BTS
BREVET DE TECHNICIEN SUPÉRIEUR
LYCÉE QUALIFIANT MOHAMED V
ISSAOUIRA

المملكة المغربية
وزارة التربية الوطنية
والتعليم الأولي والرياضة

Algorithmique et programmation

2024-2025

Chapitre 2 :

Les structures alternatives et répétitives

Les structures alternatives

Activité 1

A et B deux variables entières donnée.

Question :

Déterminer le maximum.

I- Structures conditionnelle

1-La structure séquentielle

C’est une structure dont les instructions sont exécutées l’une après l’autre en respectant l’ordre des instructions.

Exemple

Un algorithme qui permet de permuter deux entiers:

Algorithme **échange;**

Variable X, Y, Z : entier ;

Début

Ecrire (**"**donnez la valeur de X : **"**) ;
Lire (X) ;
Ecrire (**"**donnez la valeur de Y : **"**) ;
Lire (Y) ;
Z ← X ;
X←Y ;
Y← Z ;
Ecrire (**"**La valeur de X est : **",X**) ;
Ecrire (**"**La valeur de Y est : **",Y**) ;

Fin.

2-La structure Sélective:

La structure sélective est une structure dont les instructions sont exécutées selon les réponses des conditions.

1-1 Structure sélective Simple (un choix)

Syntaxe :

Si Condition **Alors**

    Instructions ;

Finsi

Exemple :

Un algorithme qui calcule le maximum de deux nombres réels.(sans prendre en considération le cas d'égalité)

```
Algorithme maximum;
Variable a,b : reel;
Début
    Ecrire ("donnez la valeur de a : ") ; Lire (a) ;
    Ecrire ("donnez la valeur de b : ") ; Lire (b) ;
    Si (a>b) alors
        Ecrire ("le max est a ") ;
    Finsi
    Si (a<b) alors
        Ecrire ("le max est b ") ;
    Finsi
Fin.
```

2-2 Structure alternative (deux choix)

Syntaxe :

Si Condition **Alors**
    Instruction**1** ;
Sinon
    Instruction**2** ;
Finsi

Explication:

- Une condition est une expression logique ou une variable logique évaluée à Vrai ou Faux.

- La condition est évaluée. **Si** elle est **vraie**, la série **d'instructions 1** est exécutée et l'ensemble **d'instructions 2** est ignoré. La machine sautera directement à la première instruction située après le **FinSi**.

- De même, au cas où la condition serait **fausse** (aurait la valeur Faux), la machine sautera directement à la première ligne située après le **Sinon** et exécutera l'ensemble **d'instructions 2**.

Exemple : le dernier exemple en utilisant la structure alternative

```
Algorithme maximum;
Variable a,b : reel;
Début
    Ecrire (”donnez la valeur de a : ”) ; Lire (a) ;
    Ecrire (”donnez la valeur de b : ”) ; Lire (b) ;
    Si (a>b) alors
        Ecrire (”le max est a ”) ;
    Sinon
        Ecrire (”le max est b ”) ;
    Finsi
Fin.
```

2-3 Structure alternative imbriquée

Syntaxe :

```
Si Condition Alors
    Instructions1 ;
Sinon
    Si condition 2 Alors
            Instructions2 ;
    Sinon
            Instructions3 ;
    Finsi
Finsi
```

Exemple : Un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou nul ou négatif.

```
Algorithme Nature_nombre;
Variable n : entier ;
Début
    Ecrire ("Entrer un nombre: ") ;
    Lire (n) ;
    Si (n>0) alors
        Ecrire ("ce nombre est positif ") ;
    Sinon
        Si (n=0) Alors
                Ecrire ("ce nombre est  nul ") ;
        Sinon
                Ecrire ("ce nombre est négatif ") ;
        Finsi
    Finsi
Fin.
```

Note 1 : Les deux blocs 1 et 2 d'instruction sont équivalents.

