres réels.

Remarque : Nous verrons par la suite qu'il existe des opérations définies pour chaque type de données.

Eléments de base : Type Alphanumérique

Le type alphanumérique caractérise une variable dont le contenu peut être des lettres, des caractères spéciaux, des espaces ou même des chiffres. Le type alphanumérique regroupe les caractères et les chaînes de caractères.

- Un caractère (noté CARACTERE) peut être : une lettre('a', 'b',..., 'z', 'A', 'B',..., 'Z'), un chiffre ('0', '1',..., '9'), un caractère spécial('|', '+', '$', ';' ,'[',...) ou encore un espace(noté ' '). Un caractère est toujours délimité par les quotes simples.

- Une chaîne de caractères (noté CHAINE) est une suite finie de caractères quelconques. Cette suite peut aussi être vide(ne contient aucun caractère) ou contenir un seul caractère. Exemples : "Hello world", "a", "0123dns", ...

Eléments de base : Type Booléen

Le dernier type de variables est le type booléen : on y stocke uniquement les valeurs logiques VRAI et FAUX.

On peut représenter ces notions abstraites de VRAI et de FAUX par tout ce qu'on veut : de l'anglais (TRUE et FALSE) ou des nombres (0 et 1).

Eléments de base : Déclaration de variables

Avant de pouvoir utiliser une variable, il faut d'abord la déclarer au début de l'algorithme. La déclaration des variables permet de réserver de l'espace mémoire pour stocker les données représentées par ces variables. Pour déclarer une variable, il faut connaître le type de la variable et repérer la variable par un identifiant

identificateur : type

On peut déclarer plusieurs variables en une seule fois.

Exemples:

1. Variables : a : Entier
2. Variables : nom, prenom : Chaîne
   i : Entier
   b, surface : Réel

Eléments de base : Déclaration de variables

Pour une constante, sa déclaration est toujours associée à son initialisation(i.e. lui donner une valeur de départ).

Syntaxe:

constante type identificateur valeur

La déclaration se fait avec le mot-clé **constante**.

Exemples:

1. constante Réel prix 2500 // Durant tout l'algorithme, prix sera égal à 2500
2. Constante Chaîne hello "Hello, World!"

Eléments de base : Opérateurs Numériques

Un opérateur est un signe(ou symbole) qui permet de faire des calculs sur des variables.
Les opérateurs dépendent du type des variables.

Pour le type entier

+	Addition
-	Soustraction
*	Multiplication
/	Division entière
mod	Division modulo

Pour le type réel

+	Addition
-	Soustraction
*	Multiplication
/	Division

Eléments de base : Opérateurs Alphanumériques

Ce sont les opérateurs applicables sur caractères d'une part, et sur les chaines de caractères d'autre part. Ces opérateurs sont appelés opérateurs relationnels, c'est-à-dire qu'ils permettent de faire des comparaisons.

Pour les caractères

==	Egalité
!=	Different
<	Inférieur
<=	Inférieur ou Egale
>	Supérieur
>=	Supérieur ou égale

Remarque : Il existe un ordre défini sur l'ensemble des caractères. Par exemple, on a : ' '< '0' <'1' < ... < '9' < 'A' < 'B'< ... <'Z' < 'a' < 'b'< ... <'z'.

Eléments de base : Opérateurs Alphanumériques

Pour les chaines de caractères, on retrouve les mêmes opérations que pour les caractères. Mais il existe en plus une autre opération spécifique aux chaînes de caractères qu'on appelle la concaténation. Elle est notée ||. La concaténation permet de "coller" deux chaînes entre elles afin d'en créer une nouvelle.

Exemple :

1. "conca" || "ténation" = "concaténation"
2. "bon" || "jour" = "bonjour"

Eléments de base : Opérateurs Booléens

On définit trois opérations pour les variables de type booléen : la négation, l'union et l'intersection.

Soient A et B deux expressions logiques.

