Module PROG & ALGORITHMIQUE
TD 2
Alternative

Hervé Owsinski
Herve.Owsinski@uphf.fr

Semestre 1

2026-2027 (v2)

Définitions et cours

Définitions

Définition 1 : Structure de contrôle

Les **structures de contrôle** sont des outils qui permettent de **contrôler l'ordre** dans lequel sont exécutées les instructions de l'algorithme. Deux éléments sont directement liées aux structures de contrôle :

- la notion de **condition/alternative** (aujourd'hui) ;
- la notion de **boucle** (prochain thème).

Définitions

Définition 1 : Structure de contrôle

Les **structures de contrôle** sont des outils qui permettent de **contrôler l'ordre** dans lequel sont exécutées les instructions de l'algorithme. Deux éléments sont directement liées aux structures de contrôle :

- la notion de **condition/alternative** (aujourd'hui) ;
- la notion de **boucle** (prochain thème).

Remarque : sans structure de contrôle ?

Sans structure de contrôle, rien ne vient perturber la séquentialité.

Définitions

Définition : Expression

Une **expression** est une combinaison d'**opérateurs** (comme $+$, -, ET $\wedge$, OU $\vee$...) et d'**opérandes** (variables, constantes, valeurs...).

- Expression arithmétique : **a** $\times$ **x** $+$ **b**
- Expression logique : **a** $\wedge$ **b**, qu'on peut noter **a ET b**

Condition ou prédicat : une expression booléenne

En logique, une **condition** est soit vraie, soit fausse.
La **condition** fournie doit donc être une **expression booléenne** :

une expression obligatoirement évaluée à **Vrai** ou à **Faux**

On utilise également le mot **prédicat** si la valeur de la condition dépend d'une variable (ex : $x \geq 5$).

Définitions

Définition 2 : La structure de contrôle "alternative"

Le pseudo-code choisi a une structure proche de la syntaxe de beaucoup de langages de programmation :

```
13 ...
14 Si (condition) alors
15 |  instructionA1 ;
16 |  instructionA2 ;
17 Sinon
18 |  instructionB1 ;
19 |  instructionB2 ;
20 fin Si
```

Attention : on exécute soit le bloc A, soit le bloc B. **Jamais les deux**.
Si condition vaut Vrai : L14 - **L15 - L16** - L20
Si condition vaut Faux : L14 - L17 - **L18 - L19** - L20

Définitions

Définition 2 : déroulé si condition Vrai

Le pseudo-code choisi a une structure proche de la syntaxe de beaucoup de langages de programmation :

```
13 ...
14 Si (condition) alors
15 |  instructionA1 ;
16 |  instructionA2 ;
17 Sinon
18 |  instructionB1 ;
19 |  instructionB2 ;
20 fin Si
```

Attention : on exécute soit le bloc A, soit le bloc B. **Jamais les deux**.
Si **condition vaut Vrai** : L14 - **L15 - L16** - L20
Si condition vaut Faux : L14 - L17 - **L18 - L19** - L20

Définitions

Définition 2 : déroulé si condition Vrai

Le pseudo-code choisi a une structure proche de la syntaxe de beaucoup de langages de programmation :

```
13 ...
14 Si (condition) alors
15   |  instructionA1 ;
16   |  instructionA2 ;
17 Sinon
18   |  instructionB1 ;
19   |  instructionB2 ;
20 fin Si
```

Attention : on exécute soit le bloc A, soit le bloc B. **Jamais les deux**.
Si **condition vaut Vrai** : L14 - **L15 - L16** - L20
Si condition vaut Faux : L14 - L17 - **L18 - L19** - L20

Définitions

Définition 2 : déroulé si condition Vrai

Le pseudo-code choisi a une structure proche de la syntaxe de beaucoup de langages de programmation :

```
13 ...
14 Si (condition) alors
15 |  instructionA1 ;
16 |  instructionA2 ;
17 Sinon
18 |  instructionB1 ;
19 |  instructionB2 ;
20 fin Si
```

Attention : on exécute soit le bloc A, soit le bloc B. **Jamais les deux**.
Si **condition vaut Vrai** : L14 - **L15 - L16** - L20
Si condition vaut Faux : L14 - L17 - **L18 - L19** - L20

Définitions

Définition 2 : déroulé si condition Vrai

Le pseudo-code choisi a une structure proche de la syntaxe de beaucoup de langages de programmation :

```
13 ...
14 Si (condition) alors
15 |  instructionA1 ;
16 |  instructionA2 ;
17 Sinon
18 |  instructionB1 ;
19 |  instructionB2 ;
20 fin Si
```

Attention : on exécute soit le bloc A, soit le bloc B. **Jamais les deux**.
Si **condition vaut Vrai** : L14 - **L15 - L16** - L20
Si condition vaut Faux : L14 - L17 - **L18 - L19** - L20

Définitions

Définition 2 : déroulé si condition Faux

Le pseudo-code choisi a une structure proche de la syntaxe de beaucoup de langages de programmation :

```
13 ...
14 Si (condition) alors
15 |  instructionA1 ;
16 |  instructionA2 ;
17 Sinon
18 |  instructionB1 ;
19 |  instructionB2 ;
20 fin Si
```

Attention : on exécute soit le bloc A, soit le bloc B. **Jamais les deux**.
Si condition vaut Vrai : L14 - **L15 - L16** - L20
Si **condition vaut Faux** : L14 - L17 - **L18 - L19** - L20

Définitions

Définition 2 : déroulé si condition Faux

Le pseudo-code choisi a une structure proche de la syntaxe de beaucoup de langages de programmation :

```
13 ...
14 Si (condition) alors
15 |  instructionA1 ;
16 |  instructionA2 ;
17 Sinon
18 |  instructionB1 ;
19 |  instructionB2 ;
20 fin Si
```

Attention : on exécute soit le bloc A, soit le bloc B. **Jamais les deux**.
Si condition vaut Vrai : L14 - **L15 - L16** - L20
Si **condition vaut Faux** : L14 - L17 - **L18 - L19** - L20

Définitions

Définition 2 : déroulé si condition Faux

Le pseudo-code choisi a une structure proche de la syntaxe de beaucoup de langages de programmation :

```
13 ...
14 Si (condition) alors
15   | instructionA1 ;
16   | instructionA2 ;
17 Sinon
18   | instructionB1 ;
19   | instructionB2 ;
20 fin Si
```

Attention : on exécute soit le bloc A, soit le bloc B. **Jamais les deux**.
Si condition vaut Vrai : L14 - **L15 - L16** - L20
Si **condition vaut Faux** : L14 - L17 - **L18 - L19** - L20

Définitions

Définition 2 : déroulé si condition Faux

Le pseudo-code choisi a une structure proche de la syntaxe de beaucoup de langages de programmation :

```
13 ...
14 Si (condition) alors
15 |  instructionA1 ;
16 |  instructionA2 ;
17 Sinon
18 |  instructionB1 ;
19 |  instructionB2 ;
20 fin Si
```

Attention : on exécute soit le bloc A, soit le bloc B. **Jamais les deux**.
Si condition vaut Vrai : L14 - **L15 - L16** - L20
Si **condition vaut Faux** : L14 - L17 - **L18 - L19** - L20

Définitions

Définition 2 : déroulé si condition Faux

Le pseudo-code choisi a une structure proche de la syntaxe de beaucoup de langages de programmation :

```
13 ...
14 Si (condition) alors
15 |  instructionA1 ;
16 |  instructionA2 ;
17 Sinon
18 |  instructionB1 ;
19 |  instructionB2 ;
20 fin Si
```

Attention : on exécute soit le bloc A, soit le bloc B. **Jamais les deux**.
Si condition vaut Vrai : L14 - **L15 - L16** - L20
Si **condition vaut Faux** : L14 - L17 - **L18 - L19** - L20

Définitions

Définition 2 (version alternative limitée)

Parfois, on ne désire rien faire lorsque la condition est évaluée à Faux. Dans ce cas, on peut utiliser ceci :

```
13 ...
14 Si (condition) alors
15 |  instructionA1 ;
16 |  instructionA2 ;
17 fin Si
```

Attention : on va faire soit le bloc A, soit... rien.