Bloc 1		Bloc 2
Si condition Alors Instructions(1) Sinon Instructions(2) FinSI	Le bloc 1 est équivalent au bloc 2	Si Non (condition) Alors Instructions(2) Sinon Instructions(1) FinSI

Exemple :

Bloc 1		Bloc 2
Si note > = 10 Alors Ecrire("Note acceptable") Sinon Ecrire("Mauvaise Note") FinSI	Le bloc 1 est équivalent au bloc 2	Si note < 10 Alors Ecrire("Mauvaise Note") Sinon Ecrire("Note acceptable") FinSI

Note 2 : Conditions composées

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

Par exemple : la condition "x est inclus dans l'intervalle [10, 20]" est composée de deux conditions simples qui sont : "x est supérieur à 10" et "x est inférieur à 20" reliées par l'opérateur logique Et.

2-4 Structure à choix multiple

Syntaxe :

Cas Variable ou Expression **Vaut**
    Val 1 : Instructions 1 ;
    Val 2 : Instructions 2 ;
    ……
    Val n : Instructions n ;
Sinon
    Autres Instructions ;
Fin Cas

Exemple:

```
Algorithme Nom_chiffre ;
Variable n : entier ;
Début
    Ecrire ('donnez votre chiffre entre 0 et 4 : ') ;
    Lire (n) ;
    Cas n vaut
        0 : Ecrire (' Zéro') ;
        1 : Ecrire ('Un') ;
        2 : Ecrire ('Deux') ;
        3 : Ecrire ('Trois') ;
        4 : Ecrire ('Quatre') ;
    Sinon
        Ecrire ('erreur de la saisie') ;
    Fin cas
Fin.
```

Remarque :Présentation de l'algorithme ou du programme

➢Certaines parties de l'algorithme ou du code sont en retrait par rapport à d'autres, c'est ce qu'on appelle l'indentation. Celle-ci est très importante pour la lisibilité de l'algorithme ou du programme. Elle montre rapidement le début et la fin de chaque instruction alternative ou répétitive ainsi le début et la fin de l'algorithme ou du programme.

➢De plus, pour faciliter la compréhension, toutes vos instructions doivent être commentées. Un développeur est souvent amené à modifier un code, par conséquent, des commentaires de qualité rendent cette tâche plus facile et plus rapide.

Les Structures répétitives

Problème : Distribution de bonbons

- Imagine que tu veux distribuer des bonbons à un groupe d'enfants et tu veux t'assurer que chaque enfant reçoive un bonbon.
- Tu devras répéter l'action de donner un bonbon à chaque enfant jusqu'à ce que tous les enfants en aient reçu un.

Algorithme distributeur_bonbon;

Début :

```
    Ecrire("Je donne un bonbon à l'enfant numéro 1 ") ;
    Ecrire("Je donne un bonbon à l'enfant numéro 2 ") ;
    Ecrire("Je donne un bonbon à l'enfant numéro 3 ") ;
    Ecrire("Je donne un bonbon à l'enfant numéro 4 ") ;
    Ecrire("Je donne un bonbon à l'enfant numéro 5 ") ;
    Ecrire("Je donne un bonbon à l'enfant numéro 6 ") ;
```

Fin

Les types de boucles:

- On distingue 2 types de boucles:
  - Les boucles à événement ou indéfinie
    - On ne sait pas à l'avance le nombre de fois que la boucle sera exécutée.
      - Ça peut dépendre du nombre de données à traiter.
      - Ça peut dépendre du nombre d'essais que l'usager a effectués.
      - Ex : **la boucle TantQue** et **la boucle jusqu'à (Faire TantQue)**
  - Les boucles à compteur ou définie
    - On sait à l'avance combien de fois la boucle devra tourner et une variable (le compteur) compte les répétitions
      - Choisir 10 nombres au hasard. On fera dix fois l'opération choisir un nombre au hasard.
      - Ex : **la boucle Pour**

