République Algérienne Démocratique et Populaire
Ministère de l'enseignement Supérieur et de la Recherche Scientifique

Informatique (programmation)

Première année licence TL

Dr. S. BOUZOUBIA

Affectation

Ordre des instructions

l'ordre dans lequel les instructions sont écrites va jouer un rôle essentiel dans le résultat final. Considérons les deux algorithmes suivants :

Exemple 1

Variable A **: entier;**
Début
A ← 34;
A ← 12;
Fin

Exemple 2

Variable A **: entier;**
Début
A ← 12;
A ← 34;
fin

le premier cas la valeur finale de A est 12, dans l'autre elle est 34 .

Affectation

EXPRESSIONS ET OPERATEURS

dans une instruction d'affectation, on trouve :

- à gauche de la flèche, un nom de variable, et uniquement cela.
- à droite de la flèche, ce qu'on appelle une **expression**. Une expression est un ensemble de valeurs, reliées par **des opérateurs**, et équivalent à une seule valeur.
- Par exemple, voyons quelques expressions de type **numérique**. Ainsi :

7
5+4
123-45+844
Toto-12+5-Riri

- l'expression située à droite de la flèche soit du même type que la variable située à gauche.

Affectation

EXPRESSIONS ET OPERATEURS

- On va maintenant détailler ce que l'on entend par le terme d' **opérateur**.
- Un opérateur est un signe qui relie deux valeurs, pour produire un résultat.

- opérations arithmétiques tout ce qu'il y a de classique.

  \+ : addition

  \- : soustraction

  * : multiplication

  / : division

- Mentionnons également le ^ qui signifie « puissance ».

  exemple: 45 au carré s'écrira donc 45 ^ 2.

Affectation

EXPRESSIONS ET OPERATEURS

- on a le droit d'utiliser les parenthèses, avec les mêmes règles qu'en mathématiques. La multiplication et la division ont « naturellement » priorité sur l'addition et la soustraction. Les parenthèses ne sont ainsi utiles que pour modifier cette priorité naturelle.
- Cela signifie qu'en informatique, 12 * 3 + 5 et (12 * 3) + 5 valent strictement la même chose, à savoir 41.

Affectation

Opérateur alphanumérique :

Cet opérateur permet de **concaténer**, autrement dit d'agglomérer, deux chaînes de caractères.

- Par exemple :

```
Variables  A, B, C: Caractere;
Début
A ← "Gloubi "   ;
B ← "Boulga ";
C ← A & B;
Fin
```

La valeur de C à la fin de l'algorithme est "GloubiBoulga"

Affectation

- **Opérateurs logiques (ou booléens) :**

Il s'agit du ET, du OU, du NON et du XOR. Table logique

DEUX REMARQUES POUR TERMINER:

j'attire votre attention sur la trompeuse similitude de vocabulaire entre les mathématiques et l'informatique:

- En mathématiques, une « variable » est généralement une inconnue, qui recouvre un nombre non précisé de valeurs. Exemple: y = 3 x + 2
- **En informatique, une variable possède à un moment donné une valeur et une seule**. Cette valeur ne « varie » pas à proprement parler. Du moins ne varie-t-elle que lorsqu'elle est l'objet d'une instruction d'affectation.

Affectation

DEUX REMARQUES POUR TERMINER:

La deuxième remarque concerne le signe de l'affectation. En algorithmique, comme on l'a vu, c'est le signe ←. Mais en pratique, la quasi totalité des langages emploient le signe égal. Et là, pour les débutants, la confusion avec les maths est également facile.

En maths, A = B et B = A sont deux propositions strictement équivalentes. En informatique, absolument pas, puisque cela revient à écrire A ← B et B ← A, deux choses bien différentes.

Lecture / Ecriture

DE QUOI S'AGIT-IL ?

Imaginons que nous ayons fait un programme pour calculer le carré d'un nombre, mettons **12**. Si on a fait au plus simple, on a écrit un truc du genre :

Algorithme carre

Variable A : **Entier;**

Debut

A ← 12^2;

Fin

- **Mais si l'on veut le carré d'un autre nombre que 12, il faut réécrire le programme.**
- **L'utilisateur qui fait exécuter ce programme ne saura jamais quel est le carré de 12**

Lecture / Ecriture

DE QUOI S'AGIT-IL ?