L'utilisation du mot-clé **Sinon** est donc facultative.

Si **condition vaut Vrai** : L14 - **L15 - L16** - L17
Si **condition vaut Faux** : L14 - L17

Complément

Pas d'ambiguïté sur la structure

L'utilisation des mots clés **si**, **alors**, **sinon** et **fin si** permet de délimiter précisement chacun des blocs.

```
Si (condition) alors
    instructions du bloc A ;
Sinon
    instructions du bloc B ;
fin Si
```

Dans le cas d'une alternative avec un **sinon** :

- Le bloc A est délimité par **alors** et **sinon** ;
- Le bloc B est délimité par **sinon** et **fin Si** ;
- La structure est délimitée par **Si** et **fin Si**.

Complément

Pas d'ambiguïté sur la structure

L'utilisation des mots clés **si**, **alors**, **sinon** et **fin si** permet de délimiter précisement chacun des blocs.

```
Si (condition) alors
    instructions du bloc A ;
Sinon
    instructions du bloc B ;
fin Si
```

Dans le cas d'une alternative avec un **sinon** :

- Le bloc A est délimité par **alors** et **sinon** ;
- Le bloc B est délimité par **sinon** et **fin Si** ;
- La structure est délimitée par **Si** et **fin Si**.

Complément

Pas d'ambiguïté sur la structure

L'utilisation des mots clés **si**, **alors**, **sinon** et **fin si** permet de délimiter précisement chacun des blocs.

```
Si (condition) alors
    instructions du bloc A ;
Sinon
    instructions du bloc B ;
fin Si
```

Dans le cas d'une alternative avec un **sinon** :

- Le bloc A est délimité par **alors** et **sinon** ;
- Le bloc B est délimité par **sinon** et **fin Si** ;
- La structure est délimitée par **Si** et **fin Si**.

Complément

Pas d'ambiguïté sur la structure

L'utilisation des mots clés **si**, **alors**, **sinon** et **fin si** permet de délimiter précisement chacun des blocs.

```
Si (condition) alors
    instructions du bloc A ;
Sinon
    instructions du bloc B ;
fin Si
```

Dans le cas d'une alternative avec un **sinon** :

- Le bloc A est délimité par **alors** et **sinon** ;
- Le bloc B est délimité par **sinon** et **fin Si** ;
- La structure est délimitée par **Si** et **fin Si**.

Complément

Pas d'ambiguïté sur les Si imbriqués, représentation en blocs

```
1   pluie ← Vrai ;
2   pluie_forte ← Faux ;
3   si (pluie) alors
4       si (pluie_forte) alors
5           Écrire("Parapluie !") ;
6       sinon
7           Écrire("Rien de grave !") ;
8       fin si
9   sinon
10      Écrire("Cool.") ;
11  fin si
```

Un **fin Si** rencontré correspond au dernier **Si** ouvert et pas encore fermé. Comme un système de parenthèses.

Complément

Pas d'ambiguïté sur les Si imbriqués, repésentation en lignes

```
1   pluie ← Vrai ;
2   pluie_forte ← Faux ;
3   si (pluie) alors
4   |   si (pluie_forte) alors
5   |   |   Écrire("Parapluie !") ;
6   |   sinon
7   |   |   Écrire("Rien de grave !") ;
8   |   fin si
9   sinon
10  |   Écrire("Cool.") ;
11  fin si
```

Un **fin Si** rencontré correspond au dernier **Si** ouvert et pas encore fermé. Comme un système de parenthèses.

Complément

Pas d'ambiguïté sur les Si imbriqués, représentation en couleurs

```
pluie ← Vrai ;
pluie_forte ← Faux ;
si (pluie) alors      <- Début d'un Si "violet"
  si(pluie_forte) alors
    Écrire("Parapluie !") ;
  sinon
    Écrire("Rien de grave !") ;
  fin si
sinon
    Écrire("Cool.") ;
fin si
```

Complément

Pas d'ambiguïté sur les Si imbriqués, représentation en couleurs

```
pluie ← Vrai ;
pluie_forte ← Faux ;
si (pluie) alors
   si(pluie_forte) alors     <- Début d'un Si "jaune"
     Écrire("Parapluie !") ;
   sinon
     Écrire("Rien de grave !") ;
   fin si
sinon
     Écrire("Cool.") ;
fin si
```

Complément

Pas d'ambiguïté sur les Si imbriqués, représentation en couleurs

```
pluie ← Vrai ;
pluie_forte ← Faux ;
si (pluie) alors
   si(pluie_forte) alors
     Écrire("Parapluie !") ;
   sinon
     Écrire("Rien de grave !") ;
   fin si      <- Fin du Si "jaune"
sinon
     Écrire("Cool.") ;
fin si
```

Complément

Pas d'ambiguïté sur les Si imbriqués, représentation en couleurs

```
pluie ← Vrai ;
pluie_forte ← Faux ;
si (pluie) alors
   si(pluie_forte) alors
     Écrire("Parapluie !") ;
   sinon
     Écrire("Rien de grave !") ;
   fin si
sinon
     Écrire("Cool.") ;
fin si    <- Fin du Si "violet"
```

Complément

Pas d'ambiguïté sur les Si imbriqués, représentation en couleurs

```
pluie ← Vrai ;
pluie_forte ← Faux ;
si (pluie) alors
    si(pluie_forte) alors
        Écrire("Parapluie !") ;
    sinon
        Écrire("Rien de grave !") ;
    fin si
sinon
    Écrire("Cool.") ;
fin si
```

Bloc "A"

Bloc "B"

Complément

Pas d'ambiguïté du tout !

```
pluie ← Vrai ;
pluie_forte ← Faux ;
si (pluie) alors                                  Bloc "A"
    si(pluie_forte) alors                         Bloc "A"
        Écrire("Parapluie !") ;
    sinon                                         Bloc "B"
        Écrire("Rien de grave !") ;
    fin si
sinon                                             Bloc "B"
    Écrire("Cool.") ;
fin si
```

Complément

Pas d'ambiguïté sur les Si imbriqués, représentation en couleurs

```
pluie ← Vrai ;
pluie_forte ← Faux ;
si (pluie) alors                                   Bloc "A"
    si(pluie_forte) alors                          Bloc "A"
        Écrire("Parapluie !") ;    ?
    sinon                                          Bloc "B"
        Écrire("Rien de grave !") ;    ?
    fin si
sinon                                              Bloc "B"
    Écrire("Cool.") ;    ?
fin si
```

Un **fin Si** rencontré correspond au dernier **Si** ouvert et pas encore fermé. Comme un système de parenthèses.

Complément

Pas d'ambiguïté sur les Si imbriqués, représentation en couleurs

```
pluie ← Vrai ;
pluie_forte ← Faux ;
si (pluie) alors                                          Bloc "A"
    si(pluie_forte) alors                                 Bloc "A"
        Écrire("Parapluie !") ;    pluie et pluie_forte
    sinon                                                 Bloc "B"
        Écrire("Rien de grave !") ;    ?
    fin si
sinon                                                     Bloc "B"
    Écrire("Cool.") ;    ?
fin si
```

Un **fin Si** rencontré correspond au dernier **Si** ouvert et pas encore fermé. Comme un système de parenthèses.

Complément

Pas d'ambiguïté sur les Si imbriqués, représentation en couleurs

```
pluie ← Vrai ;
pluie_forte ← Faux ;
si (pluie) alors                                          Bloc "A"
    si(pluie_forte) alors                                 Bloc "A"
        Écrire("Parapluie !") ;    pluie et pluie_forte
    sinon                                                 Bloc "B"
        Écrire("Rien de grave !") ;    pluie mais faible
    fin si
sinon                                                     Bloc "B"
    Écrire("Cool.") ;    ?
fin si
```

Un **fin Si** rencontré correspond au dernier **Si** ouvert et pas encore fermé. Comme un système de parenthèses.

