

République Algérienne Démocratique et Populaire
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département Informatique



Langage Algorithmique (1)

Cours Algorithmique de 1ere Année MI
Présenté par : Medjadba(s.medjadba@gmail.com)

Etapes de résolution d'un problème

Analyse du problème : qu'est ce qu'on veut faire ?

Définir les données : leurs caractéristiques, leurs types.

Définir les résultats : leurs caractéristiques, leurs types.

Définir les relations entre résultats-données : comment passer des données aux résultats ?

Exemple : Préparer un litre de jus d'orange naturel concentré à 60%.

Analyse

La préparation d'un jus d'orange naturel concentré à 60% consiste à extraire 60 cl de jus du fruit orange, puis lui rajouter 40 cl d'eau. Pour le sucrer, on peut rajouter du miel ou des dattes.

Les données

5 à 8 oranges suivant la qualité (plus ou moins juteuse), 40 cl d'eau, du miel ou quelques dattes, suivant le gout qu'on veut.

En plus des outils : couteau, presse (ou mixeur), récipient, bouteille, passoire, entonnoir.

Le résultat

1 litre de jus (liquide) concentré à 60%.

Relations entre résultats-données

Pour passer des données au résultat, on exécute les **actions** suivantes :

- 1- **Laver** les oranges **si** elle ne sont pas propres.
- 2- **Couper** en deux pour une presse ou **peler** puis **couper** en morceaux pour un mixeur puis **enlever** les pépins.
- 3- **Si** on utilise les dattes, **dénoyer** puis **couper** en morceaux.
- 4- **Presser** les oranges ou **mixer** les oranges et les dattes avec l'eau (**suivant l'outil**).

- 5- **Si** on utilise une presse, **diluer** le miel dans l'eau.
- 6- **Mettre** le tous dans le récipient.
- 7- **Verser** dans la bouteille à travers l'entonnoir. **Utiliser** la passoire **si** on veut un jus clair.

C'est ce qu'on appelle

Un Algorithme

Définition

Un algorithme est une suite finie d'actions élémentaires exécutées dans un ordre précis sur un ensemble de données permettant de résoudre un problème.

Pour un problème informatique

L'analyse consiste à :

Définir tous les **objets** utilisés représentant les données en **entrée** et les résultats en **sortie**.

Définir toutes les **actions élémentaires** qui seront exécutées dans l'**ordre** pour aboutir aux résultats.

C'est quoi un objet ?

Un objet est l'**entité** manipulée par une **action**. On distingue deux classes d'objets: les **constantes** et les **variables**.

Exemple

L'action $S = A + 2$ manipule **trois** objets. 2 objets **variables** (S et A) et un objet **constant** (2).

Quelle est la différence entre variable et constante ?

Un objet de classe **variable**, peut **changer de valeur** dans l'algorithme et peut être **modifié**. (**c'est comme les oranges**).

Par contre, un objet de classe **constante**, garde la même valeur pendant toute l'exécution de l'algorithme. (**c'est comme le couteau**)

Caractéristiques d'un Objet

Un objet est caractérisé par :

1- Nom : appelé **identificateur**, il permet d'identifier l'objet.

2- Type : l'ensemble des valeurs que peut prendre un objet (réel, entier, caractère...), c'est le domaine de définition.

3- Valeur : un élément du type pris à un instant donné.

Exemple

Nom : Age

Type : entier

Valeur : 18

Règle de construction des identificateurs

Un identificateur doit respecter certaines règles de construction:

- Il est formé des caractères de l'alphabet (**A** à **Z** ou **a** à **z**) des chiffres (**0** à **9**) et du caractère souligné (**_** : tiret du 8).
- Peut être au moins un caractère.
- Le premier caractère doit être une lettre.