- Les instructions permettent à l'utilisateur d'entrer des valeurs au clavier pour qu'elles soient utilisées par le programme. Cette opération est la **lecture**.
- Dans l'autre sens, d'autres instructions permettent au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran. Cette opération est **l'écriture**.

Lecture / Ecriture

LES INSTRUCTIONS DE LECTURE ET D'ECRITURE

- **Lire** variable;

Dès que le programme rencontre une instruction Lire, l'exécution s'interrompt, attendant la frappe d'une valeur au clavier. Dès lors, aussitôt que la touche Entrée (Enter) a été frappée, l'exécution reprend.

- **Ecrire** expression;

Avant de Lire une variable, il est très fortement conseillé d'écrire des **libellés** à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper Avant de Lire une variable

Ecrire "Entrez votre nom : " ;
Lire NomFamille;

Les tests (structure alternatives)

De quoi s'agit-il ?

Reprenons le cas de notre « programmation algorithmique du touriste égaré ». Normalement, l'algorithme ressemblera à quelque chose comme : « *Allez tout droit jusqu'au prochain carrefour, puis prenez à droite et ensuite la deuxième à gauche, et vous y êtes* ».

- Mais en cas de doute légitime de votre part, cela pourrait devenir : « *Allez tout droit jusqu'au prochain carrefour et là regardez à droite. Si la rue est autorisée à la circulation, alors prenez la et ensuite c'est la deuxième à gauche. Mais si en revanche elle est en sens interdit, alors continuez jusqu'à la prochaine à droite, prenez celle-là, et ensuite la première à droite* ».

Les tests (structure alternatives)

- **STRUCTURE D'UN TEST**

Il n'y a que **deux formes possibles** pour un test ; la première est la plus simple, la seconde la plus complexe.

Si booléen **Alors**
  Instructions
Finsi

Si booléen **Alors**
  Instructions 1
Sinon
  Instructions 2
Finsi

- Un **booléen** est une **expression** dont la valeur est VRAI ou FAUX. Cela peut donc être :
- une **variable** (ou une expression) de type booléen
- une **condition**

Les tests (structure alternatives)

- **QU'EST CE QU'UNE CONDITION ?**

Une condition est une comparaison

Cette définition est essentielle ! Elle signifie qu'une condition est composée de trois éléments :

- une valeur
- un **opérateur de comparaison**
- une autre valeur

Les valeurs peuvent être a priori de n'importe quel type (numériques, caractères…). Mais si l'on veut que la comparaison ait un sens, il faut que les deux valeurs de la comparaison soient du même type !

Les tests (structure alternatives)

- **QU'EST CE QU'UNE CONDITION ?**

Les **opérateurs de comparaison** sont :

- égal à…
- différent de…
- strictement plus petit que…
- strictement plus grand que…
- plus petit ou égal à…
- plus grand ou égal à…

Les tests (structure alternatives)

- **Remarque très importante**
  En formulant une condition dans un algorithme, il faut se méfier comme de la peste de certains raccourcis du langage courant, ou de certaines notations valides en mathématiques, mais qui mènent à des non-sens informatiques. Prenons par exemple la phrase « Toto est compris entre 5 et 8 ». On peut être tenté de la traduire par : **5 < Toto < 8**
  Or, une telle expression, qui a du sens en français, comme en mathématiques, **ne veut rien dire en programmation**. En effet, elle comprend deux opérateurs de comparaison, soit un de trop, et trois valeurs, soit là aussi une de trop. On va voir dans un instant comment traduire convenablement une telle condition.

Les tests (structure alternatives)

- **CONDITIONS COMPOSÉES**
- Reprenons le cas « Toto est inclus entre 5 et 8 ». En fait cette phrase cache non une, mais **deux** conditions. Car elle revient à dire que « Toto est supérieur à 5 et Toto est inférieur à 8 ». Il y a donc bien là deux conditions, reliées par ce qu'on appelle un **opérateur logique**, le mot ET.
- Comme on l'a évoqué plus haut, l'informatique met à notre disposition quatre opérateurs logiques : ET, OU, NON, et XOR.

