مدرسة علوم المعلومات
ⵜⵉⵎⵔⵔⴰⵜ ⵏ ⵜⵓⵙⵙⵏⴰⵡⵉⵏ ⵉⵏⴼⵓⵔⵎⴰⵜⵉⵡⵏ
ECOLE DES SCIENCES DE L'INFORMATION
www.esi.ac.ma

Algorithmique

3. Les structures alternatives

Sommaire

Rappel

Rappel

- Algorithme informatique
  - Suite d'instructions ordonnée qui décrit de façon exhaustive les différentes étapes à suivre par un processeur pour résoudre un problème donné en un temps fini

Rappel

- L'ordinateur n'est pas capable d'initiative mais peut stocker des programmes et les exécuter
- Programme informatique
  - Implémentation d'un ou plusieurs **algorithme(s)** dans un **langage de programmation** donné

Rappel

- Un algorithme est destiné à des êtres humains qui vont généralement l'implémenter dans un langage de programmation quelconque
  - Il est indépendant du langage de programmation
  - Son écriture doit détailler le plus possible son fonctionnement et sa structure

Rappel

- Pseudo-code algorithmique
  - Formalisation de l'écriture des algorithmes dans une langue humaine (généralement l'anglais) en adoptant quelques conventions proche des langages de programmation mais sans les contraintes syntaxiques de ces derniers

Rappel

- Pseudo-code algorithmique

ALGORITHME *nom_de_l'algorithme*

<partie des déclarations>

DEBUT

<partie des instructions>

//commentaire

FIN

Rappel

- Variable
  - Cellule mémoire désignée par un **identificateur** et possédant un contenu qui est consultable et modifiable par des programmes
  - Possède un **type** donné
  - Possède une valeur qui évolue au cours de l'exécution de l'algorithme/programme

Rappel

- Identificateur
  - Nom associé à une variable
  - Suite de lettre et de chiffres devant commencer par une lettre ou "_"
    - Interdits: espaces, lettres accentuées ou avec cédille, caractères spéciaux sauf "_"
  - Sensible à la casse

Rappel

- Identificateur
  - Devrait être le plus significatif possible
  - Par convention
    - Notation camel case pour les variables
    - Majuscules pour les constantes
  - Fixé lors de la déclaration de la variable et ne change plus

Rappel

- Type
  - Ensemble des valeurs qu'une variable peut avoir
  - Peut être prédéfini ou défini par l'utilisateur
    - Entier Réel Booléen Caractère Chaîne
  - Chacun définit un ensemble d'opérations possibles
  - Fixé lors de la déclaration de la variable et ne change plus

Rappel

- Les variables sont déclarées en début de l'algorithme

ALGORITHME *nom_de_l'algorithme*

<partie des déclarations>

DEBUT

<partie des instructions>

//commentaire

FIN

<déclarations des constantes>

<déclarations des variables>

<déclarations des routines>

Rappel

- Les variables sont déclarées en début de l'algorithme
  - Syntaxe de la déclaration

VAR ***identificateur*** **:** ***type***

VAR nomEtudiant : **chaîne**

VAR absent : **booléen**

VAR lettre : **caractère**

VAR longueur, largeur : **réel**

Rappel

- Affectation
  - Opération permettant d'attribuer une valeur à une variable
  - Établissement d'un lien entre l'identificateur et l'emplacement mémoire de la valeur correspondante
  - Syntaxe

ma_variable ← constante
ma_variable ← expression

Rappel

- Expression
  - Suite d'**opérateurs** et de termes qui est compréhensible et qu'on peut calculer

Rappel

- Opérateur

Opérateur arithmétique	Description	Opérandes	Type du résultat
–	Soustraction Changement de signe (opérateur unaire)	Entiers ou réels	Même type que les opérandes
+	Addition	Entiers ou réels	Même type que les opérandes
*	Multiplication	Entiers ou réels	Même type que les opérandes
/	Division flottante	Entiers ou réels	Réel
DIV	Division entière	Entiers seulement	Entier
MOD	Modulo	Entiers seulement	Entier

Rappel

- Opérateur

Opérateur alphanumérique	Description	Opérandes	Type du résultat
&	Concaténation	Caractères et chaînes de caractères	Chaînes de caractères

