# Introduction à l'algorithmique et à la programmation

C. BENSARI

# Plan

- Notions sur la programmation
- Découverte de l'algorithmique
- Eléments de base d'un algorithme
- Les structures conditionnelles
- Les structures répétitives
- Les tableaux
- Les procédures et fonctions
- Les chaines de caractères
- La complexité d'un algorithme

# Notions sur la programmation

- Un programme informatique est une suite d'instructions qui peuvent être interprétées par une machine après des étapes de transformation du programme depuis le langage de haut niveau vers le langage de bas niveau
- Un programme à le cycle de vie suivant:
  - Conception et modélisation
  - Programmation
  - Compilation
  - Exécution

# Conception et modélisation

- Pour mieux élaborer un programme, il est nécessaire d'essayer de schématiser le déroulement de ce dernier
- Avant de définir un algorithme, il faut penser à élaborer un organigramme/logigramme en amont
- Un organigramme est normalisé et est constitué de formes qui ont une signification et qui suivent un ordre précis (le sens des flèches)
- Un programme commence par un début et se termine par une fin. La représentation de ces deux étapes en organigramme est la suivante :

# Découverte de l'algorithmique

- Description d'un traitement automatisé de données
- Le traitement doit être réalisé sur un ordinateur
- Un algorithme doit être traduit dans un langage de programmation (Pascal, Java, PHP, …)
- La mise en place d'un algorithme doit être précédé par une phase d'analyse du problème à résoudre
- La solution décrite par l'algorithme doit être totalement indépendante du langage de programmation qui sera utilisé
- Conclusion: Un algorithme est une suite d'opérations élémentaires devant être exécutées dans un ordre donné, pour accomplir une tâche donnée

# Exemple d'un algorithme dans le monde réel

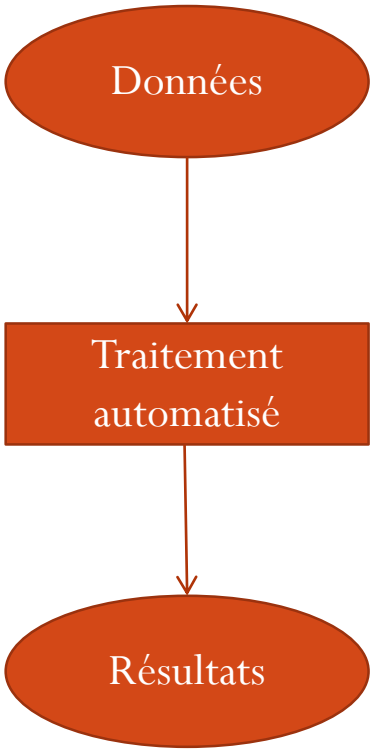
- Préparation du riz :
  1- Remplir une casserole d'eau
  2- Mettre une pincée de sel
  3- Mettre la casserole sur le feu
  4- Attendre jusqu'à l'ébullition d'eau
  5- Mettre le riz dans la casserole
  6- Laisser cuire 10 à 15 minutes
  7- Égoutter le riz

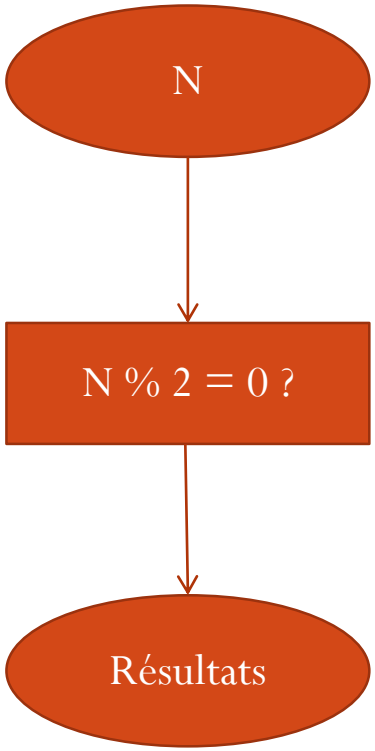
# Exemple d'un problème à résoudre

- Un nombre entier N est-il impair ?
- Solution:
  - Analyse: Un nombre N est impair si le reste de la division N par 2 est égale à 1
  - Solution algorithmique
    1. Calculer le reste R de la division de N par 2
    2. Si R est égale à 1 alors N est impair
    3. Sinon N n'est pas impair

# Principe général

- Un traitement automatisé consiste à faire des opérations sur des données (informations) fournies en entrée et afficher les données résultantes du traitement en sortie



# Eléments de base d'un algorithme

- Variables
  - Les données et résultats  sont des grandeurs qui sont susceptibles de varier => **variables**
  - Les noms de variables doivent commencer par une lettre, un underscore « _ » ou un dollar « $ »
  - Les conventions de nommages préconisent d'utiliser la forme camelCase (première lettre en minuscule et  chaque nouveau mot du nom doit commencer par une majuscule : maMagnifiqueVariable)