Complément

Pas d'ambiguïté sur les Si imbriqués, représentation en couleurs

```
pluie ← Vrai ;
pluie_forte ← Faux ;
si (pluie) alors                                          Bloc "A"
    si(pluie_forte) alors                                 Bloc "A"
        Écrire("Parapluie !") ;    pluie et pluie_forte
    sinon                                                 Bloc "B"
        Écrire("Rien de grave !") ;    pluie mais faible
    fin si
sinon                                                     Bloc "B"
    Écrire("Cool.") ;    pas de pluie
fin si
```

Un **fin Si** rencontré correspond au dernier **Si** ouvert et pas encore fermé. Comme un système de parenthèses.

Un exercice corrigé

TD1 : "Travail à faire pour la séance suivante"

Compléter l'algorithme ci-dessous, en remplaçant les ... par du texte en rapport avec les alternatives et leurs conditions.

```
1  Programme préparation_alternatives
2     VAR a, b : Entier ;
3  début
4     Écrire("Entrer deux valeurs entières : ") ;
5     Lire(a, b) ;
6     si (a * b > 0) alors
7        Écrire(" ... ") ;
8     sinon
9        si (a * b < 0) alors
10          Écrire(" ... ") ;
11       sinon
12          Écrire(" ... ") ;
13       fin Si
14    fin Si
15 fin
```

TD1 : "Travail à faire pour la séance suivante"

Compléter l'algorithme ci-dessous, en remplaçant les ... par du texte en rapport avec les alternatives et leurs conditions.

```
1  Programme préparation_alternatives
2     VAR a, b : Entier ;
3  début
4     Écrire("Entrer deux valeurs entières : ") ;
5     Lire(a, b) ;
6     si (a * b > 0) alors     /* a * b strictement positif ? */
7     |  Écrire(" ??? ") ;
8     sinon
9     |  si (a * b < 0) alors
10    |  |  Écrire(" ... ") ;
11    |  sinon
12    |  |  Écrire(" ... ") ;
13    |  fin Si
14    fin Si
15 fin
```

TD1 : "Travail à faire pour la séance suivante"

Compléter l'algorithme ci-dessous, en remplaçant les ... par du texte en rapport avec les alternatives et leurs conditions.

```
1  Programme préparation_alternatives
2     VAR a, b : Entier ;
3  début
4     Écrire("Entrer deux valeurs entières : ") ;
5     Lire(a, b) ;
6     si (a * b > 0) alors
7     |  Écrire("a et b ont le même signe") ;
8     sinon
9     |  si (a * b < 0) alors
10    |  |  Écrire(" ??? ") ;
11    |  sinon
12    |  |  Écrire(" ... ") ;
13    |  fin Si
14    fin Si
15 fin
```

TD1 : "Travail à faire pour la séance suivante"

Compléter l'algorithme ci-dessous, en remplaçant les ... par du texte en rapport avec les alternatives et leurs conditions.

```
1  Programme préparation_alternatives
2     VAR a, b : Entier ;
3  début
4     Écrire("Entrer deux valeurs entières : ") ;
5     Lire(a, b) ;
6     si (a * b > 0) alors
7     |  Écrire("a et b ont le même signe") ;
8     sinon
9     |  si (a * b < 0) alors
10    |  |  Écrire(" a et b ont un signe différent ") ;
11    |  sinon
12    |  |  Écrire("???") ;
13    |  fin Si
14    fin Si
15 fin
```

TD1 : "Travail à faire pour la séance suivante"

Compléter l'algorithme ci-dessous, en remplaçant les ... par du texte en rapport avec les alternatives et leurs conditions.

```
1  Programme préparation_alternatives
2     VAR a, b : Entier ;
3  début
4     Écrire("Entrer deux valeurs entières : ") ;
5     Lire(a, b) ;
6     si (a * b > 0) alors
7     |  Écrire("a et b ont le même signe") ;
8     sinon
9     |  si (a * b < 0) alors
10    |  |  Écrire(" a et b ont un signe différent ") ;
11    |  sinon
12    |  |  Écrire("a ou b est nul, ou les deux sont nuls") ;
13    |  fin Si
14    fin Si
15 fin
```

TD1 : "Travail à faire pour la séance suivante"

IMPORTANT : complémentarité

Si la condition est $(a * b > 0)$:

- Si la condition est Vrai : $a * b > 0$
- Sinon , c'est que : $a * b \leq 0$

Exercices

Exo 1

Exercice 1 Simuler l'exécution de l'algorithme suivant, en expliquant ce qu'il est censé faire selon les saisies de l'utilisateur.

```
1  Programme exo1
2     VAR n : Entier ;
3         rép : Caractère ;
4  début
5     Écrire("Voulez-vous saisir un entier ?") ;
6     Lire(rép) ;
7     si (rép = 'o' OU rép = 'O') alors
8        Lire(n) ;
9        si ((n - (n DIV 2) * 2) ≠ 0) alors
10          n ← n + 1 ;
11       fin si
12    fin si
13    Écrire(n) ;
14 fin
```

Exo 1

Exercice 1 Simuler l'exécution de l'algorithme suivant, en expliquant ce qu'il est censé faire selon les saisies de l'utilisateur.

```
1  Programme exo1
2     VAR n : Entier ;
3         rép : Caractère ;
4  début
5     Écrire("Voulez-vous saisir un entier ?") ;
6     Lire(rép) ;
7     si (rép = 'o' OU rép = 'O') alors
8     |   Lire(n) ;
9     |   si ((n - (n DIV 2) * 2) ≠ 0) alors
10    |   |   n ← n + 1 ;
11    |   fin si
12    fin si
13    Écrire(n) ;
14 fin
```

Rappels ? Opérateurs $=$ et $\neq$

- $=$ est l'opérateur d'EGALITE du pseudo-code :
  le prédicat $a = b$ donne Vrai si les évaluations de a et b donnent la même chose, Faux si les deux sont différents.

  L'expression $15 \ = \ 15$ est évaluée à Vrai.
  L'expression $20 \ = \ 15$ est évaluée à Faux.

Rappels ? Opérateurs $=$ et $\neq$

- $=$ est l'opérateur d'EGALITE du pseudo-code :
  le prédicat $a = b$ donne Vrai si les évaluations de a et b donnent la même chose, Faux si les deux sont différents.

  L'expression $15 = 15$ est évaluée à Vrai.
  L'expression $20 = 15$ est évaluée à Faux.

- $\neq$ est l'opérateur de DIFFERENCE du pseudo-code :
  le prédicat $a \neq b$ donne Vrai si les évaluations de a et b donnent des résultats différents, Faux si les deux sont identiques.

  L'expression $15 \neq 15$ est évaluée à Faux.
  L'expression $20 \neq 15$ est évaluée à Vrai.

Rappels ? Opérateurs ET et OU

- **ET** est l'opérateur de **CONJONCTION** du pseudo-code :
  le prédicat **a ET b** donne Vrai si et seulement **les deux expressions** sont évaluées à Vrai, Faux sur tous les autres cas.

Rappels ? Opérateurs ET et OU

- **ET** est l'opérateur de **CONJONCTION** du pseudo-code :
  le prédicat **a ET b** donne Vrai si et seulement **les deux expressions** sont évaluées à Vrai, Faux sur tous les autres cas.

- **OU** est l'opérateur de **DISJONCTION** du pseudo-code :
  le prédicat **a OU b** donne Vrai si **au moins l'une des deux expressions** est évaluée à Vrai, Faux si les deux sont Faux.

Rappels ? Opérateurs ET et OU

- **ET** est l'opérateur de **CONJONCTION** du pseudo-code :
  le prédicat **a ET b** donne Vrai si et seulement **les deux expressions** sont évaluées à Vrai, Faux sur tous les autres cas.

