

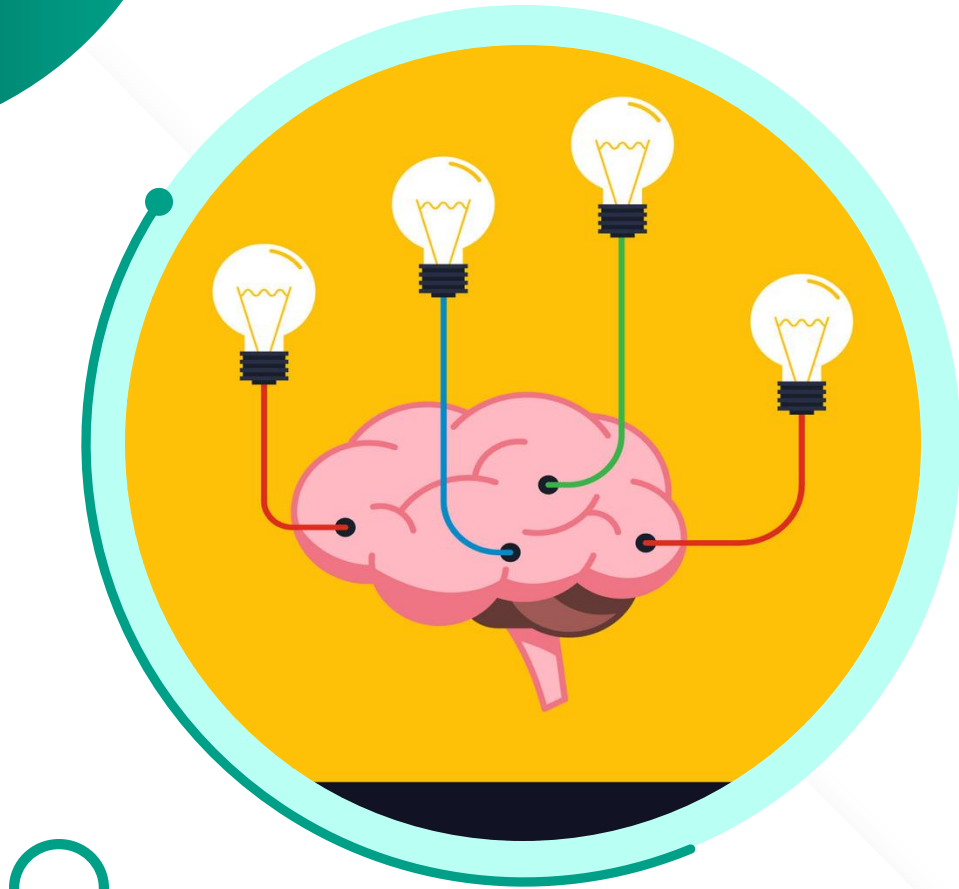
```
    cin = true;

    while (again) {
        iN = -1;
        again = false;
        getline(cin, sInput);
        system("cls");
        stringstream(sInput) >> dblTemp;
        iLength = sInput.length();
        if (iLength < 4) {
            again = true;
            continue;
        } else if (sInput[iLength - 3] != '.') {
            again = true;
            continue;
        } while (++iN < iLength) {
            if (isdigit(sInput[iN])) {
                continue;
            } if (iN == (iLength - 3)) {
                // ...
            }
        }
    }
}
```

# ALGORITHMIE

## Base de la programmation

CESI – Mehdi LITTAMÉ – 14/04/2025



 Presentation

# Qu'est ce qu'un **algo** ?

Suite d'instructions simples, qui une fois exécutée correctement, conduit à un résultat donné.