    **Exemples**:

    **variables**

    rayonCercle, surfaceCercle : réels

- Types
  - Les numériques: les nombres entiers et réels (ex. 3, -7, 1,25 E3)
  - Les chaines de caractères  (ex. " bonjour")
  - Le type booléen (deux valeurs possible: vrai ou faux)

# Eléments de base d'un algorithme

- Les constantes :
  - Certains types de données (informations) ne sont pas amenées à changer pendant tout le programme => **constantes**
  - Les conventions de nommage préconisent d'utiliser des majuscules et des underscores pour le nom de constantes

**Exemples**:

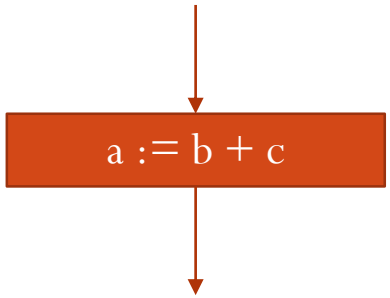
**constantes**

PI = 3,14159

NB_JOURS_DE_SEMAINE = 7

# Eléments de base d'un algorithme

- Expressions et affectations
  - Un algorithme est une suite d'opérations (**instructions**). De manière générale il s'agit d'évaluer une **expression** comportant :
    - Des variables et des constantes
    - Des opérations : +, -, *, /, >, <, >=, <=, =, % (%: reste de la division)
    - Des fonctions plus complexes
  - Les résultats de l'évaluation de ces expressions sont généralement « rangées » dans des variables. On dit qu'il s'agit d'une affectation du résultat d'une expression à une variable.
  - Pour l'affectation on utilise le symbole «:=» ou « ← »
  - Les expressions et les affectations sont représentées dans un organigramme avec le symbole rectangle :



# Eléments de base d'un algorithme

- Exemple d'une affectation simple:

**variables**

a, b, c : **entier**;

a:= 5; % affecter le nombre 5 à la variable **a** %

b:=10;

- Exemple d'une affectation avec une expression :

**variables**

rayon, surface : réels;

**constantes**

PI := 3.14;

surface := PI * rayon * rayon; % affectation du résultat de l'expression à la variable surface %

# Eléments de base d'un algorithme

- Attention aux règles de priorité :

  1- Les fonctions mathématiques en premier

  2- La multiplication et la division

  3- L'addition et la soustraction

  Exemple :

  c := sqr(a) + 5 * b; % sqr est une fonction mathématique calculant la racine carré d'un nombre %

  Pour **a = 4** et **b = 3**, **c** sera égale à **17** et non pas **21**

# Eléments de base d'un algorithme

- Opérations d'entrée/sortie :

  Dans n'importe quel programme, un échange a lieu entre l'utilisateur et la machine :

  La saisie par clavier → données d'entrée

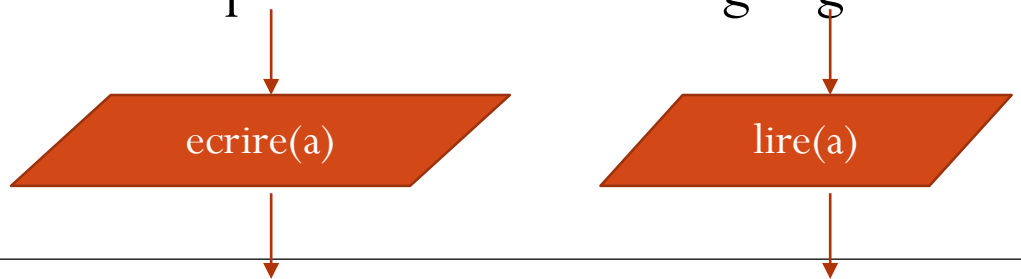
  Affichage sur écran → données en sortie