- **OU** est l'opérateur de **DISJONCTION** du pseudo-code :
  le prédicat **a OU b** donne Vrai si **au moins l'une des deux expressions** est évaluée à Vrai, Faux si les deux sont Faux.

a	b	a ET b	a OU b
Faux	Faux	**Faux**	**Faux**
Faux	Vrai	**Faux**	**Vrai**
Vrai	Faux	**Faux**	**Vrai**
Vrai	Vrai	**Vrai**	**Vrai**

[Pour les TP] Traduction pseudo-code <-> Python

Equivalences pseudo-code et Python

Notion	En pseudo-code	En Python
Affectation	`a` $\leftarrow$ `b`	`a = b`
Egalité	`a = b`	`a == b`
Différence	`a` $\neq$ `b`	`a != b`
Conjonction	`a ET b`	`a and b`
Disjonction	`a OU b`	`a or b`
Quotient	`a DIV b`	`a \\ b`
Reste	`a MOD b`	`a % b`
Ecrire	`Ecrire(a)`	`print(a)`
Lire (un caractère)	`Lire(a)`	`a = input()`
Lire (un entier)	`Lire(a)`	`a = int(input())`
Lire (un flottant)	`Lire(a)`	`a = float(input())`

Exo 1 - Quelques explications

```
1  Programme exo1
2     VAR n : Entier ;
3         rép : Caractère ;
4  début
5     Écrire("Voulez-vous saisir un entier ?") ;
6     Lire(rép) ;
7     si (rép = 'o' OU rép = 'O') alors
8     |  Lire(n) ;
9     |  si ((n - (n DIV 2) * 2) ≠ 0) alors
10    |  |  n ← n + 1 ;
11    |  fin si
12    fin si
13    Écrire(n) ;
14 fin
```

- **(n DIV 2) * 2)** : division euclidienne par 2 puis multiplication par deux ;
- n pair : **n = 2p** donne p puis **2p**. Donc **n** - **2p** donne 0 ;
- n impair : **n = 2p+1** donne p puis **2p**. Donc **n** - **2p** donne 1.

Exo 1 - Quelques explications

```
1  Programme exo1
2     VAR n : Entier ;
3         rép : Caractère ;
4  début
5     Écrire("Voulez-vous saisir un entier ?") ;
6     Lire(rép) ;
7     si (rép = 'o' OU rép = 'O') alors
8     |  Lire(n) ;
9     |  si ((n - (n DIV 2) * 2) ≠ 0) alors
10    |  |  n ← n + 1 ;
11    |  fin si
12    fin si
13    Écrire(n) ;
14 fin
```

- Conclusion : Si n est impair **n - (n DIV 2) * 2) ≠ 0** ;
- n pair : Lignes 09-11, on sort du Si sans rien faire, n reste pair.
- n impair : Lignes 09-10-11, on incrémente n de 1 et n devient donc pair !

Exo 1 - Conclusion

Cet algorithme propose à l'utilisateur de fournir un entier.
Si cet entier est pair, il l'affiche.
Si cet entier est impair, il l'incrémente. On affichera donc le nombre pair qui succède à l'entier impair initial.
Fonctionne avec les nombres positifs et négatifs.

Ce programme permet donc d'afficher **le plus petit nombre pair supérieur ou égal à l'entier fourni** par l'utilisateur.

Remarque : le calcul compliqué de la ligne 09 permet en réalité d'obtenir le reste de la division euclidienne par 2. Nous sommes donc parvenu à recréer l'opérateur MOD en le construisant à l'aide de l'opérateur DIV. Pratique si MOD n'existe pas dans le langage de programmation voulu, mais lourd à écrire si MOD est accessible directement.

Remarque 1 : détecter rapidement un nombre pair ou impair

Pour savoir si un nombre est pair ou impair, il suffit d'utiliser MOD 2. Voici un algorithme similaire utilisant directement MOD.

```
1  Programme exo1
2     VAR n : Entier ;
3         rép : Caractère ;
4  début
5     Écrire("Voulez-vous saisir un entier ?") ;
6     Lire(rép) ;
7     si (rép = 'o' OU rép = 'O') alors
8        Lire(n) ;
9        si (n MOD 2 ≠ 0) alors
10          n ← n + 1 ;
11       fin si
12    fin si
13    Écrire(n) ;
14 fin
```

Remarque 2 : obtenir le bon nombre pair

Si n est pair, son reste par 2 est nul. Si n est impair, son reste par 2 vaut 1.
On peut donc faire ceci de façon à se passer du Si :

```
1  Programme exo1
2     VAR n : Entier ;
3         rép : Caractère ;
4  début
5     Écrire("Voulez-vous saisir un entier ?") ;
6     Lire(rép) ;
7     si (rép = 'o' OU rép = 'O') alors
8        Lire(n) ;
9
10       n ← n + n MOD 2 ;
11
12    fin si
13    Écrire(n) ;
14 fin
```

Remarque 3 : taper N au début ?

Si l'utilisateur tape N au début, on atteint la ligne Ecrire(n) sans avoir défini le contenu de n. C'est imprudent.

```
1  Programme exo1
2     VAR n : Entier ;
3         rép : Caractère ;
4  début
5     Écrire("Voulez-vous saisir un entier ?") ;
6     Lire(rép) ;
7     si (rép = 'o' OU rép = 'O') alors
8        Lire(n) ;
9
10       n ← n + n MOD 2 ;
11       Écrire(n) ;
12    fin si
13 fin
```

Le comportement de Ecrire(n) est indéfini et va dépendre du langage du programmation : 0, erreur ou nombre "aléatoire". Autant placer l'instruction dans le bloc Si comme ci-dessus.

Exo 2

Exercice 2

Déterminer ce que fait l'algorithme suivant, en utilisant les trois **jeux de données** pour les variables A,B,C ci-après :

1. A = 5, B = 1, C = 2
2. A = 2, B = 1, C = 5
3. A = 1, B = 5, C = 2

Peut-on faire plus simple ? Si oui, comment ?