Exemple

Identificateurs corrects : TTC, Gr3, sect, Nom_Et, Note_1_2_3, X, y

Identificateurs incorrects : 9TH, Gr 3, S?, Nom-Et

Remarque

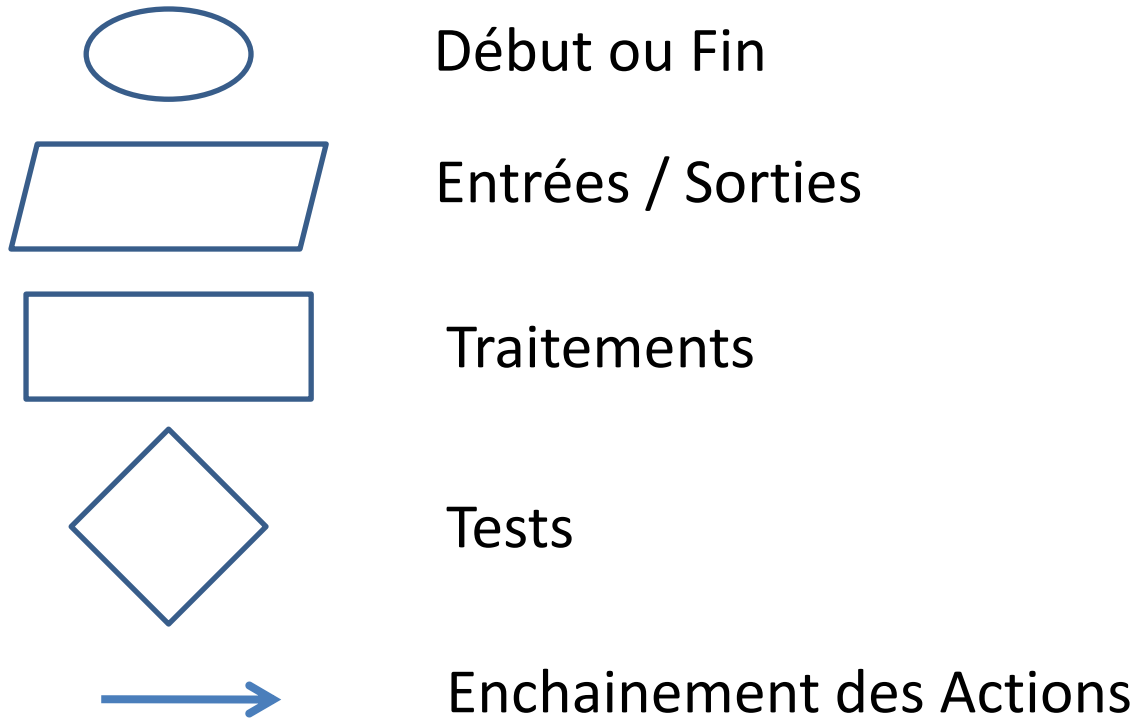
- De préférence choisir des noms significatifs.
- Certains mots ne sont pas permis ([voir mots clés](#)).
- Eviter les noms longs.