  **Exemple** : calcul de la surface d'un cercle

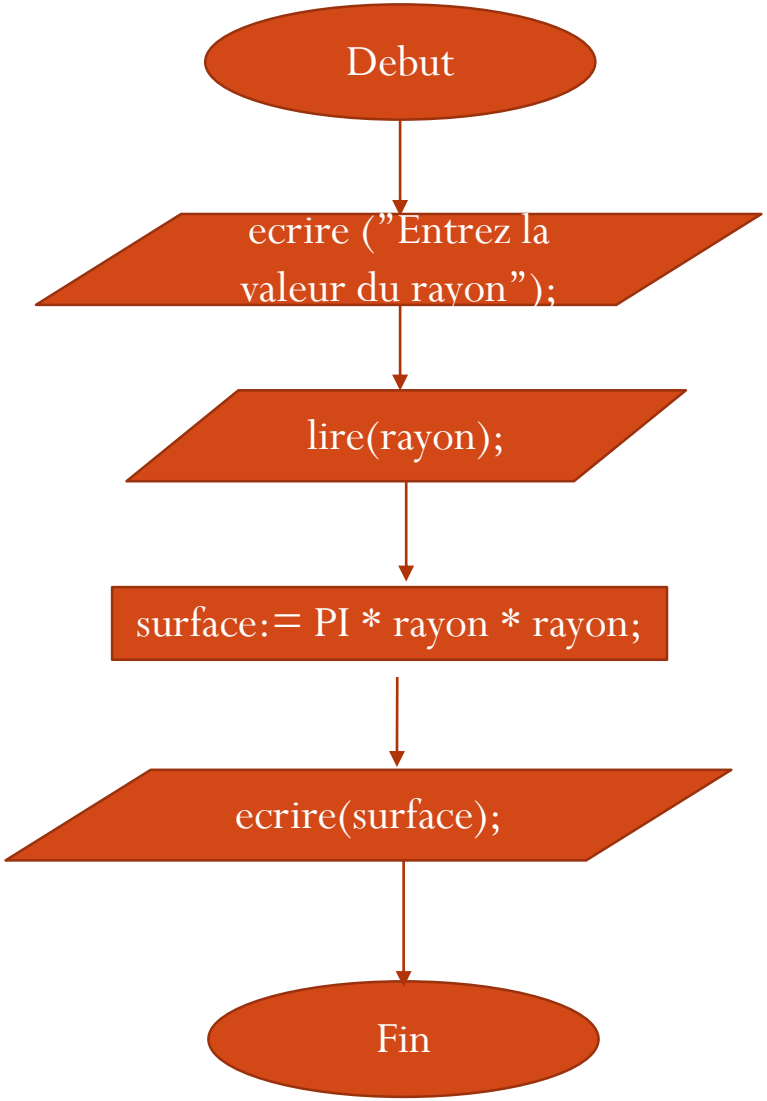
  - Saisir la valeur du rayon du cercle (donnée d'entrée ou lecture)
  - Affecter à une variable **surface** le résultat de l'expression $\pi*(\text{rayon})^2$
  - Afficher le résultat (donnée de sortie ou écriture)

  → Cette suite d'opérations est appelé « **Séquence d'instructions** »

- Les opérations d'entré/sortie sont représentées dans un organigramme avec un rectangle incliné



# Organigramme de calcule de la surface d'un cercle



# Syntaxe générale d'un algorithme

```
algorithme calcul_surface_cercle
constantes
   PI := 3.14;
variables
   rayon, surface : reels;
Debut
   ecrire("Entrez la valeur du rayon");
   lire(rayon);
   surface := PI * rayon * rayon;
   ecrire(surface);
fin
```

# Les structures conditionnelles

- Une structure conditionnelle permet de faire un traitement selon une ou plusieurs conditions
- Structure conditionnelle simple
  - La structure conditionnelle simple se présente sous la forme suivante :

```
Si (condition) alors début
                    /*Séquence d'instructions*/
                    fin
FinSi
```

**Exemple :**

```
variables
        a : booléen;
        b : entier;
lire(b);
a :=  b > 5;
si (a) alors    /* fonctionnera aussi avec : si (b>5) */
        ecrire("la valeur de a est supérieure à 5");
finsi
```

# Les structures conditionnelles

- La structure conditionnelle composée se présente sous la forme suivante :

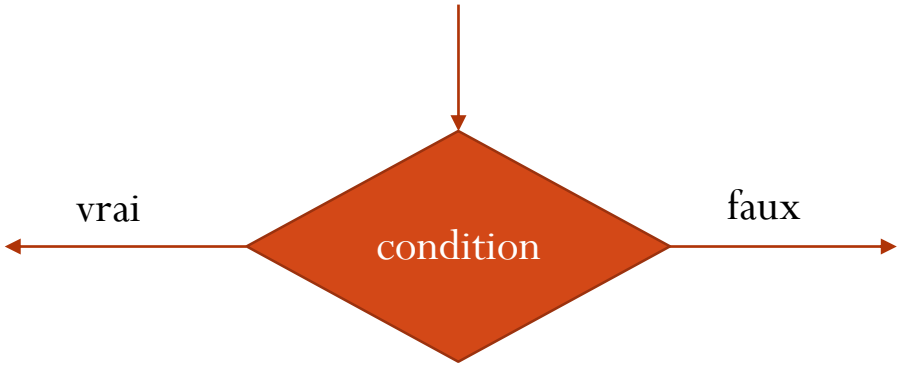
```
Si (condition) alors début
                    /*Séquence d'instructions*/
                    fin
                    sinon début
                    /*Séquence d'instructions*/
                    fin
FinSi
```

**Exemple :**

```
variables
        a : booléen;
        b : entier;
lire(b);
si (b > 5) alors   ecrire("la valeur da est supérieure à 5");
          sinon  ecrire("la valeur da est inférieure ou égale à 5");
finsi
```