La négation

A	!A
Vrai	Faux
Faux	Vrai

L'union

A **OU** B	A VRAI	A FAUX
B VRAI	VRAI	VRAI
B FAUX	VRAI	FAUX

L'intersection

A **ET** B	A VRAI	A FAUX
B VRAI	VRAI	FAUX
B FAUX	FAUX	FAUX

Eléments de base : Expressions

Une expression est une combinaison d'opérateurs et d'opérandes dont l'évaluation(durant l'exécution de l'algorithme) fournit une valeur d'un certain type. Les opérandes intervenant dans l'expression peuvent être des variables, des constantes, des valeurs littérales.

Exemples:

- "hello world"
- i+1
- a+b

Instructions

Il existe quatre familles instructions compréhensibles par les ordinateurs. Ces quatre familles d'instructions sont :

- L'affectation
- La lecture/l'écriture
- Les tests
- Les boucles

Instructions : Affectation

L'affectation consiste à donner une valeur à une variable. En pseudo-code, l'affectation est notée par la flèche ←. La syntaxe est :

identificateur ← valeur ;

La valeur peut être :

- du même type que identificateur; A ← 24

  Ceci sous-entend impérativement que A soit une variable de type entier. Si A a été défini dans un autre type, il faut bien comprendre que cette instruction provoquera une erreur.

- une variable du même type que l'identificateur; Z ← A

  Signifie que la valeur de Z est maintenant celle de A.

  Notez bien que cette instruction n'a en rien modifié la valeur de A : une instruction d'affectation ne modifie que ce qui est situé à gauche de la flèche.

Instructions : Affectation

- une expression dont l'évaluation fournit un résultat du même type que identificateur.

  Z ← A + 4

  Si A contenait 12, Z vaut maintenant 16. De même que précédemment, A vaut toujours 12.

  Z ← Z + 1

  Si Z valait 6, il vaut maintenant 7. La valeur de Z est modifiée, puisque Z est la variable située à gauche de la flèche.

Il va de soi que l'ordre dans lequel les instructions sont écrites va jouer un rôle essentiel dans le résultat final.

```
Variable A : Entier
Début
        A ← 37
        A ← 43
Fin
```

```
Variable A : Entier
Début
        A ← 43
        A ← 37
Fin
```

Instructions : Lecture/Ecriture

En algorithmique, il existe aussi des instructions qui permettent à la machine de dialoguer (et donc d'avoir des interactions) avec l'utilisateur. On les appelle instructions **d'entrée/sortie** ou de **lecture/écriture**.

Remarque: lecture et écriture sont des termes qui doivent être compris du point de vue de la machine qui sera chargée de les exécuter, et non de celui de l'utilisateur.

Pour afficher une information sur l'écran, on utilise la commande **ECRIRE**. Pour lire une information saisie par l'utilisateur au clavier, on utilise la commande **LIRE**.

Exemple:

```
ALGORITHME: EntreeSortie
VARIABLE age :Entier
DEBUT
    ECRIRE ("Entrez votre age :")
    LIRE(age)
FIN
```

```
ALGORITHME: EntreeSortie
VARIABLE nom, prenom :CHAINE
DEBUT
    ECRIRE ("Entrez votre prenom et nom de famille :")
    LIRE(prenom, nom)
    ECRIRE ("Hello ", prenom, " ", nom)
FIN
```

Instructions : Structures de contrôle

Il est possible de demander à l'ordinateur d'effectuer des instructions selon que la situation se présente d'une manière ou d'une autre. On parle alors de structures de contrôle. Il existe deux grandes familles de structures de contrôle :

les structures conditionnelles (ou tests) ;

les structures répétitives (ou boucles).

Instructions : Tests

Il n'y a que deux formes possibles pour un test ; la première est la plus simple, la seconde la plus complexe.

```
Si booleen alors
        Instruction(s)
FinSi
```

```
Si booleen alors
        Instruction(s)
Sinon
        Instruction(s)
FinSi
```

Un booléen est une expression dont la valeur est VRAI ou FAUX. Cela peut donc être (il n'y a que deux possibilités) :

- une variable (ou une expression) de type booléen
- une condition

Qu'est ce qu'une condition ?

Une condition est une comparaison.

Instructions : Tests

Cette définition est essentielle ! Elle signifie qu'une condition est composée de trois éléments:

- une valeur
- un opérateur de comparaison
- une autre valeur

Les valeurs peuvent être a priori de n'importe quel type (numériques, caractères…). Mais si l'on veut que la comparaison ait un sens, il faut que les deux valeurs de la comparaison soient du même type !