Suite d'instructions écrite en langage d'algorithme qui résout un problème et qui peuvent être programmé par n'importe quel langage

Il existe plusieurs algorithmes pour arriver au même résultat : on va chercher la version optimale

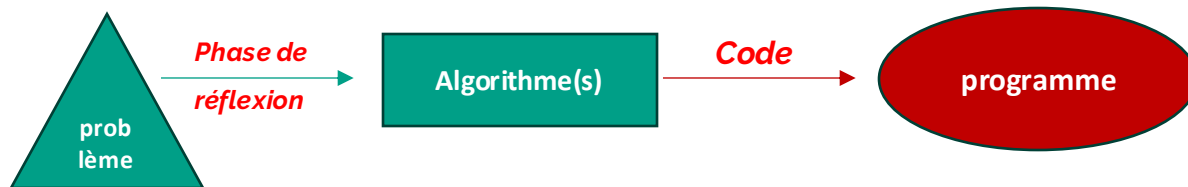
Ex de la vie courante : recette de cuisine, montage de meuble IKEA,...



# Methodologie

Un programme se présente comme un problème. On va devoir passer par une suite d'instructions (algorithmes) pour traiter les informations et arriver à un résultat

**Il est donc impératif de réfléchir à l'algo avant de programmer**



1. Comprendre le problème
2. Déterminer les données d'entrées et de sorties
3. Déterminer le processus



# Methodologie


On va pouvoir identifier plusieurs types d'instruction :

- Qui s'enchaines les une à la suite des autres : **séquence**
- Dans certains cas et pas d'autres : **structure alternative**
- Qui peuvent se répéter en boucle : **structure répétitive**

On va utiliser un pseudo-code ici, car l'interet est de comprendre la logique de ces instructions et non la syntaxe d'un langage en particulier. L'algorithme qui en découle pourra être adapter à n'importe quel langage par la suite si besoin



# 1. Déclaration **variable**

 Par définition une variable n'a pas de valeur tant qu'elle n'est pas définie

**Chaque variable doit être définie par**

- **Son nom (invariable)**
- **Son type (invariable)**
- **Sa valeur (optionnel) qui peut varier au cours d'un programme**

*Dans la mesure du possible, trouver un nom adéquat, lisible et représentatif*

Une variable doit toujours être déclarée et initialisée avant d'être utilisée

**Var entier** NomDeLaVariable = 5;



Var pour définir une variable



On attribut la valeur à celle-ci



# 1. Les types de variable (simples)

Il existe différents types de variables simples :

- **Entier** : nombre entier positif ou négatif

*var entier Chiffre = 3;*

- **Réel** : variable numérique avec décimale

*var reel Chiffre = 1,5;*

- **Chaine de caractère** : txt de plusieurs caractères

*var Cdc phrase = « Bonjour tout le monde » ;*

- **Booléen** : soit vrai soit faux

*var bool etat = vrai;*

**Var entier NomDeLaVariable = 5;**



Var pour définir une variable



On attribut la valeur à celle-ci



# 1. Les opérations

Pour chaque type de variable, des opérations sont possibles

Type var	Exemple	Opérations	Notation
Réel Entier	-2,5 3	Addition Soustraction Multiplication Division Reste de la div (modulo) Comparaison	+ - * / MOD <, <=, >=, >, =, !=
Caractère Chaine	'c'	Comparaison Concaténer	<, <=, >=, >, =, != +
Booléen	Faux	Comparaison Conjonction Disjonction	=, != ET OU

# Affectation de variable

On veut manipuler les valeurs de nos variables. L'affectation c'est placer une valeur dans une variable défini lors de sa déclaration ou bien après.

**Var entier NomDeLaVariable;** On définit notre variable

**NomDeLaVariable = 3 ;** On lui affecte une valeur avec le =

**NomDeLaVariable = NomDeLaVariable \* 3;** On lui affecte une valeur en l'utilisant avec le :=







 Instructions

## Entrées/sorties

On va pouvoir demander des saisies claviers « **Entrées** »

On va pouvoir afficher des données écran « **Sorties** »

- saisir une **entrée** :

**Saisir(NomDeLaVariable);**

- Afficher une **sortie** :

**Afficher(NomDeLaVariable);**



# Syntaxe d'une fonction algorithmique

Un algo sera composé :