Rappel

- Opérateur

Opérateur logique	Description	Opérandes	Type du résultat
NON	Négation logique	Booléens	Booléen
ET	Et logique (AND)	Booléens	Booléen
OU	Ou logique (OR)	Booléens	Booléen
OUEX	Ou exclusif (XOR)	Booléens	Booléen

Rappel

- Opérateur

Opérateur relationnel	Description	Opérandes	Type du résultat
=	Égale	Types compatibles	Booléen
<>	Différent (noté aussi !=)	Types compatibles	Booléen
<	Inférieur à	Types compatibles	Booléen
>	Supérieur à	Types compatibles	Booléen
<=	Inférieur ou égal à	Types compatibles	Booléen
>=	Supérieur ou égal à	Types compatibles	Booléen

Rappel

- Priorité des opérateurs

Opérateurs unaires	– NON
Opérateurs multiplicatifs	* / DIV MOD ET
Opérateurs additifs	+ – OU
Opérateurs relationnels	= < > ≤ ≥ <>

Rappel

- Évaluation des expressions
  - Effectuée selon la priorité des opérateurs
  - Les expressions entre parenthèses sont évaluées avant d'intervenir dans le reste des calculs
  - Le membre de droite est évalué avant d'attribuer sa valeur au membre de gauche
    - Exemple: incrémentation i ← i + 1

Rappel

- Une instruction est une action à accomplir par l'algorithme
- On compte quatre instructions de base
  - Déclaration (mémoire)
  - Affectation (calcul)
  - Lecture (entrées)
  - Écriture (sorties)

Rappel

- Instruction d'écriture **Afficher()**
  - Permet de visualiser (i.e. afficher sur l'écran) du texte ou les valeurs des variables
  - On précise entre ()
    - Les identificateurs des variables à visualiser
    - Une expression dont la valeur calculée sera affichée
    - Du texte brute entre " "

Rappel

- Instruction de lecture **Lire()**
  - Permet de lire des valeurs saisies au clavier ou contenues dans des fichiers et de les affecter à des variables
  - On met entre () les noms des variables à saisir

Exercices

- Écrire un algorithme qui demande à un utilisateur de saisir un nombre puis calcule et affiche le carré de ce nombre

Exercices

- Écrire un algorithme qui demande à un utilisateur de saisir un nombre puis calcule et affiche le carré de ce nombre

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
   Afficher("Saisir un nombre")
   Lire(x)
   Afficher(x*x)
FIN
```

Exercices

- Écrire un algorithme qui lit le prix HT d'un article et le taux de TVA puis qui calcule et affiche le prix TTC

Exercices

- Écrire un algorithme qui lit le prix HT d'un article et le taux de TVA puis qui calcule et affiche le prix TTC

```
ALGORITHME mon_algo
VAR prixHT, tva, prixTTC : réel
DEBUT
   Afficher("Saisir le prix HT et la TVA")
   Lire(prixHT, tva)
   prixTTC ← prixHT * (1+tva)
   Afficher("Le prix TTC est : ", prixTTC)
FIN
```

Exercices

- Écrire un algorithme qui lit le prix HT d'un article et le taux de TVA puis qui calcule et affiche le prix TTC

```
ALGORITHME mon_algo
VAR prixHT, tva, prixTTC : réel
DEBUT
   Afficher("Saisir le prix HT et la TVA")
   Lire(prixHT, tva)
   Afficher("Le prix TTC est : ", prixHT * (1+tva))
FIN
```

Problématique

- Que se passe-t-il si l'utilisateur saisit un texte ?

```
ALGORITHME mon_algo
VAR prixHT, tva, prixTTC : réel
DEBUT
   Afficher("Saisir le prix HT et la TVA")
   Lire(prixHT, tva)
   Afficher("Le prix TTC est : ", prixHT * (1+tva))
FIN
```

Problématique

- Que se passe-t-il si l'utilisateur veut faire ce calcul sur plusieurs produits ?

