UNIVERSITE DE DJIBOUTI

FACULTE DE SCIENCES

DEPARTEMENT INFORMATIQUES

COURS PROGRAMMATION PROCEDURAL & ALGORITHME

❖ Planning de l'UE

Semestre étudié	Titre des Chapitres
S2	**Chapitre 1** : **Tableau à un dimension**
	Chapitre 2 : **Tableau à N dimensions**
	Chapitre 3 : **Trie sur les tableaux**
	Chapitre 4 : **Fonctions & Procedures**
	Chapitre 5 : Pointeur

Sommaire générale

Chapitre 1 : Tableau à une dimension

Exemple introductif

Supposons qu'on veut conserver les notes d'une classe de 12 étudiants pour extraire quelques informations. Par exemple : calcul du nombre d'étudiants ayant une note supérieure à 10

Le seul moyen dont nous disposons actuellement consiste à déclarer 12 variables, par exemple N1, …, N12. Après 30 instructions lire, on doit écrire 30 instructions Si pour faire le calcul

Exemple calcul de la moyenne

- Moy ← (N1+N2+N3+N4+N5+N6+N7+N8+N9+N10+N11+N12)/12

```
nbre ← 0

Si (N1  >10) alors nbre ←nbre+1 FinSi

….

Si (N12>10) alors nbre ←nbre+1 FinSi
```

C'est lourd à écrire, Heureusement, les langages de programmation offrent la possibilité de rassembler toutes ces variables dans une seule structure de donnée appelée tableau

- rassembler toutes ces variables en une seule, au sein de laquelle chaque valeur sera désignée par un numéro

1. Définition

Un tableau est une structure de données permettant d'effectuer un même traitement sur des données de même nature.

tableau à **une** dimension

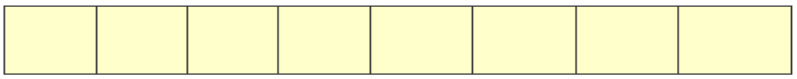
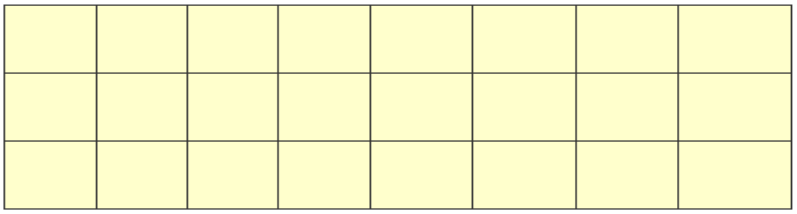
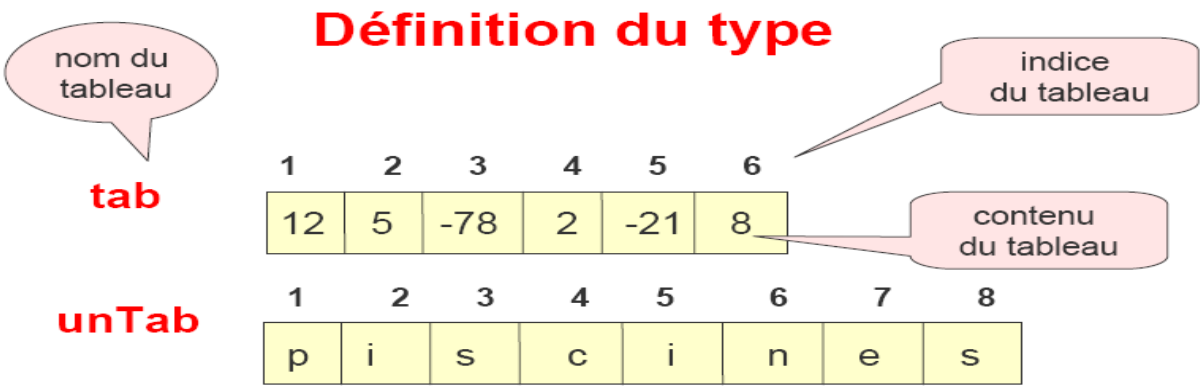


