Algorithmique
Cours 2

IUT Informatique de Lens, 1ère Année
Université d'Artois

Frédéric Koriche
koriche@cril.fr
2011 - Semestre 1

Sommaire

L'objectif de ce cours est d'étudier en pseudo-code les types de données simples et les principales instructions.

Sommaire

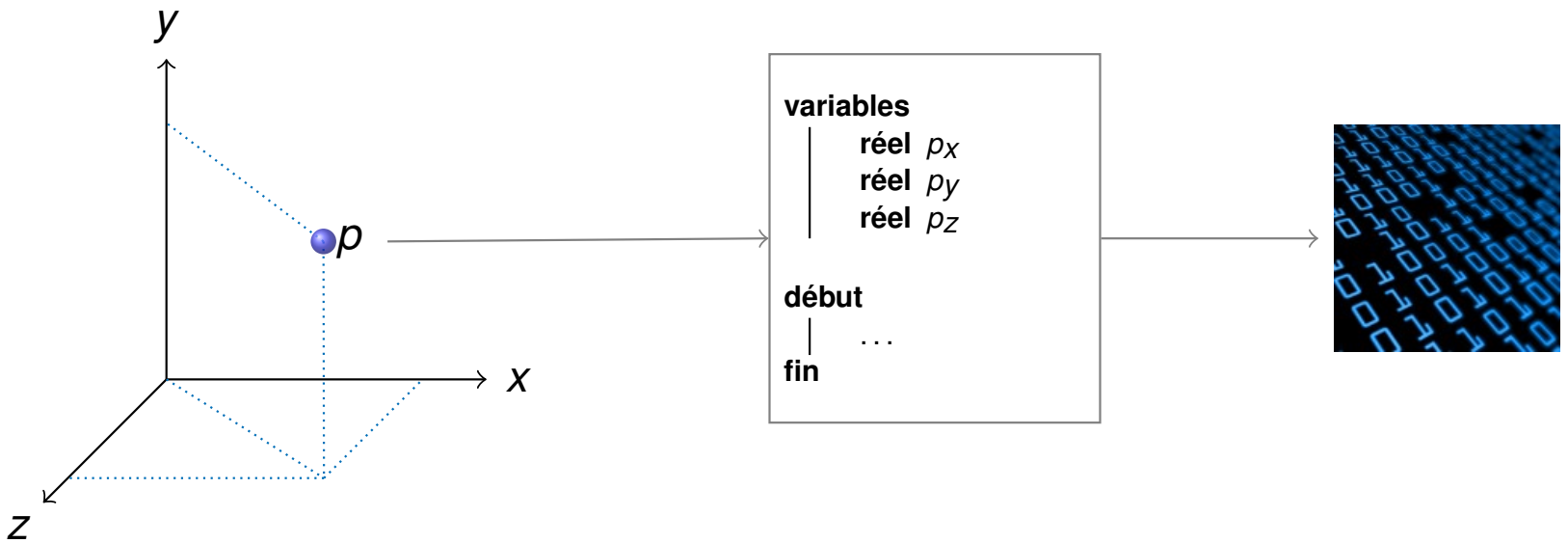
L'objectif de ce cours est d'étudier en pseudo-code les types de données simples et les principales instructions.

Données

En informatique, les objets (abstraits ou concrets) du monde sont représentés par des **données**. Chaque donnée est **codée** et **stockée** dans une mémoire.

Données

En informatique, les objets (abstraits ou concrets) du monde sont représentés par des **données**. Chaque donnée est **codée** et **stockée** dans une mémoire.



Par exemple, un point en trois dimensions est représenté par trois variables de type réel, chacune étant codée et stockée dans la mémoire

Définir une donnée

Toute donnée est spécifiée par :

- son **nom** : désigne la donnée dans l'algorithme (doit être compréhensible).
- son **type** : décrit le domaine de valeurs que peut prendre la donnée.
- sa **nature** : variable (la valeur peut changer) ou constante (la valeur est fixe).

Définir une donnée

Toute donnée est spécifiée par :

- son **nom** : désigne la donnée dans l'algorithme (doit être compréhensible).
- son **type** : décrit le domaine de valeurs que peut prendre la donnée.
- sa **nature** : variable (la valeur peut changer) ou constante (la valeur est fixe).

Données simples

Type	Domaine de valeurs
booléen	{vrai,faux} noté $\mathbb{B}$
caractère	ensemble ordonné des symboles ASCII
entier	$\mathbb{Z}$
réel	$\mathbb{R}$

Définir une donnée

Toute donnée est spécifiée par :

- son **nom** : désigne la donnée dans l'algorithme (doit être compréhensible).
- son **type** : décrit le domaine de valeurs que peut prendre la donnée.
- sa **nature** : variable (la valeur peut changer) ou constante (la valeur est fixe).

Données simples

Type	Domaine de valeurs
booléen	{vrai,faux} noté $\mathbb{B}$
caractère	ensemble ordonné des symboles ASCII
entier	$\mathbb{Z}$
réel	$\mathbb{R}$

Données structurées

Type	Désignation
tableau (statique)	séquence de valeurs de longueur constante
chaîne	séquence de caractères de longueur variable

Opérateurs

Un opérateur est une **fonction** : il associe à une série de valeurs d'entrée une valeur de sortie.

- L'**arité** d'un opérateur est donnée par le nombre de ses entrées
- La **position** d'un opérateur peut être préfixe (devant), infixe (milieu) ou postfixe (derrière)
- Le **type** d'un opérateur est donné par le type de ses entrées et celui de sa sortie.

Opérateurs

Un opérateur est une **fonction** : il associe à une série de valeurs d'entrée une valeur de sortie.

- L'**arité** d'un opérateur est donnée par le nombre de ses entrées
- La **position** d'un opérateur peut être préfixe (devant), infixe (milieu) ou postfixe (derrière)
- Le **type** d'un opérateur est donné par le type de ses entrées et celui de sa sortie.

Exemple : L'opérateur d'addition sur les entiers

Opérateur binaire, noté + (position infixe), dont le type est :

$$+ : \textbf{entier} \times \textbf{entier} \rightarrow \textbf{entier}$$

Opérateurs

Un opérateur est une **fonction** : il associe à une série de valeurs d'entrée une valeur de sortie.