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;
5     si(A > B ET A > C) alors
6        Écrire(A) ;
7     sinon
8        si(A > B) alors
9           Écrire(C) ;
10       sinon
11          si(B < C) alors
12             Écrire(C) ;
13          sinon
14             Écrire(B) ;
15          fin si
16       fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;
5     si(A > B ET A > C) alors
6     |  Écrire(A) ;
7     sinon
8     |  si(A > B) alors
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 5, B = 1, C = 2   */
5     si(A > B ET A > C) alors
6     |  Écrire(A) ;
7     sinon
8     |  si(A > B) alors
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 5, B = 1, C = 2   */
5     si(A > B ET A > C) alors     /* Vrai : 5 > 1 ET 5 > 2 */
6     |  Écrire(A) ;
7     sinon
8     |  si(A > B) alors
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 5, B = 1, C = 2   */
5     si(A > B ET A > C) alors     /* Vrai : 5 > 1 ET 5 > 2 */
6     |  Écrire(A) ;               /* Affiche 5             */
7     sinon
8     |  si(A > B) alors
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 5, B = 1, C = 2   */
5     si(A > B ET A > C) alors     /* Vrai : 5 > 1 ET 5 > 2 */
6     |  Écrire(A) ;               /* Affiche 5             */
7     sinon
8     |  si(A > B) alors
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si                       /* fin du Si bleu    l   */
18 fin                            /* fin du programme      */
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 2, B = 1, C = 5   */
5     si(A > B ET A > C) alors
6     |  Écrire(A) ;
7     sinon
8     |  si(A > B) alors
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 2, B = 1, C = 5   */
5     si(A > B ET A > C) alors     /* Faux : 2 > 1 ET 2 > 5 */
6     |  Écrire(A) ;
7     sinon
8     |  si(A > B) alors
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 2, B = 1, C = 5   */
5     si(A > B ET A > C) alors     /* Faux : 2 > 1 ET 2 > 5 */
6     |  Écrire(A) ;
7     sinon                        /* on arrive ici         */
8     |  si(A > B) alors
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 2, B = 1, C = 5   */
5     si(A > B ET A > C) alors     /* Faux : 2 > 1 ET 2 > 5 */
6     |  Écrire(A) ;
7     sinon                        /* on arrive ici         */
8     |  si(A > B) alors           /* Vrai : 2 > 1          */
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 2, B = 1, C = 5   */
5     si(A > B ET A > C) alors     /* Faux : 2 > 1 ET 2 > 5 */
6     |  Écrire(A) ;
7     sinon                        /* on arrive ici         */
8     |  si(A > B) alors           /* Vrai : 2 > 1          */
9     |  |  Écrire(C) ;           /* Affiche 5             */
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;                /* A = 2, B = 1, C = 5   */
5     si(A > B ET A > C) alors       /* Faux : 2 > 1 ET 2 > 5 */
6     |  Écrire(A) ;
7     sinon                          /* on arrive ici         */
8     |  si(A > B) alors             /* Vrai : 2 > 1          */
9     |  |  Écrire(C) ;             /* Affiche 5             */
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si                      /* Fin du Si rouge       */
17    fin si                         /* fin du Si bleu        */
18 fin                               /* fin du programme      */
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 1, B = 5, C = 2   */
5     si(A > B ET A > C) alors
6     |  Écrire(A) ;
7     sinon
8     |  si(A > B) alors
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 1, B = 5, C = 2   */
5     si(A > B ET A > C) alors     /* Faux : 1 > 5 ET 1 > 2 */
6     |  Écrire(A) ;
7     sinon
8     |  si(A > B) alors
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;    /
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;              /* A = 1, B = 5, C = 2   */
5     si(A > B ET A > C) alors     /* Faux : 1 > 5 ET 1 > 2 */
6     |  Écrire(A) ;
7     sinon                        /* on arrive ici         */
8     |  si(A > B) alors
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;                /* A = 1, B = 5, C = 2   */
5     si(A > B ET A > C) alors       /* Faux : 1 > 5 ET 1 > 2 */
6     |  Écrire(A) ;
7     sinon                          /* on arrive ici         */
8     |  si(A > B) alors             /* Faux : 1 > 5          */
9     |  |  Écrire(C) ;
10    |  sinon
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;                /* A = 1, B = 5, C = 2   */
5     si(A > B ET A > C) alors       /* Faux : 1 > 5 ET 1 > 2 */
6     |  Écrire(A) ;
7     sinon                          /* on arrive ici         */
8     |  si(A > B) alors             /* Faux : 1 > 5          */
9     |  |  Écrire(C) ;
10    |  sinon                       /* on arrive ici         */
11    |  |  si(B < C) alors
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;                /* A = 1, B = 5, C = 2   */
5     si(A > B ET A > C) alors       /* Faux : 1 > 5 ET 1 > 2 */
6     |  Écrire(A) ;
7     sinon                          /* on arrive ici         */
8     |  si(A > B) alors             /* Faux : 1 > 5          */
9     |  |  Écrire(C) ;
10    |  sinon                       /* on arrive ici         */
11    |  |  si(B < C) alors          /* FAUX : 5 < 2          */
12    |  |  |  Écrire(C) ;
13    |  |  sinon
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;                /* A = 1, B = 5, C = 2   */
5     si(A > B ET A > C) alors       /* Faux : 1 > 5 ET 1 > 2 */
6     |  Écrire(A) ;
7     sinon                          /* on arrive ici         */
8     |  si(A > B) alors             /* Faux : 1 > 5          */
9     |  |  Écrire(C) ;
10    |  sinon                       /* on arrive ici         */
11    |  |  si(B < C) alors          /* FAUX : 5 < 2          */
12    |  |  |  Écrire(C) ;
13    |  |  sinon                    /* on arrive ici         */
14    |  |  |  Écrire(B) ;
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;                /* A = 1, B = 5, C = 2   */
5     si(A > B ET A > C) alors       /* Faux : 1 > 5 ET 1 > 2 */
6     |  Écrire(A) ;
7     sinon                          /* on arrive ici         */
8     |  si(A > B) alors             /* Faux : 1 > 5          */
9     |  |  Écrire(C) ;
10    |  sinon                       /* on arrive ici         */
11    |  |  si(B < C) alors          /* FAUX : 5 < 2          */
12    |  |  |  Écrire(C) ;
13    |  |  sinon                    /* on arrive ici         */
14    |  |  |  Écrire(B) ;           /* Affiche 5             */
15    |  |  fin si
16    |  fin si
17    fin si
18 fin
```

Exo 2

Exercice 2

```
1  Programme exo2
2     VAR A, B, C : Entier ;
3  début
4     Lire(A, B, C) ;                /* A = 1, B = 5, C = 2   */
5     si(A > B ET A > C) alors       /* Faux : 1 > 5 ET 1 > 2 */
6     |  Écrire(A) ;
7     sinon                          /* on arrive ici         */
8     |  si(A > B) alors             /* Faux : 1 > 5          */
9     |  |  Écrire(C) ;
10    |  sinon                       /* on arrive ici         */
11    |  |  si(B < C) alors          /* FAUX : 5 < 2          */
12    |  |  |  Écrire(C) ;
13    |  |  sinon                    /* on arrive ici         */
14    |  |  |  Écrire(B) ;           /* Affiche 5             */
15    |  |  fin si                   /* Fin du Si orange      */
16    |  fin si                      /* Fin du Si rouge       */
17    fin si                         /* fin du Si bleu        */
18 fin                               /* fin du programme      */
```

Exo 2

L'algorithme permet donc d'afficher la valeur maximale parmi un ensemble de 3 entiers.
Pour faire la même chose avec moins de Si imbriqués, il faudrait :

- Comparer A et B et stocker dans une variable M la plus grande des deux.
- Ensuite, on compare C et M et on affiche la plus grande des deux.

Exo 2

```
1  Programme exo2
2     VAR A, B, C, M : Entier ;
3  début
4     Lire(A, B, C) ;
5     si (A > B) alors
6     |  M ← A ;
7     sinon
8     |  M ← B ;
9     fin Si
10    si (C > M) alors
11    |  M ← C ;
12    fin Si
13    Écrire(M) ;
14 fin
```

Exo 3

Écrire un algorithme qui lit deux valeurs entières et les affiche dans l'ordre décroissant.

Exo 3

```
1  Programme exo3
2     VAR a, b : Entier ;
3  début
4     Lire(a, b) ;
5     si (a > b) alors
6        Écrire(a) ;
7        Écrire(b) ;
8     sinon
9        Écrire(b) ;
10       Écrire(a) ;
11    fin Si
12 fin
```

Exo 3

```
1  Programme exo3
2     VAR a, b, max, min : Entier ;
3  début
4     Lire(a, b) ;
5     si (a > b) alors
6        max ← a ;
7        min ← b ;
8     sinon
9        max ← b ;
10       min ← a ;
11    fin Si
12    Écrire(max) ;
13    Écrire(min) ;
14 fin
```

Exo 4

Écrire un algorithme qui **détermine** et **affiche** le minimum entre deux entiers lus au clavier.
Puis, écrire un autre algorithme qui fait la même chose mais n'utilise pas de sinon et un seul si.

Remarque : attention à **détermine** et **affiche** dans un énoncé d'algorithmique : cela veut dire qu'on veut le mémoriser puis l'afficher. Pas juste le mémoriser ou juste l'afficher.

Exo 4

```
1  Programme exo4A
2     VAR a, b, minimum : Entier ;
3  début
4     Lire(a, b) ;
5     si (a < b) alors
6        minimum ← a ;
7     sinon
8        minimum ← b ;
9     fin Si
10    Écrire(minimum) ;
11 fin
```

Exo 4