L'ensemble des trois éléments composant la condition constitue donc, si l'on veut, une affirmation, qui à un moment donné est VRAIE ou FAUSSE.

A Faire:

1. Ecrire un algorithme qui teste si un nombre saisi par l'utilisateur est nul ou non
2. Ecrire un algorithme qui calcule la valeur absolue de la différence de 2 nombres.

Instructions : Tests

```
ALGORITHME : NombreNulOuNon
VARIABLES : n : ENTIER;
DEBUT
    ECRIRE("Entrez un entier")
    LIRE(n)
    SI (n = 0) ALORS
        ECRIRE(" Le nombre saisi est nul")
    SINON
        ECRIRE(" Le nombre saisi est non nul")
    FIN SI
 FIN
```

```
ALGORITHME : ValeurAbsolueDifference
VARIABLES : a,b, diffValAbs : ENTIER;
DEBUT
    ECRIRE("Entrez deux entiers")
    LIRE(a,b)
    diffValAbs ← a - b
    SI (diffValAbs < 0) ALORS
        diffValAbs ← b - a
    FIN SI
    ECRIRE(" La difference de la valeur absolue
des 2 nombres est ", diffValAbs)
 FIN
```

NB: Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple exposée ci-dessus.

Exemple: b est inclus entre 14 et 29 .

Instructions : Tests

Parfois, une seule clause si n'est pas suffisante pour évaluer tous les cas de figure. Il est alors possible d'imbriquer plusieurs boucles de tests (pouvant être des SI/SINON ou SI/FIN SI). Ci-dessous un exemple de syntaxe:

```
SI (condition1) ALORS
    SI (condition2) ALORS
        Instructions
    SINON
        Instructions
    FIN SI
SINON
    SI (condition3) ALORS
      Instructions
    FIN SI
FIN SI
```

```
ALGORITHME : Temperature
VARIABLE temp : ENTIER;
DEBUT
    ECRIRE("Entrez une température")
    LIRE(temp)
    SI (temp <= 0) ALORS
        ECRIRE("Glace")
    SINON
        SI (temp < 100) ALORS
            ECRIRE("Eau")
        SINON
            ECRIRE("Vapeur")
        FIN SI
    FIN SI
FIN
```

Instructions : Tests

Une structure de choix permet d'exécuter une action dépendant de la valeur (discrète) d'une certaine variable. La syntaxe est la suivante.

```
SELON variable FAIRE
    valeur1 : action1
    valeur2 : action2
    ……..
    valeurN : actionN
    SINON : action par défaut
FINSELON
```

L'action par défaut sera exécutée si la variable ne prend aucune des valeurs précitées.

Instructions : Les boucles

1. La boucle Pour

Elle permet de répéter une suite d'instructions en un nombre voulu de fois. Ce nombre est connu d'avance et est spécifié par les valeurs valeur-initiale et valeur-finale. La syntaxe est la suivante :

```
POUR identificateur ← valeur-initiale A valeur-finale FAIRE
        Instructions
FINPOUR
```

NB: L'identificateur est un **entier positif** qui va varier entre valeur-initiale et valeur-finale.

Instructions : Les boucles

2. La boucle Tant Que

Elle permet de répéter une suite d'instructions tant qu'une certaine condition est vérifiée. La syntaxe est la suivante :

```
TANT QUE (condition) FAIRE
        Instructions
FINTANTQUE
```

- La condition est testée avant la première exécution de l'algorithme.
- L'action sera exécutée tant que la condition restera vérifiée.
- La sortie de la boucle correspondra alors à la non-satisfaction de la condition.

Instructions : Les boucles

3. La boucle Répéter

On répète une certaine action jusqu'à ce qu'une certaine condition soit vérifiée. La syntaxe est la suivante :

```
REPETER
        Instructions;
JUSQU'A (condition);
```

- La condition est testée après une première exécution de l'algorithme.
- Les instructions sont répétées jusqu'à ce que la condition de sortie soit vérifiée.