- L'**arité** d'un opérateur est donnée par le nombre de ses entrées
- La **position** d'un opérateur peut être préfixe (devant), infixe (milieu) ou postfixe (derrière)
- Le **type** d'un opérateur est donné par le type de ses entrées et celui de sa sortie.

Exemple : L'opérateur d'addition sur les entiers

Opérateur binaire, noté + (position infixe), dont le type est :

$$+ : \textbf{entier} \times \textbf{entier} \rightarrow \textbf{entier}$$

Exemple : L'opérateur d'inversion de signe sur les réels

Opérateur unaire, noté − (position préfixe), dont le type est :

$$- : \textbf{réel} \rightarrow \textbf{réel}$$

Opérateurs arithmétiques (entiers)

Toutes les entrées sont des **entiers**
La sortie est un **entier**

Nom	Symbole
addition	$+$
soustraction	$-$
multiplication	$\times$
division entière	$/$
reste	mod
inversion de signe	$-$

Opérateurs arithmétiques (entiers)

Toutes les entrées sont des **entiers**
La sortie est un **entier**

Nom	Symbole
addition	+
soustraction	−
multiplication	×
division entière	/
reste	mod
inversion de signe	−

Opérateurs arithmétiques (réels)

Au moins une entrée est un **réel**
La sortie est un **réel**

Nom	Symbole
addition	+
soustraction	−
multiplication	×
division réelle	/
signe inverse	−

Opérateurs arithmétiques (entiers)

Toutes les entrées sont des **entiers**
La sortie est un **entier**

Nom	Symbole
addition	+
soustraction	−
multiplication	×
division entière	/
reste	mod
inversion de signe	−

Opérateurs arithmétiques (réels)

Au moins une entrée est un **réel**
La sortie est un **réel**

Nom	Symbole
addition	+
soustraction	−
multiplication	×
division réelle	/
signe inverse	−

Opérateurs de comparaison

Les deux entrées sont des **entiers**, **caractères*** ou **réels**
La sortie est un **booléen**

Nom	Symbole
est égal à	=
est plus petit que	<
est plus grand que	>
est plus petit ou égal à	≤
est plus grand ou égal à	≥

*Les caractères ASCII sont comparables !

Opérateurs arithmétiques (entiers)

Toutes les entrées sont des **entiers**
La sortie est un **entier**

Nom	Symbole
addition	+
soustraction	−
multiplication	×
division entière	/
reste	mod
inversion de signe	−

Opérateurs arithmétiques (réels)

Au moins une entrée est un **réel**
La sortie est un **réel**

Nom	Symbole
addition	+
soustraction	−
multiplication	×
division réelle	/
signe inverse	−

Opérateurs de comparaison

Les deux entrées sont des **entiers**, **caractères*** ou **réels**
La sortie est un **booléen**

Nom	Symbole
est égal à	=
est plus petit que	<
est plus grand que	>
est plus petit ou égal à	≤
est plus grand ou égal à	≥

*Les caractères ASCII sont comparables !

Opérateurs logiques

Les deux entrées sont des **booléens**
La sortie est un **booléen**

Nom	Symbole
conjonction	et
disjonction	ou
négation	non

Expression

Une expression est une **composition** d'opérations.

- Une expression est **bien formée** si chaque opération s'applique sur des entrées de type autorisé
- Le **type** d'une expression est donné par le type de la valeur de sortie

Expression

Une expression est une **composition** d'opérations.

- Une expression est **bien formée** si chaque opération s'applique sur des entrées de type autorisé
- Le **type** d'une expression est donné par le type de la valeur de sortie

Exemple

Supposons que i et j soient des entiers.

- $(i > 0)$ **et** $(j < 0)$ est une expression bien formée de type booléen
- $(i + j)$ **ou** $(j < 0)$ est une expression mal formée !

Expression

Une expression est une **composition** d'opérations.

- Une expression est **bien formée** si chaque opération s'applique sur des entrées de type autorisé
- Le **type** d'une expression est donné par le type de la valeur de sortie

Exemple

Supposons que i et j soient des entiers.

- $(i > 0)$ **et** $(j < 0)$ est une expression bien formée de type booléen
- $(i + j)$ **ou** $(j < 0)$ est une expression mal formée !

Recommandation

Utiliser les parenthèses pour expliciter l'ordre des combinaisons d'opérations.

$(x$ **ou** $y)$ **et** $(u$ **ou** $v)$ est une expression différente de x **ou** $(y$ **et** $u)$ **ou** v

La déclaration

L'instruction de **déclaration** consiste à définir une donnée, avec son nom, son type et sa nature.

La déclaration

L'instruction de **déclaration** consiste à définir une donnée, avec son nom, son type et sa nature.

Exemple

Un début d'algorithme

```
constantes
    réel pi ← 3.14158
    entier c ← 299792458

variables
    réel x, y
    entier m, n

début
    ...
fin
```

La déclaration

L'instruction de **déclaration** consiste à définir une donnée, avec son nom, son type et sa nature.

Exemple

Un début d'algorithme

```
constantes
    réel pi ← 3.14158
    entier c ← 299792458

variables
    réel x, y
    entier m, n

début
    ...
fin
```

Bloc de déclaration des constantes, chacune étant affectée à une valeur

La déclaration

L'instruction de **déclaration** consiste à définir une donnée, avec son nom, son type et sa nature.

Exemple

Un début d'algorithme

```
constantes
    réel pi ← 3.14158
    entier c ← 299792458

variables
    réel x, y
    entier m, n

début
    ...
fin
```

Bloc de déclaration des constantes, chacune étant affectée à une valeur

Bloc de déclaration des variables

L'affectation

L'instruction d'**affectation** consiste à assigner une valeur à une donnée.

L'affectation

L'instruction d'**affectation** consiste à assigner une valeur à une donnée.

Exemple

Un algorithme

variables
  réel x, y

début
  $x \leftarrow 2.0$
  $y \leftarrow 4.0$
  $y \leftarrow y + 1$
  $x \leftarrow y$
fin

L'affectation

L'instruction d'**affectation** consiste à assigner une valeur à une donnée.

Exemple

Un algorithme

variables
  réel x, y

début
  $x \leftarrow 2.0$ ← x prend la valeur 2.0
  $y \leftarrow 4.0$
  $y \leftarrow y + 1$
  $x \leftarrow y$
fin

L'affectation

L'instruction d'**affectation** consiste à assigner une valeur à une donnée.