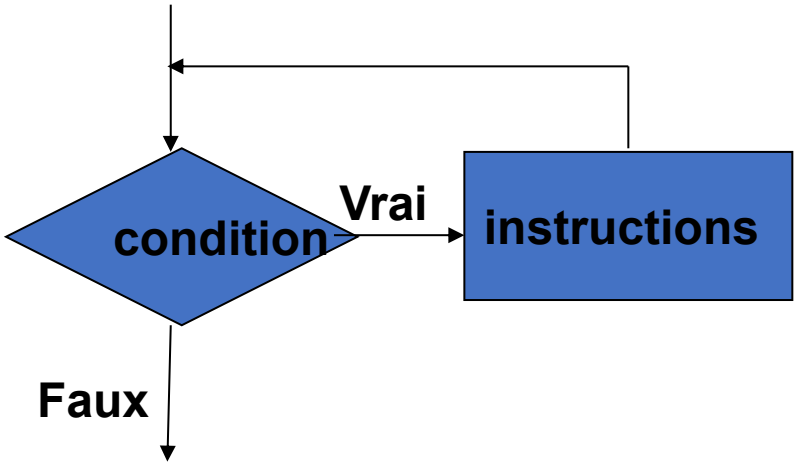
Les boucles Tant que

Les boucles Tant que

TantQue (condition) **Faire**

instructions

FinTantQue



Boucle TantQue : exemple

```
Algorithme Plus_Grand_Element
Variables grand, S, i,n: entiers
Début
    i←2; n←4;
    Ecrire(“donnez un nombre ”);
    Lire(S);
    grand ←S;
    TantQue (i ≤ n) Faire
                    Ecrire(“donnez le nombre ”,i);
                    Lire( S);
                    Si (S > grand) alors // une plus grande
    valeur a été trouvée
                    grand ← S;
                    FinSi
    i ← i+1;
    FinTantQue
    Ecrire (grand ,i );
Fin
```

Entrée: *n* entiers $S_1,\ldots, S_n$
Sortie: *grand* contenant le plus grand élément

Trace de l'algorithme:

n=4;	*S=-2*	*S=6*	*S =5*	*S=6*

grand =-2	*i = 2*
grand = 6	*i = 3*
	i = 4
	i = 5

Boucle TantQue : exercice

Ecrire un algorithme qui détermine le premier nombre entier N tel que la somme de 1 à N dépasse strictement 100, 1000?

Généralisez cet Algorithme?

Boucle TantQue : solution

Un algorithme qui détermine le premier nombre entier N tel que la somme de 1 à N dépasse strictement 100

```
Algorithme Pnombre
Variables som, i : entier
Debut
        i ← 0;
        som← 0;
        TantQue (som <=100) Faire
                        i ←  i+1;
                som ←  som+i;
        FinTantQue
        Ecrire (" La valeur cherchée est N=  ", i);
Fin
```

Algorithme d'Euclide

- Trouver le plus grand diviseur commun (pgcd) de deux entiers

Définition*:* *q* est un diviseur commun de *m* et *n* si *q* divise à la fois *m* et *n* (c-a-d le reste de la division entière est 0)

Le pgdc de m et n est le plus grand entier *q* divisant à la fois *m* et *n*.

Exemple: pgcd(4, 6) = 2; pgcd(3,8) = 1;
pgcd(9, 12) = 3;

Utile pour la simplification de fractions:
9/12 = 3*3/4*3 = 3/4

Trouver le PGCD de *a* et *b*

Exemple: pgcd(30, 105)

- 1ère méthode: Trouver tous les diviseurs de *a* et *b,* et trouver le diviseur commun le plus grand
  - Diviseurs de 30: 1, 2, 3, 5, 6, 10, 15, 30
  - Diviseurs de 105: 1, 3, 5, 7, 15, 21, 35, 105