Les structures de données

Supposons que vous ayez à traiter simultanément plusieurs variables, par exemple des notes pour calculer une moyenne.

```
Algorithme: CalculMoyenne
Variables: a1, a2, a3, a4, a5, a6, moyenne : Réel
Début
    a1 ← 7
    a2 ← 1
    a3 ← 16
    a4 ← 23
    a5 ← 19
    a6 ← 31
    moyenne ← (a1+a2+a3+a4+a5+a6)/6
    Ecrire "La moyenne vaut ", moyenne
Fin
```

Il peut alors devenir très vite laborieux de déclarer et initialiser les variables une à une, surtout s'il s'agit de centaines voire de milliers de valeurs à traiter, ou si l'on ne connait pas à l'avance le nombre de valeurs à traiter. On définit de nouveaux types permettant de regrouper toutes ces variables en une seule. On parle alors de structures de données.

On parlera essentiellement de deux structures de données :

- les tableaux : lorsque les variables traitées sont de même type;
- les enregistrements : pour des types différents.

Les structures de données : Tableaux

Un tableau est un type structuré qui permet de stocker un ensemble fini de données de même type. Les données sont repérées par un identifiant unique qui est le nom du tableau.

Exemple: Supposons que l'on dispose des mesures pluviométriques mensuelles d'une région pour calculer pluviométrie moyenne sur une période donnée. On stockera alors les données(mesures pluviométriques mensuelles) dans un tableau(TabPluieMensuelles) de12 éléments.

33	0	4	0	0	56	400	1735	749	109	0	1
1	2	3	4	5	6	7	8	9	10	11	12

Chaque élément du tableau est repéré par un nombre appelé **indice**.

L'indice est un nombre compris entre 1 et le nombre total d'éléments du tableau.

Les structures de données : Tableaux

Pour déclarer un tableau, on utilise la syntaxe suivante :

TABLEAU nomDuTableau(nbreTotalElements) : TYPE

Exemple: TABLEAU TabPluieMensuelles(12) : Réel

Accès à un élément du tableau :

Le nom donné au tableau identifie de manière globale toutes les données qui y sont stockées. Pour accéder à un seul élément du tableau, on le repère par l'indice qui indique sa position dans le tableau.

Exemple:

TABLEAU TabPluieMensuelles(12) : Réel

TabPluieMensuelles[1]; désigne l'élément à la première position dans le tableau.

TabPluieMensuelles[12]; désigne l'élément à la dernière position dans le tableau.

Remarque: L'indice est un entier positif qui ne doit pas dépasser la taille du tableau.

Les structures de données : Tableaux

Tableaux et boucles :

Les tableaux peuvent être manipulés (affectation, lecture, ...) en utilisant les boucles. L'idée est alors de parcourir le tableau avec un compteur.

```
ALGORITHME: Initisalisation
TABLEAU tab(7) : Entier
VARIABLE i : Entier
Début
    POUR i ← 1 A 7 FAIRE
        tab[i] ←i + 2
    FINPOUR
Fin
```

```
ALGORITHME: InitialisationEtAffichageTableau
TABLEAU tab(12) : Réel
VARIABLE i : Entier
Début
POUR i ← 1 A 12 FAIRE
        ECRIRE("Entrez une valeur du tableau")
        LIRE(tab[i])
FINPOUR
Pour i ← 1 A 12 FAIRE
        ECRIRE(tab[i])
FINPOUR
FIN
```

Les structures de données : Tableaux

Recherche d'un élément dans un tableau

Il est possible de rechercher la présence d'un élément dans un tableau. L'idée est alors de parcourir un à un les éléments du tableau jusqu'à rencontrer l'élément recherché.

Nous allons maintenant nous intéresser au maniement habile d'une variable booléenne : la technique dite du « flag ».

Le flag est un petit drapeau, qui va rester baissé aussi longtemps que l'événement attendu ne se produit pas. Et, aussitôt que cet événement a lieu, le petit drapeau se lève (la variable booléenne change de valeur). Ainsi, la valeur finale de la variable booléenne permet au programmeur de savoir si l'événement a eu lieu ou non.

A faire: Ecrire un algorithme qui va demander à l'utilisateur d'entrer 12 valeurs réelles dans un premier temps. Il demande ensuite à l'utilisateur d'entrer une valeur qu'il désire rechercher dans la suite de valeurs. Le programme doit afficher à la fin si la valeur recherchée existe ou pas dans le tableau.