Exemple

Un algorithme

variables
  réel x, y

début
  $x \leftarrow 2.0$
  $y \leftarrow 4.0$ ← y prend la valeur 4.0
  $y \leftarrow y + 1$
  $x \leftarrow y$
fin

L'affectation

L'instruction d'**affectation** consiste à assigner une valeur à une donnée.

Exemple

Un algorithme

variables
  réel x, y

début
  $x \leftarrow 2.0$
  $y \leftarrow 4.0$
  $y \leftarrow y + 1$ ← y est incrémenté de 1 et prend donc la valeur 5.0
  $x \leftarrow y$
fin

L'affectation

L'instruction d'**affectation** consiste à assigner une valeur à une donnée.

Exemple

Un algorithme

variables
  réel x, y

début
  $x \leftarrow 2.0$
  $y \leftarrow 4.0$
  $y \leftarrow y + 1$
  $x \leftarrow y$ ← x prend la valeur de y c'est-à-dire 5.0
fin

Les entrées-sorties

- L'instruction **lire** *x* permet de recevoir une valeur entrée au clavier et de l'affecter à *x*
- L'instruction **afficher** *x* permet d'afficher à l'écran la valeur de *x*

Les entrées-sorties

- L'instruction **lire** *x* permet de recevoir une valeur entrée au clavier et de l'affecter à *x*
- L'instruction **afficher** *x* permet d'afficher à l'écran la valeur de *x*

Exemple

Algorithme : calculeVitesse

variables
  réel distance, temps, vitesse

début
  afficher "Distance : "
  lire distance ;
  afficher "Temps : "
  lire temps ;
  vitesse ⟵ distance / temps
  afficher "Vitesse : " vitesse
fin

Les entrées-sorties

- L'instruction **lire** *x* permet de recevoir une valeur entrée au clavier et de l'affecter à *x*
- L'instruction **afficher** *x* permet d'afficher à l'écran la valeur de *x*

Exemple

Algorithme : calculeVitesse

variables
  réel distance, temps, vitesse

début
  afficher "Distance : "
  lire distance ;
  afficher "Temps : "
  lire temps ;
  vitesse ⟵ distance / temps
  afficher "Vitesse : " vitesse
fin

L'algorithme indique à l'écran d'afficher la chaîne Distance :

Les entrées-sorties

- L'instruction **lire** *x* permet de recevoir une valeur entrée au clavier et de l'affecter à *x*
- L'instruction **afficher** *x* permet d'afficher à l'écran la valeur de *x*

Exemple

Algorithme : calculeVitesse

variables
  réel distance, temps, vitesse

début
  afficher "Distance : "
  lire distance ;
  afficher "Temps : "
  lire temps ;
  vitesse ⟵ distance / temps
  afficher "Vitesse : " vitesse
fin

L'algorithme lit la valeur entrée au clavier et l'affecte à la variable distance

Les entrées-sorties

- L'instruction **lire** *x* permet de recevoir une valeur entrée au clavier et de l'affecter à *x*
- L'instruction **afficher** *x* permet d'afficher à l'écran la valeur de *x*

Exemple

Algorithme : calculeVitesse

```
variables
    réel distance, temps, vitesse

début
    afficher "Distance : "
    lire distance ;
    afficher "Temps : "
    lire temps ;
    vitesse ⟵ distance / temps
    afficher "Vitesse : " vitesse
fin
```

L'algorithme indique à l'écran d'afficher la chaîne Vitesse : suivie de la valeur de la variable vitesse

Le bloc

Le **bloc d'instruction** est une liste ordonnée d'instructions : il est identifié par une barre verticale terminée pas le mot-clé **fin**. On peut omettre la barre (et **fin**) si le bloc se limite à une seule instruction.

Le bloc

Le **bloc d'instruction** est une liste ordonnée d'instructions : il est identifié par une barre verticale terminée pas le mot-clé **fin**. On peut omettre la barre (et **fin**) si le bloc se limite à une seule instruction.

Exemple

Un corps d'algorithme

```
début
    ...
    si a < b alors
        afficher "correct"
    sinon
        c ← a
        a ← b
        b ← c
    fin
    ...
fin
```

Le bloc

Le **bloc d'instruction** est une liste ordonnée d'instructions : il est identifié par une barre verticale terminée pas le mot-clé **fin**. On peut omettre la barre (et **fin**) si le bloc se limite à une seule instruction.

Exemple

Un corps d'algorithme

```
début
    ...
    si a < b alors
        afficher "correct"
    sinon
        c ← a
        a ← b
        b ← c
    fin
    ...
fin
```

Bloc d'instructions de l'algorithme

Le bloc

Le **bloc d'instruction** est une liste ordonnée d'instructions : il est identifié par une barre verticale terminée pas le mot-clé **fin**. On peut omettre la barre (et **fin**) si le bloc se limite à une seule instruction.

Exemple

Un corps d'algorithme

```
début
|   ...
|   si a < b alors
|   |   afficher "correct"
|   sinon
|   |   c ← a
|   |   a ← b
|   |   b ← c
|   fin
|   ...
fin
```

Bloc d'instructions de l'algorithme

Bloc d'instructions du "si alors sinon"

L'instruction si alors

Dans l'instruction **si condition alors bloc**, la condition est une expression booléenne, et le bloc n'est exécuté que si la condition est vraie.

L'instruction si alors

Dans l'instruction **si** **condition** **alors** **bloc**, la condition est une expression booléenne, et le bloc n'est exécuté que si la condition est vraie.

Exemple

Algorithme : valeurAbsolue

```
variable
    réel x, y

début
    lire x
    y ← x
    si y < 0 alors
        y ← −y
    fin
    afficher y
fin
```

L'instruction si alors

Dans l'instruction **si** **condition** **alors** **bloc**, la condition est une expression booléenne, et le bloc n'est exécuté que si la condition est vraie.

Exemple

Algorithme : valeurAbsolue

```
variable
    réel x, y

début
    lire x                          ← Supposons que x = −10.0
    y ← x
    si y < 0 alors
        y ← −y
    fin
    afficher y
fin
```

L'instruction si alors

Dans l'instruction **si** **condition** **alors** **bloc**, la condition est une expression booléenne, et le bloc n'est exécuté que si la condition est vraie.

Exemple

Algorithme : valeurAbsolue

```
variable
    réel x, y

début
    lire x                    ← Supposons que x = −10.0
    y ← x                     ← y prend la valeur −10.0
    si y < 0 alors
        y ← −y
    fin
    afficher y
fin
```

L'instruction si alors

Dans l'instruction **si** **condition** **alors** **bloc**, la condition est une expression booléenne, et le bloc n'est exécuté que si la condition est vraie.