  $\Rightarrow$ pgcd(30, 105) = 15

- 2ème méthode: **la méthode d'Euclide**
  - 105 = 30*3 + 15. Donc pgcd(105, 30) = pgcd(30,15)
  - 30 = 15*2 + 0. Donc pgcd(30, 15) = pgcd(15,0)
  - pgcd(15,0)=15

  $\Rightarrow$**pgcd(105,30)=pgcd(30,15)=pgcd(15,0)=15**

Méthode d'Euclide : Algorithme

Soient r_0, r_1 deux entiers strictement positifs, tels que $r_0=r_1.q+r_2$ avec $0 \leq r_2 < r_1$

- Si $r_2 = 0$, pgcd $(r_0, r_1) = r_1$
- Sinon, rediviser r_i par r_{i+1} tant que r_{i+1} est différent de 0
- Si r_n est le premier reste nul, alors

$\text{pgcd}(r_0,r_1) = \text{pgcd}(r_1,r_2) = \ldots = \text{pgcd}(r_{n-1},r_n) = \text{pgcd}(r_{n-1},0) = r_{n-1}$

Algorithme d'Euclide

```
Algorithme pgcd_deux nombres

    variables r,a,b: entiers
 Début
  Ecrire(" donner le premier nombre");
  Lire(a);
  Ecrire("donner le deuxieme nombre");
  Lire(b);

   TantQue b <> 0 Faire

//diviser a par b:  a  = b.q+r,  0 ≤ r < b;

    r←a mod b;

    a←b;

    b← r;

 Ecrire (a,b,r);

      FinTantQue

Ecrire (" le pgcd de",a, ' ' et b", ' 'est :", r);

Fin
```

Entrée: a, b deux entiers positifs

Sortie: pgcd(a,b)

Donner la trace de l'algorithme lors que:

a=504 et b=396 ?

Algorithme d'Euclide

Entrée: a, b deux entiers positifs

Sortie: pgcd(a,b)

TantQue *b <> 0* **Faire**

//diviser *a* par *b:* *a = b.q+r, 0 ≤ r < b;*

r←a%b;

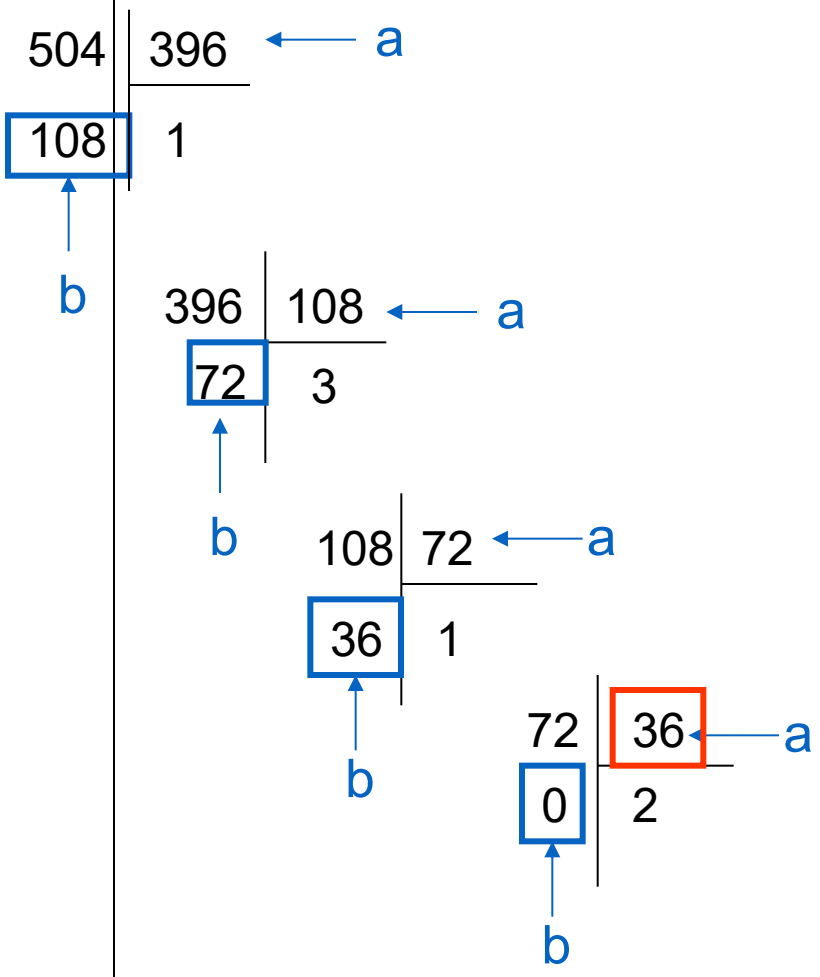
a←b;

