Université Ibn Zohr

Faculté des Sciences

Département de Mathématiques et Informatique

Ce support de cours est destiné aux étudiants des filières SMC2, SMP2, SV4 et SM2. Il comporte les éléments de base pour écrire des algorithmes simples.

Introduction à la programmation

- Pour résoudre un problème donné par l'informatique

  l'utilisateur de l'ordinateur doit mettre au point un programme et le faire exécuter par la machine.
- L'ordinateur se chargera de traiter les instructions des programmes et restituer les résultats demandés en fonction des données qui lui sont fournies.
- Un programme est une succession logique et ordonnée d'instructions.

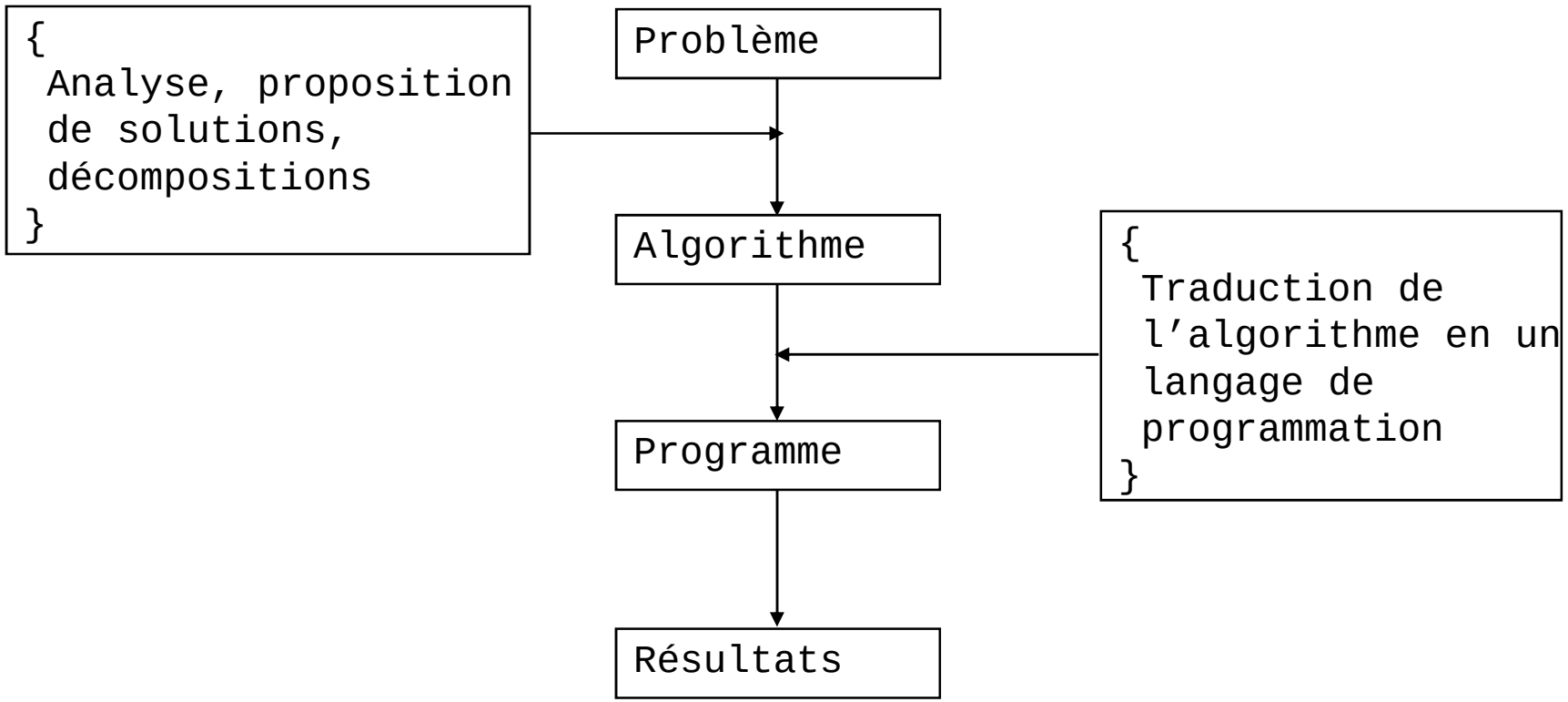
Étapes du processus de programmation

Le processus de programmation comprend les étapes suivantes :

- Spécifier le problème à résoudre, il s'agit de préciser quel est le problème, de déterminer les besoins et de fixer les objectifs,
- Trouver un algorithme comme résultat de la spécification
- Connaître un langage compréhensible par la machine
- Tester le programme,

La démarche à suivre dans la résolution d'un problème en informatique est donc

Étapes du processus de programmation



Définition et objectif d'un algorithme

Un algorithme est la description de la solution d'un problème sous la forme d'une suite finie d'opérations à effectuer sur les données du problème.

Pour fonctionner, un algorithme doit contenir uniquement les instructions compréhensibles par celui qui devra l'exécuter. Son fonctionnement nécessite un certain nombre d'objets, cette ensemble s'appelle : environnement de l'algorithme.

Instructions élémentaires en algorithmique

Le langage algorithmique offre trois instructions élémentaires de base :

- l'affectation,
- entrée de données,
- sortie de données,

Instruction d'affectation variable (1)

La base de l'informatique est le stockage et la manipulation de données. Ceci se fait essentiellement par des variables (stockage) et l'instruction *d'affectation* (transfert d'information).