Exemple

Algorithme : valeurAbsolue

```
variable
    réel x, y

début
    lire x                    ← Supposons que x = −10.0
    y ← x                     ← y prend la valeur −10.0
    si y < 0 alors            ← y est négatif donc le bloc est exécuté
        y ← −y
    fin
    afficher y
fin
```

L'instruction si alors

Dans l'instruction **si** **condition** **alors** **bloc**, la condition est une expression booléenne, et le bloc n'est exécuté que si la condition est vraie.

Exemple

Algorithme : valeurAbsolue

```
variable
    réel x, y

début
    lire x                  ← Supposons que x = −10.0
    y ← x                   ← y prend la valeur −10.0
    si y < 0 alors          ← y est négatif donc le bloc est exécuté
        y ← −y              ← y prend donc la valeur 10.0
    fin
    afficher y
fin
```

L'instruction si alors

Dans l'instruction **si** **condition** **alors** **bloc**, la condition est une expression booléenne, et le bloc n'est exécuté que si la condition est vraie.

Exemple

Algorithme : valeurAbsolue

```
variable
    réel x, y

début
    lire x                          ← Supposons maintenant que x = 5.0
    y ← x
    si y < 0 alors
        y ← −y
    fin
    afficher y
fin
```

L'instruction si alors

Dans l'instruction **si** **condition** **alors** **bloc**, la condition est une expression booléenne, et le bloc n'est exécuté que si la condition est vraie.

Exemple

Algorithme : valeurAbsolue

```
variable
    réel x, y

début
    lire x                  ←— Supposons maintenant que x = 5.0
    y ← x                   ←— y prend la valeur 5.0
    si y < 0 alors
        y ← −y
    fin
    afficher y
fin
```

L'instruction si alors

Dans l'instruction **si** **condition** **alors** **bloc**, la condition est une expression booléenne, et le bloc n'est exécuté que si la condition est vraie.

Exemple

Algorithme : valeurAbsolue

```
variable
    réel x, y

début
    lire x                  ←— Supposons maintenant que x = 5.0
    y ← x                   ←— y prend la valeur 5.0
    si y < 0 alors          ←— y est positif donc le bloc n'est pas exécuté
        y ← −y
    fin
    afficher y
fin
```

L'instruction si alors

Dans l'instruction **si** **condition** **alors** **bloc**, la condition est une expression booléenne, et le bloc n'est exécuté que si la condition est vraie.

Exemple

Algorithme : valeurAbsolue

```
variable
    réel x, y

début
    lire x                  ←— Supposons maintenant que x = 5.0
    y ← x                   ←— y prend la valeur 5.0
    si y < 0 alors          ←— y est positif donc le bloc n'est pas exécuté
        y ← −y
    fin
    afficher y              ←— y garde donc la valeur 5.0
fin
```

L'instruction si alors sinon

Dans l'instruction **si** condition **alors** bloc 1 **sinon** bloc 2, la condition est une expression booléenne. Le bloc 1 est exécuté si la condition est vraie ; le bloc 2 est exécuté si la condition est fausse.

L'instruction si alors sinon

Dans l'instruction **si** **condition** **alors** **bloc 1** **sinon** **bloc 2**, la condition est une expression booléenne. Le bloc 1 est exécuté si la condition est vraie ; le bloc 2 est exécuté si la condition est fausse.

Exemple

Algorithme : racineCarrée

```
variable
    réel x, y

début
    lire x
    si x ≥ 0 alors
        y ← sqrt(x)
        afficher y
    sinon
        afficher "Valeur indéfinie"
    fin
fin
```

L'instruction si alors sinon

Dans l'instruction **si** **condition** **alors** **bloc 1** **sinon** **bloc 2**, la condition est une expression booléenne. Le bloc 1 est exécuté si la condition est vraie ; le bloc 2 est exécuté si la condition est fausse.

Exemple

Algorithme : racineCarrée

```
variable
    réel x, y

début
    lire x                          ← Supposons que x = 2.0
    si x ≥ 0 alors
        y ← sqrt(x)
        afficher y
    sinon
        afficher "Valeur indéfinie"
    fin
fin
```

L'instruction si alors sinon

Dans l'instruction **si** **condition** **alors** **bloc 1** **sinon** **bloc 2**, la condition est une expression booléenne. Le bloc 1 est exécuté si la condition est vraie ; le bloc 2 est exécuté si la condition est fausse.

Exemple

Algorithme : racineCarrée

```
variable
    réel x, y

début
    lire x                          ← Supposons que x = 2.0
    si x ≥ 0 alors                  ← x est positif donc le bloc alors est exécuté
        y ← sqrt(x)
        afficher y
    sinon
        afficher "Valeur indéfinie"
    fin
fin
```

L'instruction si alors sinon

Dans l'instruction **si** **condition** **alors** **bloc 1** **sinon** **bloc 2**, la condition est une expression booléenne. Le bloc 1 est exécuté si la condition est vraie ; le bloc 2 est exécuté si la condition est fausse.

Exemple

Algorithme : racineCarrée

```
variable
    réel x, y

début
    lire x                          ← Supposons que x = 2.0
    si x ≥ 0 alors                  ← x est positif donc le bloc alors est exécuté
        y ← sqrt(x)                 ← y prend la valeur 1.414…
        afficher y
    sinon
        afficher "Valeur indéfinie"
    fin
fin
```

L'instruction si alors sinon

Dans l'instruction **si** **condition** **alors** **bloc 1** **sinon** **bloc 2**, la condition est une expression booléenne. Le bloc 1 est exécuté si la condition est vraie ; le bloc 2 est exécuté si la condition est fausse.

Exemple

Algorithme : racineCarrée

variable
  réel x, y

début
  lire x ← Supposons maintenant que $x = -1.0$
  si $x \geq 0$ **alors**
    $y \leftarrow \text{sqrt}(x)$
    afficher y
  sinon
    afficher "Valeur indéfinie"
  fin
fin

L'instruction si alors sinon

Dans l'instruction **si** **condition** **alors** **bloc 1** **sinon** **bloc 2**, la condition est une expression booléenne. Le bloc 1 est exécuté si la condition est vraie ; le bloc 2 est exécuté si la condition est fausse.