# Les structures conditionnelles

- Dans un organigramme, la structure conditionnelle est représentée avec la forme suivante :



# Les structures conditionnelles

- **Structure conditionnelle multiple**
  - Dans une structure conditionnelle multiple on peut comparer notre objet (variable ou expression) avec toute une série de valeurs et d'exécuter un ensemble d'instructions en fonction de la valeur effective
  - Une séquence par défaut peut être prévue dans le cas où l'objet n'est égale à aucune des valeurs énumérées
  - Syntaxe d'une structure conditionnelle multiple:

```
suivant [variable_ou_expression] faire
        valeur_1 : /* instructions*/
        valeur_2 : /* instructions*/
        valeur_3 : /* instructions*/
        sinon /* instructions par défaut */
fin_suivant
```

# Les structures conditionnelles

- Exemple d'utilisation d'une structure conditionnelle multiple :

```
…..
lire(jour);
suivant jour faire
        ”SAMEDI” : ecrire(” C’est le weekend !”);
        ”DIMANCHE” : ecrire(” C’est le weekend !”);
         sinon ecrire(” Ce n’est pas le weekend !”);
fin_suivant
……
```

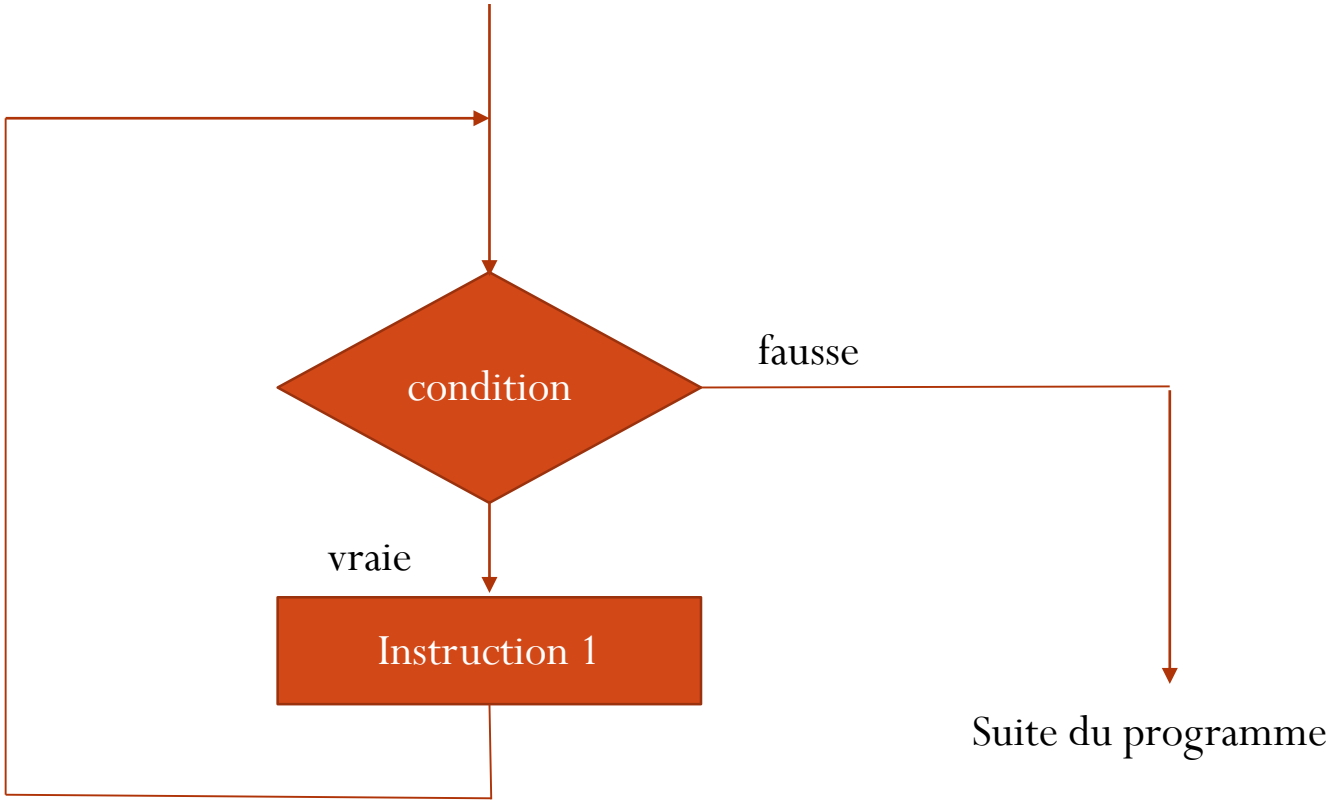
# Les structures répétitives (boucles)