b← r;

FinTantQue

Trace de l'algorithme pour

a=504 et b=396



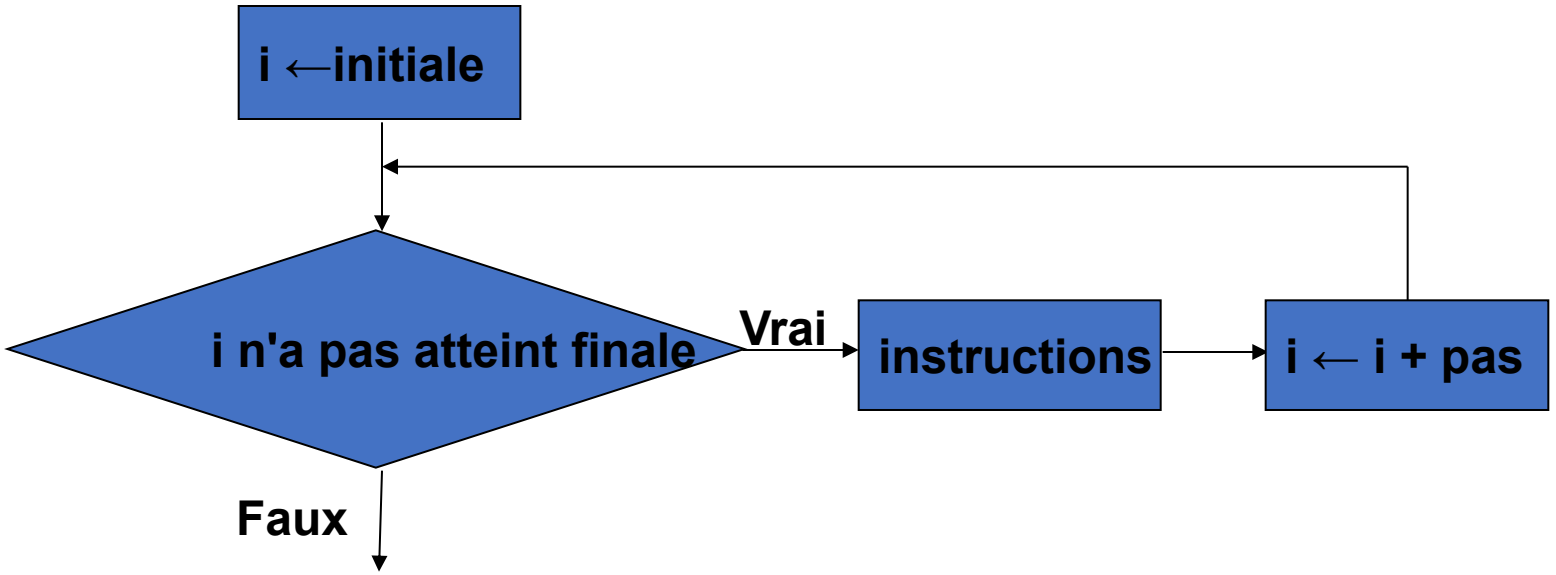
Les boucles Pour

Les boucles Pour

Pour compteur **allant de** initiale **à** finale par **pas** valeur du pas

instructions

FinPour



Les boucles Pour

Remarques :