Une variable est caractérisée par :

- Un nom symbolique (Identificateur),
- Un type,
- Une valeur,

Instruction d'affectation variable (2)

Le nom symbolique attribué aux objets doit respecter quelques règles :

- La suite des caractères peut être composée soit de lettres non accentuées (a..z, A..Z) soit de chiffres (0..9) soit du caractères de soulignement (_),
- Il ne doit contenir ni espace ni de caractères spéciaux,

Identificateur

Identificateur correct	Identificateur incorrect
Ce_nombre lePGCD Jours1	tel Nombre calcul Surface 1mois

Remarque :

De préférence le nom (identificateur) est choisi en rapport avec le rôle de l'objet pour faciliter la lecteur de l'algorithme à un autre programmeur.

Instruction d'affectation variable (3)

Le type d'une variable permet de :

- définir l'ensemble de valeurs que peut prendre la variable,
- fixer la taille, en cases mémoires (octets), de la variable,
- définir la nature des opérations autorisées sur la variable

Les types utilisés en langage algorithmique sont :

- Entier pour représenter les entiers positifs ou négatifs,
- Réel pour représenter les nombres à virgule,
- Caractère pour représenter des caractères,
- Chaîne pour représenter des phrases

Instruction élémentaire : affectation

L'affectation permet d'assigner une valeur à un objet. Elle est représentée en algorithmique par le symbole

$\longleftarrow$

Syntaxe :

Identificateur $\longleftarrow$ valeur

Le membre droit d'une affectation (valeur) peut être soit :

- Une variable de même type que identificateur,
- Une constante de même type que identificateur,
- Une expression dont l'évaluation produit un résultat final de même type que identificateur

Instruction élémentaire : affectation

Les affectations suivantes sont incorrectes :

```
16    ←——   valeur
X*Y   ←——   15
```

Le membre gauche d'une affectation (***lvalue***) doit être une variable déclarée. Il doit correspondre à une case mémoire qui peut recevoir une valeur.

De même la partie droite d'une affectation doit être une quantité bien définie, c à d une structure ayant une évaluation qui fournit une valeur résultat,

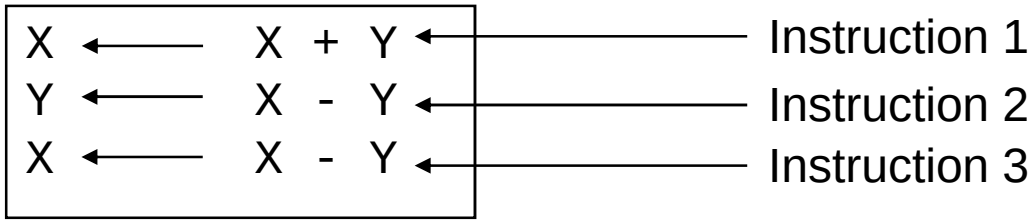
Instruction élémentaire : affectation

Les types de deux parties de l'affectation doivent être les mêmes

Exemple d'affectation

Soit deux variables X et Y de type entier, Supposons que la variable X contient la valeur 16 et que la variable Y contient 25.

Que valent les valeurs de X et Y après les instructions d'affectation suivantes ?

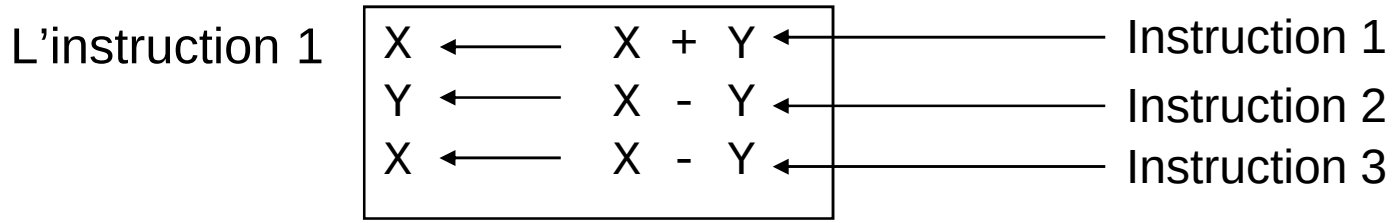


Instruction élémentaire : affectation

Schématisons un peu !

X contient 16 et Y contient 25, veut dire que les cases mémoire nommées respectivement X , Y et qui sont situées en mémoire aux adresses adr1200 et adr1204 (par exemple) contiennent respectivement les valeurs 16 et 25.