```
ALGORITHME mon_algo
VAR prixHT, tva, prixTTC : réel
DEBUT
   Afficher("Saisir le prix HT et la TVA")
   Lire(prixHT, tva)
   Afficher("Le prix TTC est : ", prixHT * (1+tva))
FIN
```

Problématique

- Un programme n'est pas purement séquentiel ➔ nécessité d'avoir des **structures de contrôle**

1. Les structures alternatives (tests)
2. Les structures itératives (boucles)

Structure d'un test

Structure d'un test

- SI … ALORS …

SI *condition* **ALORS**

séquence

FINSI

  - Une **condition** est un booléen
    - Si sa valeur est ***VRAI*** la séquence d'instructions ***séquence*** est exécutée

Structure d'un test

- SI … ALORS … SINON …

SI *condition* **ALORS**

séquence I

SINON

séquence2

FINSI

  - Si la valeur de la condition est ***VRAI***, ***séquence I*** est exécutée. Si la valeur est ***FAUX***, **séquence2** est exécutée

Structure d'un test

ALGORITHME *nom_de_l'algorithme*
<partie des déclarations>
DEBUT
séquence1
SI *condition1* **ALORS**
séquence2
FINSI
séquence3
FIN

Structure d'un test

ALGORITHME *nom_de_l'algorithme*
<partie des déclarations>
DEBUT
séquence1
SI *condition1* **ALORS**
séquence2
FINSI
séquence3
FIN

Déroulement de l'algorithme

- *sequence1* est exécutée
- Si *condition1* est VRAI alors *sequence2* est exécutée
- *sequence3* est exécutée

Structure d'un test

ALGORITHME *nom_de_l'algorithme*
<partie des déclarations>
DEBUT
séquence1
SI *condition1* **ALORS**
séquence2
SINON
séquence3
FINSI
séquence4
FIN

Structure d'un test

ALGORITHME *nom_de_l'algorithme*
<partie des déclarations>
DEBUT
séquence1
SI *condition1* **ALORS**
séquence2
SINON
séquence3
FINSI
séquence4
FIN

Déroulement de l'algorithme

- *sequence1* est exécutée
- Si *condition1* est VRAI alors
  - *sequence2* est exécutée
  - *sequence4* est exécutée
- Si *condition1* est FAUX alors
  - *sequence3* est exécutée
  - *sequence4* est exécutée

Conditions simples et composées

Structure d'un test

- Une **condition** est un booléen qui peut être
  - Valeur booléenne

```
SI b ALORS
    Afficher("Vrai")
SINON
    Afficher("Faux")
FINSI
```

Structure d'un test

- Une **condition** est un booléen qui peut être
  - Valeur booléenne
  - Expression booléenne

```
SI x OU y ALORS
    Afficher("Vrai")
SINON
    Afficher("Faux")
FINSI
```

Structure d'un test

- Une **condition** est un booléen qui peut être
  - Valeur booléenne
  - Expression booléenne
  - Comparaison
    - Entre deux valeurs

Structure d'un test

- Une **condition** est un booléen qui peut être

Opérateur relationnel	Description	Opérandes	Type du résultat
=	Égale	Types compatibles	Booléen
<>	Différent (noté aussi !=)	Types compatibles	Booléen
<	Inférieur à	Types compatibles	Booléen
>	Supérieur à	Types compatibles	Booléen
<=	Inférieur ou égal à	Types compatibles	Booléen
>=	Supérieur ou égal à	Types compatibles	Booléen

Structure d'un test

- Une **condition** est un booléen qui peut être
  - Valeur booléenne
  - Expression booléenne
  - Comparaison
    - Entre deux valeurs

```
SI x = y ALORS
    Afficher("x et y sont égaux")
SINON
    Afficher("x et y ne sont pas égaux")
FINSI
```

Structure d'un test

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI x >= 0 ALORS
     Afficher("x est un nombre positif")
  SINON
     Afficher("x est un nombre négatif")
  FINSI
FIN
```

Structure d'un test

- Une **condition** est un booléen qui peut être
  - Valeur booléenne
  - Expression booléenne
  - Comparaison
    - Entre deux valeurs
    - Entre plusieurs valeurs ?
      - Par exemple: 0 < x < 10