Représentation d'un Algorithme

Historiquement, il y a deux façons pour représenter un algorithme

1- Organigramme

C'est une représentation **graphique** en utilisant des **symboles**

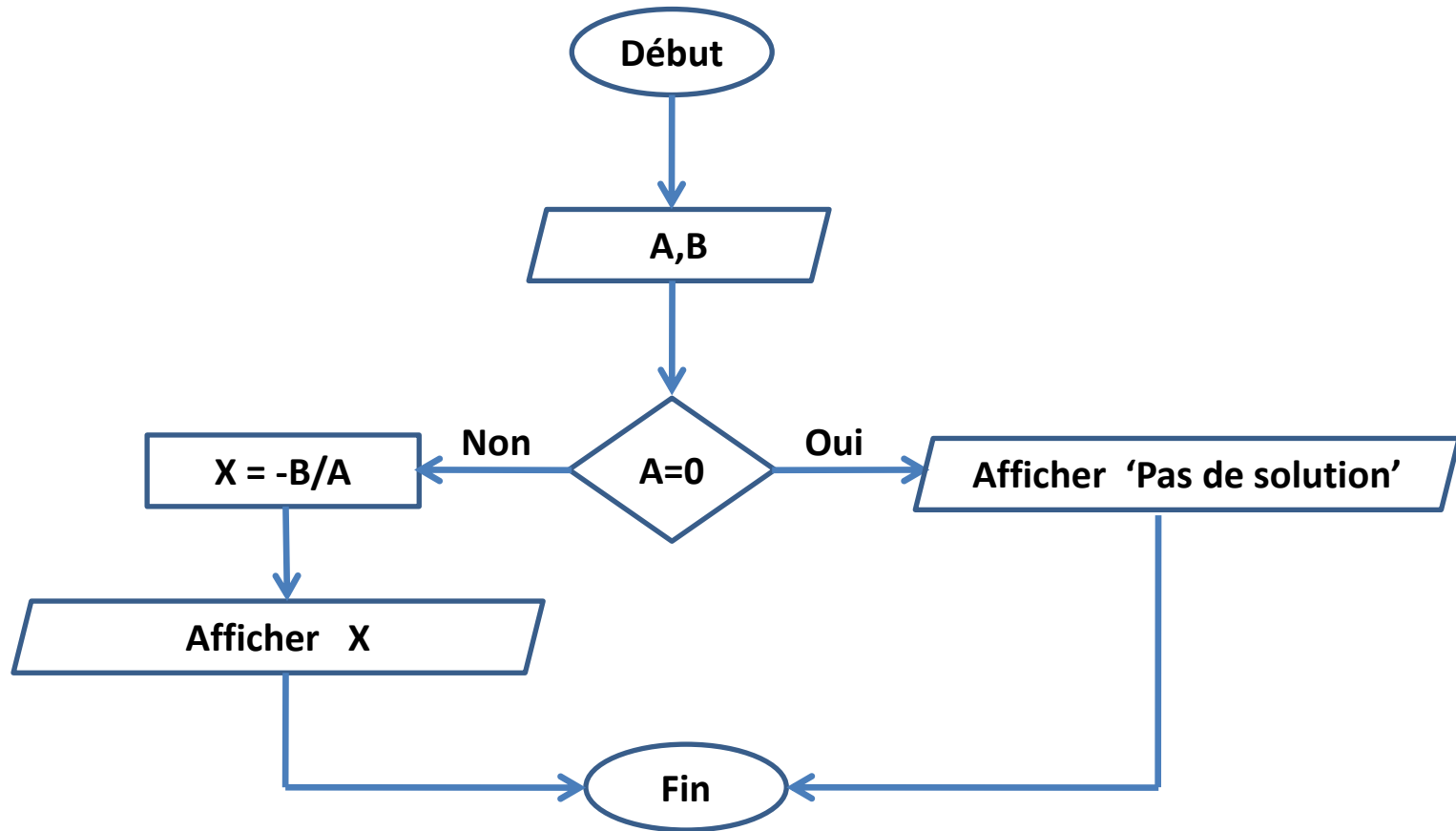


Offre une vue d'ensemble de l'algorithme. Mais elle est quasiment **abandonnée** aujourd'hui

Exemple

Résoudre une équation : $Ax + B = 0$

C'est une équation du premier degré. Sa solution est $-B/A$ à condition que $A \neq 0$. Si A est nul, l'équation n'a pas solution dans R.



2- Algorithme

C'est une représentation textuelle **normalisée**.

La structure générale d'un algorithme **ressemble** à une recette de cuisine, il est formé de trois parties.

Titre

Ingrédients

Préparation

Partie Entête

Partie Déclaration

Partie Action

Au lieu d'utiliser des symboles, on utilise des mots **spéciaux** appelés :
Mots Réservés ou **Mots Clés**

Partie Entête

Elle permet de définir le nom de l'algorithme. Pour cela on utilise un mot clé : **Algorithme**, et on l'utilise suivant une **syntaxe** qu'il faut respecter.

Algorithme <nomAlgo> ;

<nomAlgo> : c'est un **identificateur** représentant le nom de l'algorithme. On peut donner n'importe quel nom, mais il est préférable qu'il soit représentatif.

Exemple

Algorithme Facture;

Algorithme equation;

Algorithme calcul_somme;

Partie Déclaration

Dans cette partie on déclare tous les objets manipulés dans **l'algorithme** (constantes et variables) en spécifiant deux caractéristiques :

Le **nom** et le **type** pour les **variables**.

Le **nom** et la **valeur** pour les **constantes**.

La **syntaxe** de déclaration pour les deux types d'objets est :

Const <IdObj> = <ValeurObj> ;

Var <IdObj> : <TypeObj> ;

<IdObj> : est l'**identificateur** de l'objet.

<ValeurObj> : est la **valeur** de l'objet

<TypeObj> : est le **type** de l'objet.

Exemple : **Const** Pi = 3,141592 ; **Var** Age : entier ;

Remarque : Les objets de même type peuvent être regroupés :

Exemple: **Var** Note1, Note2, Moyenne : reel ;



Quel est l'effet de la déclaration dans le système ?

Les déclarations permettent au compilateur (pendant l'exécution) de **réserver** un espace mémoire à chaque objet dont la **taille** varie suivant le **type** de l'identificateur déclaré.

Nous avons vu que la machine possède une **mémoire**. Cette mémoire peut recevoir des ordres de mémorisation.

Alors chaque déclaration correspond à un **ordre** qui dit à la mémoire :
« *Préparer moi un espace appelé <IdObj>* »

Exemple

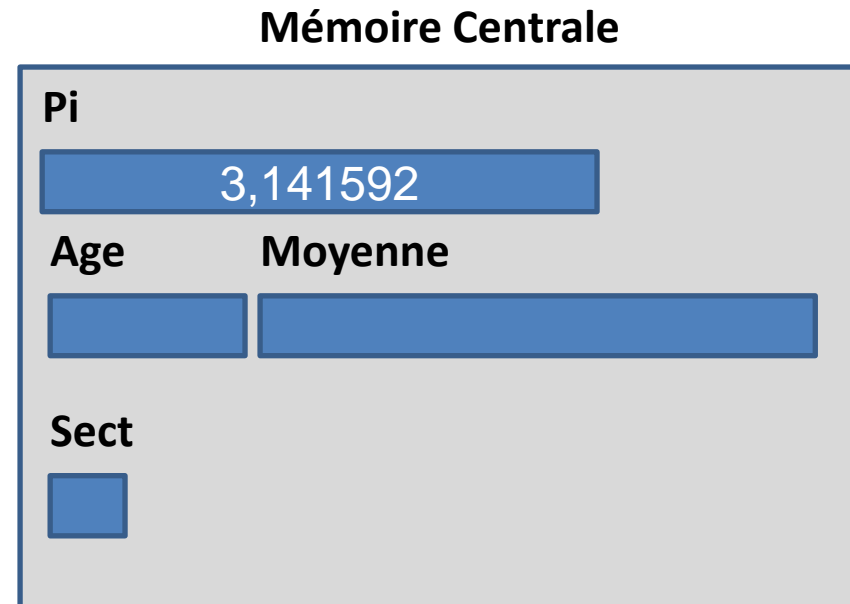
Const Pi = 3,141592;

Var Age : **entier**;

Moyenne : **reel**;

Sect : **caractère**;

Mais c'est quoi entier, reel, caractère ?