Exemple

Algorithme : racineCarrée

variable
  réel x, y

début
  lire x ← Supposons maintenant que $x = -1.0$
  si $x \geq 0$ **alors** ← x est négatif donc le bloc sinon est exécuté
    $y \leftarrow \text{sqrt}(x)$
    afficher y
  sinon
    afficher "Valeur indéfinie"
  fin
fin

L'instruction si alors sinon

Dans l'instruction **si** **condition** **alors** **bloc 1** **sinon** **bloc 2**, la condition est une expression booléenne. Le bloc 1 est exécuté si la condition est vraie ; le bloc 2 est exécuté si la condition est fausse.

Exemple

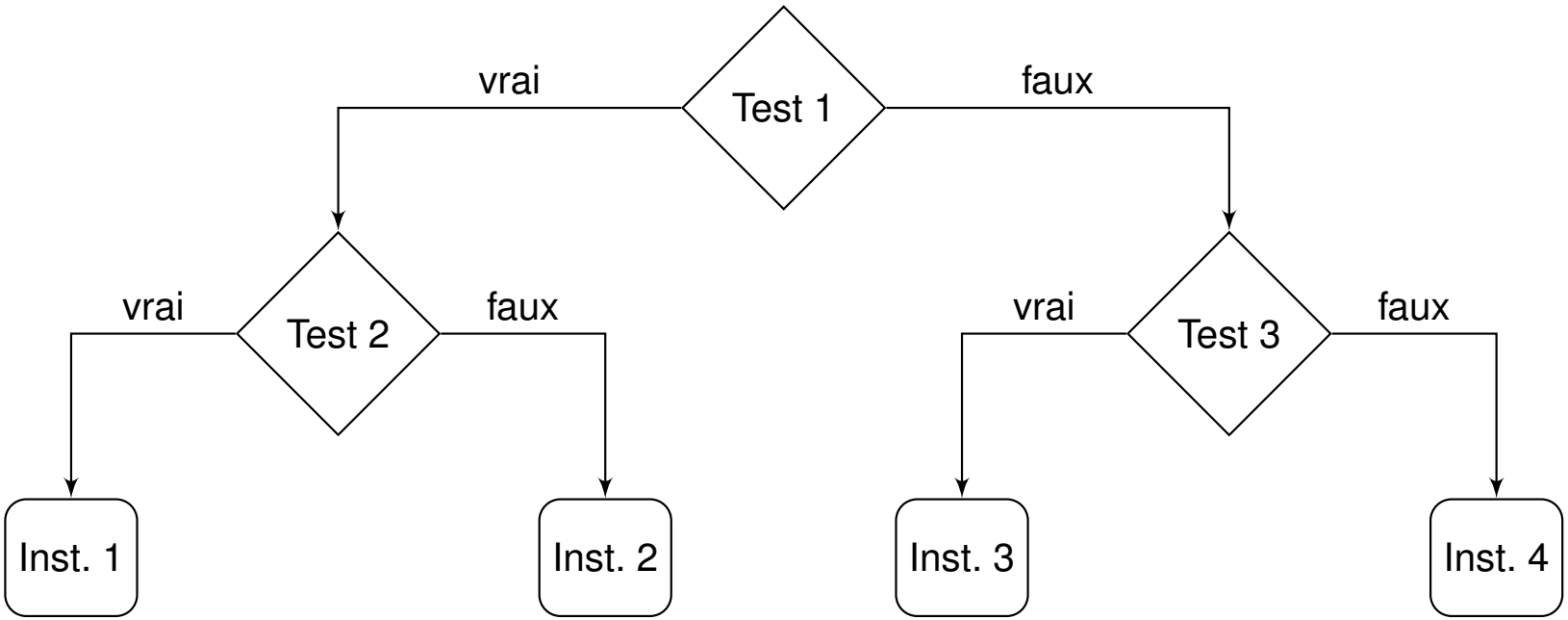
Algorithme : racineCarrée

```
variable
    réel x, y

début
    lire x                          ← Supposons maintenant que x = −1.0
    si x ≥ 0 alors                  ← x est négatif donc le bloc sinon est exécuté
        y ← sqrt(x)
        afficher y
    sinon
        afficher "Valeur indéfinie" ← L'algorithme affiche Valeur indéfinie et aucune valeur n'est affectée à y
    fin
fin
```

Arbres de Décision

Un arbre de décision est une représentation graphique des **tests imbriqués**.

- chaque sommet interne représente la condition d'un "si alors (sinon)".
- chaque feuille représente un bloc d'instructions.