Les tests (structure alternatives)

- **CONDITIONS COMPOSÉES**

C1 et C2	C2 Vrai	C2 Faux
C1 Vrai	Vrai	Faux
C1 Faux	Faux	Faux

C1 ou C2	C2 Vrai	C2 Faux
C1 Vrai	Vrai	Vrai
C1 Faux	Vrai	Faux

Non C1	
C1 Vrai	Faux
C1 Faux	Vrai

Les tests (structure alternatives)

- **TESTS IMBRIQUES**

Par exemple, un programme devant donner l'état de l'eau selon sa température doit pouvoir choisir entre trois réponses possibles (solide, liquide ou gazeuse).

Les tests (structure alternatives)

- **TESTS IMBRIQUES: LA PREMIÈRE SOLUTION**

```
Variable Temp: Entier;
Debut
Ecrire "Entrez la température de l'eau : "  ;
Lire Temp;
Si Temp =< 0 Alors
  Ecrire "C'est de la glace ";
FinSi
Si Temp > 0 Et Temp < 100 Alors
  Ecrire "C'est du liquide ";
Finsi
Si Temp > 100 Alors
  Ecrire "C'est de la vapeur ";
Finsi
Fin
```

Les tests (structure alternatives)

- **TESTS IMBRIQUES: LA DEUXIEME SOLUTION**

```
Variable Temp en Entier
Debut
        Ecrire "Entrez la température de l'eau :"
        Lire Temp
        Si Temp =< 0 Alors
           Ecrire "C'est de la glace"
        Sinon
                Si Temp < 100 Alors
                  Ecrire "C'est du liquide"
                Sinon
                        Ecrire "C'est de la vapeur"
                Finsi
        Finsi
Fin
```

Les tests (structure alternatives)

- **VARIABLES BOOLÉENNES**

On peut entrer des conditions dans ces variables, et tester ensuite la valeur de ces variables. Reprenons l'exemple de l'eau. On pourrait le réécrire ainsi :

Les tests (structure alternatives)

Algorithme: exemple3

Variable: Temp: Entier;

A, B: Booléen;

Début

Ecrire ("Entrez la température de l'eau :");

Lire (Temp);

A ← Temp =< 0;

B ← Temp < 100;

Si A **Alors**

Ecrire "C'est de la glace"

Sinon Si B **Alors**

Ecrire "C'est du liquide"

Sinon

Ecrire "C'est de la vapeur"

Finsi

Fin

Les tests (structure alternatives)

- **REMARQUE**

Dans une condition composée employant à la fois des opérateurs ET et des opérateurs OU, la présence de parenthèses possède une influence sur le résultat, tout comme dans le cas d'une expression numérique comportant des multiplications et des additions. Exemple: D ← (A ET B) OU C ; E ← A ET (B OU C);

Activités:

- Ecrire un algorithme qui demande la valeur d'un nombre et afficher si il est pair ou impair.
- Ecrire un algorithme qui lit une variable ensuite, il donne sa valeur absolue.

Les boucles (structures répétitives)

A QUOI CELA SERT-IL DONC ?

Prenons le cas d'une saisie au clavier (une lecture), où par exemple, le programme pose une question à laquelle l'utilisateur doit répondre par O (Oui) ou N (Non). Mais tôt ou tard, l'utilisateur, facétieux ou maladroit, risque de taper autre chose que la réponse attendue. Dès lors, le programme peut planter soit par une erreur d'exécution (parce que le type de réponse ne correspond pas au type de la variable attendu) soit par une erreur fonctionnelle (il se déroule normalement jusqu'au bout, mais en produisant des résultats fantaisistes).

Alors, dans tout programme un tant soit peu sérieux, on met en place ce qu'on appelle **un contrôle de saisie**, afin de vérifier que les données entrées au clavier correspondent bien à celles attendues par l'algorithme.