Bonne remarque.

Et bien ces mots sont des mots **clés**, ils correspondent à ce qu'on appelle des **types de base**.

Au fait, le **type** définit la **taille** de l'espace nécessaire à la sauvegarde d'un objet.

Et justement, pour ne pas **gaspiller** l'espace mémoire on propose différents types avec différentes tailles pour répondre aux besoins de l'utilisateur.

C'est comme des moyens de transport.

Pour transporter **3** personnes, une **voiture** suffit.

Pour transporter **50** personnes, il faut un **bus**.

Mais **mobiliser** un **bus** pour **3** personnes, **c'est... ?!**

Types de base (Simples)

On définit **4** types de base : **Entier, Reel, Caractère et Booléen**.

Le type Entier

Le type **entier** représente un sous-ensemble **fini** de nombres entiers relatifs ($\in \mathbf{Z}$).

En générale, l'espace réservé pour un entier est **4 Octets = 32 bits**.

Mais on utilise aussi des entiers sur **8 Octets = 64 bits** (**entier long**).

Entier	Valeur minimale	Valeur maximale
2 Octets = 16 bits	-32768	+32767
4 Octets = 32 bits	-2 147 483 648	+2 147 483 647
8 Octets = 64 bits	-2^{63}	$+(2^{63} - 1)$

Remarque : en algorithmique, on utilise le type entier sans préciser s'il est court ou long.

Opérations sur les entiers

Opérations arithmétiques : $+$, $-$, $*$, **div** , **mod** , ($/$: résultat réel)

mod : reste de la division (ex: $5 \bmod 2 = 1$), **div** : (ex: $5 \text{ div } 2 = 2$)

Opérations de relations : $<$, \leq , $>$, \geq , $=$, \neq (\leq , \geq , \neq)

Le type Reel

Le type **reel** représente approximativement un sous-ensemble **fini** de nombres **réels** ($\in \mathbf{R}$).

Les réels sont représenté au format **scientifique** ($\pm M 10^E$).

Comme le nombre de bits est **limité**, la représentation sera **approximative**.

En générale, l'espace réservé pour un réel est **4 Octets = 32 bits**.

Mais on utilise aussi des réels sur **8 Octets = 64 bits** (**réel double**).

Réel	Nombres négatifs	Nombres positifs
4 Octets = 32 bits	$-3,4 \times 10^{38}$ à $-1,17 \times 10^{-38}$	$+1,17 \times 10^{-38}$ à $+3,4 \times 10^{38}$
8 Octets = 64 bits	-1.79×10^{308} à $-2,2 \times 10^{-308}$	$+2,2 \times 10^{-308}$ à $+1.79 \times 10^{308}$

Remarque : en algorithmique, on utilise le type **reel** sans préciser s'il est simple ou double.

Opérations sur les réels

Opérations arithmétiques : $+, -, *, /$

Opérations de relations : $<, \leq, >, \geq, =, \neq$ (\leq, \geq, \neq)

Le type Caractere

Le type **caractere** représente tous les symboles manipulés (alphabet, chiffres, signe de ponctuation, ...).

Il prend un espace de **1 Octet = 8 bits** en mémoire.

L'ensemble de ces caractères est regroupé dans une table, appelée :
Table **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)

Opérations sur les caractères

Les seules opérations permises sur les caractères sont les

Opérations de relations : $<, \leq, >, \geq, =, \neq$ (\leq, \geq, \neq)

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Le type Booleen

Le type **booléen** représente des objets booléens (logiques) ne pouvant prendre que deux valeurs: **Vrai** ou **Faux**.

Il prend un espace de **1 Octet = 8 bits** en mémoire.

Opérations sur les booléens

Les seules opérations permises sur les booléens sont les

Opérations logiques : **ET, OU, NON** (négation).

Opérations de relations : **=, ≠ (<>)**

Rappels

A	B	NON A	A ET B	A OU B
F	F	V	F	F
F	V	V	F	V
V	F	F	F	V
V	V	F	V	V

Partie Action

Cette partie contient les différentes **actions** élémentaires décrivant la solution.

Debut

Sa syntaxe est :

<Action1> ;

<Action2> ;

<ActionN> ;

Fin.

Nous allons commencer à apprendre ces action qui nous permettent de communiquer avec notre fameuse machine ?!

Actions de Lecture/Ecriture (Entrée/Sortie)

Actions de Lecture (Entrée)

Cette action permet d'**introduire** les données d'une **unité d'entrée** (**Clavier**) vers la **mémoire centrale**.

Elle **initialise** un **objet** dont l'espace mémoire est déjà **réservé**.