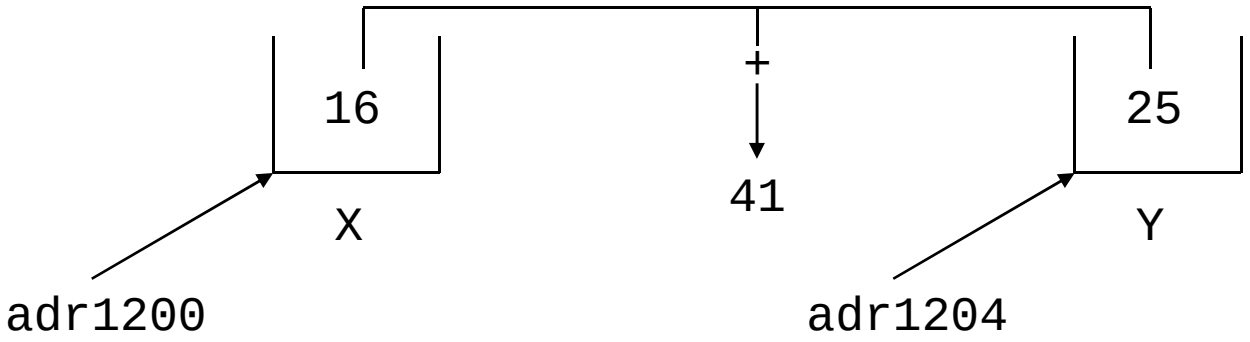
Instruction élémentaire : affectation



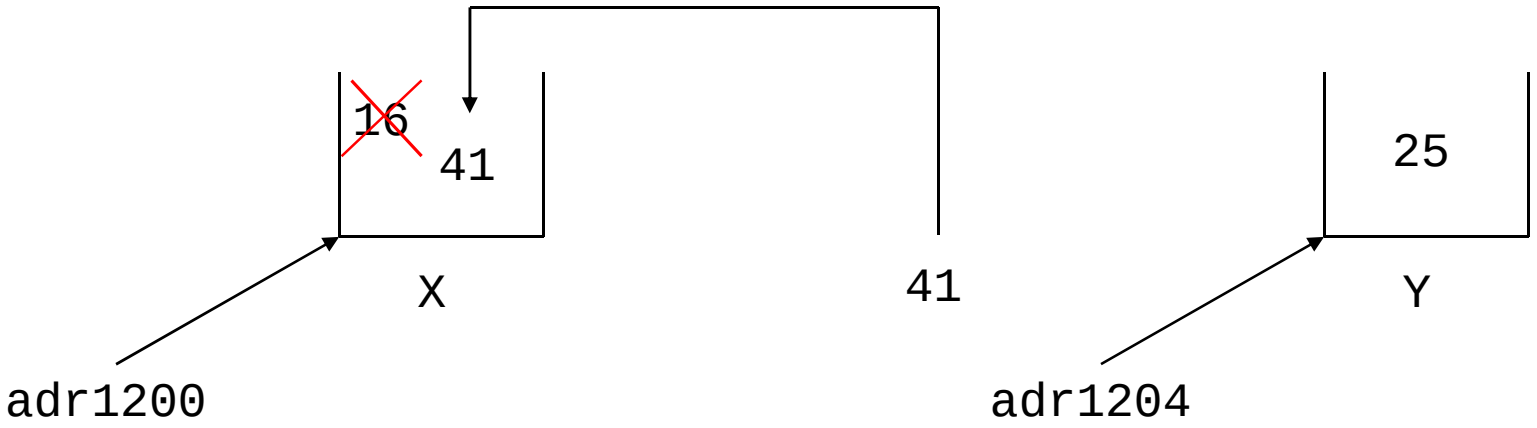
L'instruction 1 peut être explicitée comme ceci :

1. Évaluation de l'expression X + Y

   Addition des valeurs de X et Y
2. Affectation du résultat de l'expression X + Y à X



Instruction élémentaire : affectation



Après l'instruction 1, X contient 41 et la variable Y n'a pas été modifiée.

Après l'instruction 2, X vaut 41 (elle n'a pas été modifiée) la variable Y contient 16.

Après l'instruction 3, X contient 25 et Y 16.

Instruction élémentaire : affectation

Remarques :

- L'ensemble des instructions 1, 2 et 3 permet de permuter les valeurs de deux variables.
- L'opérateur d'affectation détruit complètement et définitivement le contenu d'une variable.

Instruction élémentaire : affectation

```
Algorithme Affectation1
Var
   X, Y : entier
Début
   X ⟵ 15
   Y ⟵ 25
   X ⟵ Y
   Y ⟵ X
Fin
```

```
Algorithme Affectation2
Var
   X, Y : entier
Début
   X ⟵ 15
   Y ⟵ 25
   Y ⟵ X
   X ⟵ Y
Fin
```

Dans les deux algorithmes ci-dessus, les instructions sont exécutées dans l'ordre, séquentiellement : l'une après l'autre.
Du fait de cet ordre séquentiel, les deux séquences ci-dessus ont des effets très différents.

Les valeurs de sorties pour les variables X et Y sont : 25 pour l'algorithme 1 et 15 pour l'algorithme 2.

Instruction d'entrée

Elle permet d'introduire une valeur d'une variable. L'entrée de données peut être effectuée soit par clavier ou directement par lecture des données stockées sur le disque.

En langage algorithmique la fonction utilisée pour lire des données est :

```
Lire(maVar)
```

cette instruction permet d'affecter à la variable *maVar*, la valeur lue sur le périphérique d'entrée.

Instruction de sortie

La sortie, qui constitue le résultat, peut être affiché à l'écran imprimé sur papier ou stocké dans une mémoire magnétique.

La fonction de sortie utilisée en algorithmique est la fonction Écrire :

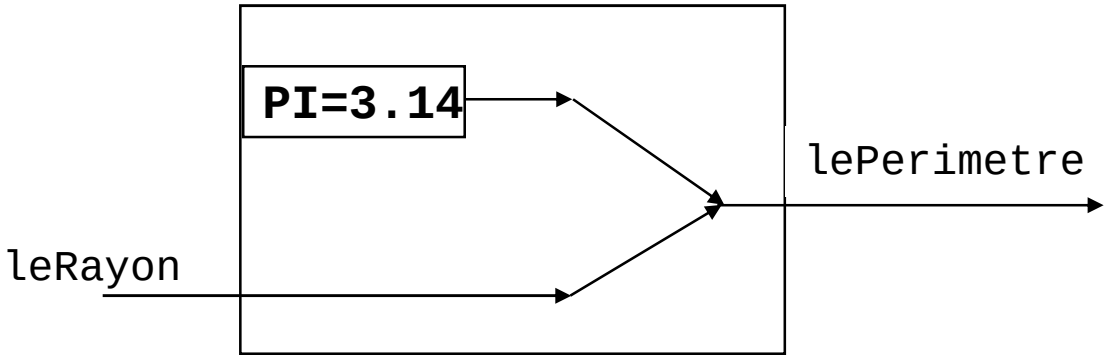
```
Écrire(monRésultat)
```

cette instruction permet de transférer la valeur *monRésultat* vers le périphérique de sortie.