Les boucles (structures répétitives)

A première vue , on pourrait essayer avec **un SI.** Voyons voir ce que ça donne :

Algorithme: exemple4

Variable Rep: Caractere;

Début

```
    Ecrire ("Voulez vous un café ? (O/N) ");
    Lire (Rep);
    Si Rep <> "O" et Rep <> "N" Alors
        Ecrire ("Saisie erronnée. Recommencez");
        Lire (Rep);
    FinSi
```

Fin

Les boucles (structures répétitives)

tant que l'utilisateur ne se trompe qu'une seule fois l'algorithme est correcte, et d'entrer une valeur correcte à la deuxième demande. Si l'on veut également bétonner en cas de deuxième erreur, il faudrait rajouter un SI. Et ainsi de suite, on peut rajouter des centaines de SI, et écrire un algorithme aussi lourd. **Solution??**

La seule issue est d'utiliser une structure de boucle

Les boucles (structures répétitives)

Primitive répétitive est une instruction qui permet de répéter un certain nombre de fois une série d'instructions simples ou composées constituant un bloc.

Les primitives répétitives sont:

- Tant que …..faire
- Pour…. Allant de…. à…. Faire
- Répéter…… jusqu'à……

Les boucles (structures répétitives)

1- La boucle Tant que:

```
TantQue booléen
  …
  Instructions
  …
FinTantQue
```

Principe

Le programme arrive sur la ligne du **Tant Que**. Il examine alors l'expression booléenne. Si cette valeur est VRAI, le programme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne **Fin Tant Que**. Il retourne ensuite sur la ligne du **Tant Que**, procède au même examen, et ainsi de suite. Le processus ne s'arrête que lorsque le booléen prend la valeur FAUX.

Les boucles (structures répétitives)

Illustration avec notre problème de contrôle de saisie. Une première approximation de la solution consiste à écrire :

Algorithme: exemple4

Variable Rep: Caractere;

Début

Ecrire ("Voulez vous un café ? (O/N) ");

tant que Rep <> "O" et Rep <> "N" **faire** → **FAUX**

Lire (Rep);

FinTant que

Fin

Les boucles (structures répétitives)

Illustration avec notre problème de contrôle de saisie. Une première approximation de la solution consiste à écrire :

Algorithme: exemple5

Variable Rep: Caractere;

Début

Ecrire ("Voulez vous un café ? (O/N) ");

Lire (rep);

tant que Rep <> "O" et Rep <> "N" **faire**

Lire (Rep);

FinTant que

Fin

Les boucles (structures répétitives)

Pour créer un dialogue

Algorithme: exemple5

Variable Rep: Caractere;

Début

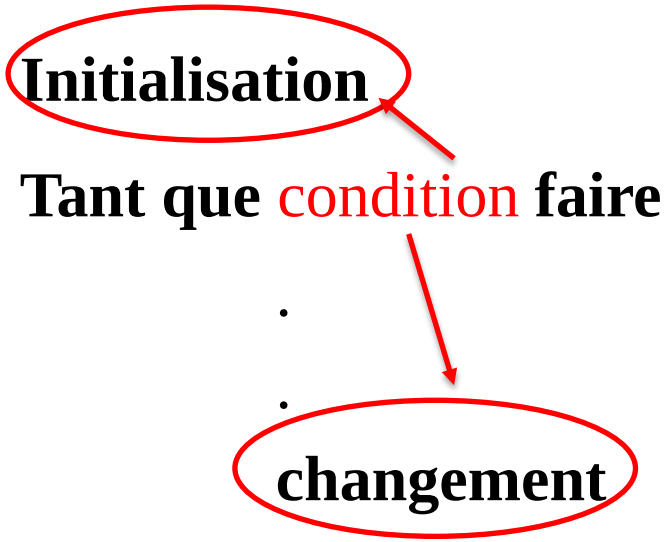
```
        Ecrire ("Voulez vous un café ? (O/N) ");
        tant que Rep <> "O" et Rep <> "N" faire
                Ecrire ("Vous devez répondre par O ou N. Recommencez");
                 Lire (Rep);
         FinTant que
         Ecrire ("Saisie acceptée");
```

Fin

Les boucles (structures répétitives)

1- La boucle Tant que:

La boucle '**Tant Que**' est dépendante de la condition de l'initialisation et changement.



exemple

co ← 1

Tant que co < N faire

.

.

co ← co+1;

Fin tant que