tableau à **deux** dimensions

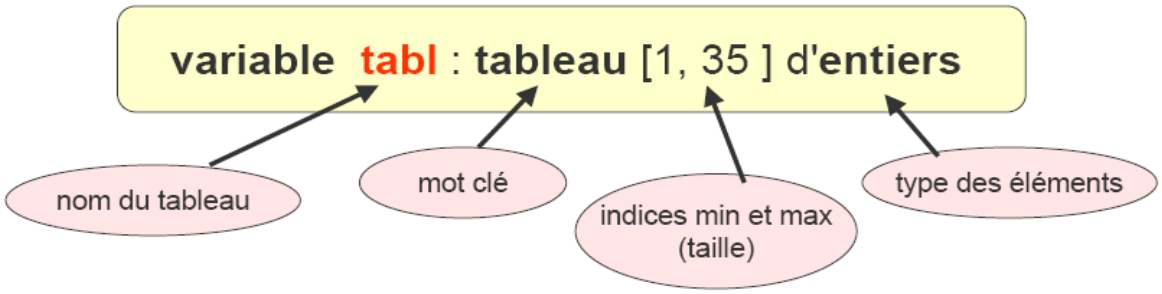




Remarques :
- Indices : en général, démarrage à 1, **mais en C++, démarrage à 0**
- Nombre d'octets occupés : dépend du type des valeurs enregistrées

2. *Déclaration d'un tableau*

Exemple : déclaration d'un tableau pouvant contenir jusqu'à 35 entiers



Autre exemple : déclaration d'un tableau qui contiendra les fréquences des températures comprises entre –40°C et 50°C

variable températures : tableau [-40, 50] de réels

- ✓ Un *tableau* possède un nom (ici note) et un nombre d'éléments (de cases) qui représente sa taille (ici 12).
- ✓ Tous les éléments d'un tableau ont le même type (c'est normal car ils représentent des valeurs logiquement équivalentes).
- ✓ Pour designer un élément, on indique le nom du tableau suivi son indice (son numéro) entre crochets:

3. *Les variables d'un tableau*

La notion de ≪case contenant une valeur ≫doit faire penser a celle de variable. Et, en effet, les cases du tableau, encore appelées éléments du tableau, sont des variables, qualifiées d'indices.

- ✓ *Différence entre variables classiques et variables indices:*

_ Les *variables classiques* sont déclarées individuellement et ont un nom distinct ;
_ Les *variables indices* (constituant le tableau) sont implicitement déclarées lors de la déclaration du tableau. Pour bien montrer que ces l'endroit qu'il faudra remplir.

❖ *Utilisation d'un tableau : par les indices*

- Accès en lecture :
  - **afficher**(tabl[4]) *{le contenu du tableau à l'indice 4 est affiché à l' écran}*
- Accès en écriture :
  - tabl[3] ←18 *{la valeur 18 est placée dans le tableau à l'indice 3}*
  - **saisir**(tabl[5]) *{la valeur entrée par l'utilisateur est enregistrée dans le tableau à l'indice 5}*
  - attention!

~~tabl ← 18~~ ~~nom[2] ← 3~~

Remarque:
Dans un programme, chaque élément d'un tableau est repéré par un indice. Dans la vie courante, nous utilisons souvent d'autres façons de repérer une valeur. Par exemple, au lieu de parler de note [1], note [2], note [3], nous préférons parler des notes des étudiants. Le tableau ne permet pas de repérer ses valeurs autrement que par un numéro d'indice. Donc si cet indice n'a pas de signification, un tableau ne permet pas de savoir a quoi correspondent les différentes valeurs.