Exercice

Proposer une analyse pour calculer le périmètre d'un cercle (`2*Π*leRayon`)

Analyse du problème



Algorithme de résolution

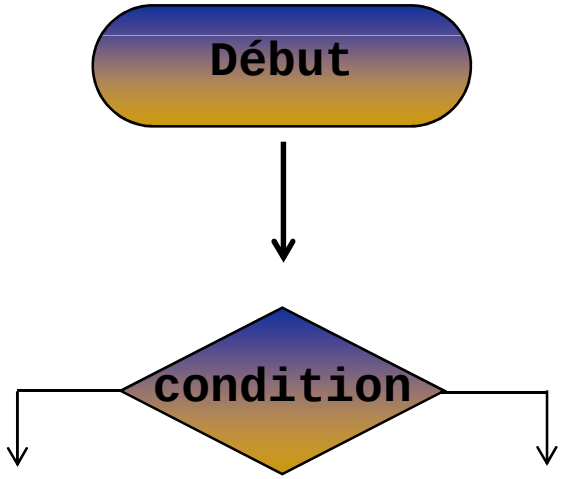
Algorithme A1

```
Algorithme perimetreCercle
const PI ←——3.14
Var
     lePerimetre, leRayon : réels
Début
     Lire(leRayon)
     lePerimetre    ←—2*PI*leRayon
     Ecrire(leRayon)
Fin
```

Organigrammes (1)

Un organigramme est une représentation schématique d'un algorithme mettant en valeur sa structure. Il permet de mieux présenter les différents modules de traitement et d'expliquer la succession des opérations d'un travail.

Les symboles utilisés dans un organigramme sont :



Début : démarrage d'un traitement

Séquence : marque le passage d'une action à une autre

Test et décision : marque un choix conditionnel d'une action à exécuter suivant que la condition est vérifiée ou non.

Organigrammes (2)

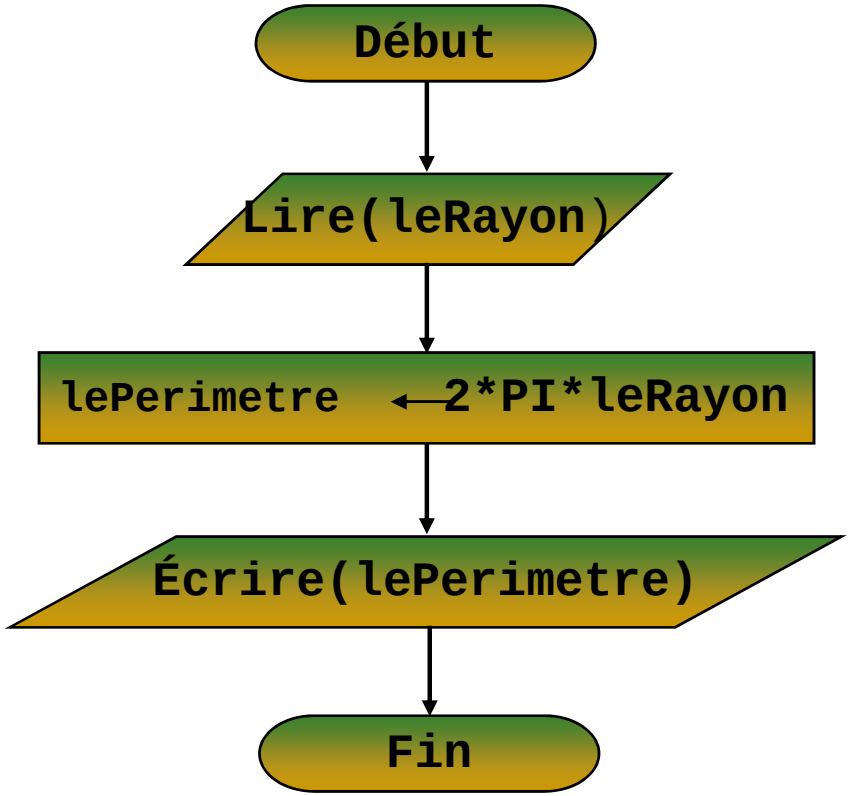
Variante de l'alternative : selon la valeur de la variable cas, il y aura branchement vers l'exécution de l'instruction correspondante.

Opération : calcul modifie une variable par l'affectation d'une nouvelle valeur.

Instruction d'entrée/sortie : entrée de données standard à partir du clavier ou sortie de données standard vers l'écran.

Fin : Arrêt d'un traitement

Exemple traduction de l'algorithme A1



Instructions composées

Les algorithmes précédents présentent un enchaînement d'exécution linéaire et séquentiel, dans la pratique, il peut y avoir des cas où l'on exige une exécution par morceau, c à d des sauts ou des répétitions d'un même bloc d'instructions. Ceci peut se faire à l'aide des primitives de base structurées.

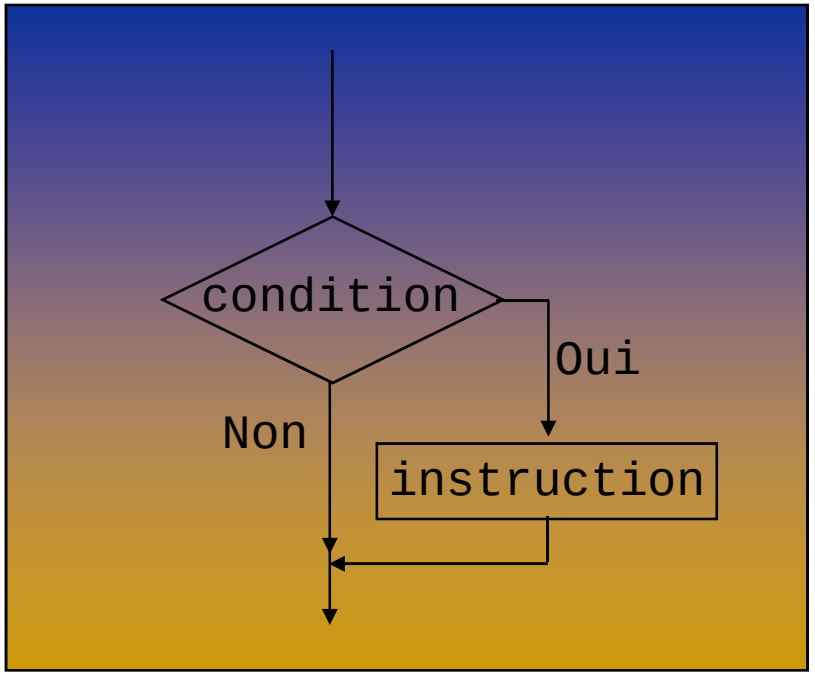
Il existe deux types d'instructions composées :