Structure d'un test

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI  0 < x< 10 ALORS
     Afficher("x est compris entre 0 et 10")
  SINON
     Afficher("x est <= 0 ou >= 10")
  FINSI
FIN
```

Structure d'un test

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI  0 < x < 10 ALORS
     Afficher("x est compris entre 0 et 10")
  SINON
     Afficher("x est <= 0 ou >= 10")
  FINSI
FIN
```

Valide mathématiquement mais non algorithmiquement

Structure d'un test

- Certains tests nécessitent d'utiliser des **conditions composées**
  - Expression conditionnelle composée de deux ou plusieurs conditions reliées par des opérateurs logiques

    Condition1 **OPL** Condition2 **OPL** … **OPL** Condition3
  - Évaluée avec des tables de vérité

Structure d'un test

- Certains tests nécessitent d'utiliser des **conditions composées**
  - Exemple: 0 < x < 10 peut être représenté par une condition composée *Condition1* ***ET*** *Condition2*

    Condition1 : x > 0

    Condition2 : x < 10

Structure d'un test

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI x > 0 ET x< 10 ALORS
     Afficher("x est compris entre 0 et 10")
  SINON
     Afficher("x est <= 0 ou >= 10")
  FINSI
FIN
```

Structure d'un test

- Une **condition** est un booléen qui peut être
  - Valeur booléenne
  - Expression booléenne
  - Comparaison
- Pour qu'un test soit utile, il faut que la condition ne prenne pas toujours la même valeur (i.e. ne soit pas toujours fausse ou toujours vraie)

Structure d'un test

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI x < 0 ET x > 10 ALORS
     Afficher("Test non réalisable !")
  SINON
     Afficher("Ce message sera toujours affiché !")
  FINSI
FIN
```

Exemples

```
ALGORITHME mon_algo
VAR prixHT, tva, prixTTC : réel
DEBUT
 Afficher("Saisir le prix HT et la TVA")
 Lire(prixHT, tva)
 SI (prixHT est réel) ET (tva est réel) ALORS
    Afficher("Le prix TTC est : ", prixHT * (1+tva))
 FINSI
FIN
```

Exemples

```
ALGORITHME mon_algo
VAR prixHT, tva, prixTTC : réel
DEBUT
 Afficher("Saisir le prix HT et la TVA")
 Lire(prixHT, tva)
 SI (prixHT est réel) ET (tva est réel) ALORS
    Afficher("Le prix TTC est : ", prixHT * (1+tva))
 SINON
    Afficher("Veuillez saisir des nombres réels")
 FINSI
FIN
```

Exemples

```
ALGORITHME mon_algo
VAR mot_1, mot_2 : chaîne
DEBUT
 Afficher("Saisir deux mots différents")
 Lire(mot_1, mot_2)
 SI mot_1 < mot_2 ALORS
    Afficher("Le premier mot est : ", mot_1)
 SINON
    Afficher("Le premier mot est : ", mot_2)
 FINSI
FIN
```

Exemples

```
ALGORITHME mon_algo
VAR mot_1, mot_2 : chaîne
DEBUT
 Afficher("Saisir deux mots différents")
 Lire(mot_1, mot_2)
 SI mot_1 < mot_2 ALORS
    Afficher("Le premier mot est : ", mot_1)
 SINON
    Afficher("Le premier mot est : ", mot_2)
 FINSI
FIN
```

mot_1 : *université* ; mot_2 : *faculté*

mot_1 : *Université* ; mot_2 : *faculté*

mot_1 : *4* ; mot_2 : *7*

Exemples

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre puis qui retourne la valeur absolue de ce nombre

Exercices

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Afficher("Saisir un nombre réel")
  Lire(x)
  SI x >= 0 ALORS
     Afficher("La valeur absolue est : ", x)
  SINON
     Afficher("La valeur absolue est : ",  -x)
  FINSI
FIN
```

Exercices

```
ALGORITHME mon_algo
VAR x, y : réel
DEBUT
  Afficher("Saisir un nombre réel")
  Lire(x)
  y ← x
  SI x < 0 ALORS
     y ← – x
  FINSI
  Afficher("La valeur absolue est : ", y)
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre puis qui retourne la racine carrée de ce nombre
  - Utiliser la fonction racine()