- Les structures répétitives sont utilisées lorsque une ou plusieurs instructions doivent être exécutées plusieurs fois
- Trois éléments nécessaires dans une structure répétitive:
  - L'initialisation : précise le point de départ de la boucle
  - Condition d'arrêt de la boucle : vérifie si la boucle doit continuer à répéter le traitement
  - L'incrémentation: elle change la valeur de la variable utilisée dans la condition d'arrêt
- Trois types de boucles utilisables en programmation :
  - La boucle **tant que** … **faire** :

```
tant que (condition) faire
        /* Instructions dont l'incrémentation*/
fin_tant_que
```

# Les structures répétitives (boucles)

- La représentation dans un organigramme d'une structure répétitive est faite de la manière suivante :



# Les structures répétitives (boucles)

- Exemple d'utilisation de la boucle **tant que … faire**
  - Algorithme permettant d'afficher les nombre entiers de 1 à 10

```
algorithme tant_que
variables
        indice :entier;
début
        indice := 1;        /* initialisation */
        tant que (indice <= 10) faire       /* condition d'arrêt */
                écrire(indice);
                indice := indice + 1;              /* incrémentation */
        fin_tant_que
fin
```

# Les structures répétitives (boucles)

- La structure **répéter … jusqu'à**
  - Syntaxe de la boucle **répéter … jusqu'à** :

```
répéter
        /* instructions */
Jusqu'à (condition)
```

Exemple :

```
i:= 20;
répéter
        ecrire(i);
        i--;
jusqu'à (i = 10)
```

- A la différence de la structure **tant que**, la structure **répéter** positionne la condition d'arrêt à la fin de la boucle et non pas au début. Dans ce cas, les instructions à l'intérieur du **bloc** de la boucle sont exécutées **au moins une fois**

# Les structures répétitives (boucles)

- La structure **pour … faire**
  - Cette structure permet de répéter une séquence un nombre connu de fois
  - Syntaxe de la boucle **pour … faire** :

```
pour (initialisation;  test_d'arrêt;  incrémentation) faire
        /* instructions */
fin_pour;
```

Exemple :

```
pour (i:=1; i <= 10; i++) faire
        écrire(i);
fin_pour;
```

# Les structures répétitives (boucles)

- Les boucles imbriquées
  - Il est possible d'imbriquer des boucles les une dans les autres. Une boucle **tant que** peut contenir une autre boucle **tant que** comme elle peut contenir aussi tout autre type de boucles.
  - Exemple (affichera : 4, 8, 5, 10, 6, 12, 7 et 8)

```
    pour (i:=1; i<=
5; i++) faire
            somme := 0;
            tant que (somme < 7) faire
                somme : = somme + i +3;
                ecrire(somme);
            fin_tant_que;
    fin_pour;
```

# Exercices

**Exercice 1 :**

Ecrire un algorithme (organigramme) permettant d'afficher la table de multiplication (de 1 à 10) d'un nombre saisi par l'utilisateur

**Exercice 2 :**

Ecrire un algorithme (organigramme) permettant à un utilisateur de faire des calculs (**autant qu'il veut**) entre deux nombres (addition, multiplication, division)

**Exercice 3 :**

Ecrire un algorithme qui permet de calculer le factoriel d'un nombre

**Exercice 4 :**

Ecrire un algorithme (organigramme) permettant d'afficher toutes les tables de multiplication de 1 à 10

# Exercices

**Exercice 5**

Ecrire un algorithme qui permet de saisir une série de nombres entiers positifs puis qui propose indéfiniment à l'utilisateur, par l'intermédiaire d'une sorte de menu à choix multiple, d'afficher la valeur minimale, la valeur maximale, la somme ou la moyenne des nombres entrés, ou encore de quitter le programme.

# Les tableaux

- Un ensemble de données du même type
- Chaque valeur du tableau possède un indice (position dans le tableau). La première valeur du tableau se trouve à l'indice 1.
- Exemple de déclaration d'un tableau:

notes(1, 10) : tableau de réels

notes: nom du tableau (variable)

1: premier indice du tableau

10: dernier indice du tableau


- Dans un algorithme, le nom d'un tableau n'est jamais écris tout seul, il est toujours suivi d'un indice entouré de parenthèses :
  - ecrire(notes(2)); → affichera 10
  - notes(4) := 15; → affectation de la valeur 15 à la case 4 du tableau notes

# Création d'un tableau

**Algorithme** saisir_tableau

**Variables** tableau(1 : 10) : **tableau d'entiers**

i: **entier**

**Debut**

**Pour** i:=1 **à** 10 **faire**

**écrire**('Entrez un nombre : ');

**lire**(tableau(i)); -- > permet de remplir la case i

**ecrire**(tableau(i)); -->permet d'afficher le contenue de la case i

**Fin_pour**

**fin**

# Affichage d'un tableau

- Pour afficher un tableau, il faut parcourir le tableau élément par élément en faisant varier l'indice et afficher le contenu de chaque cellule