L'indice d'un élément peut être:

- *directement une valeur ex: Note [10]*
- *une variable ex: note [i]*
- *une expression entière ex: note [k+1] avec k de type entier*

Quelque soit sa forme, la valeur de l'indice doit être :

- *entière*
- *comprise entre les valeurs minimales et maximales déterminées à la déclaration du tableau.*

4. *Méthode d'écriture et de lecture d'un tableau*

✓ Méthode d'écriture :

Pour saisir (remplir) un tableau, il il existe deux façon de le faire :

SOIT JE CONNAIS LE NOMBRE DE CASE (DIMENSION DU TABLEAU) DANS CE CAS:

```
Ecrire (veuillez entrer les 12 notes dans le tableau)
Pour i allant de 1 à 12 faires
Lire (Note[i])
Fin pour
```

II. *SOIT JE NE CONNAIS PAS LE NOMBRE DE CASE* (DIMENSION DU TABLEAU) ET DANS CE CAS :

```
//Demande de saisir la dimension du tableau
Ecrire (veuillez entrer la dimension du tableau)
```

```
Lire (N)
//Ensuite on demande de saisir les valeurs du tableau
Ecrire (« veuillez entrer les 12 notes dans le tableau »)
Pour i allant de 1 à N faire
Lire (Note[i])
Fin pour
```

Fin du chapitre1

Chapitre 2 : Tableau à une dimension

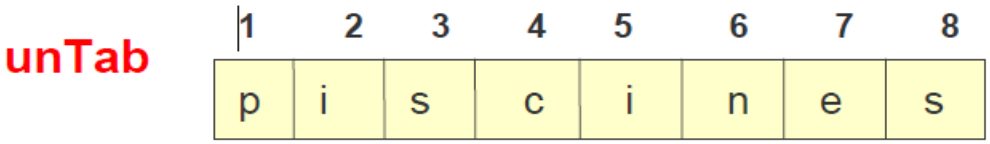
❖ Introduction :

Une seule ne suffisait-elle pas déjà amplement à notre bonheur, me demanderez-vous ? Certes, répondrai-je, mais vous allez voir qu'avec deux (et davantage encore) c'est carrément insuffisant.

Alors Prenons le cas de la modélisation du saisi de 4 notes par étudiants, comment cela va se passer ?

Avec les outils que nous avons abordés jusque là, le plus simple serait évidemment de modéliser une note et de les remplir dans un tableau :

Exemple



❖ Définition

La création d'un tableau à deux dimensions (aussi appelé matrice) permet l'accès à plus d'une valeur à partir d'un indice. Autrement dit, un numéro d'index pointe sur un autre tableau.

On représente un tableau à deux dimensions de cette manière :