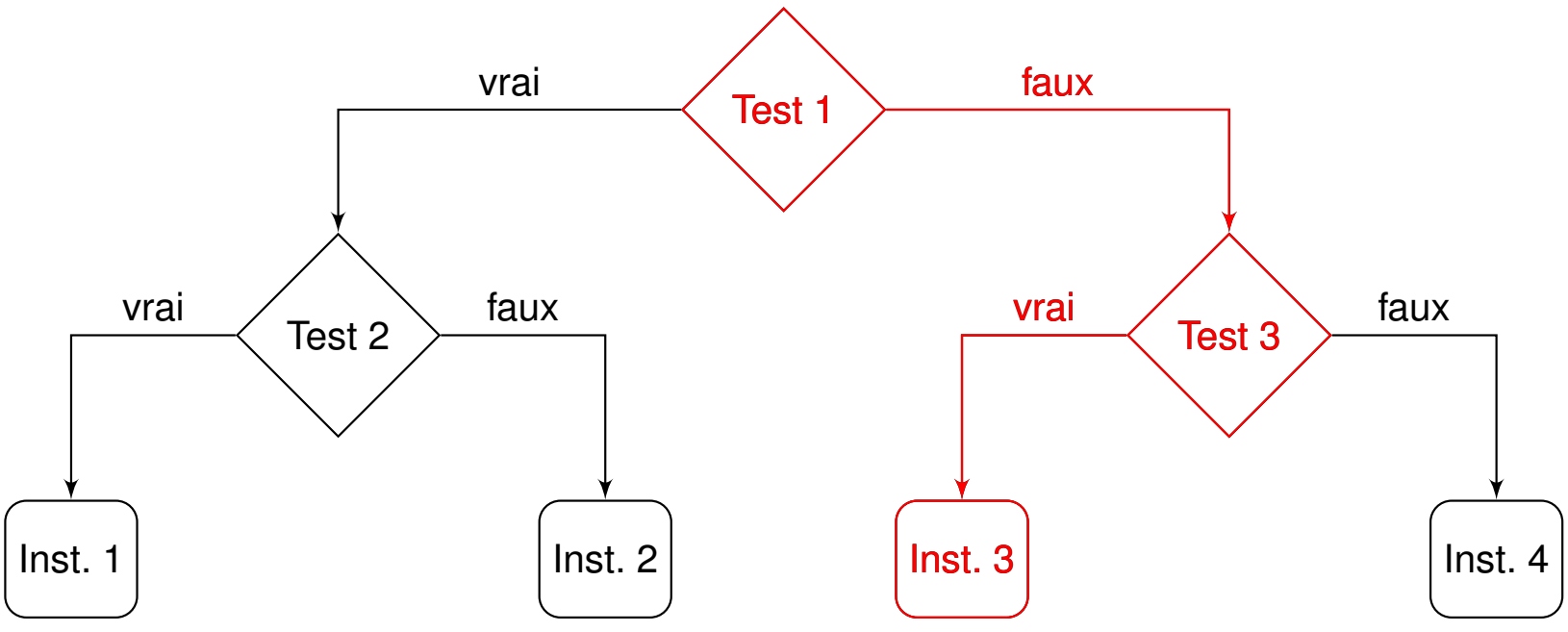


Arbres de Décision

Un arbre de décision est une représentation graphique des **tests imbriqués**.

- chaque sommet interne représente la condition d'un "si alors (sinon)".
- chaque feuille représente un bloc d'instructions.

L'exécution de l'algorithme est un chemin depuis la racine jusqu'à l'une des feuilles.



Un algorithme et son arbre de décision

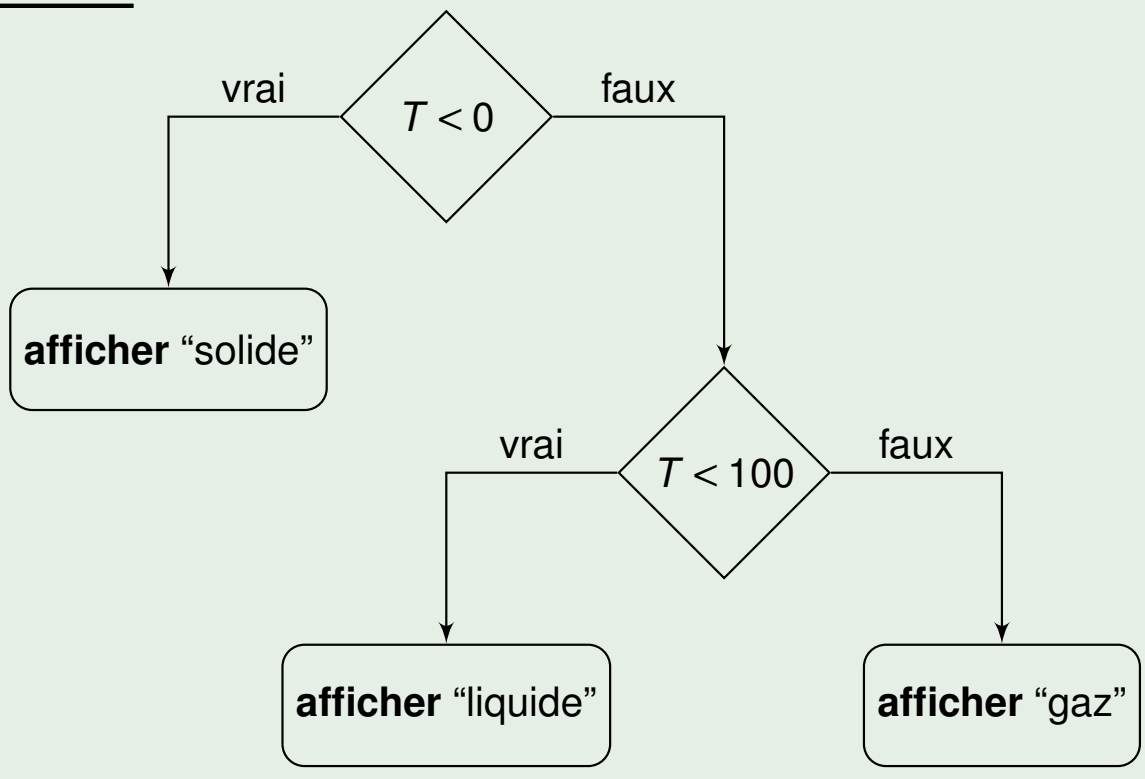
Algorithme : étatDeLeau

```
variable
    réel T

début
    lire T
    si T < 0 alors
        afficher "solide"
    sinon
        si T < 100 alors
            afficher "liquide"
        sinon
            afficher "gaz"
        fin
    fin
fin
```

Un algorithme et son arbre de décision

Algorithme : étatDeLeau

```
variable
    réel T

début
    lire T
    si T < 0 alors
        afficher "solide"
    sinon
        si T < 100 alors
            afficher "liquide"
        sinon
            afficher "gaz"
        fin
    fin
fin
```



L'instruction suivant cas

Dans l'instruction **suivant condition cas où** v_1 **bloc 1 cas où** v_2 **bloc 2** ..., la condition est une expression pouvant prendre plusieurs valeurs $v_1, v_2, \ldots$. Selon la valeur de la condition, le bloc du cas correspondant est exécuté.

L'instruction suivant cas

Dans l'instruction **suivant** **condition** **cas où** v_1 **bloc 1** **cas où** v_2 **bloc 2** . . . , la condition est une expression pouvant prendre plusieurs valeurs $v_1, v_2, \ldots$. Selon la valeur de la condition, le bloc du cas correspondant est exécuté.

Exemple

Algorithme : choixDuMenu

```
variable
    entier menu

début
    lire menu
    suivant menu faire
        cas où 1
            afficher "Menu enfants"
        fin
        cas où 2
            afficher "Menu végétarien"
        fin
        autres cas
            afficher "Menu standard"
        fin
    fin
fin
```

L'instruction suivant cas

Dans l'instruction **suivant** **condition** **cas où** v_1 **bloc 1** **cas où** v_2 **bloc 2** . . . , la condition est une expression pouvant prendre plusieurs valeurs $v_1, v_2, \ldots$. Selon la valeur de la condition, le bloc du cas correspondant est exécuté.