- Les primitives de choix,
- Les primitives d'itérations

Instructions conditionnelles

Syntaxe :

```
Si (condition) Alors
   instruction
FinSi
```

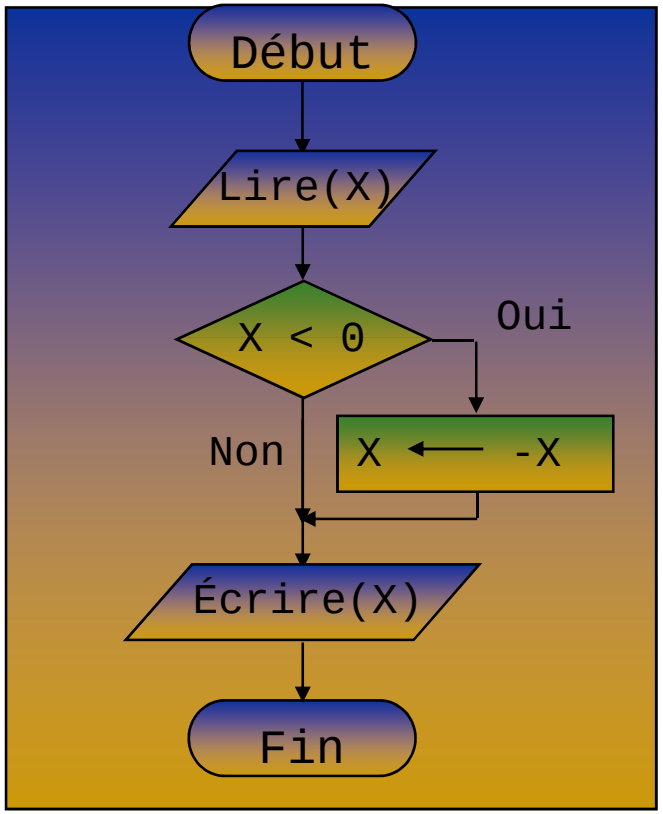


L'instruction (ou le bloc d'instructions), sera exécutée si la condition est vérifiée (valeur booléenne à VRAI).

Exemple

Exemple Valeur absolue d'un nombre réel,

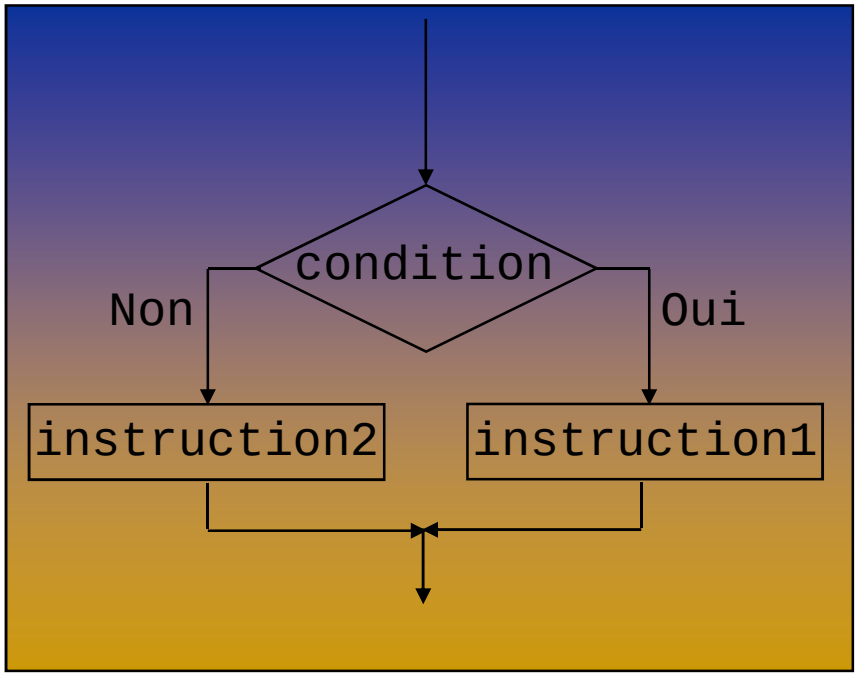
```
Algorithme ValAbs
Var
   X : réel
Début
   Écrire('Saisir un nombre réel')
   Lire(X)
   Si(X < 0)
        X ←——— -X
   FinSi
   Écrire('La valeur absolue est', X)
Fin
```



Instructions alternatives

Syntaxe :

```
Si (condition) Alors
   instruction1
Sinon
   instruction2
FinSi
```