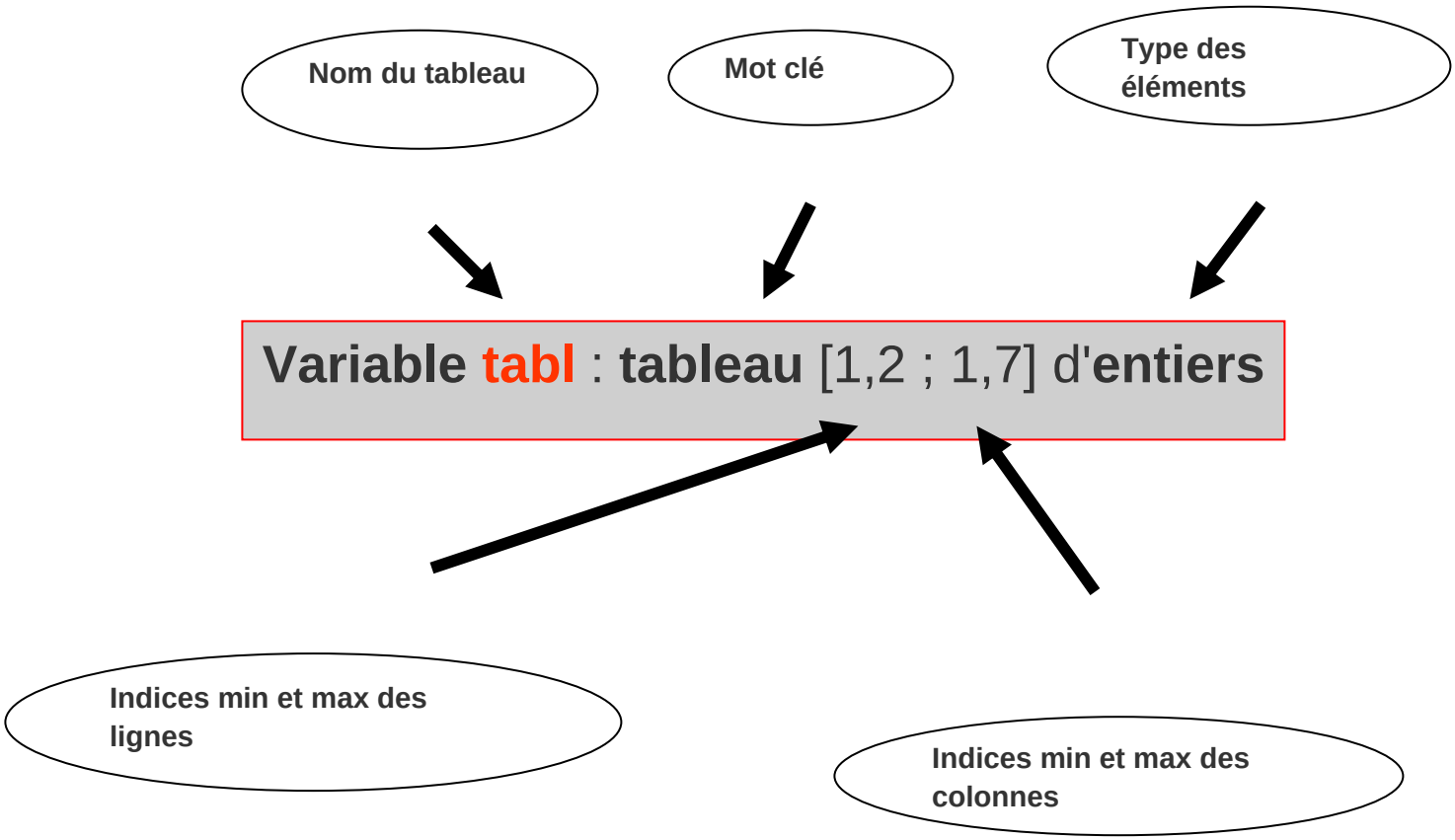
	1	2	3	4	5	6	7
1	10	3	25	14	2	1	8
2	9	20	7	12	2	4	7

❖ Déclaration :

L'informatique nous offre la possibilité de déclarer des tableaux dans lesquels les valeurs ne sont pas repérées par une seule, mais par deux coordonnées.

Pour cela nous avons trois façons de les déclarer :

1er cas :



Exemple : points : **tableau [1,2 ; 1,7] d'entiers**

2eme cas :

Variable tabl : tableau [2 ; 7] d'**entiers**

C'est le même principe mais cette fois-ci ns déclarons que le maximum

Cela veut dire : réserve moi un espace de mémoire pour 8 x 8 entiers, et quand j'aurai besoin de l'une de ces valeurs, je les repèrerai par deux indices.

3eme cas :

tab[3][2];

Remarque importante :

Il n'y a aucune différence qualitative entre un tableau à deux dimensions (i, j) et un tableau à une dimension (i * j). Tout problème qui peut être modélisé d'une manière peut aussi être modélisé de l'autre. Simplement, l'une ou l'autre de ces techniques correspond plus spontanément à tel ou tel problème, et facilite donc (ou complique, si on a choisi la mauvaise option) l'écriture et la lisibilité de l'algorithme.