```
1  Programme exo4A
2     VAR a, b, minimum : Entier ;
3  début
4     Lire(a, b) ;
5     minimum ← a ;
6     si (b < a) alors
7        minimum ← b ;
8     fin Si
9     Écrire(minimum) ;
10 fin
```

Exo 5

Écrire un algorithme permettant de traduire le jeu suivant :
Deux joueurs A et B cachent leur main droite, choisissent de tendre un certain nombre de doigts (de 0 à 5).

Les deux joueurs se montrent la main droite en même temps.

Si la somme des nombres de doigts montrés est paire, le premier joueur a gagné, sinon c'est le second.

Exo 5

```
1  Programme exo5
2     VAR nbr1, nbr2, total : Entier ;
3  début
4     Écrire("Joueur 1 : donner votre nombre.") ;
5     Lire(nbr1) ;  /*  ensuite, il faudrait effacer l'écran */
6     Écrire("Joueur 2 : donner votre nombre.") ;
7     Lire(nbr2) ;
8     si (nbr1 ≥ 0 ET nbr1 ≤ 5 ET nbr2 ≥ 0 ET nbr2 ≤ 5) alors
9        total ← nbr1 + nbr2 ;
10       si (total MOD 2 = 0) alors      /* total pair ? */
11          Écrire("Joueur 1 gagne avec un total de ", total) ;
12       sinon
13          Écrire("Joueur 2 gagne avec un total de ", total) ;
14       fin Si
15    sinon
16       Écrire("Problème de doigts !") ;
17    fin Si
18 fin
```

Exo 5 en "algo-Python"

```
1  nbr1 :int;
2  nbr2 :int;
3  total:int ;
4  nbr1 = int (input("Joueur 1 : donner votre nombre : ") ) ;
5  # ensuite, il faudrait effacer l'écran
6  nbr2 = int (input("Joueur 2 : donner votre nombre : ") ) ;
7  if (nbr1 >= 0 and nbr1 <= 5 and nbr2 >= 0 and nbr2 <= 5) :
8      total = nbr1 + nbr2 ;
9      if (total % 2 == 0) :       # total pair ?
10         print("Joueur 1 gagne avec un total de ", total) ;
11     else:
12         print("Joueur 2 gagne avec un total de ", total) ;
13 else:
14     print("Problème de doigts !") ;
```

Les $\geq$ et $\leq$ du pseudo-code s'écrivent **>=** et **<=**.
si et **sinon** s'écrivent **if** et **else:** mais le mot clé **alors** s'écrit juste avec le caractère **:**.
Le **fin si** n'existe pas, c'est l'absence des 4 espaces qui indique la sortie du bloc.

Exo 5 en Python

```
1
2
3
4  nbr1 = int (input("Joueur 1 : donner votre nombre : ") )
5  # ensuite, il faudrait effacer l'écran
6  nbr2 = int (input("Joueur 2 : donner votre nombre : ") )
7  if  nbr1 >= 0 and nbr1 <= 5 and nbr2 >= 0 and nbr2 <= 5  :
8      total = nbr1 + nbr2
9      if  total % 2 == 0  :       # total pair ?
10         print("Joueur 1 gagne avec un total de ", total)
11     else:
12         print("Joueur 2 gagne avec un total de ", total)
13 else:
14     print("Problème de doigts !")
```

Les parenthèses ne sont pas nécessaires autour de la condition de l'alternative en Python. Néanmoins, on vous demandera de placer les parenthèses et les déclarations de types lors des TP puisqu'on travaillera en "algo-Python".

Exo 5

Remarque : Interface et vérifications

Dès qu'on a affaire à une interface utilisateur, il est prudent de vérifier les données récupérées avant de commencer le travail. Néanmoins, si on vérifie tout, tout le temps, nos programmes vont finir par être lents : ces données ont peut-être déjà été vérifiées 10 fois à 10 endroits différents !
Il faut donc définir un plan de vérification des données : à partir du moment où une donnée a été vérifiée et filtrée, il est donc inutile de continuer à le faire sur les étapes suivantes.
C'est pour cela qu'on **spécifie** habituellement les algorithmes en fournissant un ensemble de conditions qu'on estime être fiable. On nomme ces conditions les **PRECONDITIONS**.

Ici, l'énoncé était ambigüe sur les nombres de 0 à 5. Précondition (et on n'a pas besoin d'en tenir compte) ou est-ce une simple indication liée au jeu réel (dans la vraie vie, on a entre 0 et 5 doigts) ?
Si le nombre entre 0 et 5 avait été spécifié comme PRECONDITION, il n'aurait pas été nécessaire de le tester par notre programme.

Exo 6

Écrire un algorithme qui saisit les 2 notes d'un étudiant dans une matière (DS1 et DS2), puis calcule sa moyenne (en fonction des deux coefficients lus).
L'algorithme affiche alors la mention obtenue par l'étudiant en fonction de sa moyenne (ajourné, passable, assez-bien, bien, très bien).

Exo 6

Début de la solution :

```
1  Programme exo5
2     VAR ds1, coef1, ds2, coef2, moyenne : Réel ;
3  début
4     /* Acquisition des données */
5     Écrire("Donnez votre note au DS 1 puis son coef : ") ;
6     Lire(ds1, coef1) ;
7     Écrire("Donnez votre note au DS 2 puis son coef : ") ;
8     Lire(ds2, coef2) ;
9
10    /* Gestion des données */
11    moyenne ← (ds1*coef1 + ds2*coef2) / (coef1 + coef2)) ;
12    ...
13
14 fin
```

Exo 6

Solution 1 avec des si imbriqués.

```
1     /* Gestion des données */
2     moyenne ← (ds1*coef1 + ds2*coef2) / (coef1 + coef2)) ;
3     si (moyenne < 10) alors
4        Écrire("Ajourné") ;
5     sinon
6        si (moyenne < 12) alors
7           Écrire("Passable") ;
8        sinon
9           si (moyenne < 14) alors
10             Écrire("Assez-bien") ;
11          sinon
12             si (moyenne < 16) alors
13                Écrire("Bien") ;
14             sinon
15                Écrire("Très bien") ;
16             fin si
17          fin si
18       fin si
19    fin si
```

Exo 6

Solution 1 avec des si imbriqués délimités visuellement.

```
1     /* Gestion des données */
2     moyenne ← (ds1*coef1 + ds2*coef2) / (coef1 + coef2)) ;
3     si (moyenne < 10) alors
4     |  Écrire("Ajourné") ;
5     sinon
6     |  si (moyenne < 12) alors
7     |  |  Écrire("Passable") ;
8     |  sinon
9     |  |  si (moyenne < 14) alors
10    |  |  |  Écrire("Assez-bien") ;
11    |  |  sinon
12    |  |  |  si (moyenne < 16) alors
13    |  |  |  |  Écrire("Bien") ;
14    |  |  |  sinon
15    |  |  |  |  Écrire("Très bien") ;
16    |  |  |  fin si
17    |  |  fin si
18    |  fin si
19    fin si
```

Exo 6

Solution 2 avec des si successifs non imbriquées mais dont les valeurs forment des sous-ensembles disjointes.