Sa syntaxe est :

Lire(<IdObj>) ;

<IdObj> : est l'identificateur (nom) de l'objet manipulé.

Exemple

Soient deux objets:

Age : entier; **Moyenne** : reel;

Lire(Age) ;

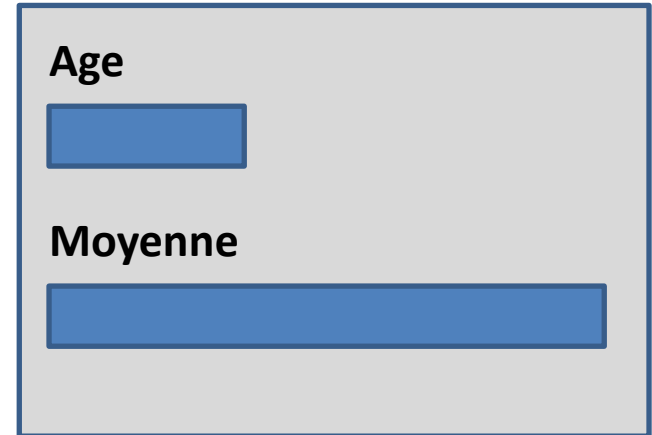
L'exécution de cette action permet de **saisir** une valeur entière au **clavier**

Si on **tape** la valeur **18** elle sera stockée dans l'espace **nommé Age**

Lire(Moyenne) ;

Si on **tape** la valeur **15.5** elle sera stockée dans l'espace **nommé Moyenne**

Mémoire Centrale



Remarque

On peut lire plusieurs objets avec une seule action en séparant les noms d'objets par des virgules.

Lire(Age); **Lire**(Moyenne); **est équivalent à** : **Lire**(Age,Moyenne);

Au moment de la saisie, on sépare les valeurs des objets par des espaces : 18 15.5

Actions d'écriture (Sortie)

Cette action permet d'**afficher** les données depuis la **mémoire centrale** vers une **unité de sortie** (**Ecran**).

Elle **affiche** la **valeur** mémorisée d'un **objet** au moment de l'exécution (à un instant **t**).

Comme elle peut afficher une constante.

Sa syntaxe est :

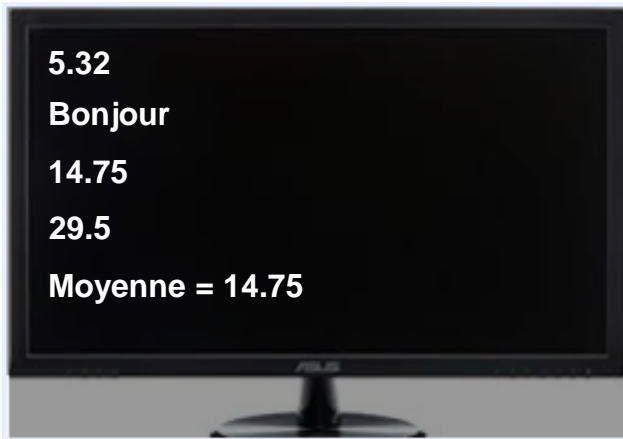
Ecrire(<Expression>) ;

<Expression> : peut prendre différentes formes.

Exemples cas possibles de <Expression>

- Peut être une constante numérique.
- Peut être un texte (entre cotes)
- Peut être un identificateur
- Peut être une expression arithmétique

```
Ecrire(5.32) ;  
Ecrire('Bonjour') ;  
Ecrire(Moyenne) ;  
Ecrire(2*Moyenne) ;
```



Moyenne

14.75

Remarque

On peut écrire plusieurs objets avec une seule action en séparant les noms d'objets par des virgules.

```
Ecrire('Moyenne= '); Ecrire(Moyenne);
```

est équivalent à :

```
Ecrire('Moyenne= ', Moyenne);
```

C'est quoi exactement une <Expression> ?

Une expression peut avoir trois formes possibles:

1- Expression de base (élémentaire): elle se réduit à une **constante** numérique (**réelle** ou **entière**), ou bien une constante **caractère** (ou suite de caractères) qui seront mis entre cotes(' ' ou " ") ou encore le **nom** d'un objet (identificateur).

2- Expression Arithmétique: c'est une combinaison d'objets de type numérique (reel, entier) et d'opérateurs arithmétiques (+, -, *, /, mod, div) et éventuellement des parenthèses (), (**formule mathématique**)

Remarque

Contrairement à une formule mathématique, l'expression en informatique doit être écrite sous forme linéaire.

$$E = \frac{X^2 + 3X - 4}{2X - 1}$$

Forme linéaire : $E = (X * X + 3 * X - 4) / (2 * X - 1)$

Linéarisation des Expressions

La linéarisation d'une expression arithmétique consiste à **transformer** l'expression en une forme **linéaire** en respectant **la priorité** des opérateurs en utilisant les **parenthèses** en cas de nécessité.