Les structures de données : Tableaux

Techniques rusées:

Il existe plusieurs stratégies possibles pour trier les éléments d'un tableau ; nous en verrons deux : le tri par sélection et le tri à bulles.

Tri par sélection

Admettons que le but de la manœuvre soit de trier un tableau de 5 éléments dans l'ordre croissant. La technique du tri par sélection est la suivante : on met en bonne position l'élément numéro 1, c'est-à-dire le plus petit. Puis en met en bonne position l'élément suivant. Et ainsi de suite jusqu'au dernier. Par exemple, si l'on part de :

45	12	78	122	28

On commence par rechercher, parmi les 5 valeurs, quel est le plus petit élément , et où il se trouve. On l'identifie en deuxième position (c'est le nombre 12), et on l'échange alors avec le premier élément (le nombre 45). Le tableau devient ainsi :

Les structures de données : Tableaux

12	45	78	122	28

On recommence à chercher le plus petit élément, mais cette fois, seulement à partir du deuxième (puisque le premier est maintenant correct, on n'y touche plus). On le trouve en dernière position (c'est le nombre 28). On échange donc le deuxième avec le dernière :

12	28	78	122	45

On recommence à chercher le plus petit élément à partir du troisième (puisque les deux premiers sont maintenant bien placés), et on le place correctement, en l'échangeant, ce qui donnera in fine :

12	28	45	122	78
12	28	45	78	122

Les structures de données : Tableaux

Nous pourrions décrire le processus de la manière suivante :

Boucle principale: prenons comme point de départ le premier élément, puis le second, ..., jusqu'à l'avant dernier.

Boucle secondaire: à partir de ce point de départ mouvant, recherchons jusqu'à la fin du tableau quel et le plus petit élément. Une fois que nous l'avons trouvé, nous l'échangeons avec le point de départ.

Les structures de données : Tableaux

Résolution de l'exemple précédent :

```
Algorithme: TriSelection
Tableau: tab(5)              : Entier
Variables: i, j, positionMin, temp  : Entier
Début
*** On suppose que le tableau est déjà rempli et non trié ***
    Pour i ← 1 à 4 Faire
        positionMin ← i
        Pour j ← i + 1 à 5 Faire
            Si tab[j] < tab[positionMin] Alors
                positionMin ← j
            FinSi
        FinPour
        temp ← tab[positionMin]
        tab[positionMin] ← tab[i]
        tab[i] ← temp
    FinPour
Fin
```

Les structures de données : Tableaux

Tri à bulles (Bubble Sort)

L'idée de départ du tri à bulles consiste à se dire qu'un tableau trié en ordre croissant, c'est un tableau dans lequel tout élément est plus petit que celui qui le suit.

En effet, prenons chaque élément d'un tableau, et comparons-le avec l'élément qui le suit. Si l'ordre n'est pas bon, on permute ces deux éléments. Et on recommence jusqu'à ce que l'on n'ait plus aucune permutation à effectuer. Les éléments les plus grands « *remontent* » ainsi peu à peu vers les dernières places, ce qui explique la dénomination de « **tri à bulle** ».

En quoi le tri à bulles implique-t-il l'utilisation d'un flag ?

On ne sait jamais par avance combien de remontées de bulles on doit effectuer. En fait, tout ce qu'on peut dire, c'est qu'on devra effectuer le tri jusqu'à ce qu'il n'y ait plus d'éléments qui soient mal classés. Ceci est typiquement un cas de question « *asymétrique* » : il suffit que deux éléments soient mal classés pour qu'un tableau ne soit pas trié. En revanche, il faut que tous les éléments soient bien rangés pour que le tableau soit trié.