Exercices

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Afficher("Saisir un nombre réel")
  Lire(x)
  SI x >= 0 ALORS
     Afficher("La racine carrée est : ", racine(x))
  SINON
     Afficher("Saisir un nombre positif")
  FINSI
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir deux nombres puis qui retourne le signe (positif ou négatif) de leur produit (sans calculer ce dernier)

Exercices

```
ALGORITHME mon_algo
VAR x, y : réel
DEBUT
  Afficher("Saisir deux nombres réel")
  Lire(x, y)
  SI (x >= 0 et y>= 0) ou (x=<0 et y=<0) ALORS
     Afficher("Le produit est positif")
  SINON
     Afficher("Le produit est négatif")
  FINSI
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir trois nombres puis qui indique s'ils sont triés avec un ordre ascendant ou non

Exercices

```
ALGORITHME mon_algo
VAR x, y, z : réel
DEBUT
  Afficher("Saisir deux nombres réel")
  Lire(x, y,z)
  SI x < y et y < z ALORS
     Afficher("Les nombres sont ordonnés")
  SINON
     Afficher("Les nombres ne sont pas ordonnés")
  FINSI
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre puis qui indique si ce nombre est pair ou impair

Exercices

```
ALGORITHME mon_algo
VAR x, modulo : réel
DEBUT
  Lire(x)
  modulo ← x MOD 2
  SI modulo = 0 ALORS
     Afficher("x est un nombre pair")
  SINON
     Afficher("x est un nombre impair")
  FINSI
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir deux mots puis qui les affiche dans l'ordre alphabétique

Exercices

```
ALGORITHME mon_algo
VAR mot_1, mot_2 : chaîne
DEBUT
  Afficher("Saisir deux mots")
  Lire(mot_1, mot_2)
  SI mot_1 < mot_2 ALORS
     Afficher(mot_1, mot_2)
  SINON
     Afficher(mot_2, mot_1)
  FINSI
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre puis qui indique si ce nombre est positif, négatif ou nul

Exercices

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI x > 0 ALORS
     Afficher("x est un nombre positif")
  FINSI
  SI x < 0 ALORS
     Afficher("x est un nombre négatif")
  FINSI
  SI x = 0 ALORS
     Afficher("x est nul")
  FINSI
FIN
```

Exercices

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI x > 0 ALORS
     Afficher("x est un nombre positif")
  FINSI
  SI x < 0 ALORS
     Afficher("x est un nombre négatif")
  FINSI
  SI x = 0 ALORS
     Afficher("x est nul")
  FINSI
FIN
```

Inconvénient ?

Tests imbriqués

Structure des tests imbriqués

- Un test imbriqué est exprimé comme suit

SI *condition1* **ALORS**

séquence1

SINON

SI *condition2* **ALORS**

séquence2

SINON

séquence3

FINSI

FINSI

Structure des tests imbriqués

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI x > 0 ALORS
     Afficher("x est un nombre positif")
  SINON
    SI x < 0 ALORS
     Afficher("x est un nombre négatif")
    SINON
     Afficher("x est nul")
    FINSI
  FINSI
FIN
```

Structure des tests imbriqués

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI x > 0 ALORS
     Afficher("x est un nombre positif")
  SINON
    SI x < 0 ALORS
     Afficher("x est un nombre négatif")
    SINON
     Afficher("x est nul")
    FINSI
  FINSI
FIN
```

Structure des tests imbriqués

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI x > 0 ALORS
     Afficher("x est un nombre positif")
  SINON
    SI x < 0 ALORS
      Afficher("x est un nombre négatif")
    SINON
      Afficher("x est nul")
    FINSI
  FINSI
FIN
```

Structure des tests imbriqués

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI x <> 0 ALORS
    SI x < 0 ALORS
       Afficher("x est un nombre négatif")
    SINON
     Afficher("x est un nombre positif")
    FINSI
  SINON
    Afficher("x est nul")
  FINSI
FIN
```

Structure des tests imbriqués

- Pour alléger l'écriture et améliorer la lisibilité, on peut fusionner SINON et SI en **SINONSI** ➔ un seul bloc de test