```
1     /* Gestion des données */
2     moyenne ← (ds1*coef1 + ds2*coef2) / (coef1 + coef2)) ;
3     si (moyenne < 10) alors
4        Écrire("Ajourné") ;
5     fin si
6     si (moyenne ≥ 10 ET moyenne < 12) alors
7        Écrire("Passable") ;
8     fin si
9     si (moyenne ≥ 12 ET moyenne < 14) alors
10       Écrire("Assez-bien") ;
11    fin si
12    si (moyenne ≥ 14 ET moyenne < 16) alors
13       Écrire("Bien") ;
14    fin si
15    si (moyenne ≥ 16) alors
16       Écrire("Très bien") ;
17    fin si
18 fin
```

Solution 2 de l'exo 6 en "algo-Python"

```
1  ds1:float    = float(input("Note du DS 1 : ") ;
2  coef1:float  = float(input("Coef du DS 1 : ") ;
3  ds2:float    = float(input("Note du DS 2 : ") ;
4  coef2:float  = float(input("Coef du DS 2 : ") ;
5  moyenne:float = (ds1*coef1 + ds2*coef2) / (coef1 + coef2)) ;
6  if (moyenne < 10) :
7      print("Ajourné") ;
8  if (moyenne >= 10 and moyenne < 12) :
9      print("Passable") ;
10 if (moyenne >= 12 and moyenne < 14) :
11     print("Assez-bien") ;
12 if (moyenne >= 14 and moyenne < 16) :
13     print("Bien") ;
14 if (moyenne >= 16) :
15     print("Très bien") ;
```

On a bien une succession de if qui seront tous testés et on fait donc beaucoup de calculs de conditions non nécessaires : si l'élève a eu 8, il est évident pour un humain qu'il ne sert à rien de tester les autres conditions.

Solution 2 de l'exo 6 en Python

```
1  ds1          = float(input("Note du DS 1 : ")
2  coef1        = float(input("Coef du DS 1 : ")
3  ds2          = float(input("Note du DS 2 : ")
4  coef2        = float(input("Coef du DS 2 : ")
5  moyenne      = (ds1*coef1 + ds2*coef2) / (coef1 + coef2))
6  if (moyenne < 10) :
7      print("Ajourné") ;
8  elif (moyenne >= 10 and moyenne < 12) :
9      print("Passable")
10 elif (moyenne >= 12 and moyenne < 14) :
11     print("Assez-bien")
12 elif (moyenne >= 14 and moyenne < 16) :
13     print("Bien")
14 else:
15     print("Très bien")
```

Python possède une structure alternative plus complexe qui intègre le elif : dans une structure **if elif elif elif elif else**, on ne réalisera qu'un seul des blocs. Dès qu'un bloc s'active, on sort de la structure automatiquement. Pratique, mais on ne l'utilisera pas en pseudo-code puisque cela n'existe pas dans tous les langages. C'est bien pénible parfois de ne pas l'avoir...

Remarque finale

En réalité, on devrait tester les valeurs fournies par l'utilisateur AVANT d'en faire la gestion. Une note de 25 sur 20 ou un coefficient de -2 n'a pas beaucoup de sens. Le début du programme devrait donc gérer cela et redemander à l'utilisateur de fournir une valeur correcte. Pour cela, nous avons besoin des schémas répétitifs (boucles non bornées notamment) que nous verrons dans le prochain TD.

Exo 7

On veut écrire un algorithme de calcul du montant d'allocations versées selon le nombre d'enfants d'un foyer, selon les règles suivantes :

- Si le nombre d'enfants est inférieur (strictement à 3) alors l'allocation est de 100 € par enfant.
- Si le nombre d'enfants est supérieur (ou égal) à trois, l'allocation est de :
  - 150 € par enfant pour les trois premiers enfants ;
  - 66 € par enfant pour les suivants.

Questions :

1. Quels jeux de données proposez-vous d'utiliser pour vérifier si un algorithme donné répond correctement à la demande ?
2. Que pensez-vous de l'algorithme fourni ?

Exo 7

Exercice 7

1. Quels jeux de données proposez-vous d'utiliser pour vérifier si un algorithme donné répond correctement à la demande ?

Il faudrait vérifier :

- Une demande négative (0 euro) ;
- Une demande nulle (0 euro) ;
- Une demande pour 1 ou 2 enfants (100 et 200 euros) ;
- Une demande pour 3 enfants (450 euros) ;
- Une demande pour 4 enfants (516 euros) ;
- Une demande pour 10 enfants (912 euros) ;
- Une demande pour 11 enfants (978 euros) ;

Le principe est de vérifier les cas problématiques (valeur maximale et ninimale possible, pair, impair, négatif, seuil où un changement doit opérer, ...)

Exo 7

Que pensez-vous de l'algorithme suivant ?

```
1  Programme exo7
2     VAR ne, m : Entier ;
3  début
4     Écrire("Entrer le nombre d'enfants : ") ; Lire(ne) ;
5     si (ne ≥ 3) alors
6     |   m ← 450 ;
7     |   si (ne > 3) alors
8     |   |   m ← m + ne * 66 ;
9     |   sinon
10    |   |   m ← ne * 100 ;
11    |   fin si
12    fin si
13    Écrire("Montant = ", m) ;
14 fin
```

Exo 7

Que pensez-vous de l'algorithme suivant ?

```
1  Programme exo7
2     VAR ne, m : Entier ;
3  début
4     Écrire("Entrer le nombre d'enfants : ") ; Lire(ne) ;
5     si (ne ≥ 3) alors   /* Pas de sinon pour 2 et - ? */
6     |   m ← 450 ;
7     |   si (ne > 3) alors
8     |   |   m ← m + ne * 66 ;  /* Mauvais calcul de 4 et plus */
9     |   sinon                  /* Sinon mal placé ! */
10    |   |   m ← ne * 100 ;
11    |   fin si
12    fin si
13    Écrire("Montant = ", m) ;
14 fin
```

Exo 7

Correction 1, qui ne gère pas le négatif.

```
1  Programme exo7
2     VAR ne, m : Entier ;
3  début
4     Écrire("Entrer le nombre d'enfants : ") ; Lire(ne) ;
5     si (ne ≥ 3) alors
6     |   m ← 450 ;
7     |   si (ne > 3) alors
8     |   |   m ← m + (ne - 3)  * 66 ;
9     |   fin si
10    sinon
11    |   m ← ne * 100 ;
12    fin si
13    Écrire("Montant = ", m) ;
14 fin
```

Exo 7

Correction 1, qui gère le négatif.

```
1  Programme exo7
2     VAR ne, m : Entier ;
3  début
4     Écrire("Entrer le nombre d'enfants : ") ; Lire(ne) ;
5     si (ne < 0) alors
6     |  Écrire("ERREUR de saisie") ;
7     sinon
8     |  si (ne ≥ 3) alors
9     |  |  m ← 450 ;
10    |  |  si (ne > 3) alors
11    |  |  |  m ← m + (ne - 3)  * 66 ;
12    |  |  fin si
13    |  sinon
14    |  |  m ← ne * 100 ;
15    |  fin si
16    |  Écrire("Montant = ", m) ;
17    fin si
18 fin
```

Exo 8

On souhaite résoudre l'équation du second degré $ax^2 + bx + c = 0$, pour trois coefficients *a*, *b*, *c* réels lus au clavier.
Pour cela, on s'autorisera l'utilisation de deux **fonctions** prédéfinies, *racine* et *puiss*, qui calculent et retournent respectivement la racine carrée d'un nombre donné et l'élévation à la puissance (entière donnée) d'un nombre donné. On pourra ainsi écrire, par exemple :

```
VAR n, r : Réel ;
r ← racine(10) ; /* exemple pour obtenir √10 dans r */
r ← puiss(5, 2) ; /* exemple pour obtenir 5^2 dans r *
Lire(n) ;
r ← racine(n) ; /* exemple pour obtenir √n dans r */
```

Nous allons fournir un programme qui comportera 4 parties pour gérer chacun des cas possibles :

- Cas A : a $= 0$, c'est une équation $bx + c = 0$ du 1er degré ;
- Cas B : Δ est positif, deux solutions réelles possibles.
- Cas D : Δ est nul, une solution réelle double.
- Cas C : Δ est négatif, deux solutions complexes

Voici la structure du programme, où la gestion des cas consiste pour le moment simplement à indiquer son nom à l'aide d'un commentaire. Il faudra bien entendu le compléter.