❖ *Structure d'un tableau à deux dimensions*

Exemple d'un algo de tableau à deux dimensions :

```
Algorithme SaisieTableau2D
{remplit un tableau à 2 dimensions }
constantes (TailleMAX : entier) ← 100
variables nbLignes, nbColonnes, indL, indC : entiers
nombres : tableau [1, TailleMAX ; 1, TailleMAX] d'entiers
début
afficher ("Combien de lignes?") ; saisir (nbLignes)
afficher ("Combien de colonnes?") ; saisir (nbColonnes)
si nbLignes > TailleMAX ou nbColonnes > TailleMAX
alors afficher ("trop de valeurs à saisir")
sinon pour indL ← 1 à nbLignes faire
pour indC ← 1 à nbColonnes faire
afficher ("Ligne" , inL, "colonne", indC, " : ")
saisir (nombres[indL indC])
fpour
fpour
fsi
fin
```

Définition

Un algorithme de tri est, en informatique ou en mathématiques, un algorithme qui permet d'organiser une collection d'objets selon un ordre déterminé. Les objets à trier font donc partie d'un ensemble muni d'une relation d'ordre (de manière générale un ordre total). Les ordres les plus utilisés sont l'ordre numérique et l'ordre lexicographique (dictionnaire).

Par ailleurs, parmi les algorithmes listés plus bas, les tris étant stables sont : le tri à bulles, le tri par insertion et le tri à sélection. Les autres algorithmes nécessitent O(n) mémoire supplémentaire pour stocker l'ordre initial des éléments. Pour cela nous verrons trois types d'algorithmes :

Le tri par insertion (facultative)

Le principe du tri par insertion est d'insérer à la n-ième itération le n-ième élément à la bonne place. La démo ci-après détaille le fonctionnement du tri par insertion :

Voici, en pseudo-code, l'algorithme du tri par insertion :

```
tri_insertion(tableau T)
  debut
    entier longueur, i, memoire, compteur;
    booleen marqueur;
    longueur<-taille(T)
        pour i=1 à (longueur-1) faire
      memoire<-T(i) //valeur à insérer au tour i
      compteur<-(i-1)
            faire
        marqueur=faux //on n'a pas fait de décalage
        si T(compteur)>memoire alors
          T(compteur+1)<-T(compteur) //décalage des plus grandes valeurs du tableau
          compteur<-compteur-1
          marqueur=vrai //on vient de faire un décalage
        fin si
        si (compteur<0) alors //on a atteint la premiére valeur du tableau
```

```
            marqueur=faux //il n'y a plus de décalages possibles
          fin si
        tantque marqueur

      T(compteur+1)<-memoire //affectation de la valeur à insérer dans la bonne case
    fin pour
fin
```

Le tri par sélection

Le principe du tri par sélection/échange (ou *tri par extraction*) est d'aller chercher le plus petit élément du vecteur pour le mettre en premier, puis de repartir du second élément et d'aller chercher le plus petit élément du vecteur pour le mettre en second, etc...

Principe : soit un tableau de n valeurs

- Recherche du minimum dans le tableau et échange du contenu d'indice 1 et d'indice correspondant à la valeur du minimum.
- Applications du même principe sur (n - 1) valeur (n - premier) pour (n - 1), ... jusqu'à traitement de 2 cases.

Code source tri par "selection"

```
Var       valeur TABLEAU [1..n] D'Entier
          inter, j, indmin, i : réel

Début
   |  i ← 1
   |  Pour i de 1 à n Faire
   |     |  Afficher (« Donner un nombre »)
   |     |  Saisir (valeur [i])
   |  Fin pour
   |  Pour i de 1 à (n – 1) Faire
   |     |  indmin ← i
   |     |  Pour i de 1 à (n + 1) Faire
   |     |     |   Si  valeur [j] < valeur [indmin] Alors
   |     |     |     |  indmin ← j
   |     |     |   FSI
   |     |  Fin pour
   |     |   Si  indmin <> i Alors
   |     |     |  inter ← valeur [i]
   |     |     |  valeur [i] ← valeur [indmin]
   |     |     |  valeur [indmin] ← inter
   |     |  FSI
   |  Fin pour
  Fin
```