Exemple

Algorithme : choixDuMenu

```
variable
    entier menu

début
    lire menu                          ← Supposons que menu = 1
    suivant menu faire
        cas où 1
            afficher "Menu enfants"
        fin
        cas où 2
            afficher "Menu végétarien"
        fin
        autres cas
            afficher "Menu standard"
        fin
    fin
fin
```

L'instruction suivant cas

Dans l'instruction **suivant** **condition** **cas où** v_1 **bloc 1** **cas où** v_2 **bloc 2** . . . , la condition est une expression pouvant prendre plusieurs valeurs $v_1, v_2, \ldots$. Selon la valeur de la condition, le bloc du cas correspondant est exécuté.

Exemple

Algorithme : choixDuMenu

```
variable
    entier menu

début
    lire menu                          ← Supposons que menu = 1
    suivant menu faire
        cas où 1                       ← le premier cas est traité
            afficher "Menu enfants"
        fin
        cas où 2
            afficher "Menu végétarien"
        fin
        autres cas
            afficher "Menu standard"
        fin
    fin
fin
```

L'instruction suivant cas

Dans l'instruction **suivant** **condition** **cas où** v_1 **bloc 1** **cas où** v_2 **bloc 2** ..., la condition est une expression pouvant prendre plusieurs valeurs $v_1, v_2, \ldots$. Selon la valeur de la condition, le bloc du cas correspondant est exécuté.

Exemple

Algorithme : choixDuMenu

```
variable
    entier menu

début
    lire menu                          ← Supposons que menu = 1
    suivant menu faire
        cas où 1                       ← le premier cas est traité
            afficher "Menu enfants"    ← l'algorithme affiche "Menu enfants"
        fin
        cas où 2
            afficher "Menu végétarien"
        fin
        autres cas
            afficher "Menu standard"
        fin
    fin
fin
```

L'instruction suivant cas

Dans l'instruction **suivant** **condition** **cas où** v_1 **bloc 1** **cas où** v_2 **bloc 2** ..., la condition est une expression pouvant prendre plusieurs valeurs $v_1, v_2, \ldots$. Selon la valeur de la condition, le bloc du cas correspondant est exécuté.

Exemple

Algorithme : choixDuMenu

```
variable
    entier menu

début
    lire menu                          ← Supposons maintenant que menu = 4
    suivant menu faire
        cas où 1
            afficher "Menu enfants"
        fin
        cas où 2
            afficher "Menu végétarien"
        fin
        autres cas
            afficher "Menu standard"
        fin
    fin
fin
```

L'instruction suivant cas

Dans l'instruction **suivant** **condition** **cas où** v_1 **bloc 1** **cas où** v_2 **bloc 2** . . . , la condition est une expression pouvant prendre plusieurs valeurs $v_1, v_2, \ldots$. Selon la valeur de la condition, le bloc du cas correspondant est exécuté.

Exemple

Algorithme : choixDuMenu

```
variable
    entier menu

début
    lire menu                          ← Supposons maintenant que menu = 4
    suivant menu faire
        cas où 1
            afficher "Menu enfants"
        fin
        cas où 2
            afficher "Menu végétarien"
        fin
        autres cas                     ← le bloc par défaut est traité
            afficher "Menu standard"
        fin
    fin
fin
```

L'instruction suivant cas

Dans l'instruction **suivant** **condition** **cas où** v_1 **bloc 1** **cas où** v_2 **bloc 2** . . . , la condition est une expression pouvant prendre plusieurs valeurs $v_1, v_2, \ldots$. Selon la valeur de la condition, le bloc du cas correspondant est exécuté.

Exemple

Algorithme : choixDuMenu

```
variable
    entier menu

début
    lire menu                          ← Supposons maintenant que menu = 4
    suivant menu faire
        cas où 1
            afficher "Menu enfants"
        fin
        cas où 2
            afficher "Menu végétarien"
        fin
        autres cas                     ← le bloc par défaut est traité
            afficher "Menu standard"   ← l'algorithme affiche "Menu standard"
        fin
    fin
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n
    s ← 0
    pour i ← 1 à n faire
        s ← s + i
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                          ← Supposons que n = 3
    s ← 0
    pour i ← 1 à n faire
        s ← s + i
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                    ← Supposons que n = 3
    s ← 0                     ← La somme s est initialisée à 0
    pour i ← 1 à n faire
        s ← s + i
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                          ← Supposons que n = 3
    s ← 0                           ← La somme s est initialisée à 0
    pour i ← 1 à n faire            ← Le compteur i est initialisé à 1
        s ← s + i
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                          <-- Supposons que n = 3
    s ← 0                           <-- La somme s est initialisée à 0
    pour i ← 1 à n faire            <-- Le compteur i est initialisé à 1
        s ← s + i                   <-- La somme s prend la valeur 0 + 1 = 1
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                          ← Supposons que n = 3
    s ← 0                           ← La somme s est initialisée à 0
    pour i ← 1 à n faire            ← Le compteur i passe à la valeur 2
        s ← s + i
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                          ← Supposons que n = 3
    s ← 0                           ← La somme s est initialisée à 0
    pour i ← 1 à n faire            ← Le compteur i passe à la valeur 2
        s ← s + i                   ← La somme s prend la valeur 1 + 2 = 3
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                          ← Supposons que n = 3
    s ← 0                           ← La somme s est initialisée à 0
    pour i ← 1 à n faire            ← Le compteur i passe à la valeur 3
        s ← s + i
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                          ← Supposons que n = 3
    s ← 0                           ← La somme s est initialisée à 0
    pour i ← 1 à n faire            ← Le compteur i passe à la valeur 3
        s ← s + i                   ← La somme s prend la valeur 3 + 3 = 6
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                          ← Supposons que n = 3
    s ← 0                           ← La somme s est initialisée à 0
    pour i ← 1 à n faire
        s ← s + i
    fin
    afficher s                      ← La somme finale est donc s = 6
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                                  ← Supposons maintenant que n = 10
    s ← 0
    pour i ← 1 à n faire
        s ← s + i
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                  ← Supposons maintenant que n = 10
    s ← 0                   ← La somme s est initialisée à 0
    pour i ← 1 à n faire
        s ← s + i
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                          ← Supposons maintenant que n = 10
    s ← 0                           ← La somme s est initialisée à 0
    pour i ← 1 à n faire            ← La boucle est exécutée 10 fois
        s ← s + i
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                          ← Supposons maintenant que n = 10
    s ← 0                           ← La somme s est initialisée à 0
    pour i ← 1 à n faire            ← La boucle est exécutée 10 fois
        s ← s + i                   ← La somme s prend la valeur 0 + 1 + ··· + 10
    fin
    afficher s
fin
```