**Début**

**Pour** i:=1 **à** 10 **faire**

**écrire**(nombres(i)); /*permet d'afficher le contenu de la case i */

**Fin_pour**

**fin**

# Rechercher le plus petit élément dans un tableau

- Pour déterminer le plus petit élément d'un tableau il faut suivre le principe suivant :
  - Supposer que la première case du tableau contient le minimum
  - Comparer le contenu de la deuxième case avec le minimum. si celui-ci est inférieur, il devient le minimum
  - Renouveler la même opération pour le reste des cases

# Trier un tableau d'entiers

- Un tableau peut être trié par ordre croissant/décroissant si le contenu d'une case « i » est inférieur/supérieur au contenu de la case « i+1 »
- L'une des manières pour trier un tableau se fait selon le principe suivant :
  - Rechercher le plus petit/grand élément du tableau et l'amener à la première case
  - Répéter l'opération pour amener le minimum/maximum des cases restantes à la deuxième case et ainsi de suite
- Pour trier un tableau il faut passer par deux boucles imbriquées l'une dans l'autre
- Attention aux pertes d'informations lors des déplacements des éléments !

# Tableau à deux dimension

- En développement il est souvent nécessaire d'utiliser plusieurs tableaux pour élaborer un programme automatisé
- Si le besoin est d'utiliser deux tableaux (ou plus) qui subissent le même traitement, il faut penser à utiliser un tableau à plusieurs dimensions
- En algorithmique, la déclaration d'une variable de type tableau à deux dimensions est la suivante

**variables**

notes (1:3, 1:5) : tableau d'entiers

1:3 = > tableau de trois lignes

1:5 => tableau de 5 colonnes

L'accès à une case du tableau se fait de la même manière qu'un tableau à une dimension en rajoutant un deuxième indice qui est celui de la colonne :

notes(1, 3) => élément à la ligne 1 et colonne 3

# Exercices sur les tableaux

**Exercice 1**:

Écrire un algorithme permettant de saisir un tableau d'entiers et de calculer la somme de tous les éléments d'un tableau

**Exercice 2**:

Écrire un algorithme permettant de saisir un tableau d'entiers et de calculer la moyenne de tous les éléments d'un tableau

**Exercice 3**:

Écrire un algorithme permettant de saisir un tableau d'entiers et de chercher le plus petit élément

**Exercice 4**:

Calculer la somme de 2 tableaux de même taille et stocker le résultat dans un troisième tableau

**Exercice 5**:

Trier un tableau avec un tri croissant (du plus petit au plus grand)

# Exercice tableau à deux dimension

**Exercice** :

Écrire un algorithme permettant de saisir un tableau de notes (3) pour deux étudiants (utiliser un n° étudiant au lieu d'un nom), d'afficher le tableau et d'afficher la moyenne de chaque étudiant

# Fonctions

- Une fonction est une suite d'instructions destinée à faire une tâche précise
- Une fonction doit être définie quelque part dans un programme et doit retourner un résultat
- Une fonction peut être réutilisée n'importe où dans un programme en faisant référence à son nom
- Certains langages de programmation définissent par défaut, des fonctions qui peuvent être directement utilisées dans les programmes
- Par exemple, le langage Pascal définit la fonction **EXP** qui permet de calculer l'exponentiel d'un nombre :

y:= **EXP**(x)

# Les fonctions

- En algorithmique, il est possible de définir une fonction dans le début de l'algorithme et de l'appeler n'importe où dans le même algorithme
- Déclaration d'une fonction:

**fonction** calculerSomme(x: **réel**, y:**réel**, z:**réel**) : **réel**

**début**

somme : = x+ y + z;

**retourner** somme;

**fin**

calculerSomme : nom de la fonction

x, y, z : arguments de la fonction

**réel** : le type de la valeur retourner par la fonction

**retourner** : instruction permettant à la fonction d'envoyer la valeur à l'appelant

- Exemple d'appel à une fonction:

```
a := 3; b := 5; c := 2.5;
resultat := calculerSomme(a, b, c);    /* resultat sera égale à 10.5 */
```

# Procédures

- Une procédure est une fonction qui ne renvoie pas de résultat
- La déclaration d'une procédure est la même que celle d'une fonction sauf que, elle ne renvoie rien et donc n'a pas de type de retour :

```
procédure afficherBonjour(nom: chaine_de_caractères)
        début
                écrire("Bonjour ", nom, "!")
        fin
```

# Portée des variables

## Variables globales

- Dans un programme, les variables peuvent ne pas être accessibles dans une partie du programme. Ceci dépend de la **porté (scope)** de ces variables
- Les variables globales sont les variables déclarées au début d'un algorithme. Elles sont accessibles partout dans l'algorithme même à l'intérieur des fonctions et procédures
- Il faut faire très attention lors de l'utilisation des variables globales dans les procédures et fonctions car si elles sont modifiées (utilisation d'une affectation) à l'intérieur d'une fonction, elles le seront pour tout le reste du programme