- D'une phase de déclaration
- D'une phase d'instruction

## Algorithme Nom\_De\_La\_fonction :

**Déclarations;** Déclarations des variables, éventuellement de fonctions

## Début {

**Instructions;** Suite d'instructions / opérations

## } Fin



# Exercice 1

Écrire un algorithme permettant de saisir un entier et de l'afficher

```
Algorithme afficher_variable  
    Var entier entierASaisir;  
  
    Debut {  
        Saisir(entierASaisir);  
        Afficher(entierASaisir);  
    }Fin
```

# Exercice 2

Écrire un algorithme permettant de saisir deux nombres et d'afficher leur produit

## Algorithme produit\_2\_nombre

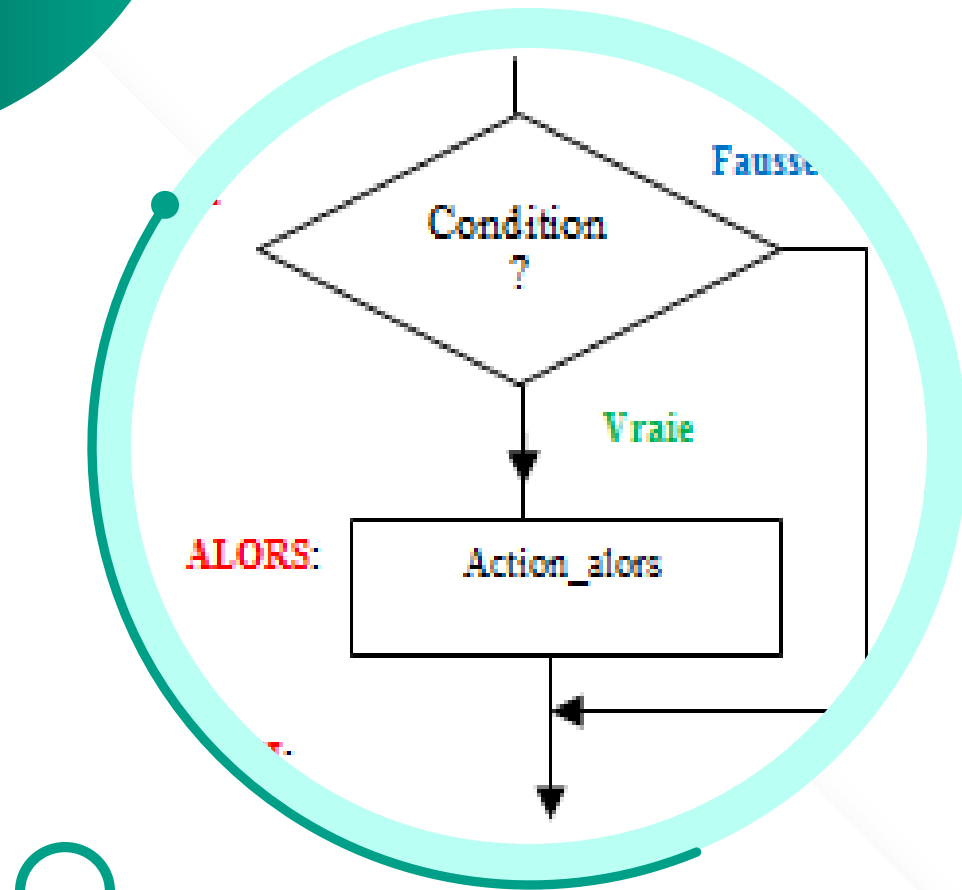
```
Var reel chiffre1;  
Var reel chiffre2;
```

```
Debut{  
  Saisir(chiffre1);  
  Saisir(chiffre2);
```

```
Afficher ( chiffre1*chiffre2);
```

```
}Fin
```





Instructions

## Structure conditionnel

On va vérifier une valeur, si c'est vérifié on fera les instructions A  
sinon on fera les instructions B

Syntaxe :