Les structures de données : Tableaux

Résolution de l'exemple précédent avec la technique du tri à bulles:

```
Algorithme: TriBulles
Tableau: tab(5)          : Entier
Variables: i, temp       : Entier
           permutation   : Booleen
Début
*** On suppose que le tableau est déjà rempli et non trié ***
    permutation ← Vrai
    TantQue (permutation) Faire
        permutation ← Faux
        Pour i ← 1 à 4 Faire
            Si tab[i] > tab[i+1] Alors
                temp ← tab[i]
                tab[i] ← tab[i+1]
                tab[i+1] ← temp
                permutation ← Vrai
            FinSi
        FinPour
    FinTantQue
Fin
```

Les structures de données : Tableaux

Tableaux multidimensionnels

Supposons qu'on dispose d'un étudiant dont on veut calculer la moyenne générale. On a ainsi un ensemble de matières (par exemple 7) dont chacune est identifiée par une note et un coefficient. On peut alors représenter cette situation par un tableau comme suit :

Note	13	08	17	12	15	13	09
Coeff	1	3	3	5	4	2	2

On obtient ainsi un tableau formé de plusieurs lignes et colonnes. En algorithmique, il est possible de représenter ces données sous forme de tableaux à deux dimensions. Ce sont des tableaux dont les éléments sont eux-mêmes des tableaux.

SYNTAXE

TABLEAU nomTableau(nbreLignes, nbreColonnes) : TYPE

Exemple: Tableau MonTableau(7,5) : CHAINE

Les structures de données : Tableaux

Accès a un élément du tableau

Pour accéder à un seul élément du tableau, on le repère par deux indices : l'un indique la ligne sur laquelle se trouve l'élément, l'autre la colonne.

Exemple:

Tableau montab(7, 5) : entier

montab[2][5]; désigne l'élément à la 2e ligne, 5e colonne.

montab[1][3]; désigne l'élément à la 1ére ligne, 3e colonne.

Tableaux multidimensionnels et boucles :

Pour utiliser les éléments d'un tableau multidimensionnel, on utilise en général deux boucles imbriquées, l'une pour parcourir les lignes, et l'autre pour parcourir les colonnes.

Les structures de données : Tableaux

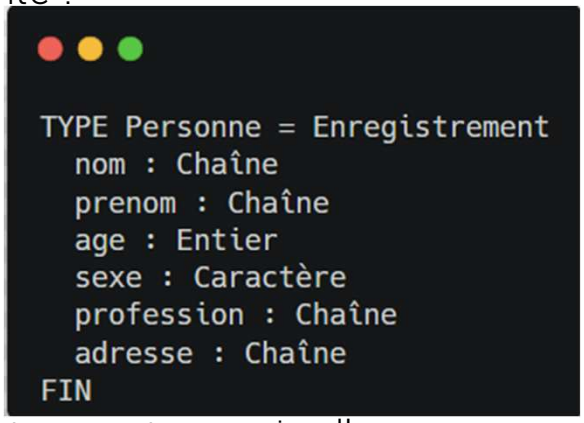
```
ALGORITHME: TableauMulti
TABLEAU Matrice(3,5) : Entier
VARIABLE i, j : Entier
Début
 POUR i← 1 A 3 FAIRE
        POUR j← 1 A 5 FAIRE
                ECRIRE("Entrez une valeur")
                LIRE(Matrice[i][j])
                ECRIRE("La valeur sur la ligne ", i, " et la colonne ", j, " est : ", Matrice[i][j])
        FINPOUR
 FINPOUR
Fin
```

Les structures de données : Enregistrements

Un enregistrement est un type structuré qui permet de regrouper des variables pouvant être de types différents au sein d'une même entité. Un enregistrement répond à un besoin et est défini par le programmeur lui-même.

La syntaxe de définition d'un enregistrement est la suivante :

```
TYPE nomEnregistrement = ENREGISTREMENT
        champ 1 : type1
        champ 2 : type2
        ...
        champ n : typeN
FIN
```



```
TYPE Personne = Enregistrement
  nom : Chaîne
  prenom : Chaîne
  age : Entier
  sexe : Caractère
  profession : Chaîne
  adresse : Chaîne
FIN
```

Remarque: On peut aussi avoir un champ de type Enregistrement au sein d'un enregistrement

Déclaration:

Variable : p : Personne

Les structures de données : Enregistrements

Initialisation:

Pour initialiser les champs de la variable personne de type Personne, on procède comme suit:

p.nom ← “Modou"
p.prenom ← “Wade"
p.age ← 19
p.sexe ← ’M’
p.profession ← “Etudiant"
p.adresse ← “Sicap"

Il existe néanmoins une autre façon d’initialiser une variable de type enregistrement.

p ← {"Modou", "Wade", 19, ’M’, "Etudiant", "Sicap"}

Cela suppose bien sûr que le type enregistrement Personne ait déjà été défini.