# Portée des variables

## Variables locales

- Une variable est dite locale si elle est déclarée dans une partie séparée de l'algorithme/programme. Exemple une fonction ou une procédure :

```
fonction maFonction() : entier
        variables
                a       /* a est un variable locale */
        début
        .....
        Fin
```

- Une variable locale n'est accessible que là où elle est déclarée. Si elle est déclarée dans une procédure/fonction alors elle n'est accessible qu'à l'intérieur de cette procédure/fonction

# Le passage des paramètres dans les procédures et fonctions

- Les procédures et fonctions peuvent utiliser des arguments (paramètres) pour effectuer des traitements précis. Pour cela, lors de l'appel d'une fonction, il est nécessaire de  le lui faire «passer» ces arguments
- Il existe deux types de passage de paramètres :
  - Passage par valeur : Dans ce passage, la fonction utilise une copie de la valeur dans son traitement et donc ne modifiera pas la valeur originelle
  - Passage par référence/adresse : Dans ce passage, la fonction/procédure utilise l'adresse mémoire du contenu pour son traitement et donc la valeur originelle sera modifiée
    - Pour ce type de passage, il faut utiliser le mot clé « **Var** » juste avant le nom du paramètre

# Le passage des paramètres dans les procédures et fonctions

- Exemple de passage par valeur

```
Algorithme Passage_par_valeur
Variables N : entier

Procédure P1(X : entier)
        Début
                X := X + 5
                écrire(X)  /* Affichera 10 */
        Fin

Début
        N := 5
        P1(N)
        écrire(N)   /* Affichera 5 */
Fin
```

# Le passage des paramètres dans les procédures et fonctions

- Exemple de passage par valeur

```
Algorithme Passage_par_référence
Variables N : entier

Procédure P1(var A : entier)
        Début
                A := A * 2
                écrire(A)  /* Affichera 10 */
        Fin

Début
        N := 5
        P1(N)
        écrire(N) /* Affichera 10 */
Fin
```

# Les fonctions récursives

- Une fonction/procédure est dite récursive si elle fait appel à elle même durant son traitement :

```
fonction maFonctionRecursive(paramètres…)
        début
                ….
                maFonctionRecursive(paramètres…)
                ….
        fin
```

- Deux conditions nécessaires pour la mise en place d'une fonction récursive :
  - Condition d'arrêt : le code de la fonction doit indiquer quand la fonction ne fera plus l'appel à elle-même
  - Il faut s'assurer que chaque appel de la fonction approchera la fonction vers la fin de récursivité (équivalent à l'incrémentation dans les boucles)

# Exercices

**Exercice 1**:

Faire les exercices 1 et 2 des tableaux en utilisant des fonctions et procédures

**Exercice 2**:

Améliorer l'algorithme de permutation en utilisant des procédures et fonctions

**Exercice 3**:

Améliorer l'algorithme du calculateur en utilisant des procédures et fonctions

**Exercice 4 :**

Améliorer l'exercice 5 (diaporama 29) en utilisant des procédures et fonctions

# Exercices

**Exercice 5 :**

Réécrire l'algorithme du factoriel en utilisant une fonction récursive :

$F(n) = n * F(n-1)$

$F(1) = 1$

$F(0) = 1$

**Exercice 6 :**

Ecrire un algorithme permettant le calcul de la suite de Fibonacci :

$F(n) = F(n-1) + F(n-2)$ pour $n > 0$

$F(1) = 1$

$F(0) = 0$

# Le type caractère

- Il existe un type **caractère** qui correspond à un symbole qui peut être alphanumérique ou tout autre symbole. Exemple: alphabet, chiffre, symboles, ..
- Exemple d'utilisation des caractères :

```
variables
        monnaie  : caractère
        prix : réel
Début
        monnaie := '€'
        prix := 50
        écrire("Le prix est à ", prix, monnaie)
        /* Affichera: Le prix est à 50€ */
        prix:=50monnaie /* Ceci n'est pas autorisé ! */
Fin
```

- Les caractères sont codés au moyen d'un code universel (ASCII: American Standard Code for Information Interchange)
- Par exemple, les lettres majuscules A à Z sont codés dans l'ordre, par les codes 65 à 90. Ces codes sont utilisés lors de la comparaison entre caractères

# Les chaînes de caractères

- Une chaîne de caractères est un tableau de **caractères**
- Chaque caractère d'une chaîne de caractères est accessible comme est accessible un élément d'un tableau

**variables**

chaine(1:20) : **tableau de caractères**

y : **caractère**

….

**lire**(chaine);

y:= chaine(2) /* y sera égale au deuxième caractère de chaine */

….