**Si(ici une condition) alors {**

**Instructions;** Suite d'instructions / opérations

**}**

Suite d'instructions / opérations

**SiNon( ){**

**Instructions;**

**}**





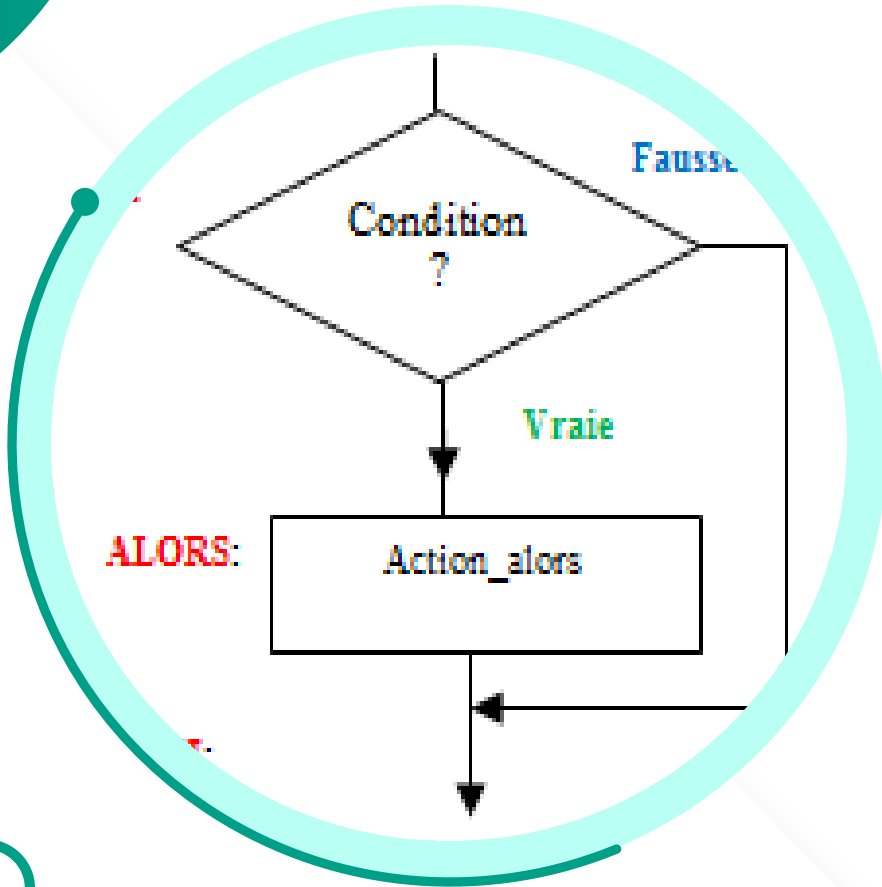
Instructions

# Les conditions

Si(ici une condition) alors ;