Faire apparaître l'opérateur de multiplication (*) d'une manière **explicite** (**2X** s'écrit **2*X**)

Priorité des Opérateurs

La priorité des opérateurs est donnée comme suit :

-	*	()
+	/	



Exemple

$$4 + 6 / 2 * 7 - 2 = 4 + \mathbf{3} * 7 - 2 = 4 + \mathbf{21} - 2 = 23$$

$$(4 + 6) / (2 * (7 - 2)) = \mathbf{10} / (2 * \mathbf{5}) = 10 / \mathbf{10} = 1$$

3- Expression Logique: c'est une combinaison d'objets de type booléen et d'opérateurs logiques (**ET, OU, NON**) et éventuellement des parenthèses **()**, ou encore combiné avec des objets de type numérique (entier, reel) avec des opérateurs de relations (**<, ≤, >, ≥, =, ≠**)

Exemple

E1 = (A **Ou** B) **Et** (**Non** C **Et** A)

E2 = (J * J **≤** N) **Et** (**Non** Trouve)

E3 = (J + 2 * K) > 2

Priorité des Opérateurs Logiques

La priorité des opérateurs est donnée comme suit :



Action d'Affectation

Cette action permet de **modifier** la **valeur** d'un **objet** au moment de l'exécution (à un instant **t**). Elle **change** donc le contenu de l'espace mémoire nommé <IdObj> réservé à l'objet.

Sa syntaxe est :

<IdObj> \leftarrow <Expression> ;

<IdObj> : l'identificateur de l'objet manipulé.

<Expression> : Expression de base, arithmétique ou logique.

Exemple : soient 2 objets **A** et **B** de type entier.

A \leftarrow 18;

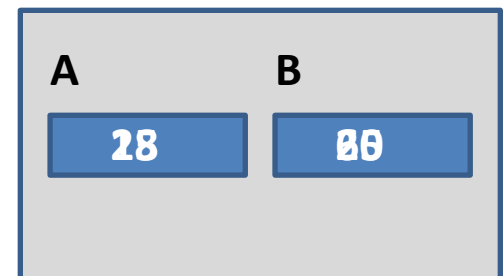
A \leftarrow B + 5;

B \leftarrow 2*B + A;

Attention

Après l'affectation, l'ancienne valeur de l'objet est perdue et on ne peut pas la récupérer.

Mémoire Centrale



Remarque

Le **type** de <Expression> doit être le **même** que le type de <IdObj>

Exemple

Ecrire un algorithme qui calcule et affiche la somme de deux entiers données.

Problème : calcul et affichage de la somme de deux entiers.

Données en entrée : on a besoin de 2 entiers (A et B).

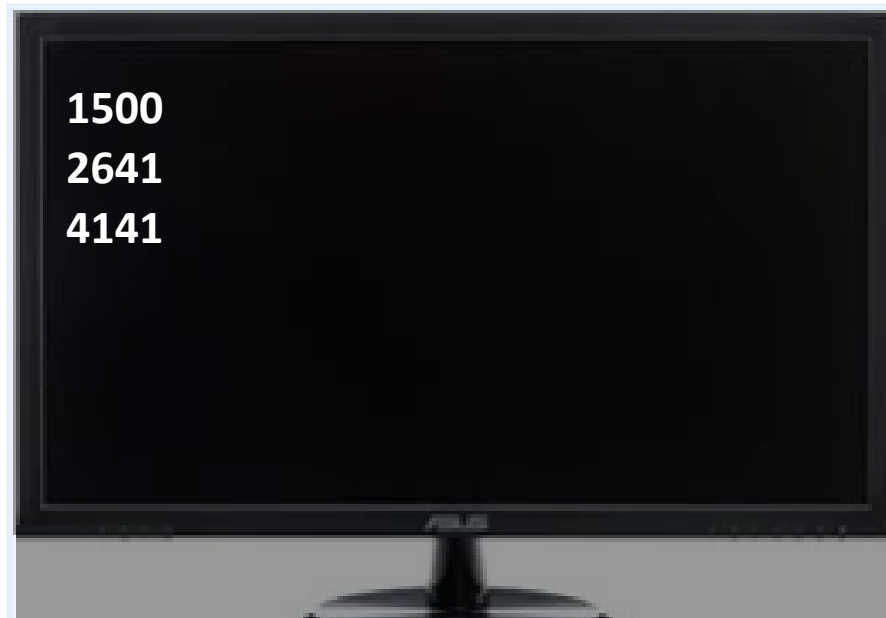
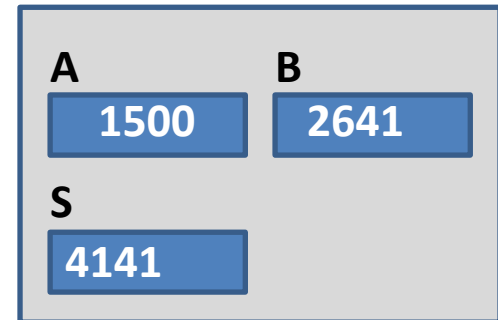
Résultat en sortie : la somme des 2 entiers (S).