- La fonction **length()** appliquée sur une chaine de caractères retourne le nombre de caractères présents dans la chaîne :

chaine := "Bonjour"

longueur := chaine.length() **/* longeur sera égale à 7 */**

# Les chaînes de caractères

- La concaténation de chaînes de caractères se fait avec le symbole « & » :

```
chaine1 := ”Bonjour”
chaine2 := ”David”
chaine := chaine1 & chaine2    /* chaine sera égale à BonjourDavid */
```

- La fonction **upcase**(caractère) renvoi le caractère en majuscule

```
chaine1 := ”Bonjour”
maj := upcase(chaine(7))       /* maj sera égale à R */
```

# Les chaînes de caractères

- La chaine X est inférieur à la chaîne Y si elle la précède dans l'ordre alphabétique. La comparaison est faite caractère par caractère :

```
chaine1 := "Legrand"
chaine2 := "Dupond"
reponse := chaine1 > chaine2     /* reponse sera égale à vrai */
chaine3 := "Dugary"
reponse := chaine3 > chaine2     /* reponse sera égale à faux */
```

# Exercices

**Exercice 1**

Ecrire un algorithme qui permet de saisir une chaîne de 20 caractères maximum et un caractère, puis affiche le nombre de fois que le caractère apparaît dans la chaîne

**Exercice 2**

Ecrire un algorithme qui permet de saisir  un tableau de chaines de caractères et le trier par ordre décroissant

**Exercice 1**

Ecrire un algorithme qui permet de saisir une chaîne de 20 caractères maximum et un deuxième chaine, puis affiche le nombre de fois que la deuxième chaine apparaît dans la chaîne initiale

# Les types définis par l'utilsateur

- Les types simples (), le type tableau et le type chaîne de caractères sont des types prédéfinis par le langage
- Tous les langages de programmation donnent la possibilité aux utilisateurs (programmeurs) de définir de nouveaux types de données qui seront adaptés et permettant d'imaginer un traitement performant et souple
- Un type de données peut être définit sous différentes structures (énumération, intervalle, ensemble, enregistrement, ..)

# Le type énumération

- Un type énuméré possède un nombre fixe de avaleurs prédéfinies. On ne peut lui affecter aucune autre valeur que celles prévues dans l'énumération
- Exemple :

```
// Définition du type
Type Couleur = {VERT, BLEU, BLANC };
// Déclaration de variable
variables couleurMur, couleurPlafond : Couleur;
.....
couleurMur :=VERT
couleurMur := ROUGE   /* ERREUR*/
```

# Le type intervalle

- Une variable de type intervalle ne peut prendre que des valeurs appartenant à l'intervalle
- Exemple :

```
// Définition du type
Type Majuscules = 'A' .. 'Z';
// Déclaration de variable
variables lettre1, lettre2 : Majuscules;
```

# Le type ensemble

- Une variable de type ensemble peut prendre un nombre limitée de valeurs de même type
- Exemple :

```
// Définition du type
Type      Couleur = {VERT, BLANC, ROUGE};
          Dessin = set of Couleur
// Déclaration de variable
var a, b, result :  Dessin;
      result2 : booleen;
a := [VERT, BLANC];
b := [ROUGE, BLANC];
result := a + b;     /* Réunion: [VERT, BLANC, ROUGE] */
result := a * b;     /* Intersection: [BLANC] */
result := a – b;     /* Différence : [VERT] */
result2 := a = b;      /* Egalité False */
result2 := a IN b;     /*  Contenance False */
```

# Le type enregistrement

- Une variable de type enregistrement permet de regrouper ensemble, des variables de types différents appelées **variables membres**
- Pou accéder à l'une des variables membres d'une variable de type enregistrement, il suffit de placer un **point** entre le nom de la variable de type enregistrement et sa **variable membre**
- Exemple :

```
// Définition du type (pas de ; à la première ligne !)
Type     Personne  = enregistrement
                        nom : String;
                        prenom : String;
                fin_ enregistrement;
// Déclaration de variable de type Personne
variables employe :  Personne;
lire(employe.nom);
ecrire(employe.nom);
```

# Le type tableau d'enregistrements

- Il est possible de déclarer un tableau dont les éléments sont de type enregistrement
- Exemple :

```
// Définition du type enregistrement
Type    Personne  = enregistrement
                    nom : String;
                    prenom : String;
              fin_ enregistrement;
// Déclaration de variable de type Personne
var employes(1..10)   : array of Personne;
.....
ecrire(employes(1).nom);
```

# Exercices

**Exercice 1**

Ecrire un algorithme permettant de chercher le nombre de voyelles dans un mot (saisi par l'utilisateur). Utiliser les ensembles

**Exercice 2**

On souhaite construire une structure « etatCivil » permettant d'enregistrer le nom d'une personne, la ville où elle réside et son année de naissance. Construire cette structure ainsi qu'un tableau permettant de stocker 10 structures de ce type