Exemple :

8	1	7	5	4
1	8	7	5	4
1	4	7	5	8
1	4	5	7	8
1	4	5	7	8

Le tri "bulle"

Le principe du tri bulle (*bubble sort*) est de comparer deux à deux les éléments e1 et e2 consécutifs d'un tableau et d'effecteur une permutation si e1 > e2. On continue de trier jusqu'à ce qu'il n'y ait plus de permutation. La démo ci-après détaille le fonctionnement du tri bulle :

Code source du tri "bulle"

```
Var     valeur TABLEAU [1..n] D Entier
        echange : booléen /*détermine si un échange de valeurs a été effectué*/
        inter : réel

Début
  | /* Saisie du tableau */
  | i ← 1
  | Pour i de 1 à n Faire
  |   | Afficher (« Donner un nombre »)
  |   | Saisir (valeur [i])
  | Fin pour
  | /* Tri du tableau */
  | Répéter
  |   | échange ← Faux
  |   | Pour i de 1 à (n – 1) Faire
  |   |   | si valeur [i] > valeur [i + 1] alors
  |   |   |   | échange ← Vrai
  |   |   |   | inter ← valeur [i]
  |   |   |   | valeur [i] ← valeur [i + 1]
  |   |   |   | valeur [i + 1] ← inter
  |   |   | FSI
  |   | Fin pour
  | Jusqu'à non échange
Fin
```

Exemple : Soit le tableau suivant à trier par ordre croissant

5	**18**	14	4	26
5	**18**	**14**	4	26
5	14	**18**	**4**	26
5	14	4	**18**	**26**
5	**14**	4	18	26
5	**14**	**4**	18	26
5	4	**14**	**18**	26
5	4	14	**18**	**26**
5	**4**	14	18	26
4	**5**	**14**	18	26
4	5	**14**	**18**	26
4	5	14	**18**	**26**
4	**5**	14	18	26
4	5	14	18	26

4	5	**14**	**18**	26
4	5	14	**18**	**26**

Il n'y a plus d'échange.

Recherche d'un élément par dichotomie dans un tableau à tri

Principe : Sur un tableau délimité par les indices borneinf et bornesup, ordonné de manière croissante, le principe de recherche est le suivant :

- Recherche de l'élément situé à l'indice médian du tableau (milieu du tableau)

- Effectuer une comparaison entre l'élément recherché et l'élément médian. Trois cas peuvent se présenter :
  - élément à chercher = élément médian : arrêt du traitement.
  - élément à chercher < élément médian : on modifie la borne supérieure de recherche dans le tableau (bornesup ← indice médian - 1) car l'élément à chercher est obligatoirement (s'il existe) dans la partie gauche du tableau.
  - élément à chercher > élément médian : on modifie la borne inférieure de recherche dans le tableau (borneinf ← indice médian + 1) car l'élément à chercher est obligatoirement (s'il existe) dans la partie droite du tableau.

Les conditions d'arrêt du traitement sont :
- égalité entre la valeur cherchée et élément médian.

- la borne supérieure est devenue inférieure à la borne inférieure car si la valeur n'existe pas, le tableau a été réduit à 0 case.

Exemple 1 : Soit le tableau

1	4	5	7	8

et l'élément à chercher = 7

1	4	5	7	8
borneinf		median		bornesup

élément à chercher > élément médian (7 > 5)
borneinf ← indice médian + 1

1	4	5	7	8
			borneinf médian	bornesup

élément à chercher = élément médian (7 = 7)

Exemple 2 : Soit le tableau

1	4	5	7	8

et l'élément à chercher = 3

élément à chercher < élément médian (3 < 5)
bornesup ← indice médian - 1

1	4	5	7	8
Borneinf		médian		bornesup

1	4	5	7	8
borneinf médian	bornesup			

élément à chercher > élément médian (3 > 1)
borneinf ← indice médian + 1

1	4	5	7	8
	borneinf bornesup médian			

élément à chercher < élément médian (3 < 4)
bornesup ← indice médian -1

1	4	5	7	8
bornesup	borneinf			

3 n'est pas dans le tableau.