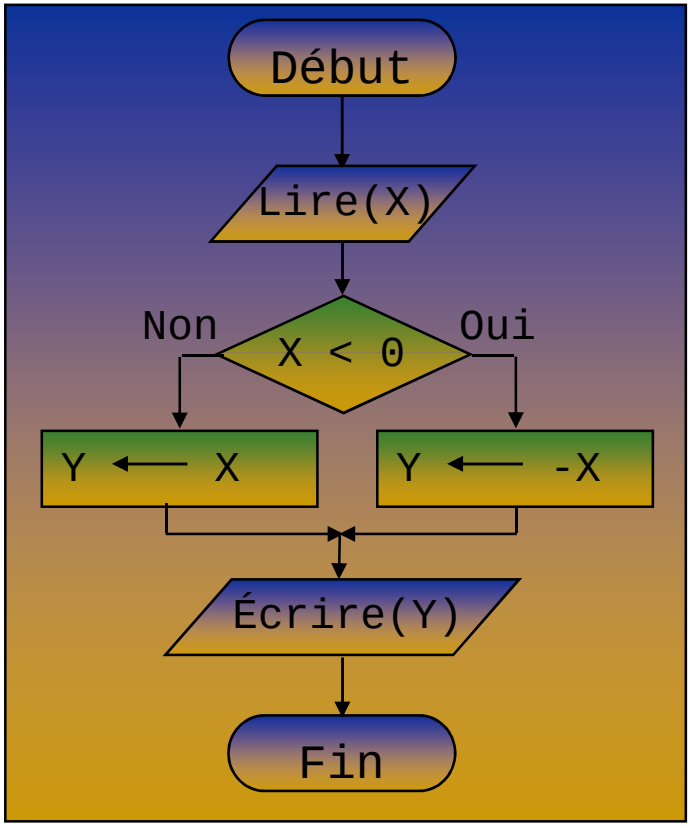


condition est une expression booléenne. Elle est évaluée, quand sa valeur vaut VRAI l'instruction1 est exécutée, quand elle vaut FAUX l'instruction2 est exécutée.

Exemple

Exemple valeur absolue d'un nombre réel,

```
Algorithme ValAbs
Var
   X, Y : réel
Début
   Écrire('Saisir un nombre réel')
   Lire(X)
   Si(X < 0)
        Y ←——— -X
   Sinon
        Y ←——— X
   FinSi
   Écrire('La valeur absolue est', Y)
Fin
```



La primitive de l'alternative
La primitive « CAS » (1)

Étude par un exemple

On désire écrire un algorithme qui permet la saisie d'un entier compris entre 0 et 4, ensuite l'algorithme doit afficher le nombre saisi en lettre.

```
Algorithme afficheEntier
Var
   lentier : entier
Début
   Écrire('Saisir un nombre entier')
   Lire(lentier)
```

La primitive CAS (2)

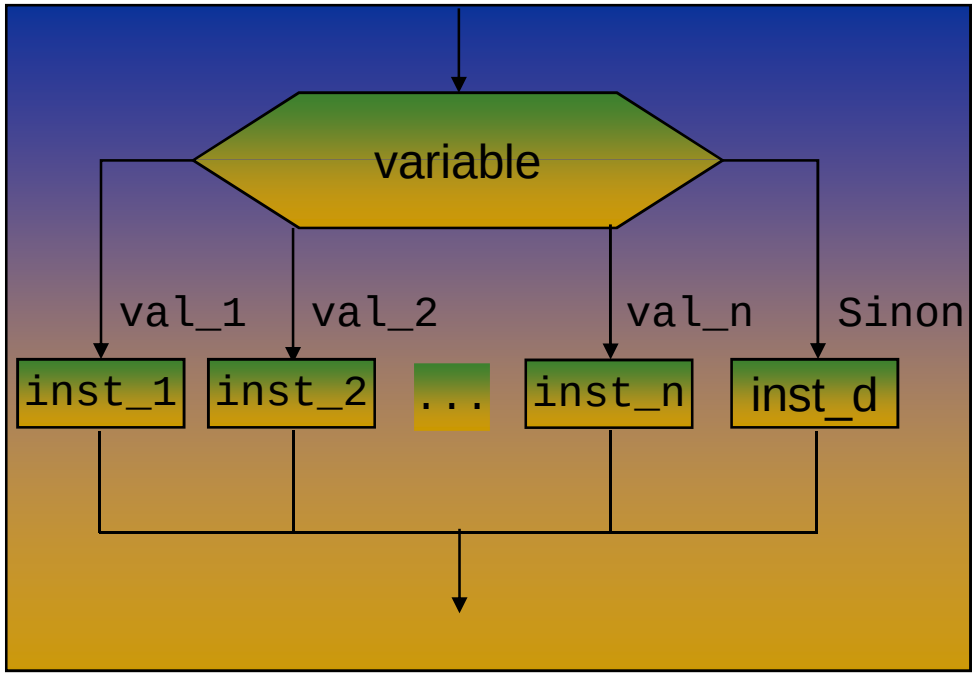
```
Si(lentier = 0)
 Écrire('Vous avez saisi Zéro')
Sinon     Si (lentier = 1)
           Écrire('Vous avez saisi Un')
          Sinon  Si (lentier = 2)
                   Écrire('Vous avez saisi Deux')
                 Sinon  Si (lentier = 3)
                          Écrire('Vous avez saisi Trois')
                        Sinon  Si (lentier = 4)
                                 Écrire('Vous avez saisi Quatre')
                               Sinon
                                 Écrire('L'entier saisi est > à 4')
                               FinSi
                        FinSi
                 FinSi
          FinSi
 FinSi
Fin
```

La primitive CAS (3)

L'algorithme précédent comporte beaucoup d'imbrication de plusieurs Si. L'oubli d'un Sinon peut entraîner des erreurs de fonctionnement de l'algorithme. La primitive CAS facilite énormément l'écriture de ce genre d'algorithme.