Sous-programmes

Lorsque l'on progresse dans la conception d'un algorithme, ce dernier peut prendre une taille et une complexité croissante. De même des séquences d'instructions peuvent se répéter à plusieurs endroits.

Un algorithme écrit d'un seul tenant devient difficile à comprendre et à gérer dès qu'il dépasse deux pages. La solution consiste alors à découper l'algorithme en plusieurs parties plus petites. Ces parties sont appelées des sous-algorithmes.

Le sous-algorithme est écrit séparément du corps de l'algorithme principal et sera appelé par celui-ci quand ceci sera nécessaire.

En algorithmique, on distingue deux types de sous-programmes :

- les procédures,
- les fonctions.

Ainsi, un algorithme sera en fait un programme principal qui fera appel à un ensemble de sous-programmes (les procédures et les fonctions).

Sous-programmes : Procédures

Une procédure est un sous-programme qui retourne zéro, une ou plusieurs valeurs.

Syntaxe:

```
PROCEDURE nomdeLaProcedure(liste des arguments : TYPE)
déclaration des variables
Début
    instructions
Fin
```

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

Sous-programmes : Procédures

Exemple :

Définir une procédure qui indique si l'année donnée en paramètre est bissextile ou pas.

```
Procédure Bissextile(X : Entier )
Début
    Si  ((X mod 4 == 0 ET X mod 100 != 0) OU (X mod 400 == 0)) Alors
        Ecrire "Année Bissextile"
    Sinon
        Ecrire "Année non Bissextile"
    FinSi
Fin
```

Sous-programmes : Procédures

Une fois la procédure définie, on peut l'utiliser au sein du programme principal en procédant à l'appel. La syntaxe est la suivante :

nomdeLaProcedure(liste des arguments)

Dans l'appel d'une procédure :

- le nombre d'arguments et l'ordre d'apparition doivent être les mêmes que lors de la définition de la procédure;
- on ne spécifie pas le type des arguments. Le programme dans lequel on fait l'appel de la procédure sera dit programme appelant.

Les arguments intervenant dans les procédures sont appelés des paramètres. Ceux qui interviennent dans la définition sont appelés paramètres fictifs et ceux intervenant lors de l'appel de la procédure sont les paramètres réels.

A Faire :

Ecrire un algorithme appelant, utilisant la procédure Bissextile de l'exemple précédent.

Sous-programmes : Passage de paramètres

Les échanges d'informations entre une procédure et l'algorithme appelant se font par l'intermédiaire de paramètres.

Il existe deux principaux types de passages de paramètres qui permettent des usages différents :

Passage par valeur :

Dans ce type de passage, le paramètre formel reçoit uniquement une copie de la valeur du paramètre effectif. La valeur du paramètre effectifs ne sera jamais modifiée.

Interprétation :

X=10	Programme Princ.	Procédure
Appel de ESSAI	X=10	i=10
Exécution de ESSAI	X=10	i=30
Après ESSAI	X=10	i=30

```
Algorithme: PassageValeur
Variable: x : Entier

Procédure ESSAI ( i : Entier )
DÉBUT
    i ← 3 * i
    Écrire "Le paramètre valeur i =" , i
Fin

Début
    Ecrire "Donner la valeur de X :"
    Lire (x)
    Ecrire "Avant appel x = " , x
    ESSAI(x)
    Ecrire "Après appel x = " , x
Fin
```

Sous-programmes : Passage de paramètres

Passage par référence ou par adresse :

Dans ce type de passage, la procédure utilise l'adresse du paramètre effectif. Lorsqu'on utilise l'adresse du paramètre, on accède directement à son contenu. La valeur de la variable effectif sera donc modifiée.