Passage des entrées aux sorties : Calcul : $S = A + B$, puis l'affichage.

Notre PREMIER ALGORITHME !!!

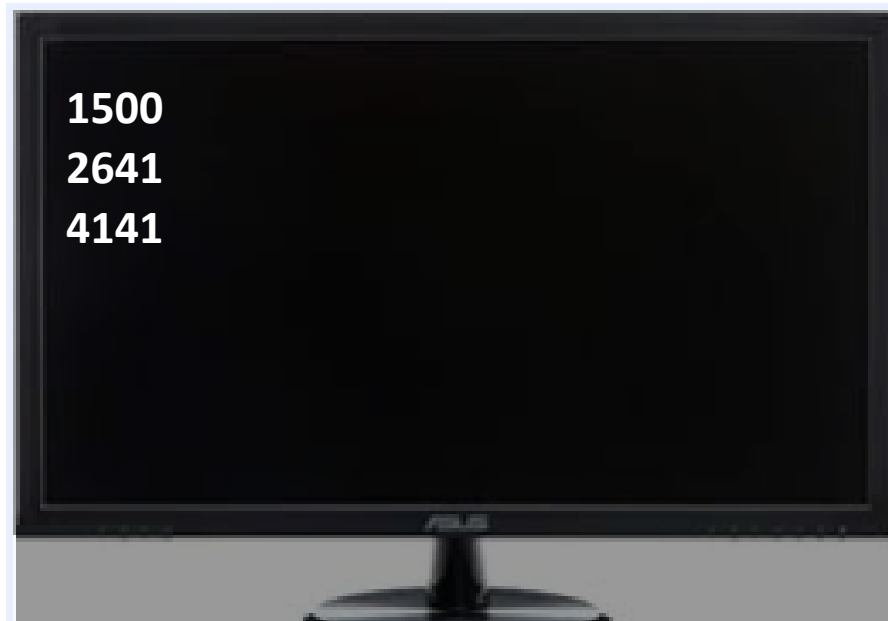
```
Algorithme somme ;  
Var A,B,S : entier;  
Debut  
    Lire (A) ;  
    Lire (B) ;  
    S ← A+B;  
    Ecrire (S) ;  
Fin.
```

Mémoire Centrale



On peut faire Mieux...

Ajouter des messages pour rendre l'exécution **conviviale**.



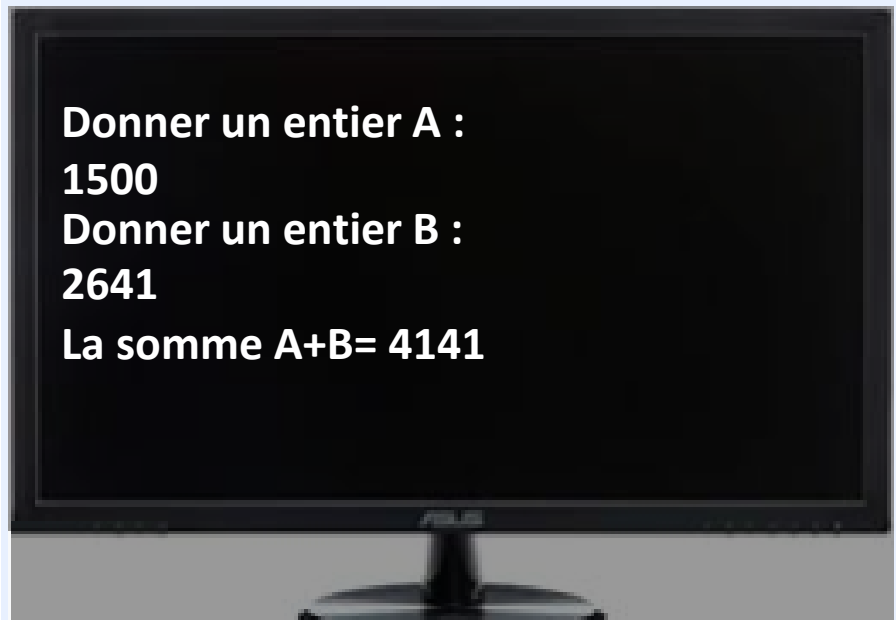
```
Algorithme somme ;  
Var A,B,S : entier;  
Debut
```

```
    Ecrire ( 'Donner un entier A :' ) ;  
    Lire (A) ;  
    Ecrire ( 'Donner un entier B :' ) ;  
    Lire (B) ;  
    S ← A+B ;  
    Ecrire ( 'La somme A+B  
            = ', S ) ;
```

```
Fin.
```

Mémoire Centrale

A	B
1500	2641
S	
4141	



Donner un entier A :
1500
Donner un entier B :
2641
La somme A+B= 4141



Les Structures de Contrôle

Les Structures Alternatives (Conditionnelles)

La structure alternative ou conditionnelle permet d'exécuter ou non une série d'instruction **selon la valeur d'une condition**.