L'instruction pour

L'instruction **pour** est utilisée lorsque le nombre d'itérations est connu :

- elle initialise un **compteur**
- elle incrémente le compteur après chaque exécution du bloc d'instructions
- elle vérifie que le compteur ne dépasse pas la borne supérieure

Exemple

Un algorithme pour la somme de 1 à n

```
variable
    entier n, s, i

début
    lire n                          ← Supposons maintenant que n = 10
    s ← 0                           ← La somme s est initialisée à 0
    pour i ← 1 à n faire            ← La boucle est exécutée 10 fois
        s ← s + i                   ← La somme s prend la valeur 0 + 1 + ··· + 10
    fin
    afficher s                      ← La somme finale est donc s = 55
fin
```

L'instruction pour

Dans une instruction **pour**, le compteur ne doit jamais être modifié par le bloc d'instructions

L'instruction pour

Dans une instruction **pour**, le compteur ne doit jamais être modifié par le bloc d'instructions

Exemple

Un algorithme mal construit

```
variable
    entier n, s, i

début
    ...
    pour i ← 1 à n faire
        ...
        si s = 10 alors i = 1
        ...
    fin
    ...
fin
```

L'instruction pour

Dans une instruction **pour**, le compteur ne doit jamais être modifié par le bloc d'instructions

Exemple

Un algorithme mal construit

```
variable
    entier n, s, i

début
    ...
    pour i ← 1 à n faire
        ...
        si ... alors i = 1
        ...
    fin
    ...
fin
```

i ne doit pas être modifié !

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n
        s ← s + n
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0                          ← La somme s est initialisée à 0
    n ← 1
    tant que n ≠ 0 faire
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n
        s ← s + n
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0          ← La somme s est initialisée à 0
    n ← 1          ← L'entrée n est initialisée à 1
    tant que n ≠ 0 faire
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n
        s ← s + n
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0                          ← La somme s est initialisée à 0
    n ← 1                          ← L'entrée n est initialisée à 1
    tant que n ≠ 0 faire           ← Comme n = 1, la condition est vraie
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n
        s ← s + n
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n                    ← Supposons que la saisie de n soit 2
        s ← s + n
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n          ← Supposons que la saisie de n soit 2
        s ← s + n       ← La somme s prend la valeur 0 + 2 = 2
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire          ← Comme n = 2, la condition est vraie
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n                    ← Supposons que la saisie de n soit 2
        s ← s + n                 ← La somme s prend la valeur 0 + 2 = 2
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n                    ← Supposons que la saisie de n soit 5
        s ← s + n
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n          ← Supposons que la saisie de n soit 5
        s ← s + n       ← La somme s prend la valeur 2 + 5 = 7
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire          ← Comme n = 5, la condition est vraie
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n                    ← Supposons que la saisie de n soit 5
        s ← s + n                 ← La somme s prend la valeur 2 + 5 = 7
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n                    ← Supposons que la saisie de n soit 0
        s ← s + n
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n          ← Supposons que la saisie de n soit 0
        s ← s + n       ← La somme s prend la valeur 7 + 0 = 7
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire          ← Comme n = 0, la condition est fausse
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n                    ← Supposons que la saisie de n soit 0
        s ← s + n                 ← La somme s prend la valeur 7 + 0 = 7
    fin
    afficher s
fin
```

L'instruction tant que

La boucle **tant que** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance. Elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire          ← Comme n = 0, la condition est fausse
        afficher "Entrer un entier (0 pour arrêter) : "
        lire n
        s ← s + n
    fin
    afficher s                    ← La somme finale s est donc égale à 7
fin
```

L'instruction répéter jusqu'à

La boucle **répéter jusqu'à** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance, et qu'il faut lancer au moins une exécution du bloc d'instructions. Elle exécute le bloc jusqu'à ce que la condition d'arrêt devienne vraie.

L'instruction répéter jusqu'à

La boucle **répéter jusqu'à** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance, et qu'il faut lancer au moins une exécution du bloc d'instructions. Elle exécute le bloc jusqu'à ce que la condition d'arrêt devienne vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire
        afficher "Entrer un entier : "
        lire n
        s ← s + n
    fin
    afficher s
fin
```

L'instruction répéter jusqu'à

La boucle **répéter jusqu'à** est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance, et qu'il faut lancer au moins une exécution du bloc d'instructions. Elle exécute le bloc jusqu'à ce que la condition d'arrêt devienne vraie.

Exemple

Algorithme : sommeDesEntrées

```
variable
    entier n, s

début
    s ← 0
    n ← 1
    tant que n ≠ 0 faire
        afficher "Entrer un entier : "
        lire n
        s ← s + n
    fin
    afficher s
fin
```

Algorithme : sommeDesEntrées'

```
variable
    entier n, s

début
    s ← 0
    n ← 0
    répéter
        afficher "Entrer un entier : "
        lire n
        s ← s + n
    jusqu'à n = 0
    afficher s
fin
```

Comparaisons

Boucle pour

La boucle

pour $i \leftarrow x$ **à** y **faire** instruction

exécute l'instruction exactement $y - x + 1$ fois.

Comparaisons

Boucle pour

La boucle

pour $i \leftarrow x$ **à** y **faire** instruction

exécute l'instruction exactement $y - x + 1$ fois.

Boucle tant que

La boucle

tant que condition **faire** instruction

exécute l'instruction jusqu'à ce que la condition soit fausse. L'instruction peut donc être exécutée zero fois si la condition est initialement fausse.

Comparaisons

Boucle pour

La boucle

pour $i \leftarrow x$ **à** y **faire** instruction

exécute l'instruction exactement $y - x + 1$ fois.

Boucle tant que

La boucle

tant que condition **faire** instruction

exécute l'instruction jusqu'à ce que la condition soit fausse. L'instruction peut donc être exécutée zero fois si la condition est initialement fausse.

Instruction répéter jusqu'à

La boucle

répéter instruction **jusqu'à** condition

exécute l'instruction jusqu'à ce que la condition soit fausse. L'instruction est donc exécutée au moins une fois.