Algorithme

Programme Dichotomie

```
Const   n = vous ne vous 5
Var     valeur TABLEAU [1..n] D'Entier
        borneinf, bornesup, median, element : entier

 Début
   | /* Saisie d'un élément a rechercher */
   | Afficher (« aucune valeur »)
   | Saisir (élément)
   | /* Recherche de l'élément */
   | Tantque (element <> valeur [median]) et (borneinf < = bornesup) Faire
   |    |       /* Comparaison */
   |    |   Si  (element < valeur[median]) alors
   |    |    |  bornesup ← (median – 1)
   |    | sinon
   |    |    |  borneinf ← (median + 1)
   |    |  FSI
   |    | median ← (borneinf + bornesup) DIV 2
   | FTQ
   |       /* Résultat de la recherche */
   |   Si  (element = valeur[median]) alors
   |    |  Afficher ( element, « trouvé à l'indice médian », median, « . »)
   | sinon
   |    |  Afficher ( « L'élément »,element, « n'est pas dans le tableau. »)
   |  FSI
  FIN
```

Fin du chapitre 3

Introduction

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.

Il existe deux sortes de sous-algorithmes : les procédures et les fonctions.

Les procédures

Une procédure est une série d'instructions regroupées sous un nom, qui permet d'effectuer des actions par un simple appel de la procédure dans un algorithme ou dans un autre sous-algorithme.
Une procédure renvoie plusieurs valeurs (par une) ou aucune valeur.

Déclaration d'une procédure

Syntaxe :

```
Procédure nom_proc(liste de paramètres)
Variables identificateurs : type
Début
        Instruction(s)
FinRoc
```

Après le nom de la procédure, il faut donner la liste des paramètres (s'il y en a) avec leur type respectif. Ces paramètres sont appelés paramètres formels. Leur valeur n'est pas connue lors de la création de la procédure.

Exemple :

Ecrire une procédure qui affiche à l'écran une ligne de 15 étoiles puis passe à la ligne suivante.

Solution :

```
Procédure Etoile()
Variables i : entier
Début
        Pour i Allant de 1 à 15 faire
            Afficher("*")
        FinPour
        //\n : retour à la ligne
        Afficher("\n")
FinProc
```

L'appel d'une procédure

Pour déclencher l'exécution d'une procédure dans un programme, il suffit de l'appeler.
L'appel de procédure s'écrit en mettant le nom de la procédure, puis la liste des paramètres, séparés par des virgules.
A l'appel d'une procédure, le programme interrompt son déroulement normal, exécute les instructions de la procédure, puis retourne au programme appelant et exécute l'instruction suivante.

Syntaxe :

```
Nom_proc(liste de paramètres)
```

Les paramètres utilisées lors de l'appel d'une procédure sont appelés paramètres effectifs. Ces paramètres donneront leurs valeurs aux paramètres formels.

Exemple :
En utilisant la procédure Etoiles déclarée dans l'exemple précédent, écrire un algorithme permettant de dessiner un carré d'étoiles de 15 lignes et de 15 colonnes.

Solution :

```
Algorithme carré_étoiles
Variables j : entier

//Déclaration de la procédure Etoiles()
Procédure Etoile()
Variables i : entier
Début
   Pour i Allant de 1 à 15 Faire
        Afficher("*")
   FinPour
   Afficher("/n")
FinProc

//Algorithme principal (Partie principale)
Début
     Pour j Allant de 1 à 15 Faire
            //Appel de la procédure Etoiles
            Etoile()
     FinPour
Fin
```

Remarque 1 :

Pour exécuter un algorithme qui contient des procédures et des fonctions, il faut commencer l'exécution à partir de la partie principale (algorithme principal)

Remarque 2 :

Lors de la conception d'un algorithme deux aspects apparaissent :

- La définition (déclaration) de la procédure ou fonction.
- L'appel de la procédure ou fonction au sein de l'algorithme principal.