- **Compteur** est une variable; Elle doit être déclarée

- **Pas** est un entier qui peut être positif ou négatif. **Pas** peut ne pas être mentionné, car par défaut sa valeur est égal à 1. Dans ce cas, le nombre d'itérations est égal à finale - initiale+ 1

- **Initiale et finale** peuvent être des valeurs, des variables définies avant le début de la boucle ou des expressions de même type que compteur

Déroulement des boucles Pour

1) La valeur initiale est affectée à la variable compteur
2) On compare la valeur du compteur et la valeur de finale :
   a) Si la valeur du compteur est > à la valeur finale dans le cas d'un pas positif (ou si compteur est < à finale pour un pas négatif), on sort de la boucle et on continue avec l'instruction qui suit FinPour
   b) Si compteur est <= à finale dans le cas d'un pas positif (ou si compteur est >= à finale pour un pas négatif), instructions seront exécutées
      i. Ensuite, la valeur de compteur est incrémentée de la valeur du pas si pas est positif (ou décrémenté si pas est négatif)
      ii. On recommence l'étape 2 : La comparaison entre compteur et finale est de nouveau effectuée, et ainsi de suite …

Boucle Pour : Exercice

Algorithme Plus-Grand-Element: Réécrire l'algorithme précédent mais avec une boucle « Pour »

Boucle Pour : exemple

Algorithme Plus-Grand-Element

//Réécriture de l'algorithme précédent mais avec une boucle ``Pour''

Variable grand,i,S,n, :entiers

Debut

N←4;

Ecrire(“donnez un nombre”);

Lire(S);

grand <- S;

Pour *i* ***allant de 2 à*** *n*

Ecrire(“donnez le nombre ”,i);

Lire(S);

Si (*S > grand*) **alors** //*une plus grande valeur a été trouvée*

grand <- S ;

FinSi

FinPour

Ecrire *(grand);*

Fin

Boucle Pour : remarque

- Il faut éviter de modifier la valeur du compteur (et de finale) à l'intérieur de la boucle. En effet, une telle action :
  - Perturbation de nombre d'itérations prévu par la boucle
  - Difficulté de lecture de l'algorithme
  - Risque d'aboutir à une boucle infinie

Exepmle : **Pour** i **allant de** 1 **à** 5
i ← i -1;
Ecrire(" i = ", i) ;
FinPour

Lien entre Pour et TantQue

La boucle Pour est un cas particulier de TantQue (cas où le nombre d'itérations est connu et fixé) . Tout ce qu'on peut écrire avec Pour peut être remplacé avec TantQue (la réciproque est fausse)

Pour compteur **allant de** initiale **à** finale par **pas** valeur du pas

instructions

FinPour

peut être remplacé par :

(cas d'un pas positif)

```
compteur ← initiale;
TantQue  compteur <= finale Faire
instructions;
compteur ←  compteur+pas;
FinTantQue
```

Lien entre Pour et TantQue: exemple

Calcul de **x** à la puissance **n** avec la boucle **Pour** et la boucle **TantQue**

x : un réel non nul

n : entier positif ou nul

Solution avec la boucle Pour

Algorithme TestPour

Variables x, puiss : réel

n, i : entier

Debut

Ecrire (" Entrez respectivement les valeurs de x et n ");

Lire (x, n);

puiss ← 1;

Pour i **allant de** 1 **à** n

puiss← puiss*x ;

FinPour

Ecrire (x, " à la puissance ", n, " est égal à ", puiss**);**

Fin

Solution avec la boucle TantQue

Algorithme TestTantQ

variables x, puiss : réel

n, i : entier

Debut

Ecrire (" Entrez respectivement les valeurs de x et n ");

Lire (x, n);

puiss ← 1; i ← 1;

TantQue (i<=n) **Faire**

puiss← puiss*x ;
i ← i+1;