Valeur1 **OPERATEUR** Valeur2

<, <=, >=, >, =, !=





Instructions

# Les conditions composées

## Condition1 **ET** Condition2

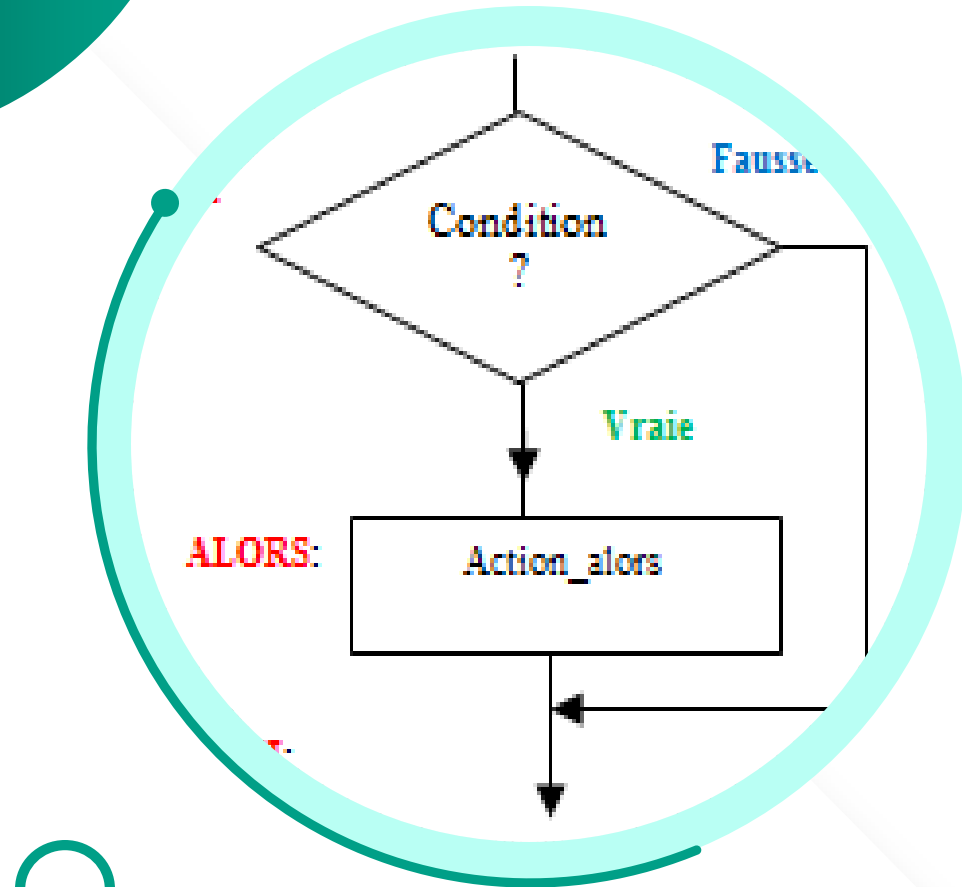
Il faut que 1 et 2 soient vrai

## Condition1 **OU** Condition2

Il faut au moins un des 2 soit vrai

## Condition1 **XOR** Condition2

Il faut uniquement un des 2 soit vrai



# Exercice 3

Écrire un programme qui permet d'afficher si un nombre entier saisi au clavier est pair ou impair.

## **Algorithme** entierPair

Var entier *entierASaisir*;

Debut {

Saisir(*entierASaisir*);

Si (*entierASaisir* % 2 = 0){

*afficher*(« Ce chiffre est pair »)

}

Sinon {

*afficher*(« Ce chiffre est pair »)

}

}Fin





## Exercice 4

Écrire un programme qui demande deux nombres  $m$  et  $n$  à l'utilisateur et l'informe ensuite si le produit de ces deux nombres est positif ou négatif. On inclut dans le programme le cas où le produit peut être nul.

### Algorithme produitPositif

Var reel  $m$ ;

Var reel  $n$ ;

Debut{

Saisir( $m$ );

Saisir( $n$ );

Si (  $m*n \geq 0$  ) {

Afficher (« le produit est positif »)

}

}Fin



## Algorithme produitPositif

Var reel m;

Var reel n;

Debut{

Saisir(m);

Saisir(n);

Si (  $m*n \geq 0$  ) {

Afficher (« le produit est positif »)

}

Si (  $m*n < 0$  ) {

Afficher (« le produit est negatif »)