Syntaxe :

```
Cas (variable)
   val_1  : inst_1
   val_2  : inst_2
   ...
   val_n  : inst_n
   Sinon  : inst_d
FinCas
```



La primitive CAS (4)

variable est une expression de type discret (caractère, entier ou booléen). Elle est évaluée, si sa valeur est égale à une des valeurs val_i, l'instruction correspondante est exécutée. En revanche si la valeur de variable ne coïncide avec aucune des valeurs val_i, c'est l'instruction (par défaut) inst_d qui est exécutée.

```
Algorithme afficheEntier
Var
   lentier : entier
Début
   Écrire('Saisir un nombre entier')
   Lire(lentier)
   Cas (lentier)
        0       : Écrire('Vous avez saisi Zéro')
        1       : Écrire('Vous avez saisi Un')
        2       : Écrire('Vous avez saisi Deux')
        3       : Écrire('Vous avez saisi Trois')
        4       : Écrire('Vous avez saisi Quatre')
        Sinon   : Écrire('L'entier saisi est > à 4')
   FinCas
Fin
```

Instructions répétitives (boucles)

La résolution de quelques problèmes passe parfois par la réalisation d'une action d'une manière répétitive.

Par exemple, enfoncer un clou dans un mur se réalise par donner des coups du marteau sur la tête du clou tant que le clou dépasse le mur.

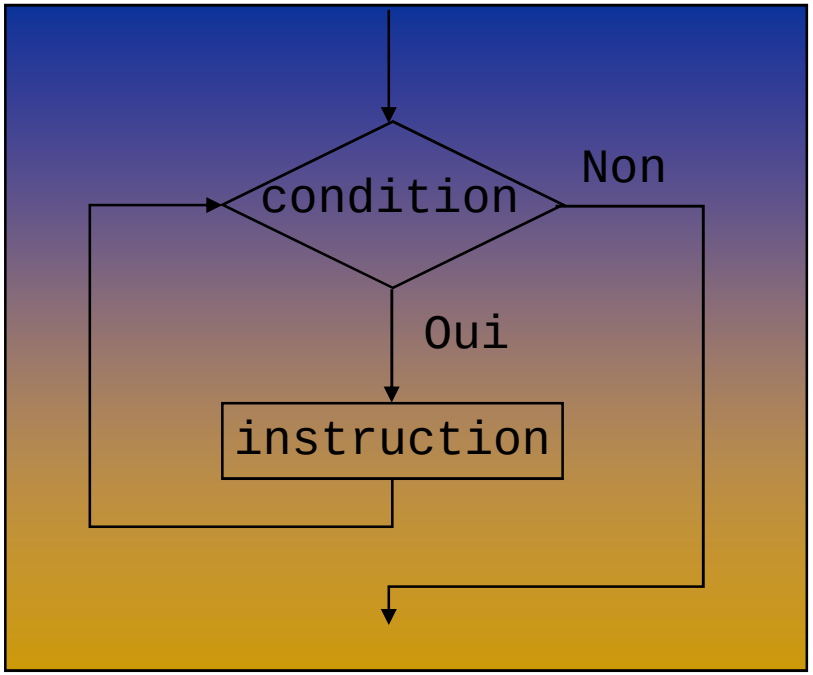
En informatique , on distingue deux types de boucles :

- Les boucles pour les quelles *le nombre d'itération n'est pas connu à l'avance*. Pour ce type de boucles le nombre d'itération est déterminé par la réalisation d'une certaine condition (Les structures TANT QUE et REPETER),
- Boucles pour les quelles *le nombre d'itération est connu à l'avance* (la structure POUR)

L'instruction Tant que … faire … FinTantque

Syntaxe :

```
Tant que (condition) Faire
   instruction
FinTantque
```



L'instruction est exécutée tant que la condition est vérifiée (valeur booléenne à VRAI). Le test est effectué à l'entrée de la boucle.

Si la condition est fausse d'entrée, l'instruction n'est jamais exécutée.

L'instruction Tant que … faire … FinTantque Exemple

Problème calcul de S = 1+2+3+…+100

```
Algorithme somme
Var
   i, S : entier
Début
   /* initialisation des variables */
   i ←——1
   S ←——0
   Tant que (i < 100) Faire
       S ←—— S + i /* calcul    */
       i ←—— i + 1 /* évolution */
   FinTantque
   Écrire('la somme de 1 à 100 est', S)
Fin
```

L'instruction Tant que … faire … FinTantque scénario de test

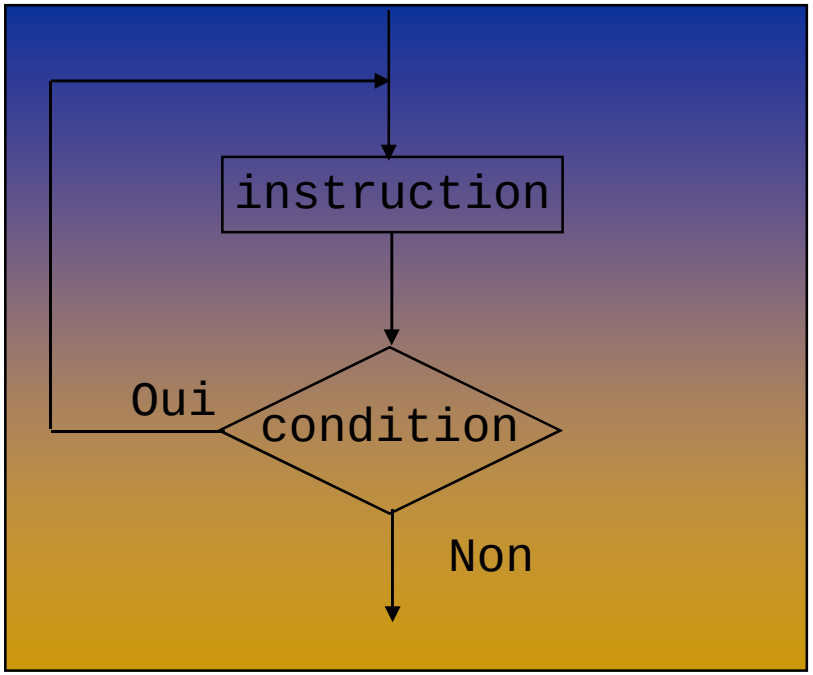
scénario de test pour le calcul de S = 1+2+3+4+5

Boucle	S	i	i < 6
Itération 0	0	1	V
Itération 1	1	2	V
Itération 2	3	3	V
Itération 3	6	4	V
Itération 4	10	5	V
Itération 5	15	6	F

L'instruction Répéter … Tant que

Syntaxe :

```
Répéter
   instruction
Tant que (condition)
```



Pour ce type de boucle, le test de contrôle est effectué à la fin la boucle.