Les paramètres passés par adresse sont précédés du mot clé Var.

```
Algorithme: PassageRéférence
Variable: x : Entier

Procédure ESSAI (VAR i : Entier )
DÉBUT
    i ← 3 * i
    Écrire "Le paramètre valeur i =" , i
Fin

Début
    Ecrire "Donner la valeur de X :"
    Lire (x)
    Ecrire "Avant appel x = " , x
    ESSAI(x)
    Ecrire "Après appel x = " , x
Fin
```

Interprétation :

X=10	Programme Princ.	Procédure
Appel de ESSAI	X=10	i=10
Exécution de ESSAI	X=30	i=30
Après ESSAI	X=30	i=30

Sous-programmes : Fonctions

Les fonctions sont des sous algorithmes admettant des paramètres et retournant un seul résultat (une seule valeur) de type simple qui peut apparaître dans une expression, dans une comparaison, à la droite d'une affectation, etc.

SYNTAXE

```
Fonction nomFonction (liste des arguments: TYPE) : TYPE
Déclaration des variables
Début
    Instruction(s)
    Retourner Expression
Fin
```

La syntaxe de la déclaration d'une fonction est assez proche de celle d'une procédure à laquelle on ajoute un type qui représente le type de la valeur retournée par la fonction et une instruction Retourner Expression. Cette dernière instruction renvoie au programme appelant le résultat de l'expression placée à la suite du mot clé Retourner.

Sous-programmes : Fonctions

Exemple :

Définir une fonction qui renvoie le plus grand de deux nombres différents

```
Fonction Max(X : Réel, Y : Réel) : Réel
Variables:
Début
    Si  X > Y Alors
        Retouner X
    Sinon
        Retouner Y
    FinSi
Fin
```

Sous-programmes : Fonctions

Appel d'une fonction :

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom suivie des paramètres effectifs. C'est la même syntaxe qu'une procédure.

A la différence d'une procédure, la fonction retourne une valeur. L'appel d'une fonction pourra donc être utilisé dans une instruction (affichage, affectation, ...) qui utilise sa valeur.

SYNTAXE

nomFonction(liste des arguments)

A Faire :

Ecrire un algorithme appelant, utilisant la fonction Max de l'exemple précédent.

Sous-programmes : Portée des variables

La portée d'une variable désigne le domaine de visibilité de cette variable. Une variable peut être déclarée dans deux emplacements distincts.

Une variable déclarée dans la partie déclaration de l'algorithme principale est appelée variable globale. Elle est accessible de n'importe où dans l'algorithme, même depuis les procédures et les fonctions. Elle existe pendant toute la durée de vie du programme.

Une variable déclarée à l'intérieur d'une procédure (ou une fonction) est dite locale. Elle n'est accessible qu'à la procédure au sein de laquelle elle définie, les autres procédures n'y ont pas accès. La durée de vie d'une variable locale est limitée à la durée d'exécution de la procédure.

```
Algorithme: PortéeVariable
Variables: X,S : Entier

Procédure Somme():
Variables: i, Som : Entier
Début
    Som ← 0
    Pour i ← 1 à 100 Faire
        Som ← Som + 1
    FinPour
    Ecrire "La somme est", Som
Fin

Début
    Ecrire "Entrez X et S"
    Lire(X,S)
    Somme()
    .....
Fin
```

X et S sont des variables globales visibles dans tout l'algorithme.
i et Som sont des variables locales visibles uniquement à l'intérieur de la procédure

Sous-programmes : Récursivité

Une procédure (ou une fonction) est dite récursive si elle s'appelle elle-même.

Exemple :

Ecrire une fonction récursive permettant de calculer la factorielle d'un entier positif.

```
Fonction Fact(n : Entier) : Entier
Début
    Si n > 1 Alors
        Retourner ( Fact(n-1)*n )
    Sinon
        Retourner 1
    FinSi
Fin
```

Dans cet exemple, la fonction renvoie 1 si la valeur demandée est inférieur à 1, sinon elle fait appel à elle même avec un paramètre inférieur de 1 par rapport au précédent. Les valeurs de ces paramètres vont en décroissant et atteindront à un moment la valeur une (1). Dans ce cas, il n'y a pas d'appel récursif et donc nous sortons de la fonction.

Note : Toute procédure ou fonction récursive comporte une instruction (ou un bloc d'instructions) nommée "point terminal" permettant de sortir de la procédure ou de la fonction.
Le "point terminal" dans la fonction récursive Fact est : retourner 1.