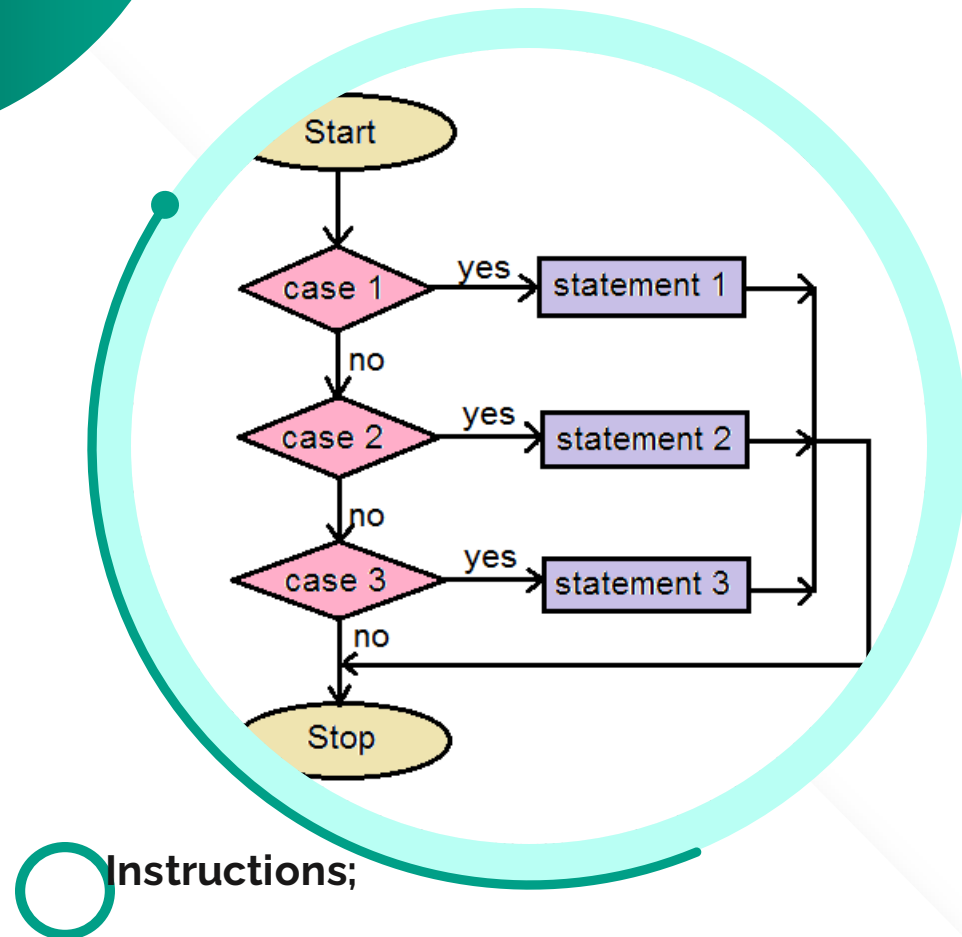
}

Si (  $m*n = 0$  ) {

Afficher (« le produit est nul »)

}

}Fin



Instructions;



Instructions

# Conditions multiple

Syntaxe :

## Selon que

Condition 1 faire

Instructions 1 ;

Ou que Condition 2 faire

Instructions 2 ;

Ou que Condition 3 faire

Instructions 3 ;

Autrement

Instructions ;

FinSelonQue



## Algorithme estAdulte

Var entier age;

Debut{  
Saisir(age);

Selon que ( age > 18){  
    Afficher (« vous etes majeur »)  
}

Ou que ( age < 18){  
    Afficher (« vous etes mineur »)  
}

Ou que ( age =< 0){  
    Afficher (« votre age n'est pas correct »)  
}

}Fin



# Exercice 5

Écrire un programme permettant d'afficher le mois en lettre selon le numéro saisi au clavier. ( Si l'utilisateur tape 1 le programme affiche janvier, si 2 affiche février , si 3 affiche mars... ). **Verfier aussi que le nombre soit compris entre 1 et 12**

## Algorithme mois

Var entier mois;

Debut{

Saisir(mois);