FinTantQue

Ecrire (x, " à la puissance ", n, " est égal à ", puiss**);**

Fin

Exemple : Algorithme de la fonction factorielle

- Écrire deux algorithmes qui calculent pour un entier positif donné **n** la valeur **n!**, un de ces algorithmes doit utilisé la boucle Pour et l'autre la boucle Tanque

Entrée : n de type naturel

Sortie : factoriel (n) = 1*2*3*…..*(n-1)*n

Algorithme de la fonction factorielle

Algorithme / TantQue

```
Algorithme Calcul_factorielle_1
Variables
    i, fact1, n : Entier
Début
Lire(n);
        i ← 1;
        f ← 1;
   TantQue (i < n) Faire
        i ← i+1;
        f ← f * i;
      FinTantQue
       Ecrire (f);
Fin
```

Algorithme / Pour

```
Algorithme Calcul_factorielle_2
Variables
    i, fact2, n : Entier
Début
 Lire(n);
 f ← 1;

Pour i Allant de 2 à n
       f ← f * i;
    FinPour
       Ecrire (f);
Fin
```

Exemple :Algorithme de la recherche des nombres premiers

- Problème: Écrire l'algorithme estPremier, qui a partir d'un entier strictement positif donné, retourne le résultat booléen VRAI ou FAUX selon le nombre est premier ou non.
- Procédure : pour déterminer si un entier *m* est un nombre premier. Il suffit de tester si *m* est divisible par un entier entre *2* et *m/2*
  - *Ex : 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47. (listes des nombres premier <=50)*

```
Algorithme estPremier;
Variable m,i:entier;
        premier : booleen;
Début
      Lire (m);
      Si m < 2 alors
            premier= FAUX ;
      Sinon
            premier ← VRAI ;
            Pour i de 2 à m/2 faire
                  Si (m mod i = 0) alors
                        premier ← FAUX ;
                  FinSi
            FinPour
      Fin Si
      Si premier = VRAI alors
         ecrire (m,  " est premier ") ;
      Sinon
         ecrire(m, " n'est pas premier ") ;
      FinSi
Fin
```

Détecter l'erreur dans les deux algorithmes

Algorithme

```
Algorithme Essai0
Variables
    n : entier
Début
        n ← 15;
    TantQue (n<>0) Faire
        Ecrire (n);
        n ← n – 2;
    FinTantQue
Fin
```

Algorithme

```
Algorithme Essai1
Variables
    k, N : entier
Début
        n ← 200;
    Pour k  Allant de 1 à n
        Ecrire (k);
        k ← n – 100;
    FinPour
Fin
```

Boucles imbriquées

- Les instructions d'une boucle peuvent être des instructions itératives. Dans ce cas, on aboutit à des **boucles imbriquées**

- **Exemple:**

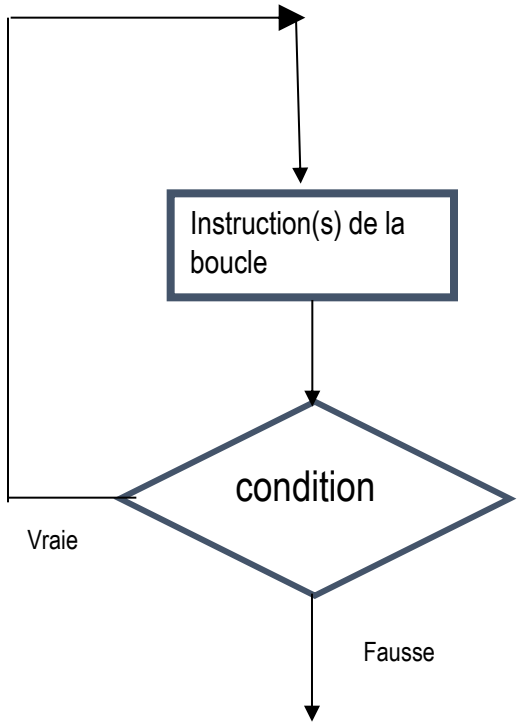
```
Pour i allant de  1 à 5
    Pour j allant de  1 à i
        Ecrire("O");
    FinPour
    Ecrire("X");
FinPour
```

Exécution?