Structure Alternative Réduite : Si ... Alors ... Fsi

Sa syntaxe

Si <Condition> **Alors** <Bloc Action> **Fsi** ;

<Condition> : est une **Expression** booléenne (**Vrai** ou **Faux**)

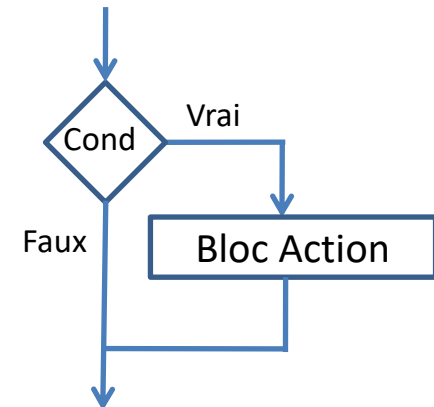
<Bloc Action> : suite d'actions séparées par des ;
(Act1; Act2; ... ; ActN ;)

Sémantique

- On évalue la <Condition>

Si sa valeur = **Vrai**, on exécute <Bloc Action>

Si sa valeur = **Faux**, on continue après **Fsi** ;



Exemple

```
Si X<0 Alors absX ← - X Fsi ;  
Si A>B ET Gr=2 Alors A← A- B;  
C←2*Gr+A;  
B←A+C Fsi ;
```

Structure Alt

ernative Complète : Si ... Alors ... Sinon ... Fsi

Sa syntaxe

Si <Condition> Alors <Bloc Action1> Sinon <Bloc Action2> Fsi ;

<Condition> : est une **Expression** booléenne (**Vrai** ou **Faux**)

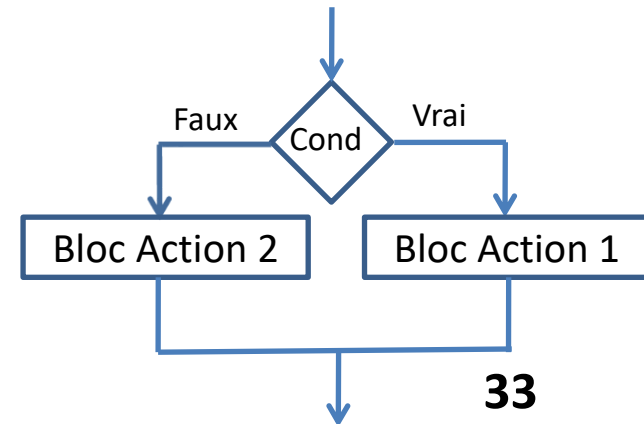
<Bloc Action**1**> ou bien <Bloc Action**2**>: suite d'actions séparées par des ;
(Act**1**; Act**2**; ... ; Act**N** ;)

Sémantique

- On évalue la <Condition>

Si sa valeur = **Vrai**, on exécute <Bloc Action1>

Si sa valeur = **Faux**, on exécute <Bloc Action2>



Exemple

Ecrire un algorithme qui détermine puis affiche le maximum entre deux entiers.

Analyse

Déterminer le maximum entre deux entiers revient à comparer les deux valeurs et prendre le plus grand.

Données entrée : deux entiers A et B.

Résultat en sortie : un entier Max.

```
Algorithme CalculMax;  
Var  A,B,Max:entier;  
Debut  
    Ecrire('Donner deux entiers A et B:');  
    Lire(A,B);  
    Si A>B Alors Max←A; Sinon Max←B; Fsi;  
    Ecrire('Le maximum est :',Max);  
Fin.
```

Exemple

Ecrire un algorithme qui détermine puis affiche le maximum entre trois entiers.

```
Algorithme CalculMax;  
Var  A,B,C,Max:entier;  
Debut  
    Ecrire('Donner deux entiers A, B et C:');  
    Lire(A,B,C);  
    Si (A ≥ B Et A ≥ C) Alors Max ← A Fsi;  
    Si (B ≥ A Et B ≥ C) Alors Max ← B Fsi;  
    Si (C ≥ A Et C ≥ B) Alors Max ← C Fsi;  
    Ecrire('Le maximum est :',Max);  
Fin.
```

On a fait combien de tests ? **6**

Est-ce-que on peut minimiser ce nombre ?

Solution 2

```
Algorithme CalculMax;  
Var  A,B,C,Max:entier;  
Debut  
    Ecrire('Donner deux entiers A, B et C:');  
    Lire(A,B,C);  
    Si A ≥ B Alors Si A ≥ C Alors Max ← A;  
                                Sinon Max ← C ;  
                                Fsi;  
    Sinon Si B ≥ C Alors Max ← B;  
                                Sinon Max ← C ;  
                                Fsi;  
  
    Fsi;  
    Ecrire('Le maximum est :',Max);  
Fin.
```

On a fait combien de tests ? **2**

Conclusion: **Eviter les conditions composées (en utilisant des tests imbriqués) tant que c'est possible**

A Suivre ...

Merci!