```
1  Programme exo8
2     VAR a, b, c : Réel :
3         delta, x1, x2, a1, a2, b1, b2 : Réel ;
4  début
5     Écrire("Entrer a, b, c : ") ; Lire(a, b, c) ;
6     si (a = 0) alors
7     |  /* CAS A (1er degré) */
8     sinon
9     |  delta ← puiss(b, 2) - 4 * a * c ;
10    |  si (delta ≥ 0) alors
11    |  |  si (delta > 0) alors
12    |  |  |  /* CAS B (deux solutions réelles) */
13    |  |  sinon
14    |  |  |  /* CAS D (solution réelle double) */
15    |  |  fin si
16    |  sinon
17    |  |  /* CAS C (deux solutions complexes) */
18    |  fin si
19    fin si
20 fin
```

Exercice 8 - Cas A : équation du 1er degré

```
1  Programme exo8
2     VAR a, b, c : Réel :
3         delta, x1, x2, a1, a2, b1, b2 : Réel ;
4  début
5     Écrire("Entrer a, b, c : ") ; Lire(a, b, c) ;
6     si (a = 0) alors
7        /* CAS A (1er degré) */
8        si (b ≠ 0) alors
9           x1 ← -c / b ;
10          Écrire("Solution x = ", x1) ;
11       sinon
12          si (c ≠ 0) alors
13             Écrire("Aucune solution.") ;
14          sinon
15             Écrire("Infinité de solutions.") ;
16          fin si
17       fin si
18    sinon
19       ... /* les autres cas */
```

Les autres cas : calcul du Δ

Si a est non nul, c'est une vraie équation du second degré. Nous allons devoir gérer les trois cas suivants après calcul de $\Delta = b^2 - 4ac$

- **Cas B :** Si $\Delta > 0$, alors l'équation admet deux solutions réelles distinctes :

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a}, \quad x_2 = \frac{-b + \sqrt{\Delta}}{2a}.$$

- **Cas D :** Si $\Delta = 0$, alors l'équation admet une solution réelle double :

$$x = \frac{-b}{2a}.$$

- **Cas C :** Si $\Delta < 0$, alors l'équation n'a pas de solution réelle, mais complexes.

$$x_1 = \frac{-b - i\sqrt{|\Delta|}}{2a}, \quad x_2 = \frac{-b + i\sqrt{|\Delta|}}{2a}.$$

```
1  Programme exo8
2     VAR a, b, c : Réel :
3         delta, x1, x2, a1, a2, b1, b2 : Réel ;
4  début
5     Écrire("Entrer a, b, c : ") ; Lire(a, b, c) ;
6     si (a = 0) alors
7     |  /* CAS A (1er degré) */
8     sinon
9     |  delta ← puiss(b, 2) - 4 * a * c ;
10    |  si (delta ≥ 0) alors
11    |  |  si (delta > 0) alors
12    |  |  |  /* CAS B (deux solutions réelles) */
13    |  |  sinon
14    |  |  |  /* CAS D (solution réelle double) */
15    |  |  fin si
16    |  sinon
17    |  |  /* CAS C (deux solutions complexes) */
18    |  fin si
19    fin si
20 fin
```

Cas B et D : équation du 2er degré, solution(s) réelle(s)

```
1  début
2      Écrire("Entrer a, b, c : ") ; Lire(a, b, c) ;
3      si (a = 0) alors
4      |   /* voir CAS A (1er degré) */
5      sinon
6      |   delta ← puiss(b, 2) - 4 * a * c ;
7      |   si (delta ≥ 0) alors
8      |   |   si (delta > 0) alors
9      |   |   |   /* CAS B (deux solutions réelles) */
10     |   |   |   x1 ← (-b - racine(delta)) / (2*a) ;
11     |   |   |   x2 ← (-b + racine(delta)) / (2*a) ;
12     |   |   |   Ecrire("Deux racines réelles : ", x1, " et ", x2) ;
13     |   |   sinon
14     |   |   |   /* CAS D (solution réelle double) */
15     |   |   |
16     |   |   |
17     |   |   fin si
18     |   sinon
19     |   :   /* Gestion du cas C (complexes) */
```

Cas B et D : équation du 2er degré, solution(s) réelle(s)

```
1  début
2      Écrire("Entrer a, b, c : ") ; Lire(a, b, c) ;
3      si (a = 0) alors
4      |   /* voir CAS A (1er degré) */
5      sinon
6      |   delta ← puiss(b, 2) - 4 * a * c ;
7      |   si (delta ≥ 0) alors
8      |   |   si (delta > 0) alors
9      |   |   |   /* CAS B (deux solutions réelles) */
10     |   |   |   x1 ← (-b - racine(delta)) / (2*a) ;
11     |   |   |   x2 ← (-b + racine(delta)) / (2*a) ;
12     |   |   |   Ecrire("Deux racines réelles : ", x1, " et ", x2) ;
13     |   |   sinon
14     |   |   |   /* CAS D (solution réelle double) */
15     |   |   |   x1 ← -b / (2*a) ;
16     |   |   |   Ecrire("Une racine réelle double : ", x1) ;
17     |   |   fin si
18     |   sinon
19     |   :   /* Gestion du cas D (complexes) */
```

Cas C : équation du 2er degré, solutions imaginaires

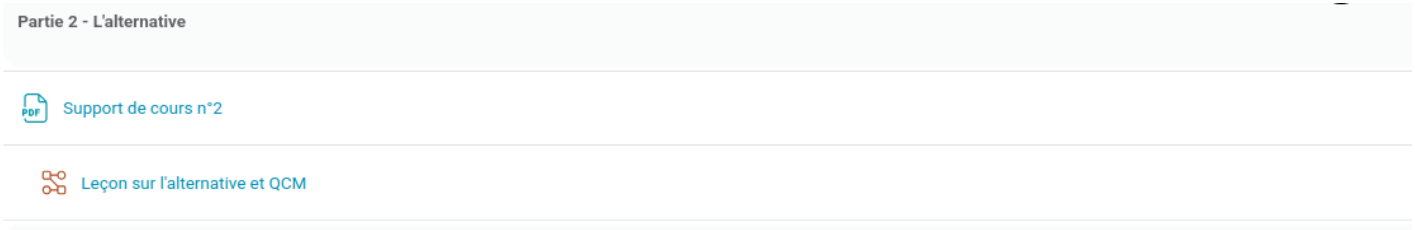
```
1  début
2      Écrire("Entrer a, b, c : ") ; Lire(a, b, c) ;
3      si (a = 0) alors
4          /* voir CAS A (1er degré) */
5      sinon
6          delta ← puiss(b, 2) - 4 * a * c ;
7          si (delta ≥ 0) alors
8              /* CAS B et C précédents */
9          sinon
10             /* Gestion du cas C (complexes) */
11             a1 ← -b / (2*a) ; a2 ← a1 ;
12             b1 ← racine(-delta) / (2*a) ; b2 ← -b1 ;
13             Ecrire("Deux racines complexes : ") ;
14             Ecrire("x1 = ", a1, " - i*", b1) ;
15             Ecrire("x2 = ", a1, " + i*", b2) ;
16         fin si
17     fin si
18 fin
```

Bilan

Bilan

- Principe de l'alternative (Si...Alors - Sinon - fin Si) ;
- Principe de l'alternative (Si...Alors - fin Si) ;
- Savoir ce qu'est une expression booléenne ;
- Savoir suivre le déroulé d'un algorithme contenant une alternative ;
- Visualiser les deux blocs d'une alternative ;
- Savoir exprimer une condition d'égalité $=$, de différence $\neq$;
- Savoir exprimer $>$, $<$, $>=$, $<=$ sans se tromper ! ;
- Connaître la table de vérité du ET et du OU ;
- Savoir gérer les alternatives imbriquées l'une dans l'autre ;
- Savoir gérer les alternatives successives et indépendantes ;
- ALGO : recherche du maximum, du minimum et les variantes ;
- ALGO : recherche des solutions d'une équation de second degré ;

Pour la semaine prochaine

Avec Moodle :

1. Finir le TD 2, notamment les deux exercices en autonomie ;
2. Moodle : lire et réaliser les deux parties proposées (vidéos, petites questions...) sur l'alternative ;
3. Réaliser **sérieusement** le QCM final de la leçon 2.