L'instruction est exécutée tant que la condition est vérifiée (valeur booléenne à VRAI). Le test est effectué à la fin la boucle.

Si la condition est fausse d'entrée, l'instruction est tout de même exécutée au moins une fois.

L'instruction Répéter … Tant que
Exemple

Problème calcul de S = somme de N entiers saisis au clavier, le dernier entier est zéro.

```
Algorithme sommeBis
Var
   lentier, S : entier
Début
   /* initialisation des variables */
   S ←—— 0
   Répéter
       Lire(lentier)
       S ←—— S + lentier /* calcul    */
   Tant que (lentier ≠ 0)
   Écrire('la somme des entiers saisies est', S)
Fin
```

L'instruction Répéter … Tant que scénario de test

scénario de test pour le calcul de S = ∑ entiers (≠ 0)

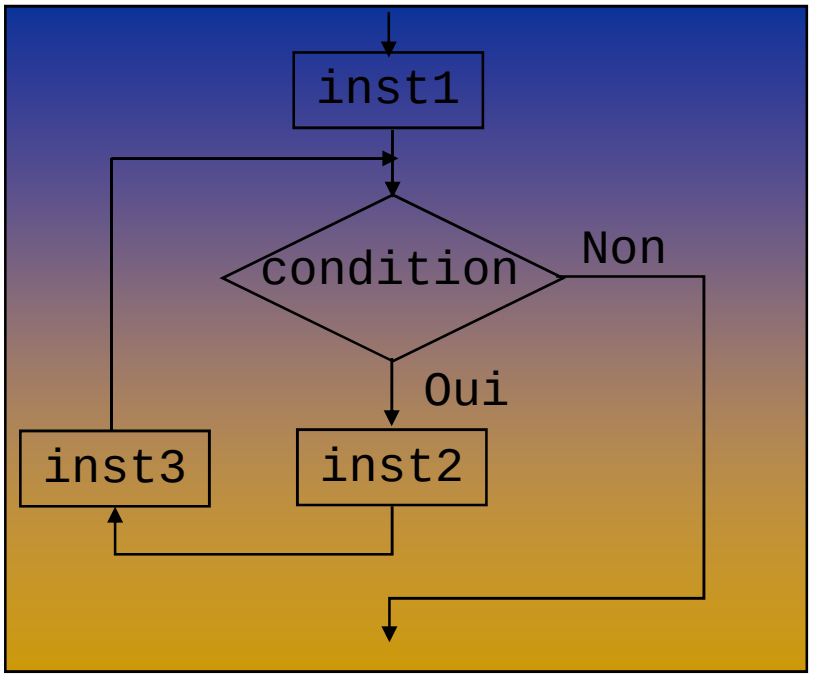
Lecteur dans l'ordre 17, 12, 5, 3 et puis 0

Boucle	lentier	S	lentier ≠0
Itération 0	17	17	V
Itération 1	12	29	V
Itération 2	5	34	V
Itération 3	3	37	V
Itération 4	0	37	F

L'instruction Pour

Syntaxe :

```
Pour(inst1; condition; inst3)
   inst2
FinPour
```



Dans ce cas le nombre d'itération est connu à l'avance.

- inst1 est exécutée, elle permet d'initialiser les variables de contrôle de la boucle.
- condition est évaluée, si sa valeur est vraie alors inst2 est exécutée. Ensuite inst3 est exécutée, elle permet de modifier les variables de contrôle de la boucle. Si la valeur de la condition est faux, la boucle s'arrête.

Exemple d'utilisation de la boucle Pour

Écrire un algorithme qui permet de calculer le factorielle d'un entier positif saisi au clavier.

```
Algorithme factorielle
Var
   lentier, i, Fact : entier
Début
   Écrire('Saisir un nombre entier')
   Lire(lentier)
   Fact ⟵ 1
   Pour (i ⟵ 1; i < lentier + 1; i ⟵ i + 1)
        Fact ⟵ Fact*i
   FinPour
   Écrire('Le factorielle est', Fact)
Fin
```

L'instruction Pour Exemple

scénario de test pour le calcul de Fact = 1*2*3*4*5

Boucle	Fact	i	i < 6
Itération 0	1		V
Itération 1	1	2	V
Itération 2	2	3	V
Itération 3	6	4	V
Itération 4	24	5	V
Itération 5	120	6	F

Mini projet (1)

Écrire un algorithme permettant de calculer la somme, la différence, le produit, le rapport de deux réels, la racine carrée ou l'inverse d'un nombre réel à partir du menu suivant :

Mini projet (2)

Fonctionnement :

L'algorithme doit contrôler le choix de l'utilisateur (entre 1 et 7), toutes les autres valeurs doivent être signalées comme message d'erreur ''Votre choix est incorrect'', Dans le cas d'un mauvais choix, vous donnez la possibilité à l'utilisateur de retourner au menu.

Lorsque une opération est terminée, on doit aussi retourner au menu principal.

Bibliographie et Webographie

- www.u-picardie.fr_ferment\initiation\index.html
- Algorithmique et technique de programmation par K. Mansouri, édition Toubkal et IGA-2003