SI *condition 1* **ALORS**
    séquence 1
SINONSI *condition2* **ALORS**
    séquence2
SINON
    séquence3
FINSI

Structure des tests imbriqués

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI x > 0 ALORS
     Afficher("x est un nombre positif")
  SINON
    SI x < 0 ALORS
     Afficher("x est un nombre négatif")
    SINON
     Afficher("x est nul")
    FINSI
  FINSI
FIN
```

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SI x > 0 ALORS
     Afficher("x est un nombre positif")
  SINONSI x < 0 ALORS
     Afficher("x est un nombre négatif")
  SINON
     Afficher("x est nul")
  FINSI
FIN
```

Structure des tests imbriqués

- Les tests imbriqués présentent plusieurs avantages
  - Simplification de l'écriture des tests
  - Amélioration de la lisibilité de l'algorithme / programme
  - Réduction du temps d'exécution

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir deux nombres puis qui retourne le signe (positif, négatif ou nul) de leur produit (sans calculer ce dernier)

Exercices

```
ALGORITHME mon_algo
VAR x, y : réel
DEBUT
  Afficher("Saisir deux nombres réel")
  Lire(x, y)
  SI x = 0 ou y = 0 ALORS
     Afficher("Le produit est nul")
  SINONSI (x > 0 et y> 0) ou (x < 0 et y < 0)
     Afficher("Le produit est positif")
  SINON
     Afficher("Le produit est négatif")
  FINSI
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir trois nombres puis qui indique s'ils sont triés avec un ordre ascendant, triés avec un ordre descendant ou non triés

Exercices

```
ALGORITHME mon_algo
VAR x, y, z : réel
DEBUT
  Afficher("Saisir deux nombres réel")
  Lire(x, y,z)
  SI x < y et y < z ALORS
     Afficher("Les nombres sont triés avec un ordre asc")
  SINONSI z < y et y < x ALORS
     Afficher("Les nombres sont triés avec un ordre desc")
  SINON
     Afficher("Les nombres ne sont triés")
  FINSI
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir son âge puis qui retourne la catégorie à laquelle il appartient
  - Age $<$ 12 ans : enfant
  - 12ans $\leq$ Age $<$ 18 ans : adolescent
  - 18 ans $\leq$ Age $<$ 60 ans : adulte
  - 60 ans $\leq$ Age : senior

Exercices

```
ALGORITHME mon_algo
VAR age: réel
DEBUT
  Afficher("Saisir votre âge")
  Lire(age)
  SI age < 12 ALORS
      Afficher("Vous être un enfant")
  SINONSI age >= 12 et age < 18 ALORS
      Afficher("Vous être un adolescent")
  SINONSI age >= 18 et age < 60 ALORS
      Afficher("Vous être un adulte")
  SINON
      Afficher("Vous être un senior")
  FINSI
FIN
```

Autres structures de test

Test « SUIVANT … CAS »

- Permet de sélectionner le bloc à exécuter en fonction de la valeur d'une variable
- Spécialisation de l'instruction SI … SINONSI
- Utile quand une variable a plusieurs valeurs à tester

Test « SUIVANT … CAS »

- Structure du test

SUIVANT *variable* **FAIRE**

CAS *valeur_1* : *sequence_1*

CAS *valeur_2* : *sequence_2*

…

CAS *valeur_n* : *sequence_n*

AUTRES CAS : sequence_autre

FINSUIVANT

Test « SUIVANT … CAS »

- Exemple

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SUIVANT x FAIRE
    CAS 0 :Afficher("Zéro")
    CAS 1 :Afficher("Un")
    CAS 2 :Afficher("Deux")
    CAS 3 :Afficher("Trois")
    AUTRES CAS :Afficher("Autre valeur")
  FINSUIVANT
   Afficher("Traitement terminé")
FIN
```

Test « SELONQUE…CAS… »

- Permet de sélectionner le bloc à exécuter en fonction de conditions
- Spécialisation de l'instruction SI … SINONSI …
- Utile quand il y a plusieurs conditions à tester

Test « SELONQUE…CAS… »

- Structure du test

SELONQUE

condition_1 : *sequence_1*

condition_2 : *sequence_2*

…

condition_n : *sequence_n*

SINON: sequence_sinon

FINSELONQUE

Test « SELONQUE…CAS… »

- Exemple