Selon que ( mois = 1){  
Afficher (« Janvier »)  
}

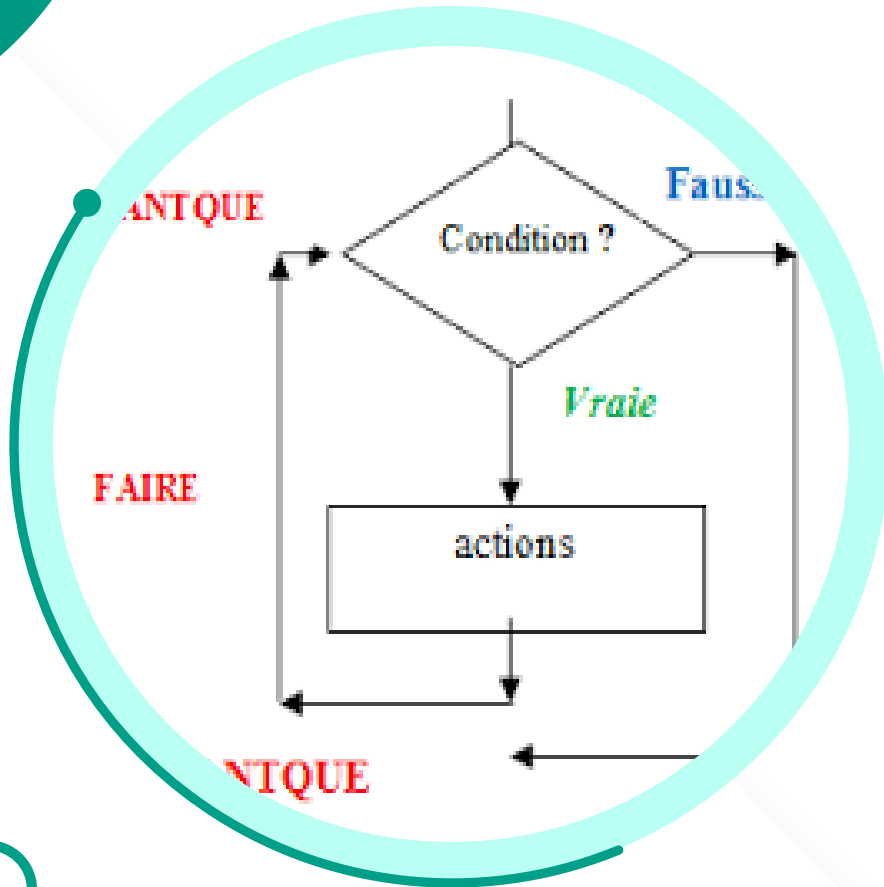
Selon que ( mois = 2){  
Afficher (« Fevrier »)  
}

.....

Selon que ( mois < 1 ET mois > 12){  
Afficher (« erreur de saisie »)  
}

}Fin





Instructions

## Boucle TantQue

Syntaxe :

**TantQue (ici une condition) Faire**

**Instructions;** Suite d'instructions / opérations

**FinTantQue;**

On arrive sur la ligne de condition, si la valeur est VRAI on exécute les instructions jusqu'à la ligne fintantque. Puis on revient à la ligne tant que et on répète l'opération jusqu'à arriver à la condition FAUX



# Exercice 6

Écrire un programme qui permet d'afficher le message "Bonsoir" 10 fois. Utilisant la boucle tant que.

```
Var entier compteur = 1;
```

```
Tant que ( compteur <=10) {
```

```
Afficher ( »Bonsoir »)
```

```
Compteur ++ // compteur = compteur +1;
```

```
}
```

# Exercice 7

Écrire un programme permettant de calculer la somme  $S = 1+2+3+...+ 10$ . Utilisant la boucle tant que

```
Var entier compteur = 1;
```

```
Var Somme = 0;
```

```
Tant que ( compteur <=10) {
```

```
  Somme = Somme + compteur
```

```
  Compteur ++ // compteur = compteur +1;
```

```
}
```

```
Afficher (Somme)
```







Instructions

# Boucle Pour (for)

Syntaxe :

**Pour** (**var** compteur; condition; incrementation) **Faire**

**Instructions;** Suite d'instructions / opérations

**FinPour;**

Permet de faire un nombre déterminé de passage de boucle. On peut ainsi définir le Compteur :

**i de 1 à 10, i++**

Variable de « départ » à « arrivée », incrémentation



Dans certains cas on peut placer l'incrémentation à la fin de la boucle



# Exercice 7

Écrire un programme permettant de calculer la somme  $S = 1+2+3+\dots+10$ . Utilisant la boucle tant que

```
Var Somme = 0;
```

```
Pour ( var compteur = 1 ; compteur<=10 ; compteur ++ ) {
```

```
  Somme = Somme + compteur
```

```
}
```

```
Afficher (Somme)
```

Pour (declaration variable compteur; **condition**; **incrementation**)



# Exercice 8

Écrire un programme qui permet de calculer la somme  $S=1+2+3+4+...+N$ . où N saisi au clavier par l'utilisateur. Utilisant la boucle Pour

```
Var Somme = 0;
```

```
Var chiffreASaisir ;
```

```
Saisir (chiffreASaisir )
```

```
Pour ( var compteur = 1 ; compteur<= chiffreASaisir ; compteur ++ ) {
```

```
    Somme = Somme + compteur
```

```
}
```

```
Afficher (Somme)
```