OX
OOX
OOOX
OOOOX
OOOOOX

La boucle Faire…Tant Que

- **Faire**
  Instruction(s)
  TantQue (condition)



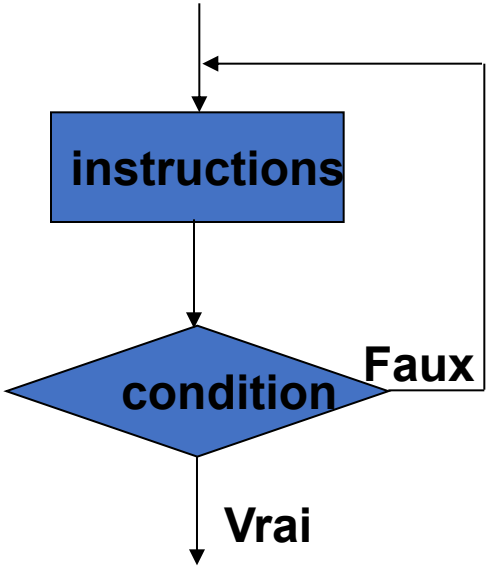
La boucle s'exécute tant que la condition est vraie. La boucle cesse lorque la condition est fausse. À utiliser si l'on veut que la boucle soit exécutée au moins une fois

Les boucles **Répéter … jusqu'à …**

Répéter

instructions

Jusqu'à (condition)



- Condition est évaluée après chaque itération
- les instructions entre *Répéter* et *jusqu'à* sont exécutées au moins une fois et leur exécution est répétée jusqu'à ce que condition soit vrai (tant qu'elle est fausse)

Boucle **Répéter jusqu'à** : exemple

Un algorithme qui détermine le premier nombre entier N tel que la somme de 1 à N dépasse strictement 100 **(version avec répéter jusqu'à)**

Algorithme Test

```
Variables som, i : entier
Debut
        som ← 0;
        i ← 0;
        Répéter
                i ← i+1;
                som ← som+i;
        Jusqu'à ( som > 100)
        Ecrire (" La valeur cherchée est N=  ", i);
Fin
```

Choix d'un type de boucle

- Si on peut déterminer le nombre d'itérations avant l'exécution de la boucle, il est plus naturel d'utiliser *la boucle Pour*

- S'il n'est pas possible de connaître le nombre d'itérations avant l'exécution de la boucle, on fera appel à l'une des *boucles TantQue ou répéter jusqu'à*

- Pour le choix entre *TantQue* et *jusqu'à :*

  - Si on doit tester la condition de contrôle avant de commencer les instructions de la boucle, on utilisera *TantQue*

  - Si la valeur de la condition de contrôle dépend d'une première exécution des instructions de la boucle, on utilisera *répéter jusqu'à ou faire tanque*

Algorithme de la racine carrée : Exercice

- Problème: Écrire l'algorithme qui calcul la racine carrée d'un nombre sans avoir recours a la fonction mathématique Racine Carrée prédéfinie.
- Procédure : si n est le nombre dont on souhaite extraire la racine carrée. On construit une suite de nombres Xi dont le premier vaut 1 et dont le terme général a pour expression :

$$X_i = (n/x_{i-1} + x_{i-1}) / 2$$

Rq : Cette suite converge systématiquement vers racarree (n)

Algorithme de la racine carrée

Algorithme Racine Carrée (RC)

```
Algorithme Racine
Variables
    n, x : réels
    i, max : entier
Début
    Lire (n);
    Lire (max);
    x ← 1;
 Pour i allant de 1 à max
        y ← ((n/x) + x) / 2;
        x←y;
 FinPour
 Ecrire (x),
Fin
```

Algorithme de la racine carrée

Exercice

Tester l'algorithme précédent avec des valeurs:

Exemples: 1, 2, 3 , 4, 9.