Passage de paramètres

Les échanges d'informations entre une procédures et le sous 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.

Exemple :

Soit l'algorithme suivant :

```
Algorithme Passage_par_valeur
Variables N : entier

//Déclaration de la procédure P1
Procédure P1(A : entier)
Début
    A ← A * 2
    Afficher(A)
FinProc

//Algorithme principal
Début
   N ← 5
   P1(N)
   Afficher(N)
Fin
```

Cet algorithme définit une procédure P1 pour laquelle on utilise le passage de paramètres par valeur.
Lors de l'appel de la procédure, la valeur du paramètre effectif N est recopiée dans le paramètres formel A. La procédure effectue alors le traitement et affiche la valeur de la variable A, dans ce cas 10.
Après l'appel de la procédure, l'algorithme affiche la valeur de la variable N dans ce cas 5.
La procédure ne modifie pas le paramètre qui est passé par valeur.

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***.

Exemple :
Reprenons l'exemple précédent :

```
Algorithme Passage_par_référence
Variables N : entier

//Déclaration de la procédure P1
Procédure P1 (Var A : entier)
Début
    A ← A * 2
    Afficher(A)
FinProc

//Algorithme Principal
Début
      N ← 5
      P1(N)
      Afficher(N)
Fin
```

A l'exécution de la procédure, l'instruction ***Afficher(A)*** permet d'afficher à l'écran 10. Au retour dans l'algorithme principal, l'instruction ***Afficher(N)*** affiche également 10.
Dans cet algorithme le paramètre passé correspond à la référence (adresse) de la variable N. Elle est donc modifiée par l'instruction :
A ← A *2

Remarque :
Lorsqu'il y a plusieurs paramètres dans la définition d'une procédure, il faut absolument qu'il y en ait le même nombre à l'appel et que l'ordre soit respecté.

Les 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.

Déclaration d'une fonction

Syntaxe :

```
Fonction nom_Fonct (liste de paramètres) : type
Variables identificateur : type
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.

Note :
Les paramètres sont facultatifs, mais s'il n'y pas de paramètres, les parenthèses doivent rester présentes.

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

Solution :

```
//Déclaration de la fonction Max
Fonction Max(X: réel, Y:réel) : réel
Début
      Si  X > Y Alors
            Retouner X
      Sinon
            Retouner Y
      FinSi
FinFonction
```

L'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

```
Nom_Fonc(list de paramètres)
```

Exemple :

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

Solution :

```
Algorithme Appel_fonction_Max
Variables A, B, M : réel

//Déclaration de la fonction Max
Fonction Max(X: réel, Y: réel) : réel
Début
        Si X > Y Alors
             Retourner X
```

```
        Sinon
             Retourner Y
        FinSi
FinFonction

//Algorithme principal
Début
        Afficher("Donnez la valeur de A :")
        Saisir(A)
        Afficher("Donnez la valeur de B :")
        Saisir(B)
        //Appel de la fonction Max
        M ← Max(A,B)
        Afficher("Le plus grand de ces deux nombres est : ", M)
Fin
```

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 Portée
Variables X, Y : Entier
Procédure P1()
Variables A : Entier
Début
....
FinProc
//Algorithme principal
Début
....
Fin
```

X et Y sont des variables globales visibles dans tout l'algorithme.

A est une variables locale visibles uniquement à l'intérieur de la procédure

Remarque :
Les variables globales sont à éviter pour la maintenance des programmes.

Bibliographie

Web :

Livre :

Chapitre 5 : Les pointeurs

Travial de recherche !!!