```
ALGORITHME mon_algo
VAR x : réel
DEBUT
  Lire(x)
  SELONQUE
    x = 0 :Afficher("x est nul")
    x > 0 :Afficher("x est positif")
    x < 0 :Afficher("x est négatif")
  FINSELONQUE
   Afficher("Traitement terminé")
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir le nombre pour sélectionner la langue d'affichage
  - 1 pour l'arabe
  - 2 pour l'anglais
  - 3 pour le français
  - 4 pour l'espagnol

Exercices

```
ALGORITHME mon_algo
VAR n: réel
DEBUT
  Afficher("Saisir le numéro de la langue à choisir")
  Lire(n)
  SUIVANT n FAIRE
      CAS 1:Afficher("Arabe")
      CAS 2:Afficher("Anglais")
      CAS 3:Afficher("Français")
      CAS 4:Afficher("Espagnol")
     AUTRES CAS :Afficher("Erreur !")
  FINSUIVANT
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir son âge puis qui retourne la catégorie à laquelle il appartient
  - Age $<$ 12 ans : enfant
  - 12ans $\leq$ Age $<$ 18 ans : adolescent
  - 18 ans $\leq$ Age $<$ 60 ans : adulte
  - 60 ans $\leq$ Age : senior

Exercices

```
ALGORITHME mon_algo
VAR age: réel
DEBUT
  Afficher("Saisir votre âge")
  Lire(age)
  SELONQUE
      age >= 0 et age <12  :Afficher("Vous être un enfant")
      age >= 12 et age < 18 : Afficher("Vous être un adolescent")
      age >= 18 et age < 60 :Afficher("Vous être un adulte")
      age >= 60 :Afficher("Vous être un senior")
  SINON :Afficher("Erreur !")
  FINSELONQUE
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir sa moyenne générale puis qui retourne la mention correspondante
  - MG $\geq$ 18 : Très bien
  - 14 $\leq$ MG $<$ 18 : Bien
  - 12 $\leq$ Age $<$ 14 : Assez-bien
  - 10 $\leq$ Age $<$ 12 : Passable

Exercices

```
ALGORITHME mon_algo
VAR moyenne : réel
DEBUT
  Afficher("Saisir votre moyenne générale")
  Lire(moyenne)
  SELONQUE
     moyenne >= 18 :Afficher("Mention Très bien")
     moyenne < 18 et moyenne >= 14 : Afficher("Mention Bien")
     moyenne < 14 et moyenne >= 12 :Afficher("Mention Assez bien")
     moyenne < 12 et moyenne >= 10  :Afficher("Mention Passable")
  SINON : Afficher("Non admis")
  FINSELONQUE
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir deux opérandes et un opérateur arithmétique, puis qui fait le calcul correspondant et retourne le résultat

E

```
ALGORITHME mon_algo
VAR x, y, resultat: réel
VAR op: chaîne
DEBUT
   Afficher("Saisir les opérandes")
   Lire(x,y)
   Afficher("Saisir l'opérateur")
   Lire(op)
   SUIVANT op FAIRE
      CAS "+": resultat ← x + y
      CAS "−": resultat ← x − y
      CAS "*": resultat ← x * y
      CAS "/": resultat ← x / y
      AUTRES CAS :Afficher("Saisir un opérateur valide")
   FINSUIVANT
   Afficher("Le résultat est : ", resultat)
FIN
```

Exercices

- Écrire un algorithme qui demande à l'utilisateur de saisir deux opérandes et un opérateur arithmétique, puis qui fait le calcul correspondant et retourne le résultat
  - Ajouter traitement pour MOD et DIV

مدرسة علوم المعلومات
ⵜⵉⵎⵔⵔⴰⵜ ⵏ ⵜⵓⵙⵙⵏⴰⵡⵉⵏ ⵉⵏⴼⵓⵔⵎⴰⵜⵉⵡⵏ
ECOLE DES SCIENCES DE L'INFORMATION
www.esi.ac.ma

Algorithmique

3. Les structures alternatives